UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESIGN AND IMPLEMENTATION OF A PREDICTIVE MODULE FOR THE INTRUSION DETECTION SYSTEM SNORT BASED ON SUPERVISED MACHINE LEARNING ALGORITHMS

RUBÉN JIMÉNEZ CALVO

2018

PROYECTO FIN DE CARRERA

Título: Diseño e implementación de un módulo predictivo para el sistema de detección de intrusos snort basado en algoritmos de aprendizaje automático supervisado

Título (inglés): Design and implementation of a predictive module for the intrusion detection system snort based on supervised machine learning algorithms

Autor: D. Rubén Jiménez Calvo

Tutor: D. Carlos A. Iglesias Fernández

Departamento: Departamento de Ingeniería de Sistemas Telemáticos

TRIBUNAL:

Presidente:

Vocal:

Secretario:

Suplente:

Fecha de lectura:

Calificación:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESIGN AND IMPLEMENTATION OF A PREDICTIVE MODULE FOR THE INTRUSION DETECTION SYSTEM SNORT BASED ON SUPERVISED MACHINE LEARNING ALGORITHMS

RUBÉN JIMÉNEZ CALVO

2018

"It is during our darkest moments that we must focus to see the light". Aristotle Onassis

Abstract

Security on computer networks has become a critical topic for many companies and organizations due the security concerns and costs associated that can have a severe impact. Due to increasing traffic using encryption techniques which on one hand increase security on the other hand it helps attackers to hide their illegitimate activities making harder for defenders to detect and protect its infrastructure. This final project defines a machine learning based approach that can be included in Snort by the addition of rules generated by machine learning algorithms. This flow can be continued over the time with supervision to update detection capabilities of the system. Algorithms C5.0, J48, random forest, generalized boosting method and JRip will be evaluated against the NSL-KDD dataset for a binary scenario (normal or anomaly) and for a multiclass scenario (Dos, probe, R2L, U2R and normal).

Keyword: Machine learning, Snort, NSL-KDD, network intrusion detection system, security

Resumen

La seguridad en las redes de computadoras se ha convertido en un tema crítico para muchas compañías y organizaciones debido a las preocupaciones y los costos asociados que pueden tener un impacto severo en ellas. Debido al aumento del tráfico encriptado que, por un lado, aumentan la seguridad, por otro lado, ayuda a los atacantes a ocultar sus actividades ilegítimas, lo que dificulta que los defensores puedan detectar y proteger su infraestructura. Este proyecto final define un enfoque basado en el aprendizaje automático que se puede incluir en Snort mediante la adición de reglas generadas por algoritmos de aprendizaje automático. Este flujo se puede continuar durante el tiempo con supervisión para actualizar las capacidades de detección del sistema. Los algoritmos C5.0, J48, random forest, generalized boosting method y JRip se evaluarán con el conjunto de datos NSL-KDD en un escenario binario (normal o anómalo) y en un escenario multiclase (Dos, probe, R2L, U2R y normal).

Palabras clave: Aprendizaje automático, Snort, NSL-KDD, sistema de detección de intrusos en red, seguridad

Agradecimientos

Agradecer a todas las personas que me han acompañado durante este tiempo.
Agradecer a Carlos Angel Iglesias por la inmensa paciencia durante este tiempo.
También especial mención a mis amigos, lolo, diego y marta que tanto apoyo me han dado durante todo este tiempo aunque fuera en la distancia.
Ha sido un largo viaje, lleno de recuerdos, de personas, de paises, y como todo, ha de terminar en algún momento.
Por último recordar a todos los que ya no pueden estar...

Contents

List of Figures

List of Tables

Acronyms

RF: Random Forest

IDS: intrusion Detection System

NIDS: Network Intrusion Detection System

HIDS: Host Based Intrusion Detection System

OISF: Open Information Security Foundation

UNB: University of New Brunswick

RMSE: Root Mean Squared Error

GBM: Generalized Boosting Method

KDD: Knowledge Discovery in Databases

CHAPTER 1

INTRODUCTION

In this chapter we will introduce the objectives of this Final Project as well as the motivation behind it. It will also describe the structure of this document.

1.1 Context

Internet services have encouraged people to communicate and exchange information on a daily basis. This provides convenience and benefits such as shortening the effective geographical distances and efficiently sharing information. On the other hand, with the information exchange over the computing environment, a possible problem is that communications over the network may be compromised by hackers.

With the rapid increase of our everyday life dependency to Internet-based services network security has become a critical issue. According to the Cisco 2017 Annual Security Report [1] the trend in the number of alerts continues upwards during the recent years.

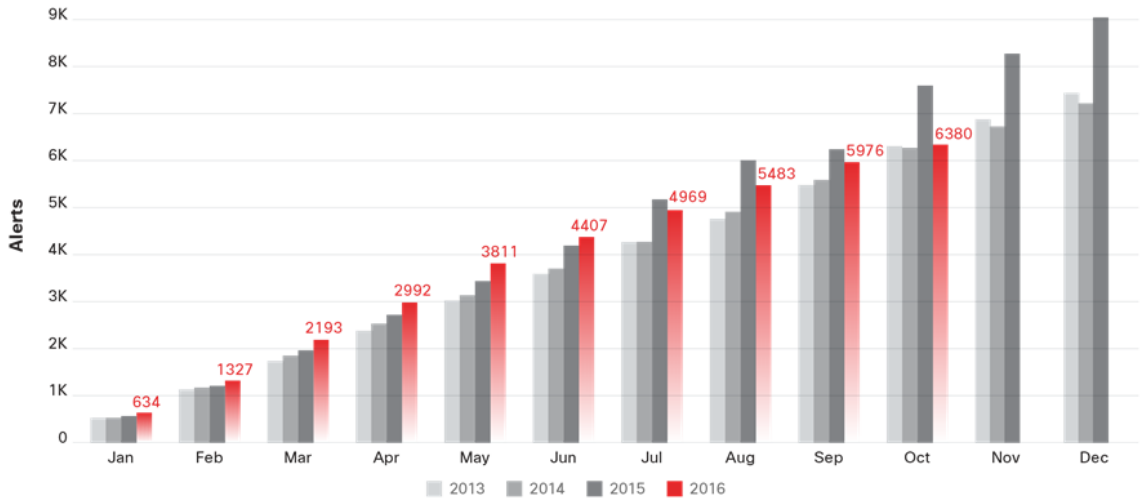


Figure 1.1: Cumulative attacks by Cisco report 2017 [1]

Computer attacks, e.g. the use of specialized methods to overcome the security policy of an organization, are becoming more and more common. Fortunately, policies and tools are being developed to provide increasingly efficient defence mechanisms. For instance IDSs are installed to identify such attacks and to react by usually generating an alert or blocking suspicious activity. The goal of intrusion detection is to identify malicious activity in a stream of monitored data.

Cisco report [1] also states that despite advances by the security industry, criminals continue to evolve their approaches to break through security defences, so keeping up to date all security systems is nowadays a big challenge for many organizations.

The majority of current intrusion detection systems (IDS) is based on signatures, specific pre-defined patterns matching known functionality of attacks. The main limitation of the signature-based approach is its inability to identify novel attacks.

Machine learning offers a major opportunity to improve quality and to facilitate up to date security system by using detected attacks. If an intrusion is detected quickly enough, an intruder can be identified quickly and ejected from the system before any damage is done or any data are compromised.

1.2 Problem statement

Networks attacks have raised concerns about security of customer personal data and impact on activities in companies from all different backgrounds. These concerns have a financial cost and issues occurring due to attacks can take months or years to resolve. All these breaches inside company infrastructure can have a real economic damage to the organizations.

As it has been highlighted in studies from Cisco 2017 Annual Security Report [1] more than half (53 percent) of all attacks resulted in financial damages of more than US *$500.000*, including, but not limited to, lost revenue, customers, opportunities, and out-of-pocket costs. This is illustrated in Figure 1.2.

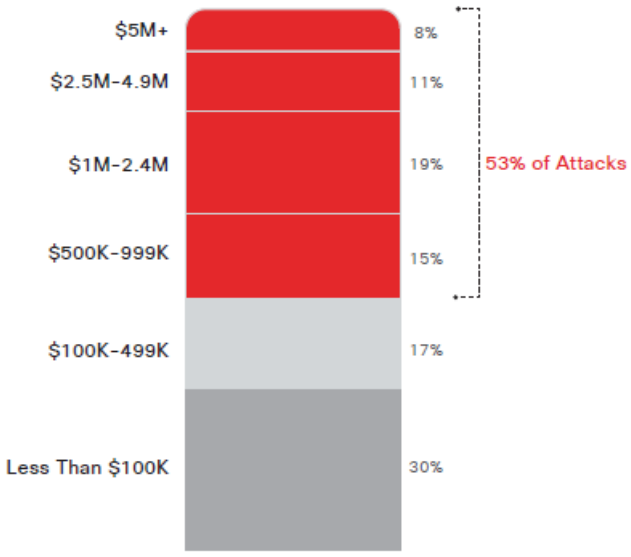


Figure 1.2: Attack cost by Cisco report 2018 [1]

1.3 Motivation and approach

In order to preserve user privacy encryption techniques over the web has been taken as a standard approach for a secure way of transferring sensitive information such passwords, credit card details. While this is positive and harden attackers to obtain illegitimate information it is also used by them to hide their illegal activities from Telecom operators and organizations. This increase in encrypted traffic by attacker is showed in Figure 1.3:

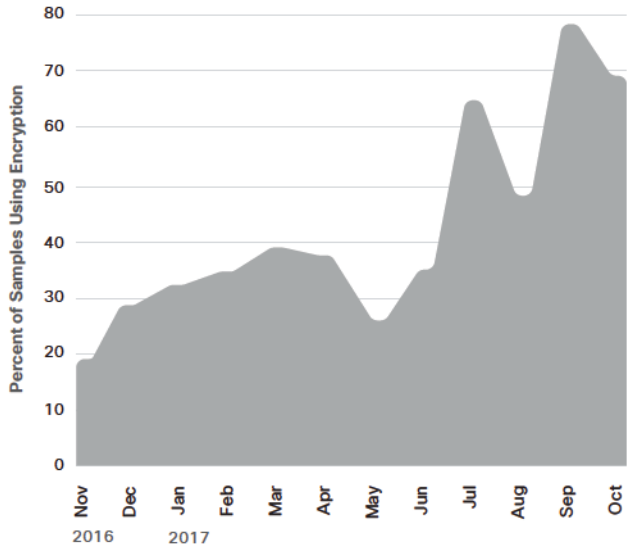


Figure 1.3: Volume of malicius encrypted network communication by Cisco report 2018 [1]

The increasing volume of encrypted traffic, whether is legitimate or malicious, create more challenges for defenders to identify and monitors possible threats to their organizations. To overcome this situation organizations would need to adapt and include more automated and advanced techniques like machine learning and artificial intelligence to enhance their prevention, monitoring, detection and correction capabilities.

These capabilities will enhance network security defences and over the long term they will "learn" how to automatically detect unusual patterns in web traffic that may indicate malicious activity both known and novel threats. This would reduce time to operate from attackers and reduce cost and impact associated with attacks. Machine learning and artificial intelligence taxonomy is showed in 1.4 by Cisco 2018 Annual Security Report.

Real value of those techniques, specially when considering encrypted traffic, is the ability to detect "known-unknown" threats (previously unseen variations of known threats, malware subfamilies, or related new threats) and "unknown-unknown" (net-new malware) threats. This project would evaluate this capabilities attempting to prove that novels attacks can be detected by learning from previous known attacks through machine learning.

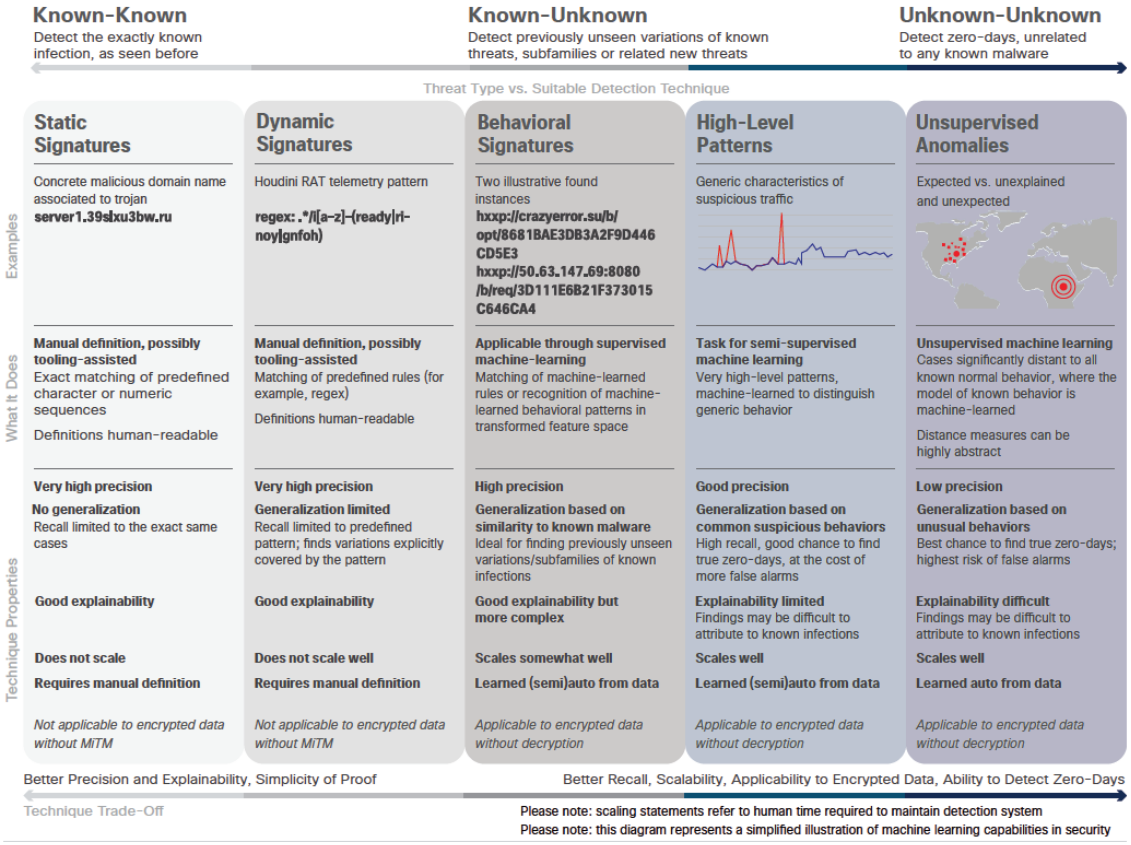


Figure 1.4: Machine learning taxonomy by Cisco report 2018 [1]

1.4 Methodology

Development of this final thesis has followed several phases which are explained in the following points:

Research. This phase has covered compilation of information in areas such as machine learning and network intrusion detection systems. Different tools and datasets were evaluated in detail to choose the one that fits more for purpose.

Development. This phase comprises dataset preparation and script development used for executing the experiments. For this phase a subset of full dataset was used in order to improve efficiency of the whole programming cycle and hence reduce time.

Performance evaluation. Once everything is in place specified experiments were executed to the whole dataset and results collected.

Results analysis and conclusion. With all results available from different algorithms, they were studied and processed to obtain the final conclusions of this project.

Documenting project work. Task of documenting all information produced during all the previous stages was done in parallel to the previous steps. However, a final review and completion of produced information was required as final phase. Final document has been produced using LaTeX.

1.5 Project Outline

This final project is divided into 7 chapters:

- Chapter 1 called Introduction: It gives an understanding of the problem statements and why a machine learning approach can provide additional value to standard NIDS.
- Chapter 2 called Background: It provides understanding of current NIDS platforms, machine learning approaches and available datasets.
- Chapter 3 called Project approach, it provides general description of the system flow including all elements involved such NSL-KDD dataset, Snort and R Caret package.
- Chapter 4 called Experiment Design outlines different algorithms used and how relevant functionality was configured.
- Chapter 5 called Result Analysis: Binary. It shows obtained results for all experiments performed on the binary scenario, i.e normal/anomaly.
- Chapter 6 called Result Analysis: Multiclass. It shows obtained results for all experiments performed in multiclass scenario that consist of several classes: normal, Dos, Probe, R2L and U2R.
- Chapter 7 called Conclusion: A final summary of the project where main conclusions obtained from the experimental results are showed.

CHAPTER 2

BACKGROUND

In this chapter an overview of intrusion detection systems, machine learning techniques, available tools and datasets is provided.

2.1 Intrusion Detection Systems

An intrusion detection system (IDS) is a device or software application that monitors a network or systems for malicious activity or policy violations.

The most common classifications are network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). A system that monitors important operating system files is an example of a HIDS, while a system that analyses incoming network traffic is an example of a NIDS. A Host based IDS resides on the system being monitored. It consists of an agent on a host which identifies intrusions by analyzing system calls, application logs, file system modifications and other host activities and state. A Network Based Intrusion Detection System monitors and analyses the traffic on its network segment to detect intrusion attempts.

It is also possible to classify IDS by detection approach: the most well-known variants are signature-based detection (recognizing bad patterns, such as malware) and anomaly-based detection (detecting deviations from a model of "good" traffic, which often relies on machine learning). This will be covered in more detail in following sections with information from [2] [3].

2.1.1 Signature Based Detection

Commonly called signature detection, this method uses specifically known patterns of unauthorized behaviour to predict and detect subsequent similar attempts. These specific patterns are called signatures. The occurrence of a signature might not signify an actual attempted unauthorized access (for example, it can be an honest mistake), but it is a good idea to take each alert seriously. Depending on the robustness and seriousness of a signature that is triggered, some alarm, response, or notification should be sent to the proper authorities.

The main advantage of misuse detection system is their ability to detect known attacks and the relatively low false alarm rate when rules are correctly defined. On the other hand only known attacks can be detected.

2.1.2 Anomaly Based Detection

These techniques are designed to uncover abnormal patterns of behaviour, the IDS establishes a baseline of normal usage patterns, and anything that widely deviates from it gets flagged as a possible intrusion.

Anomaly-based intrusion detection systems were primarily introduced to detect unknown attacks. The basic approach is to use machine learning to create a model of trustworthy activity, and then compare new behaviour against this model. Although this approach enables the detection of previously unknown attacks, it may suffer from false positives: previously unknown legitimate activity may also be classified as malicious.

As this detection technique allow detections of new attacks it will be the base for this project as one of the main goals is that through machine learning new anomalies can be detected through training with previous known attacks.

2.2 Machine Learning

Machine learning is a sub-field of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.

Although machine learning is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. Machine learning algorithms instead allow for computers to train on data inputs and use statistical analysis in order to output values that fall within a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs.

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.These are supervised learning which trains algorithms based on example input and output data that is labelled by humans, and unsupervised learning which provides the algorithm with no labelled data in order to allow it to find structure within its input data.

2.2.1 Supervised Learning

In supervised learning, the computer is provided with example inputs that are labelled with their desired outputs. The purpose of this method is for the algorithm to be able to “learn” by comparing its actual output with the “taught” outputs to find errors, and modify the model accordingly. Supervised learning therefore uses patterns to predict label values on additional unlabelled data.

Supervised learning problems can be further divided into two parts, namely classification, and regression. A classification problem is when the output variable is a category or a group and a regression problem is when the output variable is a real value.

2.2.2 Unsupervised Learning

In unsupervised learning, data is unlabelled, so the learning algorithm is left to find commonalities among its input data. As unlabelled data are more abundant than labelled data, machine learning methods that facilitate unsupervised learning are particularly valuable.

The goal of unsupervised learning may be as straightforward as discovering hidden patterns within a dataset, but it may also have a goal of feature learning, which allows the computational machine to automatically discover the representations that are needed to classify raw data.

2.3 Available NIDS

This section will cover an overview of several NIDS currently available: BRO IDS, Suricata and SNORT.

2.3.1 BRO IDS

Bro [4] is a passive, open-source network traffic analyser. It is primarily a security monitor that inspects all traffic on a link in depth for signs of suspicious activity. More generally, however, Bro supports a wide range of traffic analysis tasks even outside of the security domain, including performance measurements and helping with trouble-shooting.

Bro provides the following capabilities including:

- Deep packet inspection
- Attack and anomaly detection
- Event correlation
- Alert generation
- Full IPv6 and IPv4 support
- A powerful, flexible policy scripting language
- Scalable, clustering architecture

BRO is based in a modular software stack with three components: packet processing layer, event engine and a policy script interpreter. This IDS is both a signature and anomaly-based IDS. Its analysis engine will convert traffic captured into a series of events. An event could be a user logon to FTP, a connection to a website or practically anything. The power of the system is what comes after the event engine and that's the Policy Script Interpreter. This policy engine has it's own language (Bro-Script) and it can do some very powerful and versatile tasks.

Tool is very flexible but that comes to be a disadvantage and it is fairly complicated to use, compared to other tools, including its own scripting language for those reason this tool was discarded.

2.3.2 SNORT

Snort is a free and open source network intrusion detection and prevention tool. It was created by Martin Roesch [5] in 1998. Snort is now developed by Sourcefire which has been owned by Cisco since 2013. It is based on rules which are easy to write for intrusion detection and it is highly flexible. The main advantage of using Snort is its capability to perform real-time traffic analysis and packet logging on networks. With the functionality of protocol analysis, content searching and various pre-processors, Snort is widely accepted as a tool for detecting varied worms, exploits, port scanning and other malicious threats.

It can be configured in three main modes — sniffer, packet logger and network intrusion detection.

- In sniffer mode, the program will just read packets and display the information on the console.
- In packet logger mode, the packets will be logged on the disk.
- In intrusion detection mode, the program will monitor real-time traffic and compare it with the rules defined by the user.

This is the tool that has been chosen for this project being the main reason the easy and flexible way of rule format and also because of the good community support for solving problems which is something positive. The availability of plugins supporting KDD metrics was also a critical reason.

2.3.3 Suricata

Suricata[6] is an open source threat detection engine that was developed by the Open Information Security Foundation (OISF), a non-profit foundation committed to ensuring Suricata's development and sustained success as an open source project.

Main features of Suricata are:

- Suricata is written in C, Lua
- JSON output supported
- Support for Lua scripting
- Support for pcap (packet capture)

Suricata is rules-based and while it offers compatibility with Snort Rules, it also introduced multi-threading, which provides the theoretical ability to process more rules across faster networks, with larger traffic volumes, on the same hardware.

Suricata inspects the network traffic using a powerful and extensive rules and signature language, and has powerful Lua scripting support for detection of complex threats.

The Suricata engine is capable of real-time intrusion detection, inline intrusion prevention and network security monitoring. Suricata consists of a few modules like Capturing, Collection, Decoding, Detection and Output. It captures traffic passing in one flow before decoding, which is highly optimal.

The downside to Suricata it is a little more complex to install and the community is smaller than what Snort has amassed for that reason Snort was the preferred NIDS to use.

2.4 Datasets

Several datasets were evaluated in order to define which one to be used for the purpose of this project.

2.4.1 UNB ISCX IDS 2012

ISCX 2012 is a benchmark intrusion detection dataset with contains 7 days of synthetically recorded packet details replicating the real time network traffic by labelling the attacks. It consists of labelled network traces, including full packet payloads in pcap format.
newpage This dataset is built on the concept of profiles that include the details of intrusions.

As described in [7] and [8] where the creators of the dataset identify a set of requirements for effective dataset generation, the datasets were collected using a real-time testbed by incorporating multi-stage attacks. They used two profiles α and β during the generation of the datasets. α profiles are constructed using the knowledge of specific attacks and β profiles are built using the filtered traffic traces. Real packet traces were analysed to create α and profiles for agents that generate real-time traffic for HTTP, SMTP, SSH, IMAP, POP3 and FTP protocols. Various multi-stage attack scenarios were explored to generate malicious traffic.

The UNB ISCX 2012 Intrusion Detection Evaluation Data Set has following characteristics according to official dataset description [9]:

- Realistic network and traffic: Ideally, a dataset should not exhibit any unintended properties, both network and traffic wise. This is to provide a clearer picture of the real effects of attacks over the network and the corresponding responses of workstations. For this reason, it is necessary for the traffic to look and behave as realistically as possible. This includes both normal and anomalous traffic.
- Labelled dataset: Creating a dataset in a controlled and deterministic environment allows for the distinction of anomalous activity from normal traffic and therefore eliminates the impractical process of manual labelling.
- Total interaction capture: The amount of information available to detection mechanisms are of vital importance as this provides the means to detect anomalous behaviour. Thus, it is a major requirement for a dataset to include all network interactions, either within or between internal LANs.
- Complete capture: Most of traces are either used internally, which limits other researchers from accurately evaluating and comparing their systems, or are heavily anonymized with the payload entirely removed resulting in decreased utility to researchers. For this dataset the need of any sanitization is removed as network traces were generated in a controlled testbed environment.
- Diverse intrusion scenarios: Dataset contains several types of threats attempting to cover more complex schemes observed nowadays by using multi-stage attacks each carefully crafted and aimed towards recent trends in security threats.

Files are available for researchers and only it is required to contact researcher to get temporary access to a remote server with all files. Files have a considerable size with several GB per day, from 4GB up to 23GB. For each day it is specified if there attacks present or if it just contains normal traffic.

Besides the pcap files dataset contains more files such as listofmaliciousips which contains all IPs involved in different attack scenarios and labelled flows xml which contains detailed flow information in XML format for each day. The flows were generated using IBM QRadar appliance. The XML files contain the following features: appName, totalSourceBytes, totalDestinationBytes, totalDestinationPackets, totalSourcePackets, sourcePayloadAsBase64, destinationPayloadAsBase64, destinationPayloadAsUTF, direction, sourceTCPFlagsDescrip tion, destinationTCPFlags Description, source, "protocolName", sourcePort, destination, destinationPort, startDateTime, stopDateTime and Tag.

Due to the huge size of the dataset, the limited number of features provided in the xml files, while more could be obtained from the raw file, it was decided to chose other datasets in order to focus on the machine learning field rather than spending too much time processing the dataset.

2.4.2 KDD cup99

Since 1999, the KDDcup99 dataset [10] has been the most widely used dataset for evaluation of network based anomaly detection methods and systems.

This dataset was built based in the data captured in the DARPA98 [11] IDS evaluation program prepared and managed by MIT Lincoln Labs and it was used for a competition whose task [15] was to build a network intrusion detector, a predictive model capable of distinguishing between bad connections, called intrusions or attacks, and good normal connections. The database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks.

The raw training data was about four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was processed into about five million connection records. Similarly, the two weeks of test data yielded around two million connection records.

A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labelled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

The KDD training dataset consists of approximately 4.900.000 single connection vectors, each of which contains 41 features and is labelled as either normal or attack of a specific attack type. The test dataset contains about 300.000 samples with a total 24 training types, with an additional 14 attack types in the test dataset only.

The represented attacks are mainly four types:

- Denial of Service (DoS): An attacker attempts to prevent valid users from using a service provided by a system.
- Remote to Local (r2l): Attackers try to gain entrance to a victim machine without having an account on it.
- User to Root (u2r): Attackers have access to a local victim machine and attempt to gain privilege of a superuser.
- Probe: Attackers attempt to acquire information about the target host.

Background traffic was simulated and the attacks were all known. The testing set also consists of simulated background traffic and known attacks, including some attacks that are not present in the training set.

2.4.3 NSL-KDD

NSL-KDD is a dataset suggested to solve some of the inherent problems of the KDD'99 data set which are mentioned in [12]. NSL-KDD has been generated by removing redundant and duplicate instances, also by decreasing size of dataset.

The NSL-KDD data set has the following advantages over the original KDD data set according dataset description [13]

- This dataset doesn't contain superfluous and repeated records in the training set, so classifiers or detection methods will not be biased towards more frequent records.
- There are no duplicate records in the test set. Therefore, the performance of learners is not biased by the methods which have better detection rates on frequent records.
- The number of selected records from each difficulty level is inversely proportional to the percentage of records in the original KDD dataset. As a result, the classification rates of various machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of various learning techniques.
- The number of records in the training and testing sets is reasonable, which makes it practical to run experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research groups are consistent and comparable.

The NSL-KDD dataset consists of two parts: (i) KDDTrain+ and (ii) KDDTest+. The KDDTrain+ part of the NSL-KDD dataset is used to train a detection method or system to detect network intrusions. It contains four classes of attacks and a normal class dataset. The KDDTest+ part of NSLKDD dataset is used for testing a detection method or a system when it is evaluated for performance. It also contains the same classes of traffic present in the training set.

This dataset was chosen for using it in this project as it addressed issues present in previous KDD99 dataset and also it directly provide labelled files in text format that can be directly used as input for any machine learning framework.

2.5 Machine leaning tools

Machine learning and data analysis are two areas where open source has become almost the de facto license for innovative new tools. Both the Python and R languages have developed robust ecosystems of open source tools and libraries that help data scientists of any skill level more easily perform analytical work. Next subsections will provide an overview of R and Python related to machine learning including mention to some relevant packages [14].

2.5.1 Python

Even though Python is naturally disposed toward machine learning, it has packages that further optimize this attribute. PyBrain is a modular machine learning library that offers powerful algorithms for machine learning tasks. The algorithms are intuitive and flexible, but the library also has a variety of environments to test and compare your machine learning algorithms.

Scikit-learn is the most popular machine learning library for Python. Built on NumPy and SciPy, scikit-learn offers tools for data mining and analysis that bolster Python's already-superlative machine learning usability. NumPy and SciPy impress on their own. They are the core of data analysis in Python and any serious data analyst is likely using them raw, without higher-level packages on top, but scikit-learn pulls them together in a machine learning library with a lower barrier to entry.

When it comes to data analysis, Python receives a welcome boost from several different packages. Pandas, one of its most well-known data analysis packages, gives Python high-performance structures and data analysis tools.

2.5.2 R

R, like Python, has plenty of packages to boost its performance. When it comes to approaching parity with Python in machine learning, Nnet improves R by supplying the ability to easily model neural networks. Caret is another package that bolsters R's machine learning capabilities, in this case by offering a set of functions that increase the efficiency of predictive model creation.

But data analysis is R's domain, and there are packages to improve it beyond its already-stellar capabilities. Packages for the pre-modelling, modelling, and post-modelling stages of data analysis are available. These packages are directed at specific tasks like data visualization, continuous regression, and model validation.

Caret package provide a very easy way to run experiments and one of the very interesting options is the fact that hyperparameter tuning can be performed to chose the best set of parameters to be used. This was found a very interesting feature to not necessarily use default parameters in selected algorithm and has make R with Caret package the tool selected for the machine learning part if the project.

CHAPTER 3

PROJECT APPROACH

In this chapter the general flow of the system is described including all elements involved such NSL-KDD dataset, Snort and R Caret package.

3.1 Overview

This section will cover general flow that could be used so a NIDS could benefit from machine learning generated knowledge. The system is based on SNORT which consists of user defined rule sets for the identification of specific attacks. With the help of various plugins and rule sets, SNORT is capable of performing signature based, protocol based and anomaly based detections on given input traffic.

Main idea is that additional set of rules generated by machine learning models are included into SNORT which has some additional plugins in the preprocessor and in the output modules to use defined fields on the KDD dataset. Those plugins come from an external project covered in section 3.4.3. General flow is illustrated in Figure 3.1.

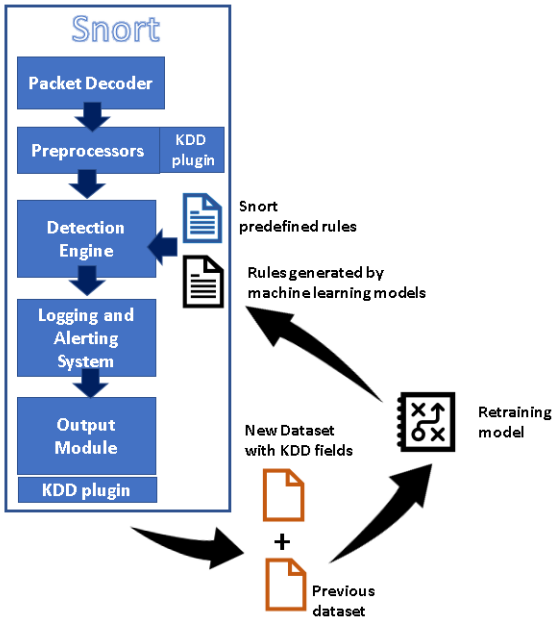


Figure 3.1: NIDS machine learning flow

Initially a set of rules is generated with the existing NSL-KDD dataset and a rule files is obtained from different models evaluated. Objective of the project is to identify best model that suits for this dataset features.

Next steps would be based on identifying new data from Snort execution on the network. This would produce a new dataset either from standard Snort rules or from rules generated by machine learning models. This dataset would be combined and reviewed by an individual to re-train the system and obtain an updated set of rules. This process would be repeated in the time so new set of rules can be developed with new attack detection capabilities.

Next sections would cover following topics:

- Highlight main objectives of the project
- Describe in detail NSL KDD dataset: available files, attacks, fields..
- SNORT description and KDD features implementation
- R package CARET used for model evaluation

3.2 Objectives

The main objective of this final project consists of evaluating the benefits of including machine learning algorithms to Snort NIDS. The main task will be to build a predictive model capable to distinguishing between attack connections and normal traffic.

Several academic research papers have suggested that anomaly detection have the potential for addressing novel attacks and also that most novel attacks are variants of known attacks. Therefore a NIDS that can effectively learn and correlate unknown attacks is desirable.

So one of the objectives will attempt to prove that new attacks can be detected by the knowledge of patterns obtained through machine learning algorithms that are defined by other attacks.

During the development of the project several algorithms, which can directly produce rules, will be evaluated and performance will be compared in order to find the optimal model.

In addition, evaluation of what is best approach for defining anomaly behaviours, meaning categorizing traffic only in normal/anomaly or having several categories for anomaly traffic, in our case DoS, probe, U2R, R2l would be performed.

3.3 NSL KDD Dataset

NSL KDD dataset has been chosen as data to be used for evaluation of different models in this project. As it has been explained in [12] this dataset address several problems on the original KDD dataset. It also provide labelled data with train and test dataset making it convenient for this project. Following sections will describe in detail this dataset that still have similarities with legacy KDD 99 dataset.

3.3.1 Dataset Overview

NSL KDD Dataset is based on original KDD data set which includes multiple attacks falling into four main categories Dos (denial-of-service, e.g. syn flood), probe (surveillance and other probing, e.g., port scanning), R2L (unauthorized access from a remote machine, e.g. guessing password) and U2R (unauthorized access to local superuser (root) privileges, e.g., various buffer overflow). In addition also normal traffic is included.

The NSL-KDD data set has the following advantages over the original KDD data set as described in [13]:

- It does not include redundant records in the train set, so the classifiers will not be biased towards more frequent records.
- There are no duplicate records in the proposed test sets; therefore, the performance of the learners are not biased by the methods which have better detection rates on the frequent records.
- The number of selected records from each difficulty level group is inversely proportional to the percentage of records in the original KDD data set. As a result, the classification rates of distinct machine learning methods vary in a wider range, which makes it more efficient to have an accurate evaluation of different learning techniques.
- The number of records in the train and test sets are reasonable, which makes it affordable to run the experiments on the complete set without the need to randomly select a small portion. Consequently, evaluation results of different research works will be consistent and comparable.

3.3.2 Data files

Full dataset includes several files:

- KDDTrain+.ARFF: The full NSL-KDD train set with binary labels in ARFF format
- KDDTrain+.TXT: The full NSL-KDD train set including attack-type labels and difficulty level in CSV format
- KDDTrain+_20Percent.ARFF: A 20% subset of the KDDTrain+.arff file
- KDDTrain+_20Percent.TXT: A 20% subset of the KDDTrain+.txt file
- KDDTest+.ARFF: The full NSL-KDD test set with binary labels in ARFF format
- KDDTest+.TXT: The full NSL-KDD test set including attack-type labels and difficulty level in CSV format

- KDDTest-21.ARFF: A subset of the KDDTest+.arff file which does not include records with difficulty level of 21 out of 21
- KDDTest-21.TXT: A subset of the KDDTest+.txt file which does not include records with difficulty level of 21 out of 21

Approach taken for this project would be make use of txt files that contains attack label and process them to generate additional columns such attack profile (Dos, Probe, U2R, R2L, normal), class (normal, anomaly) so both class and multi class evaluation can be performed, this is covered in more detail is Section 4.4.

3.3.3 Attack distributions

Dataset was loaded into R environment for inspection. This section will cover attack distribution over the train, 20 % train and test datasets.

Following tables shows traffic type distribution for binary or multi class classification obtained by inspecting dataset.

Traffic Type	Train Dataset	Train Dataset 20%	Test Dataset
Normal	67343	11743	12833
Anomaly	58630	13449	9710

Table 3.1: Normal/Anomaly samples on NSL KDD dataset files

Traffic profile distribution is presented in following table:

Traffic Type	Train Dataset	Train Dataset 20%	Test Dataset
DoS	45927	9234	7458
normal	67343	13449	9710
Probe	11656	2289	2421
R2L	995	209	2887
U2R	52	11	67

Table 3.2: Attack profile samples on NSL KDD dataset files

On the train dataset attack distribution is presented in following table:

Traffic profile	Attack	Train Dataset	Train Dataset 20%
DoS	back	956	196
	land	18	1
	neptune	41214	8282
	pod	201	38
	smurf	2646	529
	teardrop	892	188
Probe	ipsweep	3599	710
	nmap	1493	301
	portsweep	2931	587
	satan	3633	691
R2L	ftp_write	8	1
	guess_passwd	53	10
	imap	11	5
	multihop	7	2
	phf	4	2
	spy	2	1
	warezclient	890	181
	warezmaster	20	7
U2R	buffer_overflow	30	6
	loadmodule	9	1
	perl	3	NA
	rootkit	10	4
normal	normal	67343	13449

Table 3.3: Distribution of attacks in train dataset

Similarly test dataset attack distribution is showed in the following table where an additional column indicating that the specific attack is novel or not is included. An attack is consider novel if it is not part of the training dataset.

Traffic profile	Attack	Test dataset count	Is novel attack
DoS	apache2	737	**TRUE**
	back	359	FALSE
	land	7	FALSE
	mailbomb	293	**TRUE**
	neptune	4657	FALSE
	pod	41	FALSE
	processtable	685	**TRUE**
	smurf	665	FALSE
	teardrop	12	FALSE
	udpstorm	2	**TRUE**
Probe	ipsweep	141	FALSE
	mscan	996	**TRUE**
	nmap	73	FALSE
	portsweep	157	FALSE
	saint	319	**TRUE**
	satan	735	FALSE
R2L	ftp_write	3	FALSE
	guess_passwd	1231	FALSE
	httptunnel	133	**TRUE**
	imap	1	FALSE
	multihop	18	FALSE
	named	17	**TRUE**
	phf	2	FALSE
	sendmail	14	**TRUE**
	snmpgetattack	178	**TRUE**
	snmpguess	331	**TRUE**
	warezmaster	944	FALSE
	worm	2	**TRUE**
	xlock	9	**TRUE**
	xsnoop	4	**TRUE**
U2R	buffer_overflow	20	FALSE
	loadmodule	2	FALSE
	perl	2	FALSE
	ps	15	**TRUE**
	rootkit	13	FALSE
	sqlattack	2	**TRUE**
	xterm	13	**TRUE**
normal	normal	9710	FALSE

Table 3.4: Distribution of attacks in test dataset

3.3.4 Feature/fields

NSL KDD does not modify fields given in original KDD dataset therefore description is the same as the one provided for the KDD Cup task [15] a complete listing of the set of features defined for the connection records is given in the three tables below:

feature name	description	type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Table 3.5: NSL KDD Basic features of individual TCP connections

feature name	description	type
hot	number of "hot" indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of "compromised" conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if "su root" command attempted; 0 otherwise	discrete
num_root	number of "root" accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the "hot" list; 0 otherwise	discrete
is_guest_login	1 if the login is a "guest"login; 0 otherwise	discrete

Table 3.6: NSL KDD: Content features within a connection suggested by domain knowledge

feature name	description	type
count	# of connections to the same host as the current connection in the past two secs	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have "SYN" errors	continuous
rerror_rate	% of connections that have "REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	#r of connections to the same service as the current connection in the past two secs	continuous
	Note: The following features refer to these same-service connections.	
srv_serror_rate	% of connections that have "SYN" errors	continuous
srv_rerror_rate	% of connections that have "REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Table 3.7: NSL KDD features:Traffic features computed using a two-second time window

3.4 Snort

Snort is an open source network intrusion prevention and detection system initially developed by developed by Martin Roesch [5]. It uses a rule-based language combining signature, protocol and anomaly inspection methods. A NIDS, which stands for network intrusion detection system, is an intrusion detection system that tries to detect malicious activity such as denial of service attacks, port scans or even attempts to crack into computers.

3.4.1 Snort components

Snort consist of several modules. It takes as input traffic from the network and will produce as output alerts on the system and log files. This information is created based on applying a different sets of rules by the detection engine. This modular architecture provides great flexibility. Snort modules and flows are showed in Figure 3.2.

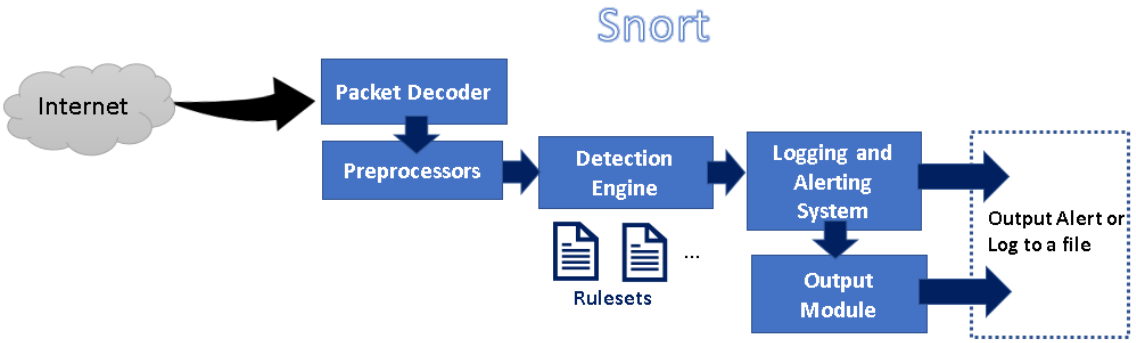


Figure 3.2: Snort component flow

Now each module will be briefly explained:

- **Packet Decoder:** Main function is to capture all network packets using libcap libray. First it identify link layer protocol, then it will decode IP layer and then will look at the transport layer to identify relevant protocol such TCP,UDP.. If there are malformed headers Snort will alert about it.
- **Preprocessors:** This module will prepare data obtained from previous module for detection by obtaining additional information. There are several types of preprocessors depending on the traffic to be analysed, for instance HTTP, telnet, IP reassembly. In our case an additional preprocessor will provide metrics defined in KDD dataset.
- **Detection engine:** Its responsibility is to detect if any intrusion activity exists in a packet. The detection engine employs Snort rule for this purpose. The rules are read into internal data structures or chains where they are matched against all packets. It a packet matches any rule, relevant action is taken, otherwise packet is discarded. Actions may be logging the packet or alert generation. It will use of several detection plugins, in our case a specific plugin that can make use of KDD features.
- **Rulesets:** It defines the sets of individual rules that will control the analysis of the detected packets. Detection plugins, protocol, flow direction are defined. More information is provided in section Section 3.4.2.
- **Logging and alerting sytem:** Depending on what is found inside the packet from the detection engine activity.
- **Output module:**It controls the type of output generated by the logging and alerting system. Those notifications (alerts, logs) can be saved in different formats: text files, database, server system logs ...

3.4.2 Snort rules

Rules are used by Snort detection engine to compare received packets and generate alerts in case packets match with the content defined on the rules. There are two types of rules: official sets of rules provided by Snort, subscription fee applies, and rules provided by the community, free to use under GPL licence.

Snort uses a simple, lightweight rules description language that is flexible. As an example one rule is showed in box below:

```
alert tcp any any ->192.168.1.0/24 111 (content:"—00 01 86 a5—"; msg: "mountd access";)
```

There are many web pages containing information about how to write Snort rules, official documentation section provide a detail of all available options, some of the fields are described below [16].

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis is the rule options. The words before the colons in the rule options are called option keywords.

Rule Actions: The rule header contains the information that defines the "who, where, and what" of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up. The first item in a rule is the rule action. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are five available default actions in Snort, alert, log, pass, activate, and dynamic.

Protocols: The next field in a rule is the protocol. There are three IP protocols that Snort currently analyses for suspicious behavior, tcp, udp, and icmp.

IP Addresses: The next portion of the rule header deals with the IP address and port information for a given rule. The keyword "any" may be used to define any address. Snort does not have a mechanism to provide host name lookup for the IP address fields in the rules file.

Port Numbers: Port numbers may be specified in a number of ways, including "any" ports, static port definitions, ranges, and by negation. "Any" ports are a wildcard value, meaning literally any port. Static ports are indicated by a single port number, such as 111 for portmapper, 23 for telnet, or 80 for http, etc. Port ranges are indicated with the range operator ":".

The Direction Operator The direction operator "->" indicates the orientation, or "direction", of the traffic that the rule applies to. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source host, and the address and port information on the right side of the operator is the destination host. There is also a bidirectional operator, which is indicated with a "<>" symbol. This tells Snort to consider the address/port pairs in either the source or destination orientation.

Rule Options: Rule options form the heart of Snort's intrusion detection engine, combining ease of use with power and flexibility. All Snort rule options are separated from each other using the semicolon ";" character. Rule option keywords are separated with a colon ":".

3.4.3 KDD features in Snort

It is possible to add features implemented in KDD database into Snort by the addition of a preprocessor and a detection-plugin. I has been found an existing implementation named s-predator [17]. Preprocessor will add implementation of the features and it is just required to include new fields to be calculated for each packet after loading of the module. Last addition to make use of KDD features is detection plugin which adds the capability to understand rules extrapolated from KDD features and will check conditions on new parameters and trigger relevant configured actions, alerts or logging.

3.5 R Caret package

Existing CARET package in R, which stands for **C**lassification **A**nd **RE**gression **T**raining, is a set of functions that attempt to simplify the process to create predictive models. The package contains tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation

For train phase the train function is specified. This function provides following functionalities: evaluate, using resampling, the effect of model tuning parameters on performance; choose the "optimal" model across these parameters and estimate model performance.

This will allow to evaluate different models and to estimate the better parameter set before evaluating the test dataset. The training and hyperparameter tuning algorithm is shown below as specified in official documentation [18] which include a detail view of all modules:

```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

Figure 3.3: Caret parameter tuning flow

3.5.1 Models available

CARET package provides a wide set of models to be used. Currently there are 238 models defined that cover both classification and regression scenarios. It is also possible to define your own model that can be integrated in the package.

For the purpose of this project evaluation of several model was done, one of the main reasons to chose an algorithm is the capabilities to generate rules so they can be easily included in the workflow. While some models provide rules directly from the trained model for some other it was investigated additional package to convert for instance trees to rules. An additional condition when choosing a model for the purpose of this project was the fact that both binary and multi class classification can be performed as some models only support binary scenario.

A total of five models have been chosen which include: C50, J48 (C4.5-like Trees), JRip, Gradient Boosting Machine and Random Forest. All these models will be explained in more detail in Section 4.3.

3.5.2 Optimizing tuning parameters

Hyperparameter tuning evaluation is performed by specifying a grid containing the parameters list and possible values to be evaluated. Argument *tuneGrid* can take a data frame with columns for each tuning parameter. If parameter *tuneGrid* is not specified default grid would be used.

In addition to the approach mentioned below, it is also possible to use a random selection of tuning parameter combination or a combination of a grid search and racing to find best suitable values.

3.5.3 Re-sampling

In CARET there are 5 different methods that you can use to estimate model accuracy. These are as specified in [19]:

Data Split

Data splitting involves partitioning the data into an explicit training dataset used to prepare the model and an unseen test dataset used to evaluate the models performance on unseen data. It is useful when you have a very large dataset so that the test dataset can provide a meaningful estimation of performance, or for when you are using slow methods and need a quick approximation of performance.

Bootstrap

Bootstrap re-sampling involves taking random samples from the dataset (with re-selection) against which to evaluate the model. In aggregate, the results provide an indication of the variance of the models performance. Typically, large number of resampling iterations are performed (thousands or tens of thousands). This resampling technique is specified as an example: *trainControl(method="boot", number=100)*

k-fold Cross Validation

The k-fold cross validation method involves splitting the dataset into k-subsets. For each subset is held out while the model is trained on all other subsets. This process is completed until accuracy is determine for each instance in the dataset, and an overall accuracy estimate is provided. The following example uses 10-fold cross validation to estimate Naive Bayes on the iris dataset. This resampling technique is specified as an example: *trainControl(method="cv", number=10)*

Repeated k-fold Cross Validation

The process of splitting the data into k-folds can be repeated a number of times, this is called Repeated k-fold Cross Validation. The final model accuracy is taken as the mean from the number of repeats. This resampling technique is specified as an example: *trainControl(method="repeatedcv", number=10, repeats=3)*

Leave One Out Cross Validation

In Leave One Out Cross Validation (LOOCV), a data instance is left out and a model constructed on all other data instances in the training set. This is repeated for all data instances. This resampling technique is specified as an example:
trainControl(method="LOOCV")

3.5.4 Performance measurements

After re-sampling, the process produces a profile of performance measures is available to guide the user as to which tuning parameter values should be chosen. There are several metrics that can be used to evaluate the machine learning algorithm in R. Caret supports a wide range of evaluation metrics:

- Accuracy and Kappa
- RMSE and R^2
- ROC (AUC, Sensitivity and Specificity)
- LogLoss

Default metrics used are accuracy for classification problems and RMSE for regression. For evaluation of different sets of parameters in this project accuracy has been used. For reference all possible options that CARET provides are briefly explained below [20]:

Accuracy and Kappa

Accuracy is the percentage of correctly classifies instances out of all instances. It is more useful on a binary classification than multi-class classification problems because it can be less clear exactly how the accuracy breaks down across those classes (e.g. you need to go deeper with a confusion matrix). Learn more about Accuracy here. Kappa or Cohen's Kappa is like classification accuracy, except that it is normalized at the baseline of random chance on your dataset. It is a more useful measure to use on problems that have an imbalance in the classes (e.g. 70-30 split for classes 0 and 1 and you can achieve 70% accuracy by predicting all instances are for class 0).

RMSE and R^2

RMSE or Root Mean Squared Error is the average deviation of the predictions from the observations. It is useful to get a gross idea of how well (or not) an algorithm is doing, in the units of the output variable. Learn more about RMSE here. R^2 which stands for R Squared or also called the coefficient of determination provides a *goodness of fit* measure for the predictions to the observations. This is a value between 0 and 1 for no-fit and perfect fit.

Area Under ROC Curve

ROC metrics are only suitable for binary classification problems (e.g. two classes). To calculate ROC information, you must change the summaryFunction in your trainControl to be twoClassSummary. This will calculate the Area Under ROC Curve (AUROC) also called just Area Under curve (AUC), sensitivity and specificity. ROC is actually the area under the ROC curve or AUC. The AUC represents a models ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predicts perfectly. An area of 0.5 represents a model as good as random. ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity. Sensitivity is the true positive rate also called the recall. It is the number instances from the positive (first) class that actually predicted correctly. Specificity is also called the true negative rate. Is the number of instances from the negative class (second) class that were actually predicted correctly.

Logarithmic Loss

Logarithmic Loss or LogLoss is used to evaluate binary classification but it is more common for multi-class classification algorithms. Specifically, it evaluates the probabilities estimated by the algorithms.

3.5.5 Best tuned parameters

For all caret models, the final model is trained on the full dataset. *caret::train* uses the cross-validation scheme you chose to select model parameters and estimate out-of-sample performance of the model. After performance of all parameter combinations is known train function automatically chooses the tuning parameters associated with the best value.

Once the cross-validation is done, caret retrains the model on the full dataset, using the parameters it selected during cross-validation. The model does not average the trained model's coefficients. It re-fits the model on the full dataset.

As model would be trained with the full train dataset for comparison purposes cross validation performance would be used to compare performance on every parameter combination.

Accuracy an kappa has been chosen as relevant metric to be optimized as it will provide a good view for both binary and multi class scenarios of the samples detected correctly even more metrics will be used for analysis purposes.

3.5.6 Runtime performance

Training task can be quite time consuming. The caret package supports parallel processing in order to decrease the compute time for a given experiment. It is supported automatically as long as it is configured. Caret package use R support for parallel core computations (by parallel package).

The train instruction of the caret package has built-in support for parallel backends, but you have to call and set it up. If you do not register a backend, train will resort to single-core computations. With a registered parallel backend, any caret model training will use multi-cores of the CPU, since by default the trainControl argument is already set as allowParallel=TRUE.

This effectively means that several cores will be running independent training for a single parameter combination which reduce total time to evaluate whole parameter grid defined.

3.5.7 Model Evaluation Metrics in R

Caret provide a very convenient way to obtain most relevant metrics by the use of the confusionmatrix() function. It compares predicted values against the reference and provide absolute values for confusion matrix, as an example for two class problem:

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

Figure 3.4: Confusion matrix example

One of the main metrics that we would consider is accuracy defined as $A+D/(A+B+C+D)$. Accuracy measures how well a binary classification test correctly identify or excludes a condition. That is, the accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined. Multiple metrics are calculated which are shown in the following picture from the official documentation [18]:

$$
\begin{aligned}
Sensitivity &= \frac{A}{A+C} \\
Specificity &= \frac{D}{B+D} \\
Prevalence &= \frac{A+C}{A+B+C+D} \\
PPV &= \frac{sensitivity \times prevalence}{((sensitivity \times prevalence) + ((1-specificity) \times (1-prevalence))} \\
NPV &= \frac{specificity \times (1-prevalence)}{((1-sensitivity) \times prevalence) + ((specificity) \times (1-prevalence))} \\
Detection\ Rate &= \frac{A}{A+B+C+D} \\
Detection\ Prevalence &= \frac{A+B}{A+B+C+D} \\
Balanced\ Accuracy &= (sensitivity + specificity)/2
\end{aligned}
$$

$$
\begin{aligned}
Precision &= \frac{A}{A+B} \\
Recall &= \frac{A}{A+C} \\
F1 &= \frac{(1+\beta^2) \times precision \times recall}{(\beta^2 \times precision) + recall}
\end{aligned}
$$

Figure 3.5: Metrics calculated from the confusion matrix Caret

For two class problems, the sensitivity, specificity, positive predictive value and negative predictive value is calculated using the positive argument. Also, the prevalence of the "event" is computed from the data, the detection rate (the rate of true events also predicted to be events) and the detection prevalence (the prevalence of predicted events).

For more than two classes, these results are calculated comparing each factor level to the remaining levels (i.e. a "one versus all" approach).

Sensitivity or Recall: When it is actually "yes" how often it predicts "yes". From the observed class 1, what percentage cases are actually classified by the model predicted class 1 is measured by a measure which is called Sensitivity.

Specificity: Specificity measures true negative rate. When it is actually "No", how often it is "No". In our case would be when traffic is normal how often it is consider as normal. Specificity= TN/(TN+FP) = TN/(Actual No)

Another useful performance measure is the balanced accuracy which avoids inflated performance estimates on imbalanced datasets. It is defined as the arithmetic mean of sensitivity and specificity, or the average accuracy obtained on either class

Cohen's Kappa: This is essentially a measure of how well the classifier performed as compared to how well it would have performed simply by chance. In other words, a model will have a high Kappa score if there is a big difference between the accuracy and the null error rate.

CHAPTER 4

EXPERIMENT DESIGN

This chapter explains how the experiments are designed, covering from the environment used, models selected, data pre-processing, rule generations and output produced from the execution.

4.1 Overview

This section will cover details on how the experiment has been designed. Main tasks of the experiment which have been performed by several R scripts developed are:

- Process NSL KDD dataset to get adequate input for algorithms
- Obtain best set of parameter for each model
- Train models with best set and evaluate performance on test dataset
- Produce rules for each algorithm

High level flow of the experiments is presented in Figure 4.1 where inputs and outputs for each script is specified. Explanation of what each script do is listed below:

- **Data Process R Script.** This script takes as input raw NSL KDD dataset, it will convert the script to add additional columns like profile attack (Dos, Probe, R2L, U2R, normal), flag for novel attack in test data, class (anomaly, normal). Details on implementation are provided in Section 4.4.
- **Parameter tuning R Script.** This script will perform the parameter tuning task by the use of Caret package. For each model a grid of parameters is specified and training is performed with a re sampling technique. Trained model is part of the output so it can be inspected at later stage and best parameter set is provided.
- **Train and Test R Script.** Once best parameters are known training on the full dataset is performed, and model is evaluated against the test data set. Rules and results for each algorithm are generated so analysis on performance can be done.

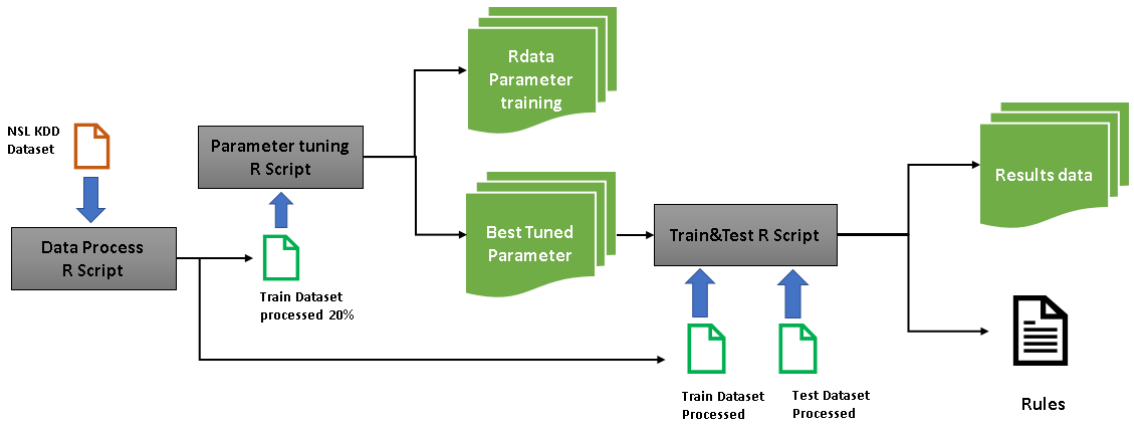


Figure 4.1: Experiment flow

Next sections will cover in more detail implementation of the experiment, what models have been selected, parameters to be tuned, files and rules generated, and different options used during the process.

4.2 Environment

All experiments were executed under a Windows 7 laptop with following characteristics:

- Intel Core i7-4810MQ CPU @2.8GHz
- 8 cores
- RAM memory 24GB
- System type 64-bit Operating System

Software levels that have been used are:

- RStudio Version 1.1.383
- R Version 3.5.0 64bit

4.3 Model selection

From all the available models in CARET package a total of five algorithms which can perform classification were chosen for evaluation.

One of the main reasons for selecting them is the fact that algorithm can produce directly rules. This is very important and relevant as rule files are the main input into Snort NIDS for attack detection.

Next sections will add more information about those models, outline available parameters that can be tuned and how rules can be generated, in some cases using others R packages which functionality is not part of CARET package itself.

A summary of selected models is presented in Table 4.1:

Model	**Method Value**	**Type**	**Libraries**	**Tunning Parameters**
C5.0	C5.0	Classification	C50, plyr	trials, model, winnow
C4.5-like Trees	J48	Classification	RWeka	C, M
Rule-Based Classifier	JRip	Classification	RWeka	NumOpt, NumFolds, MinWeights
Generalized Boosted Regression Models	gbm	Classification, Regression	gbm, plyr	n.trees, interaction.depth, shrinkage, n.minobsinnode
Random Forest Rule-Based Model	rfRules	Classification, Regression	randomForest, inTrees, plyr	mtry, maxdepth

Table 4.1: CARET models selected for evaluation

4.3.1 J48

CARET method J48 makes use of Weka J48 implementation and it is used for classification. J48 is an extension of ID3. The additional features of J48 are accounting for missing values, decision trees pruning, continuous attribute value ranges, derivation of rules, etc. In the WEKA, J48 is an open source Java implementation of the C4.5 algorithm from Ross Quinlan [21].

At a high level C4.5 tree tries to recursively partition the data set into subsets by evaluating the normalized information gain (difference in entropy) resulting from choosing a descriptor for splitting the data. The descriptor with the highest information gain is used on every step. The training process stops when the resulting nodes contain instances of single classes or if no descriptor can be found that would result to the information gain.

For this model following parameters are available for tuning, range used in the experiments is also included:

- Confidence Threshold (C, numeric). From 0 to 0.5 in 0.05 steps
- Minimum Instances Per Leaf (M, numeric). From 1 to 10 in steps of 1 to get better visualization and from 10 to 375 in steps of 25.

4.3.2 C50

C5.0 algorithm is an enhanced version of C4.5 algorithm with following enhancements [22]:

- Speed - C5.0 is significantly faster than C4.5 (several orders of magnitude)
- Memory usage - C5.0 is more memory efficient than C4.5
- Smaller decision trees - C5.0 gets similar results to C4.5 with considerably smaller decision trees.
- Support for boosting - Boosting improves the trees and gives them more accuracy.
- Winnowing - a C5.0 option automatically winnows the attributes to remove those that may be unhelpful.

Model C5.0 is specified by method = 'C5.0' in CARET. It uses packages C50 and plyr with the following tuning parameters:

- Number of Boosting Iterations (trials, numeric). After the first tree is created, weights are determined and subsequent iterations create weighted trees or rulesets. Subsequent trees (or rulesets) are constrained to be about the same size as the initial model. The final prediction is a simple averages of class probabilities generated from each tree or ruleset (i.e. no stage weights).

- Winnow (winnow, logical). Winnowing is a feature selection step conducted before modeling. The data set is randomly split in half and an initial model is fit. Each predictor is removed in turn and the effect on model performance is determined (using the other half of the random split). Predictors are flagged if their removal does not increase the error rate. The final model is fit to all of the training set samples using only the unflagged predictors.
- Model Type (model, character). The main two modes on this algorithm for model parameter are tree-based and a rule-based model which are easier to understand compared to other models. A rule based model has been selected as is able to directly produce rules which can be later processed to be included as part of Snort functionality.

4.3.3 JRip

Model named JRip in CARET terminology makes use of Weka rule classifier Jrip. This class implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), which was proposed by William W. Cohen [23] as an optimized version of IREP.

The algorithm is briefly described, according package documentation [24], in following steps:

Initialize RS = {} , and for each class from the less prevalent one to the more frequent one.

DO:

1. Building stage: Repeat 1.1 and 1.2 until the description length (DL) of the ruleset and examples is 64 bits greater than the smallest DL met so far, or there are no positive examples, or the error rate $\geq$ 50%.

    1.1. Grow phase: Grow one rule by greedily adding antecedents (or conditions) to the rule until the rule is perfect (i.e. 100 per cent accurate). The procedure tries every possible value of each attribute and selects the condition with highest information gain: p(log(p/t)-log(P/T)).

    1.2. Prune phase: Incrementally prune each rule and allow the pruning of any final sequences of the antecedents. The pruning metric is (p-n)/(p+n) – but it's actually 2p/(p+n) -1, so in this implementation we simply use p/(p+n) (actually (p+1)/(p+n+2), thus if p+n is 0, it's 0.5).

2. Optimization stage: after generating the initial ruleset {Ri}, generate and prune two variants of each rule Ri from randomized data using procedure 1.1 and 1.2. But one variant is generated from an empty rule while the other is generated by greedily adding antecedents to the original rule. Moreover, the pruning metric used here is (TP+TN)/(P+N). Then the smallest possible DL for each variant and the original rule is computed. The variant with the minimal DL is selected as the final representative of Ri in the ruleset. After all the rules in Ri have been examined and if there are still residual positives, more rules are generated based on the residual positives using Building Stage again.

3. Delete the rules from the ruleset that would increase the DL of the whole ruleset if they were in it and add resultant ruleset to RS.

END DO

For this model parameters that will be tuned and its specific range are listed below:

- MinWeight from 1 to 6
- Number of optimizations (NumOpt) from 1 to 6
- Number of folds from 2 to 7

4.3.4 GBM

Another algorithm used is named as GBM which stands for Generalized Boosted Regression Models. Model GBM is specified by method = 'gbm' in CARET and it uses GBM R package which implements extensions to Freund and Schapire's [25] AdaBoost algorithm and J. Friedman's [26] gradient boosting machine.

A very simplified explanation about how Gradient boost machine works is based on the following steps [27]:

- Fit a model to the data, $F_1(x) = y$
- Fit a model to the residuals, $h_1(x) = y - F_1(x)$
- Create a new model, $F_2(x) = F_1(x) + h_1(x)$

From there we can see that the main idea is to keep adding models that correct errors of the previous model. Since the procedure was initialized by fitting $F_1(x)$, our task at each step is to find $h_m(x) = y - F_m(x)$.

So in summary boosting is the process of iteratively adding basis functions in a greedy fashion so that each additional basis function further reduces the selected loss function. More detail is provided in the CRAN GBM package documentation [28].

For the purpose of this project default loss functions have been used which have been bernoulli for the binary classification and multinomial for multi class scenario.

CARET GBM implementation has following parameters available for tuning:

- interaction.depth. The maximum depth of variable interactions. 1 implies an additive model, 2 implies a model with up to 2-way interactions, etc.
- n.minobsinnode. minimum number of observations in the trees terminal nodes. Note that this is the actual number of observations not the total weight.
- Shrinkage. a shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction.
- n.trees. the total number of trees to fit. This is equivalent to the number of iterations and the number of basis functions in the additive expansion.

In this model a total of 480 parameter combination will be evaluated. This is the range selected for the available parameters for this model:

- interaction.depth: from 1 to 4
- n.minobsinnode: 0.1,0.2,0.3,0.4
- Shrinkage: 10
- n.trees: 10,20,40,80,100,140,180,220

4.3.5 Random Forest

Random Forest was implemented and described by Leo Breiman and Adele Cutle [29] as follow:

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.

2. If there are M input variables, a number $m << M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.

3. Each tree is grown to the largest extent possible. There is no pruning.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob (out of bag) error rate a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

Out of bag error estimation works as follow: Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree.

Put each case left out in the construction of the kth tree down the kth tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take j to be the class that got most of the votes every time case n was oob. The proportion of times that j is not equal to the true class of n averaged over all cases is the oob error estimate. This has proven to be unbiased in many tests.

Parameter range and description chosen to evaluate this algorithm are:

- mtry Number of variables randomly sampled as candidates at each split. Range selected: 2,4,6,8,10
- maxdepth. represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data. Range selected: 2,3,4,5
- ntree Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times. While not possible to use a train grid for this parameter value was fixed to 100 to reduce huge time taken to perform all test.

4.4 Data and pre-processing

As presented before NSL-KDD dataset was used in this evaluation. This dataset contains files either with binary classification i.e normal or anomaly or specific attack for each sample in a different file.

In order to perform multi classification evaluation dataset was process to map each attack to its profile: Dos, Probe, R2L, U2R and normal in a new column named profile. Once this is known another column named class is added with normal or anomaly classification.

In addition to evaluate models capabilities for detecting new attacks a new column named novel was included to indicate that the sample represent a novel attack, this means that the attack was not present in the whole train dataset but it is now present in test dataset.

Table 4.2 shows mapping performed for all the novel attacks present in the test dataset, also it illustrate new fields included in the pre-processing step.

attack	profile	class	novel
apache2	DoS	anomaly	1
sendmail	R2L	anomaly	1
sqlattack	U2R	anomaly	1
mscan	Probe	anomaly	0
normal	normal	normal	0
...	...	...	...

Table 4.2: Additional fields added to dataset, novel attacks

Random Forest had the limitation of supporting a maximum number of 52 levels. This represent a problem as one of the fields in the dataset, service, has 70 different levels. To overcome this limitation approach taken was to convert 18 less frequent services in the dataset to service other. Specifically services http_ 2784, aol, harvest, http_ 8001, tftp_ u, pm_ dump, red_ i, tim_ i, urh_ i, shell, printer, X11, pop_ 2, remote_ job, rje, ntp_ u, IRC, sql_ net were converted to other.

4.5 Resampling and model validation

From the existing option for re sampling a Cross Validation approach was chosen for validate the different models. In our case a 5-fold cross-validation was selected, meaning that for 4/5 of the data will be used for training and then predictions and evaluation will be done with the remaining 1/5. As data is split in 5 parts this process is performed the same number of times, that is, 5. This has been considered to be enough for comparison purposes of different parameter set.

This is specified in CARET environment with a trainControl object as showed below:

```
xCtrl <- trainControl(method = "cv", number=5,classProbs = TRUE,verboseIter =
TRUE,allowParallel = TRUE)
```

Parameter classProbs is configured as TRUE and it specify whether class probabilities should be computed for held-out samples during resample.

Parameter allowParallel is specified as TRUE to allow parallel processing using more than 1 core. Even this is configured number of parallel process to be used needs to be registered.

4.6 Reproducibility

As one easy way to execute fully reproducible tests in parallel mode using the caret package is by using the seeds argument when the train control is called. This is done with following command where an integer needs to be specified. Before each model is trained a seed is specified.

```
set.seed(1)
```

4.7 Parallel processing

To allow the use of all cores available on the machine we need to start with setting up a cluster, a collection of process that will be executing the training. It is possible to specify an output file to inspect the progress of the activity, note that information will not be in sequence as each process will write independently. After the cluster is created it needs to be register. At the end of the script is recommended to stop the cluster to avoid possible problems with further command executions. All this is done with following commands:

```
cl = makeCluster(detectCores()-1, outfile= "outputfile.txt"))
registerDoParallel(cl)
...
stopCluster(cl)
```

4.8 Rules generation

Selected models allow rule generation either directly from the train model produced by the training on CARET or by other packages which are explained below:

C5.0

This model provides directly rules obtained from the final model, i.e model with best parameters. This is done by selecting parameter rules from the finalModel object as shown:

```
fit.c50$finalModel$rules
```

J48

J48 model produces a tree but it can be converted to rules by using additional package partykit. First model is converted into a party object which can be used as argument to a function that will output the rules. This is exemplified in the following box:

```
library("partykit")
pres = as.party( fit.J48$finalModel)
rules = partykit:::.list.rules.party(pres)
```

JRip

This model supports as well rule generation from the final model produced after training:

```
fit.JRip$finalModel$rules
```

GBM

GBM model does not produce rules directly however it is possible to obtain them by using an additional packaged called inTrees [30] which is a framework for extracting, measuring, pruning, selecting and summarizing rules from a tree ensemble which includes random forests, xgboost and gbm models. Steps to produce rules from a GBM final model are showed in following text box:

```
treeList = GBM2List(fit.GBM$finalModel,x)
ruleExec = extractRules(treeList,x)
ruleExec = unique(ruleExec)
ruleMetric = getRuleMetric(ruleExec,x,yc)
ruleMetric = pruneRule(ruleMetric,x,yc)
ruleMetric = unique(ruleMetric)
learner = buildLearner(ruleMetric,x,yc)
readableLearner = presentRules(learner,colnames(x))
```

Random Forest

Random Forest itself does not produce rules, however RFRules method used implements inTrees package functionalities internally so rules are provide in text format as part of the final model. It can be obtained with following command:

```
fitf.RF$finalModel$model
```

4.9 Output files

The process as defined in previous sections has two main steps, one that will provide best tuned parameter for each algorithm and a last step which will use those best parameter to train the algorithms with the full train dataset and then obtain relevant results.

For first step which can be named as parameter tuning following files are produced by the script:

- train_ param.RData - File with the trained model. It has been used for later analysis
- BestTuned.txt - File contains best parameter combination i.e with highest accuracy
- train_ model.jpeg - Graph showing accuracy for each of the parameter combinations.
- PerformanceBestTuned.txt - Contains accuracy and kappa resulted from the cross validation of the best parameter combination.

Last step will perform training of the models this time using the full train dataset, and it will produce several files containing the results against the test dataset. Files are stored in a specific folder for each algorithm. These files are:

- train.RData - File with the trained model. It has been used for later analysis
- tuned_ params.txt - File contains parameter used for reference.
- times.txt - Contains time taken for executing the training.
- rules.txt - File with rules in text format.
- VarImp.jpg - Graph showing variable importance
- varImp.txt - Variable importance in text format.
- novelresults.txt - For every novel attack, i.e attacks not present in train dataset, number of detected samples per attack is provided.
- test_ model_ results.txt - Results obtained after predicting and comparing with the test dataset. It includes the confusion matrix and related metrics calculated.

4.10 Results evaluation metrics

Evaluation of the results can be divided into two different parts whether the assessment of the best parameter for each algorithm is done or whether performance is evaluated against the test dataset.

Training Phase results. This step is based on finding the best parameter combination for each model. In order to do that Accuracy vs. Parameter graphs will be plotted and inspected. Caret provides directly the best combination and its performance. Time taken for training will be collected.

Test Phase results. This phase will cover training of the models with the best set of parameters with the full train dataset. Next step is based on predicting values on the test dataset and comparing them with the expected values. This provides the confusion matrix which will be provided.

In addition metrics calculated from the confusion matrix will be included in the results. Description has been provided in Section 3.5.7 but metrics will be slightly different depending on type of classification: binary or multi class.

- Binary: Accuracy, kappa, sensitivity, specificity, balanced accuracy, detection prevalence, and prevalence
- Multi class. Overall Accuracy and Kappa and several metrics per class which include sensitivity, specificity, balanced accuracy, detection prevalence, detection rate and prevalence

In general we would consider a better algorithm when it has a high specificity, as number of wrong alerts is reduced, and high sensitivity, as more attacks can be detected.

For the multi class scenario metrics are calculated comparing each factor level to the remaining levels, using class normal as reference would give relevant information for comparison.

CHAPTER 5

RESULTS ANALYSIS: BINARY CLASSIFICATION

Following chapter cover analysis of results obtained for all algorithms: C50, J48, JRip, GBM and Random Forest in the binary scenario: normal or anomaly. Parameter tuning and evaluation of best parameter set on the test dataset is provided.

5.1 Training phase results

5.1.1 C50

Algorithm C50 was evaluated to obtain the best value for its parameters. In this case options for tuning were the number of boosting iterations, named trials, and winnow with logical value. Following graph shows obtained accuracy for all possible combination evaluated:

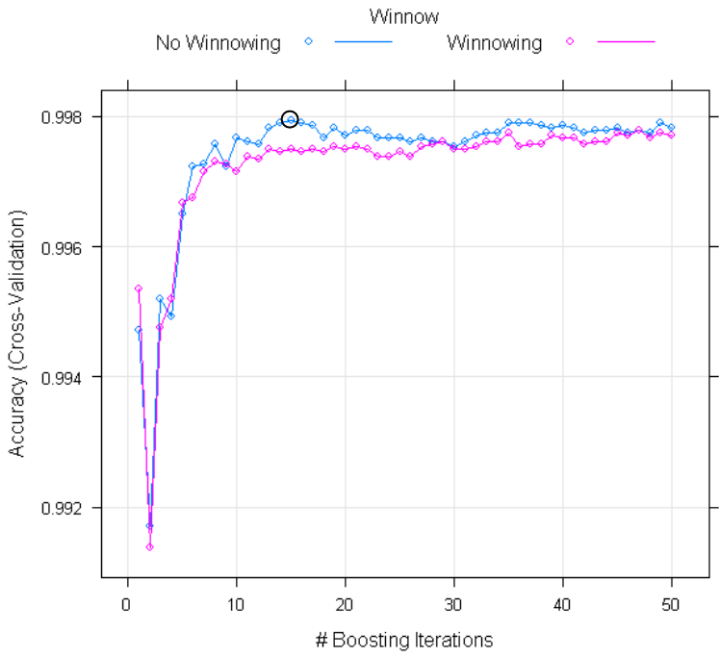


Figure 5.1: C50 class accuracy evaluation

As observed in Figure 5.1 boosting iteration has a positive effect on accuracy when it increase specially in the first values. Then accuracy remains with not major difference after approximately 10 iterations. Best parameter values was obtained when winnow is false and 15 boosting iterations obtained a training accuracy of 0.9979358 and a training Kappa of 0.9958519.

5.1.2 J48

J48 was trained using all parameter combinations defined. For this model a set of confidence threshold with range from 0 to 0.5 with 0.05 as step was configured. Similarly the minimum instances per leaf was defined in a range from 0 to 375. All results combined are presented in Figure 5.2. In this figure we can observe a negative trend when instance per leaf increase with no mayor difference in terms of the confidence threshold used, specially after 100 instances per leaf.

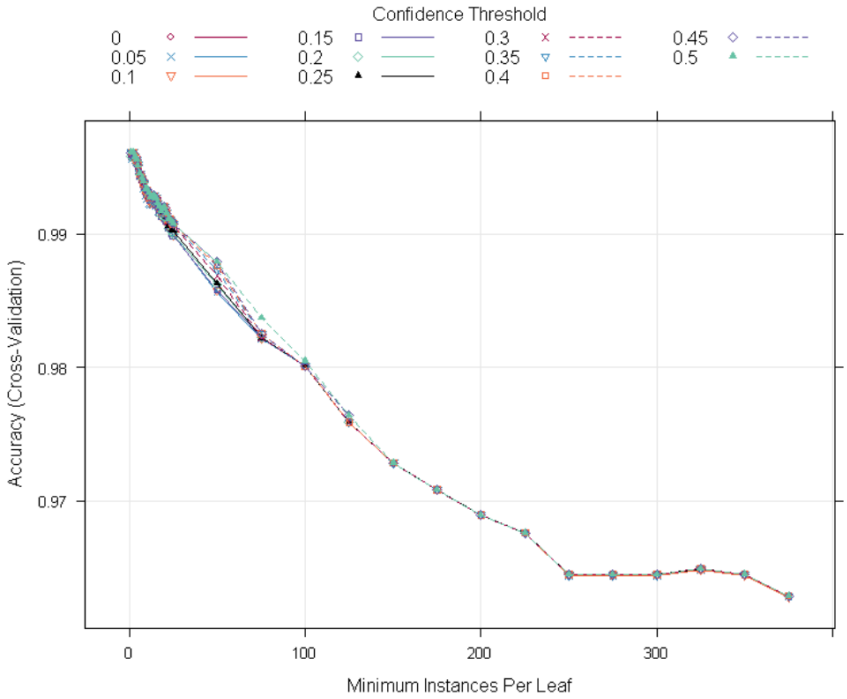


Figure 5.2: J48 class accuracy evaluation all results

Some difference is observed related to the confidence threshold used when lowest range of instances per lead is used. In this range is where highest accuracy can be observed. In order to get a better view Figure 5.3 provides a zoomed view best performance range.

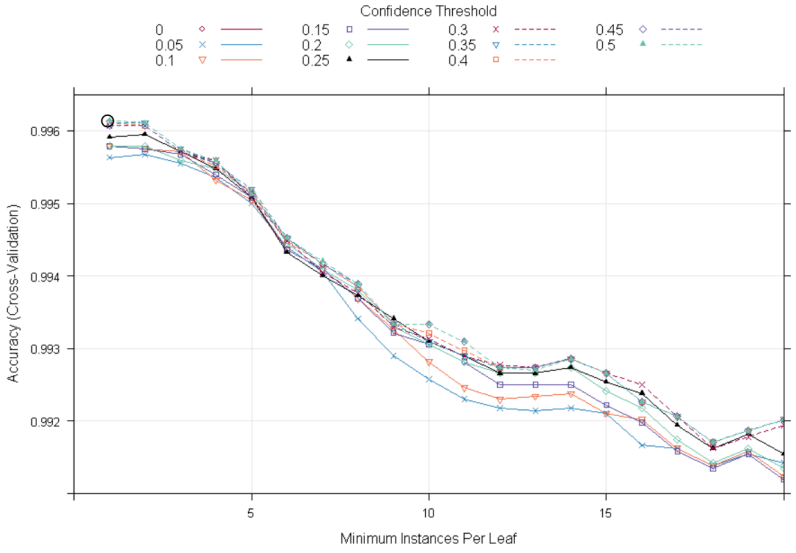


Figure 5.3: J48 class accuracy evaluation detail

Best train accuracy, 0.9961 was obtained with a 0.5 confidence threshold and 1 minimum instance per leaf of 1. Kappa was 0.9922 in this case.

5.1.3 JRip

Jrip algorithm was trained to evaluate accuracy of following parameters:

- MinWeight from 1 to 6
- Number of optimizations from 1 to 6
- Number of folds from 2 to 7

For each of the parameter combination obtained cross validation accuracy is shown in the figure below:

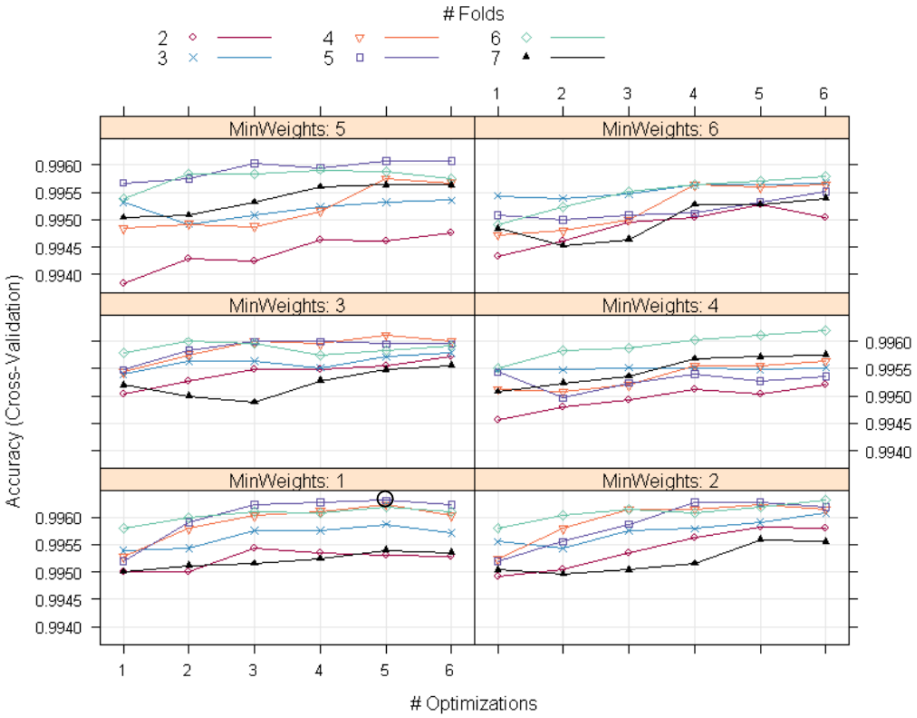


Figure 5.4: JRip class accuracy evaluation

For this algorithm best accuracy was obtained with 5 folds, a minimum weight of 1 and number of optimizations equals to 5. For that parameter combination performance is 0.9963084 for accuracy and 0.9963084 for Kappa.

5.1.4 GBM

Gradient boosting machine was trained with 4 different shrinkage values: 0.001, 0.05, 0.1 and 0.2, a maximum tree depth of 4 with a highest number of boosting iterations of 140 obtaining values for previous iteration levels. Min number of observations in the node was fixed to 10. All results obtained are plotted in the Figure 5.5.

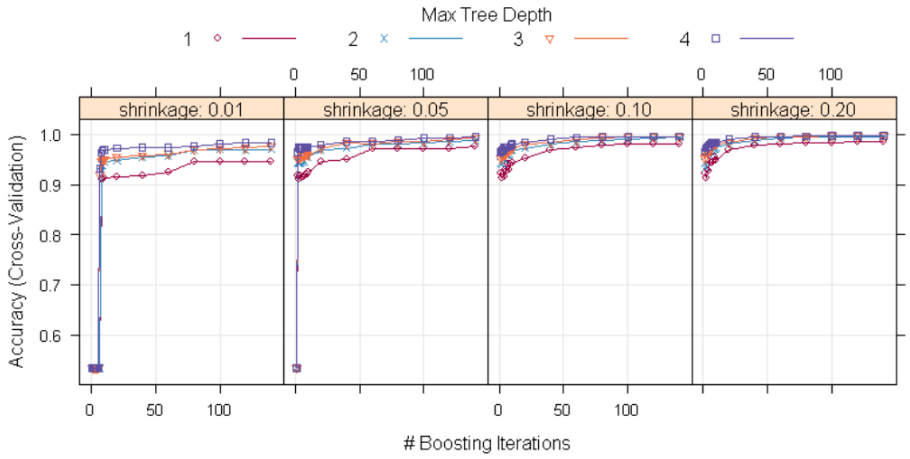


Figure 5.5: GBM class accuracy evaluation all results

Looking at all results it can be observed that accuracy is very low when shrinkage has lowest values in the defined range, i.e 0.01 and 0.02 and iterations are low around 0.53. This behaviour tends to improve when iterations increase to 6 or 7 with shrinkage of 0.01 and it is only observed with a single iteration for shrinkage of 0.02.

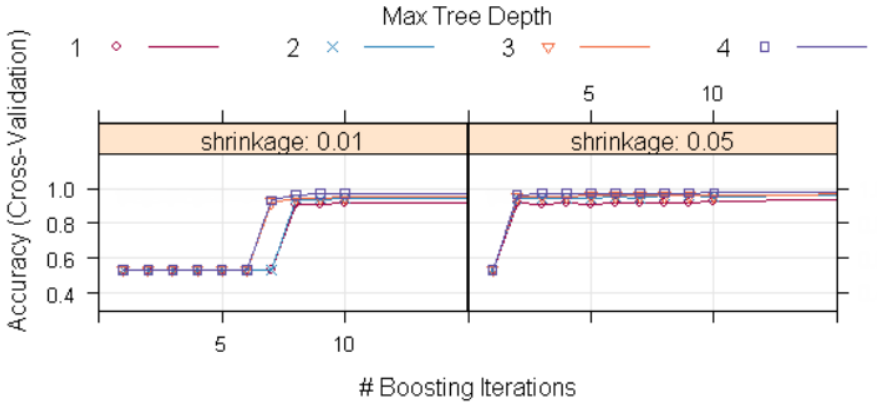


Figure 5.6: GBM class accuracy low performance zoom

In any case for the purpose of this project we are looking to identify best performance so a deeper view on highest range is needed to identify best combination. This is shown in Figure 5.7 where highest accuracy was reached with 140 trees, iteration depth of 4, shrinkage of 0.2 and 10 minimum number of observations. For this parameter combination accuracy was 0.9971021 with a value for metric Kappa of 0.9941768.

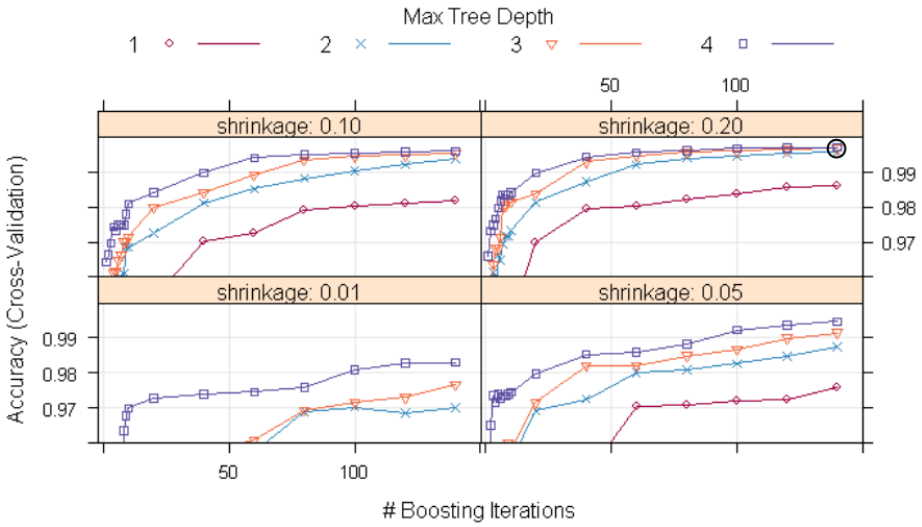


Figure 5.7: GBM class accuracy evaluation

5.1.5 Random Forest

As performed with others algorithms Random Forest was trained and accuracy obtained for each of the parameter combination which in this case cover a maximum rule depth from 2 to 5 and number of randomly selected predictors from 2 to 10 with step of 2. Results are shown in the figure below.

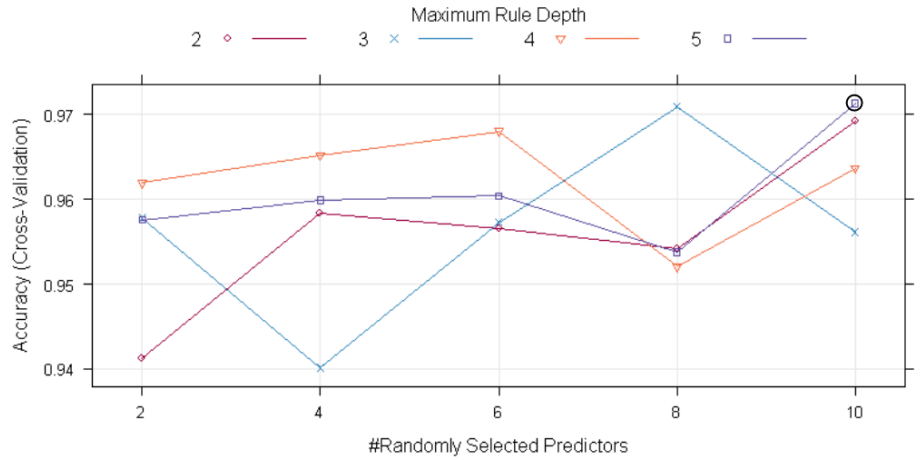


Figure 5.8: Random Forest class accuracy evaluation

From the graph can be observed that best performance is obtained with a maximum rule depth of 5 and 10 randomly selected predictors. In this case accuracy has a value of 0.97134 with a Kappa value of 0.9422744.

5.1.6 Training time consumption

It is possible to obtained the time taken by caret to perform training on each of the algorithms. Available metrics include total time pass considering all possible parameter combinations and all cross validation performed. It is also possible to get time taken for fitting the final model that is using the whole training dataset. This measure will allow to compare training time for each of the algorithms as training dataset is the same. Following table shows total time taken for training the model with full dataset, this is with the 125K+ samples:

Algorithm	**C50**	**J48**	**JRip**	**GBM**	**RF**
Total Time (s)	136.47	21.27	404.56	41.02	5279.65

Table 5.1: Training time binary classification

5.1.7 Summary

This section will summarize results obtained on training phase for all the algorithms covered.

First in Figure 5.9 performance obtained from all samples is showed. It includes minimum, first quartile, median, mean, third quartile and maximum values.

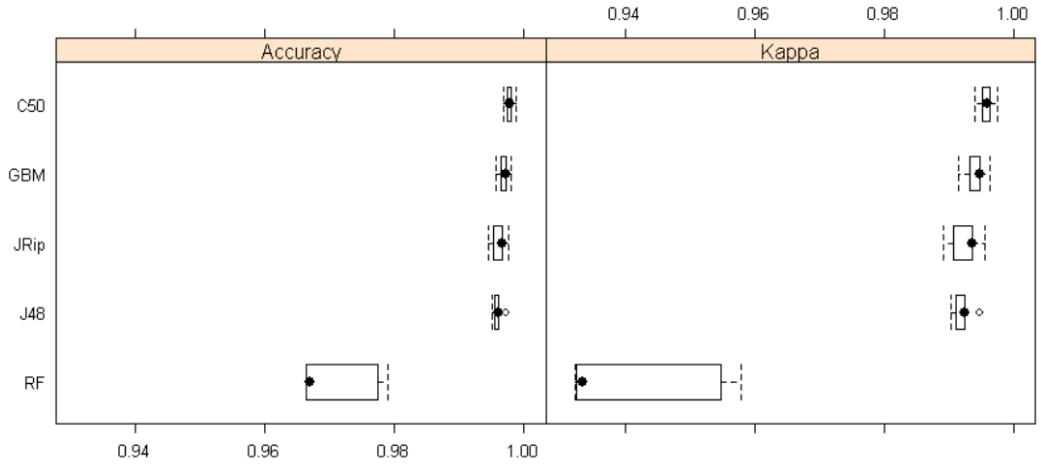


Figure 5.9: Algorithm accuracy evaluation cross validation

It can be observed in the re sampled distribution that random forest has the lowest performance with a high variance for both accuracy and kappa.

Best performing model in this case is C50 with less variance and highest performance for both accuracy and kappa.

Figure 5.10 shows accuracy and kappa obtained from the cross validation training when the best parameter set was used. Similarly as observed in re sampled distribution from Figure 5.9 best algorithm is again C50.

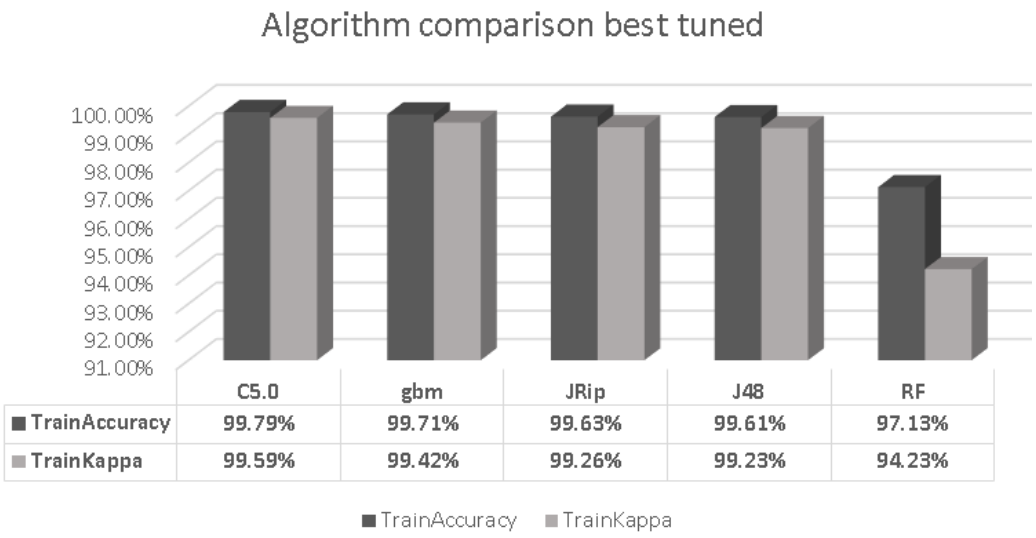


Figure 5.10: Best tuned results accuracy and kappa

It can be observed that all algorithms except random forest provide similar and good performance in the range of 99.8-99.6 per cent. Best algorithm is C50 followed in order by gbm, JRip, J48 and RF.

5.2 Test Dataset results

After training evaluation, set of parameters that provide best performance are known for each algorithm. Next sections outline performance obtained when predictions are obtained on the test dataset.

5.2.1 C50

For algorithm C50 confusion matrix and associated metrics are presented below:

	Reference	
Prediction	anomaly	normal
anomaly	9236	506
normal	3597	9204

Table 5.2: Confusion matrix C50 binary classification

Accuracy	**Kappa**	**Sensitivity**	**Specificity**
80.91%	62.76%	69.20%	96.40%
Balanced Accuracy	**Detection Rate**	**Detection Prevalence**	**Prevalence**
82.80%	39.39%	40.94%	56.92%

Table 5.3: C50 binary classification main metrics

This algorithm provides good results, with a 80.91% of accuracy meaning that around 8 of 10 samples are classified correctly. Important to note that specificity is also high 96.40% so only 3.6% of the normal traffic is considering as attack. Sensitivity has a fair value of 69.20% around 7 of 10 attacks are successfully detected.

5.2.2 J48

Below metrics calculated from confusion matrix for algorithm J48:

	Reference	
Prediction	anomaly	normal
anomaly	9349	536
normal	3484	9174

Table 5.4: Confusion matrix J48 binary classification

Accuracy	**Kappa**	**Sensitivity**	**Specificity**
82.17%	64.93%	72.85%	94.48%
Balanced Accuracy	**Detection Rate**	**Detection Prevalence**	**Prevalence**
83.67%	41.47%	43.85%	56.93%

Table 5.5: J48 binary classification main metrics

Overall this algorithm has a good performance, 94.48% of specificity so very few packets are considered as attacks when they are not, around 7 of 10 attacks are detected by a sensitivity of 72.85%. This results in a good accuracy of 82.17%.

5.2.3 JRip

Similarly for algorithm JRip results obtained are showed below:

	Reference	
Prediction	anomaly	normal
anomaly	6825	186
normal	6008	9524

Table 5.6: Confusion matrix JRip binary classification

Accuracy	**Kappa**	**Sensitivity**	**Specificity**
72.52%	47.78%	53.18%	98.08%
Balanced Accuracy	**Detection Rate**	**Detection Prevalence**	**Prevalence**
75.63%	30.28%	31.10%	56.93%

Table 5.7: JRip binary classification main metrics

In this case sensitivity is quite low, only 53.18% of the attacks are detected and even specificity is very high, 98.08% this result in a degraded accuracy of 72.52%.

5.2.4 GBM

In case of GBM confusion matrix and metrics that are calculated from it are listed below:

	Reference	
Prediction	anomaly	normal
anomaly	8936	299
normal	3897	9411

Table 5.8: Confusion matrix GBM binary classification

Accuracy	**Kappa**	**Sensitivity**	**Specificity**
81.39%	63.68%	69.63%	96.92%
Balanced Accuracy	**Detection Rate**	**Detection Prevalence**	**Prevalence**
83.28%	39.64%	40.97%	56.93%

Table 5.9: GBM binary classification main metrics

In this case it can be observed a good performing algorithm, specificity is high, less than 1 of 10 packets are considered as attacks being in reality normal traffic (specificity was 96.92%). Sensitivity is also high, 7 out of 10 attacks are detected which result in an overall accuracy of 81.39%.

5.2.5 Random Forest

Results for last algorithm Random Forest are showed below:

	Reference	
Prediction	anomaly	normal
anomaly	8880	350
normal	3953	9361

Table 5.10: Confusion matrix Random Forest binary classification

Accuracy	**Kappa**	**Sensitivity**	**Specificity**
80.91%	62.76%	69.20%	96.40%
Balanced Accuracy	**Detection Rate**	**Detection Prevalence**	**Prevalence**
82.80%	39.39%	40.94%	56.92%

Table 5.11: RF binary classification main metrics

While this algorithm was the one with worst performance from the training results evaluation it has performed well against the test dataset. It has a high specificity, 96.40%, with most of the normal traffic correctly categorized. Around 70% of the attacks are detected which results in a high accuracy of 80.91%.

5.2.6 Performance on novel attacks

Last evaluation would consider novel attacks present on test dataset, this means that those specific attacks where not part of the train dataset but they are included in the test dataset. Results can be observed in Table5.14 showing that new attacks can be detected by leaning from similar attacks. Best performing algorithm in this case was C50 with 74% of the new attacks being detected. JRip perform worse with only 20% of the attacks detected and remaining algorithms archive between 40% and 60%.

		Detected	C50	J48	Jrip	GBM	RF
Class	Attack	Total	Detected attacks				
DoS	apache2	737	732	217	205	683	132
	mailbomb	293	1	0	27	0	0
	processtable	685	685	598	161	415	685
	udpstorm	2	2	2	0	0	2
Probe	mscan	996	880	518	142	436	904
	saint	319	318	316	255	307	307
R2L	httptunnel	133	127	10	1	84	109
	named	17	8	11	2	1	3
	sendmail	14	9	0	0	3	0
	snmpget	178	0	0	0	0	0
	snmpguess	331	0	0	0	3	0
	worm	2	0	0	0	0	0
	xlock	9	0	0	0	0	0
	xsnoop	4	2	1	1	0	0
U2R	ps	15	6	4	2	0	3
	sqlattack	2	2	0	0	0	0
	xterm	13	4	3	0	0	2
	Total/ detected	3750	2776	1680	796	1932	2147
	% detected		74.03%	44.80%	21.23%	51.52%	57.25%

Table 5.12: Novel Attacks detected in Test Dataset, class scenario

5.2.7 Overall

This section will aggregate some of the main metrics obtained for all algorithms in order to evaluate and decide which one is better for potentially be included as part of a modified Snort flow with machine learning capabilities.

First on Figure 5.11 Specificity is presented. Best performing algorithm is JRip with best result of 98.08% followed by Random Forest and GBM with 96-97%. J48 and C50 have lowest performance among all evaluated algorithms with values around 94%. This metric is important because a high value will reduce the number of wrong alerts in the system which can reduce unnecessary costs.

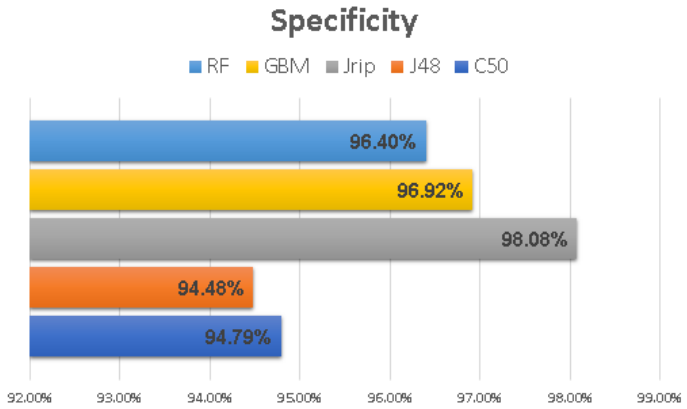


Figure 5.11: Test Dataset Specificity Results

Sensitivity is presented for each of the algorithms in Figure 5.12. Jrip is the worst algorithm in this case, only around 5 of 10 attacks are detected by the system which is not something good enough. Remaining models have around 69% in case of Random Forest and GBM and 72% for J48 and C50. Overall performance obtained has been good and many attacks are detected.

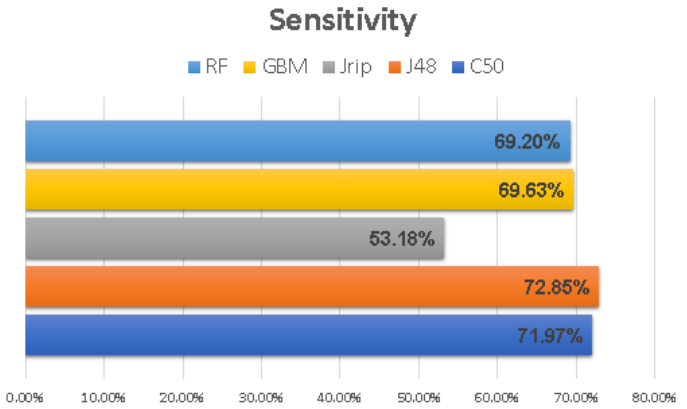


Figure 5.12: Test Dataset Sensitivity Results

Finally overall Accuracy is presented in Figure 5.13. Lowest performing algorithm is Jrip impacted by lower sensitivity. Other algorithms have similar good performance of around 80%.

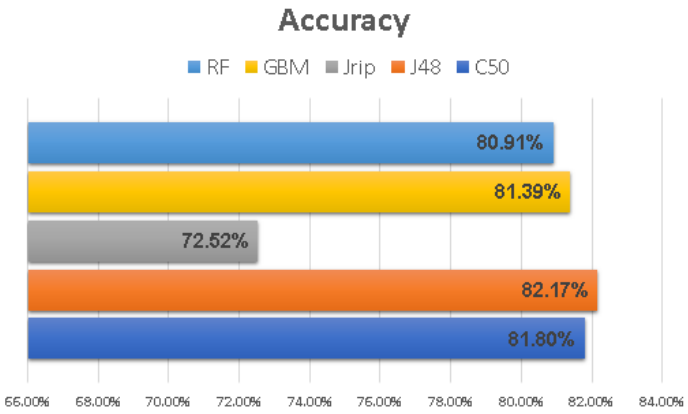


Figure 5.13: Test Dataset Accuracy Results

As a final conclusion algorithm that would be selected from the ones evaluated is GBM. Jrip algorithm would not be consider as sensitivity is too low. J48 and C50 have lower specificity than the rest and it is preferred to have a better value to reduce wrong alerts that can cause increasing costs in terms of time and resources in the organizations when there is no reason to inspect that traffic. Random forest is then discarded against GBM due to the very high training time required, as presented in Section 5.1.6 being 100 times higher than GBM.

It is also observed that algorithms are able to detect new attacks by learning from previous known attacks which aggregated for all algorithms in Figure 5.14. GBM archive a decent rate of 50% of the new attacks being detected.

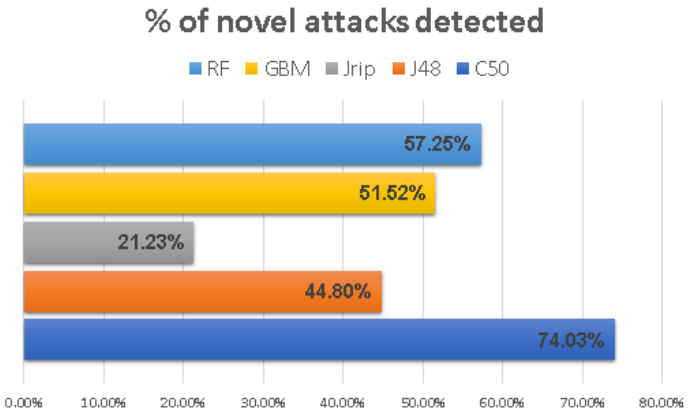


Figure 5.14: Novel attacks detection performance class

CHAPTER 6

RESULTS ANALYSIS: MULTICLASS CLASSIFICATION

Following chapter cover analysis of results obtained for all algorithms: C50, J48, JRip, GBM and Random Forest in the multiclass scenario: Dos, Probe, R2L, U2R and normal traffic. Parameter tuning and evaluation of best parameter set on the test dataset is provided.

6.1 Training phase results

6.1.1 C50

Algorithm C50 was evaluated to obtain the best value for its parameters. In this case options for tuning were the number of boosting iterations, named trials, and winnow with logical value. Following graph shows obtained accuracy for all possible combination evaluated:

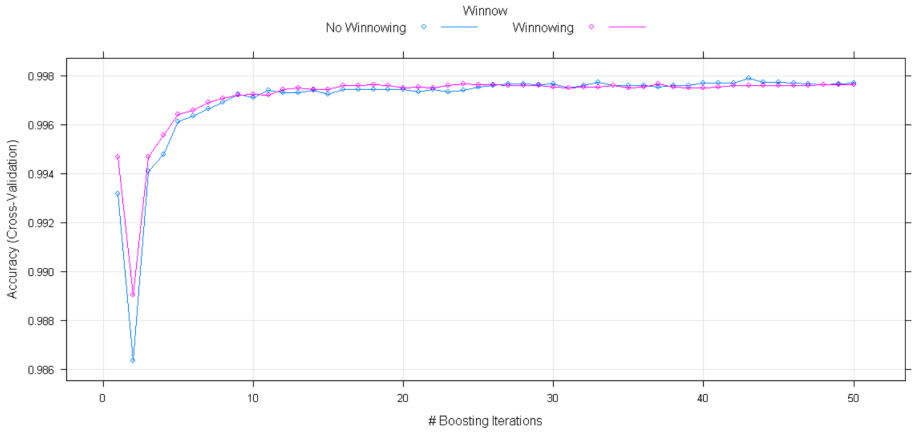


Figure 6.1: C50 multi class accuracy all results

It can be observed that there is no much difference when using winnowing or not and that performance increase with the number of iterations. From approximately nine iterations accuracy seems to converge. To get better visibility graph is zoomed around the area and shown in next figure:

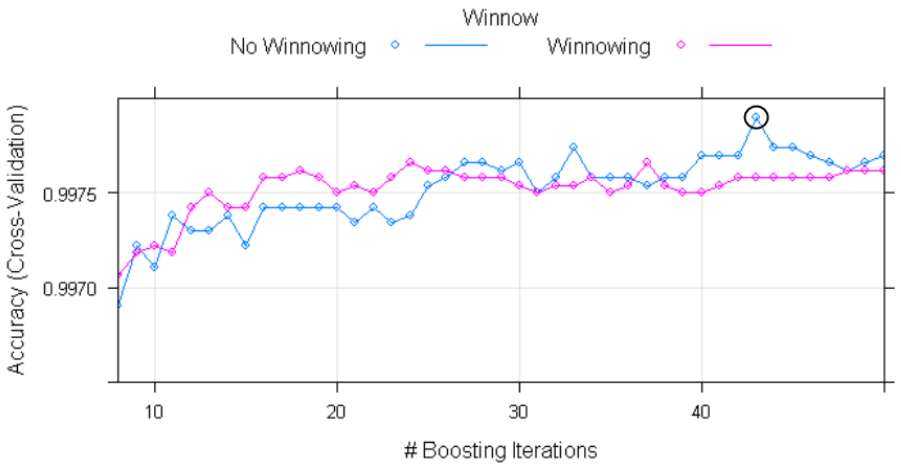


Figure 6.2: C50 multi class accuracy evaluation detail

In this case highest accuracy was obtained when number of trial is 43 and winnow is false. For that parameter set an accuracy of 0.9978961 and a kappa of 0.9963195 was reached on the cross validation.

6.1.2 J48

Model J48 was evaluated with a parameter set that includes tuning of confidence threshold and minimum instances per leaf. Values used for confidence threshold start from 0 to 0.5 using a step of 0.05, resulting in a total 11 possible values. Similarly for minimum instance per leaf a range from 0 to 375 in step of 25 was considered.

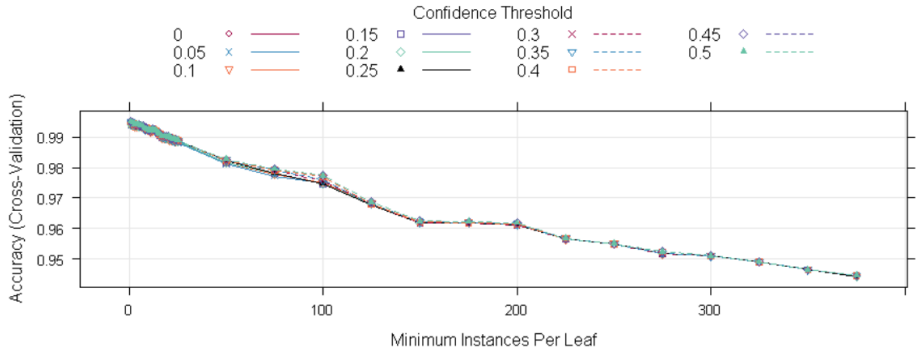


Figure 6.3: J48 multi class accuracy evaluation all results

Figure 6.3 shows results for all configured combinations. It can be observed that accuracy decrease when minimum instances per leaf increase with similar values for confidence threshold. In the lowest range of parameter minimum instances per leaf there is difference in terms of the confidence threshold used and in this area highest accuracy is archived.

A more detailed view on best accuracy range is shown in Figure 6.4 where best parameter set is highlighted and was obtained with 1 minimum instance per leaf and a confidence threshold of 0.5. For those settings train accuracy was 0.9951572 with a Kappa of 0.9915263.

6.1.3 JRip

Algorithm JRip was trained covering a range for parameters number of folds, minimum weight and number of optimization presented below:

- MinWeight from 1 to 6
- Number of optimizations from 1 to 6
- Number of folds from 2 to 7

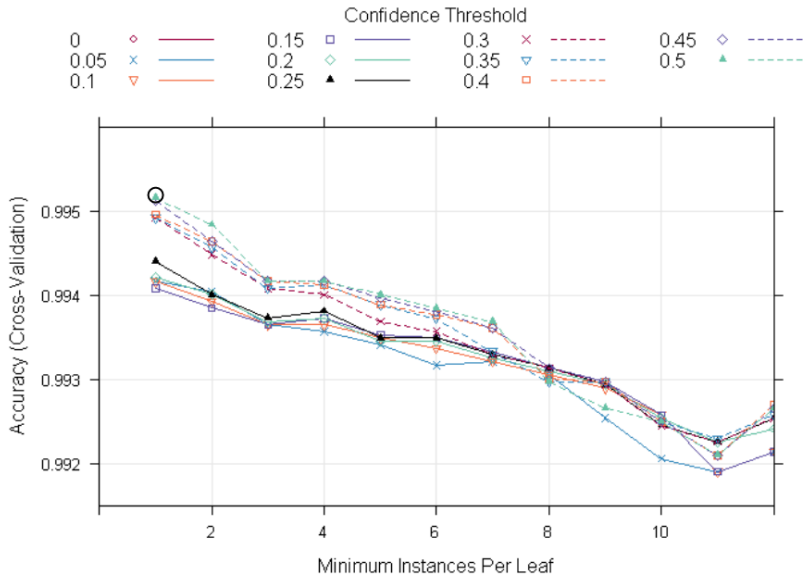


Figure 6.4: J48 multi class accuracy zoom

All results obtained are presented in Figure 6.5 where best parameter combination provided highest accuracy is highlighted. Best value was obtained with number of optimization equals to 5, number of folds equals to 3 and a minimum weight of 1. For those settings training accuracy was 0.9963084 with a Kappa of 0.9935434.

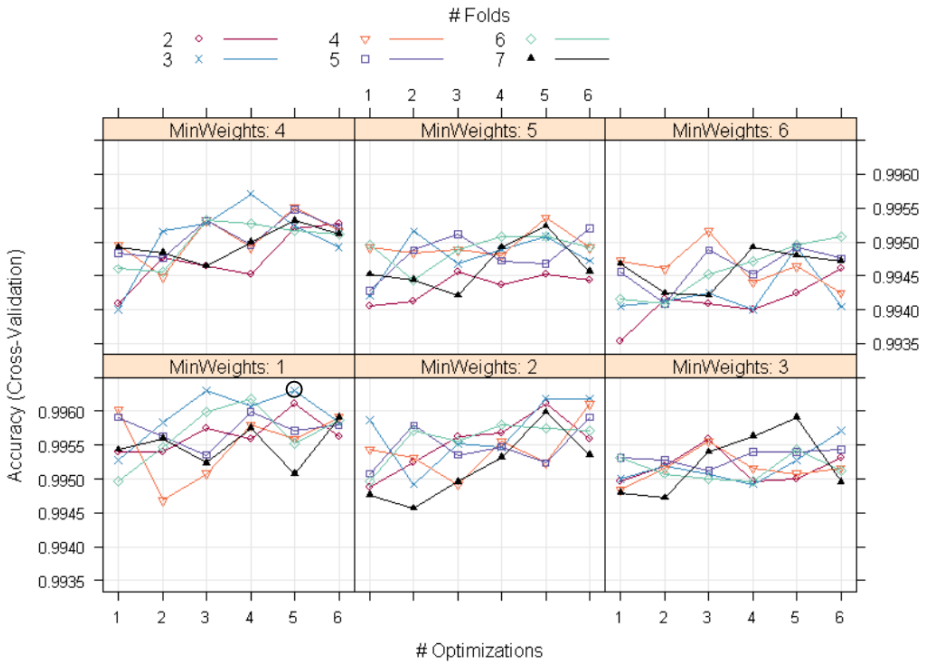


Figure 6.5: JRip multi class accuracy evaluation

6.1.4 GBM

Similarly for model GBM all results obtained from training all defined combinations of parameters are shown in 6.6. It is observed an increase in performance when a higher number of boosting iterations is performed. Also there is a correlation on maximum tree depth as accuracy tends to be higher when that parameter increase. In addition a higher value for shrinkage seems to provide better results however difference between 0.1 and 0.2 is not that big.

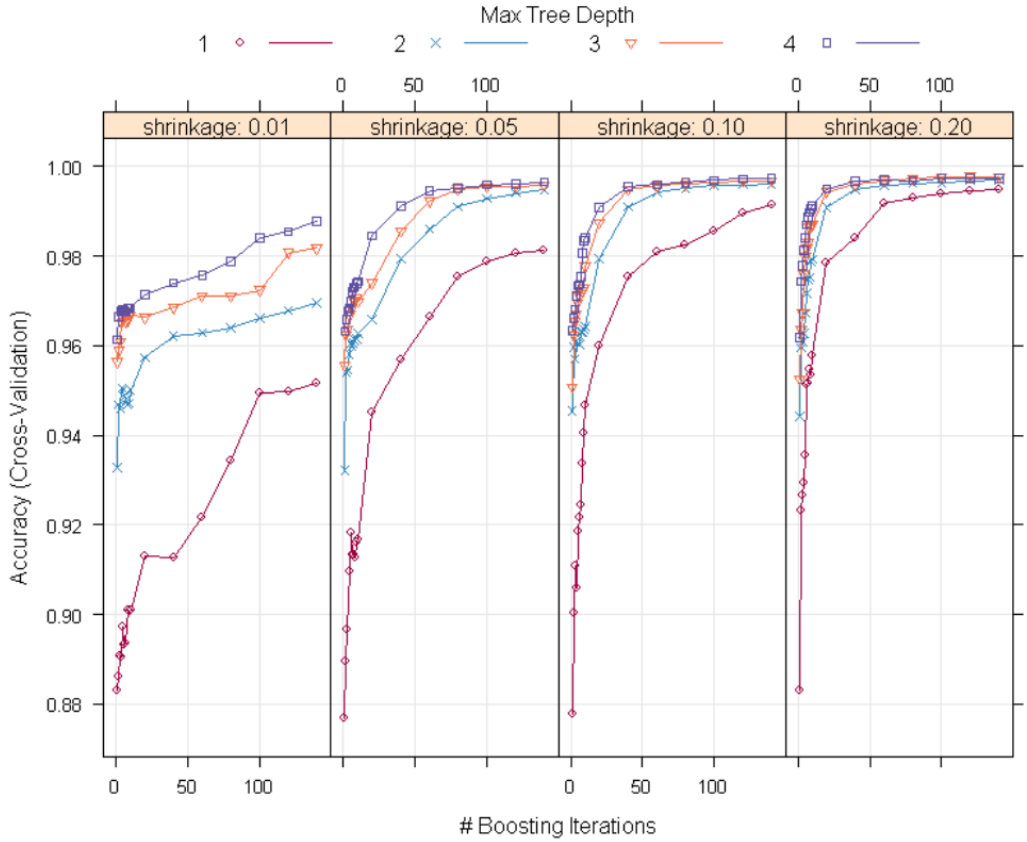


Figure 6.6: GBM multi class training accuracy all results

To visualize where best parameter combination fall a detailed view of the shrinkage equals to 0.1 and 0.2, that provide best performance, is presented in Figure 6.7. In this graph best tuned parameter is highlighted and correspond to 120 boosting iterations, a maximum tree depth of 3 and a shrinkage of 0.2. Accuracy resulting on training with those settings was 0.9977503 with a Kappa of 0.9960653.

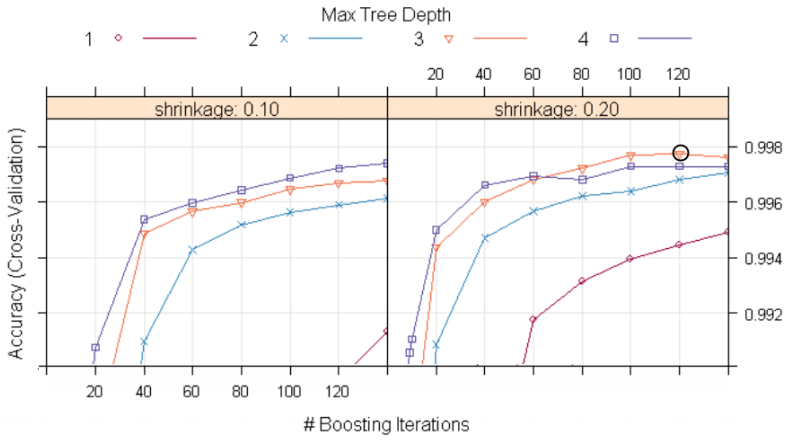


Figure 6.7: GBM multi class training accuracy zoomed

6.1.5 Random Forest

Last algorithm evaluated was random forest where performance against different values of number of random predictors and maximum rule depth was evaluated. Results are presented in Figure 6.8 where it can be observed that parameter combination that provides highest accuracy was 8 randomly selected predictors with a maximum rule depth of 5. For this configuration of the algorithm train accuracy was 0.9570497 with a subsequent kappa of 0.9237541.

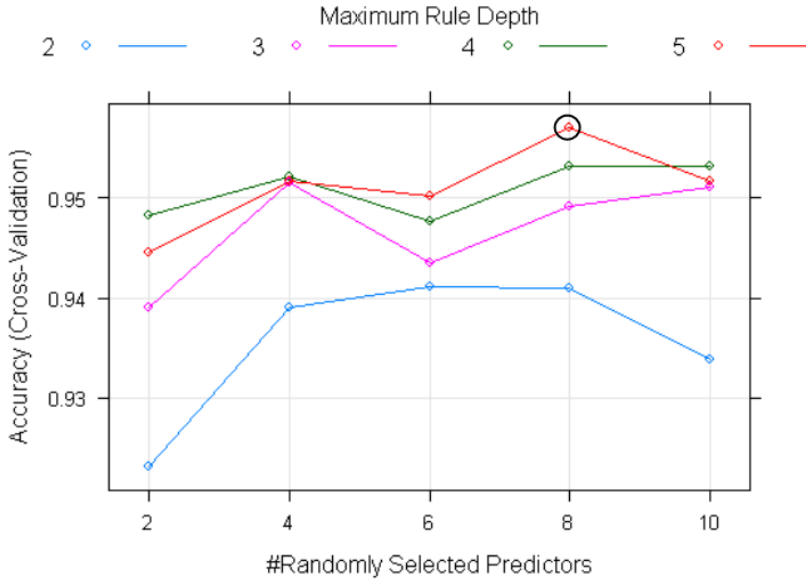


Figure 6.8: Random Forest parameter tuning multiclass

6.1.6 Training time consumption

Time consumption when the models were trained with the full train dataset, i.e 125K+ samples, was collected for evaluation. Results are showed in Table 6.1. Most of the algorithms perform fairly quick to complete the training phase, being the slowest algorithm Random forest which is 100 times more than the others.

Algorithm	**C50**	**J48**	**JRip**	**GBM**	**RF**
Total Time (s)	119.70	45.15	442.26	105.86	6371.06

Table 6.1: Training time binary classification

6.1.7 Overall

Below results obtained from all possible parameter combination and all cross validation tests performed are showed . Random Forest shows lowest performance with approximately 3% lower accuracy. Remaining algorithms have low deviation from a best tuned performance of around 99.7% as illustrated in Figure 6.10.

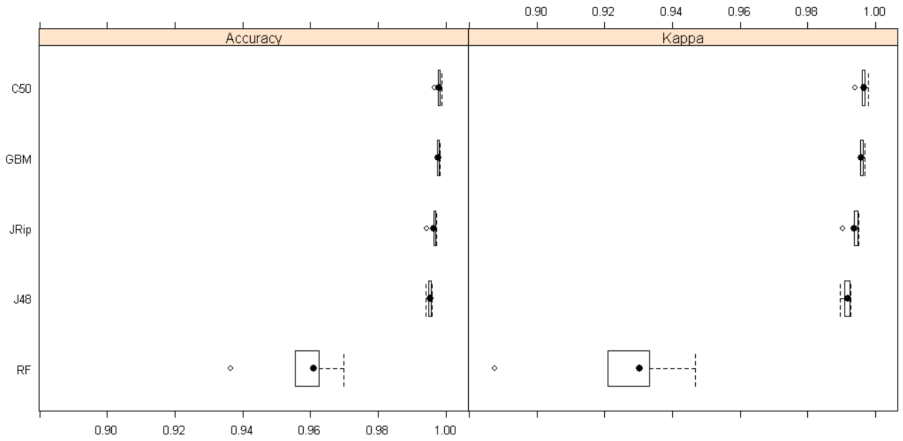


Figure 6.9: Algorithm accuracy evaluation cross validation

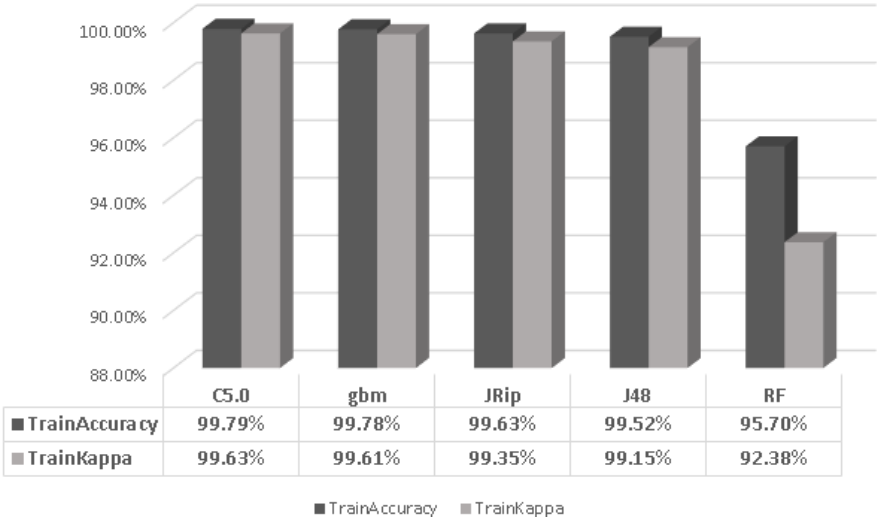


Figure 6.10: Algorithm accuracy on best parameter set

6.2 Test Dataset results

Once parameter settings are defined for each algorithm next step was to obtain predictions on the test dataset and evaluate performance. This is covered in next sections for each of the models selected: C50, J48, JRip, GBM and RF.

6.2.1 C50

After predictions are computed using C50 model a comparison with expected values is performed. This results in the well known confusion matrix which is showed in Table 6.2. While confusion matrix does not provide too much detail with absolute numbers it can be observed that no single samples for attack types R2L and U2R are detected. Number of wrong alerts is low as normal class has a sensitivity of 95.73% meaning that percentage will be correctly classified as normal when it is actually normal. Overall accuracy is of 72.19%.

	Reference				
Prediction	DoS	normal	Probe	R2L	U2R
DoS	**4800**	52	156	1	0
normal	2349	**9295**	86	2765	66
Probe	309	363	**2179**	121	1
R2L	0	0	0	**0**	0
U2R	0	0	0	0	**0**
Total	7458	9710	2421	2887	67

Table 6.2: C50 multi class confusion matrix

Overall Statistics	
Accuracy	72.19%
Kappa	56.14%

Table 6.3: C50 multi class overall statistics

Metric	**DOS**	**Normal**	**Probe**	**R2L**	**U2R**
Sensitivity	64.36%	95.73%	90.00%	0.00%	0.00%
Specificity	98.61%	58.97%	96.05%	100.00%	100.00%
Prevalence	33.08%	43.07%	10.74%	12.81%	0.30%
Detection Rate	21.29%	41.23%	9.67%	0.00%	0.00%
Detection Prevalence	22.22%	64.59%	13.19%	0.00%	0.00%
Balanced Accuracy	81.49%	77.35%	93.03%	50.00%	50.00%

Table 6.4: C50 overall statistics per class test dataset

6.2.2 J48

Next algorithm is J48 below confusion matrix, overall statistics and several metrics per class are showed. In this case we can observe that a small number of samples have been correctly classified as R2L and U2R, while sensitivity for those classes is low, 4.64% and 11.94% respectively system can detect some of them. Also number of false detected attacks is low, observed in sensitivity for normal class which is 94%.

	Reference				
Prediction	DoS	normal	Probe	R2L	U2R
DoS	**5152**	62	207	1	0
normal	2215	**9181**	610	2472	48
Probe	88	463	**1604**	277	10
R2L	3	3	0	**134**	1
U2R	0	1	0	3	**8**
Total	7458	9710	2421	2887	67

Table 6.5: J48 multi class confusion matrix

Overall Statistics	
Accuracy	71.33%
Kappa	54.52%

Table 6.6: J48 multi class overall statistics

Metric	**DOS**	**Normal**	**Probe**	**R2L**	**U2R**
Sensitivity	69.08%	94.55%	66.25%	4.64%	11.94%
Specificity	98.21%	58.35%	95.84%	99.96%	99.98%
Prevalence	33.08%	43.07%	10.74%	12.81%	0.30%
Detection Rate	22.85%	40.73%	7.12%	0.59%	0.04%
Detection Prevalence	24.05%	64.44%	10.83%	0.63%	0.05%
Balanced Accuracy	83.65%	76.45%	81.05%	52.30%	55.96%

Table 6.7: J48 Multi class metrics on test dataset

6.2.3 JRip

For algorithm JRip we can observe a fair overall accuracy of 75%. Sensitivity of class Normal is good with 97.31% of the normal samples correctly classified as normal which results in very low wrong alerts in the system. Also R2L and U2R classes shows some detection which even low is positive considering the low number of samples present in the train dataset.

	Reference				
Prediction	DoS	normal	Probe	R2L	U2R
DoS	**5355**	52	198	0	0
normal	1924	**9449**	640	2264	50
Probe	179	204	**1583**	3	0
R2L	0	3	0	**616**	7
U2R	0	2	0	4	**10**
Total	7458	9710	2421	2887	67

Table 6.8: JRip multi class confusion matrix

Overall Statistics	
Accuracy	75.47%
Kappa	61.13%

Table 6.9: JRip multi class overall statistics

Metric	**DOS**	**Normal**	**Probe**	**R2L**	**U2R**
Sensitivity	71.80%	97.31%	65.39%	21.34%	14.93%
Specificity	98.34%	61.99%	98.08%	99.95%	99.97%
Prevalence	33.08%	43.07%	10.74%	12.81%	0.30%
Detection Rate	23.75%	41.92%	7.02%	2.73%	0.04%
Detection Prevalence	24.86%	63.55%	8.73%	2.78%	0.07%
Balanced Accuracy	85.07%	79.65%	81.73%	60.64%	57.45%

Table 6.10: JRip Multi class metrics on test dataset

6.2.4 GBM

Results for GBM algorithm in multi class scenario are showed below. It can be observed that none R2L or U2R classes are detected, indicated by a of 0% for those classes. In addition 97.44% of sensitivity for normal class is a positive result as false attack rate would be low in a system implemented with this algorithm.

	Reference				
Prediction	DoS	normal	Probe	R2L	U2R
DoS	**5347**	33	252	6	1
normal	1888	**9461**	807	2878	66
Probe	223	216	**1362**	3	0
R2L	0	0	0	**0**	0
U2R	0	0	0	0	**0**
Total	7458	9710	2421	2887	67

Table 6.11: GBM multi class confusion matrix

Overall Statistics	
Accuracy	71.73%
Kappa	54.41%

Table 6.12: GBM multi class overall statistics

Metric	**DOS**	**Normal**	**Probe**	**R2L**	**U2R**
Sensitivity	71.69%	97.44%	56.26%	0.00%	0.00%
Specificity	98.06%	56.06%	97.80%	100.00%	100.00%
Prevalence	33.08%	43.07%	10.74%	12.81%	0.30%
Detection Rate	23.72%	41.97%	6.04%	0.00%	0.00%
Detection Prevalence	25.01%	66.98%	8.00%	0.00%	0.00%
Balanced Accuracy	84.88%	76.75%	77.03%	50.00%	50.00%

Table 6.13: GBM Multi class metrics on test dataset

6.2.5 Random Forest

Last algorithm evaluated is Random Forest, which archive a fair overall accuracy of 76.61%. Detection capabilities for this algorithm are good, 97.66% of normal samples, 77.89% of DoS samples and 81.66% of Probe samples are correctly classified. This translates into a good detection rate of attacks with a low number of false alerts. On the other hand no instances of U2R and R2L are detected but volumes are low.

	Reference				
Prediction	DoS	normal	Probe	R2L	U2R
DoS	**5809**	57	1	0	0
normal	1590	**9484**	443	2860	67
Probe	59	170	**1977**	27	0
R2L	0	0	0	**0**	0
U2R	0	0	0	0	**0**
Total	7458	9710	2421	2887	67

Table 6.14: RF multi class confusion matrix

Overall Statistics	
Accuracy	76.61%
Kappa	62.71%

Table 6.15: RF multi class overall statistics

Metric	**DOS**	**Normal**	**Probe**	**R2L**	**U2R**
Sensitivity	77.89%	97.66%	81.66%	0.00%	0.00%
Specificity	99.62%	61.35%	98.73%	100.00%	100.00%
Prevalence	33.08%	43.08%	10.74%	12.81%	0.30%
Detection Rate	25.77%	42.07%	8.77%	0.00%	0.00%
Detection Prevalence	26.02%	64.07%	9.91%	0.00%	0.00%
Balanced Accuracy	88.75%	79.51%	90.19%	50.00%	50.00%

Table 6.16: RF Multi class metrics on test dataset

6.2.6 Performance on novel attacks

This section will cover detection evaluation of novel attacks, i.e attacks that were not part of the train dataset. It can be observed that system was able to learn new attacks such probe saint almost fully detected by all algorithms.

For new U2R and R2L attacks detection is almost null, but itself performance on those classes is very low mainly caused by the limited number of samples in the train dataset and its imbalance.

			C50	J48	Jrip	GBM	RF
Class	Attack	Total	# Detected as specific class				
DoS	apache2	737	147	80	638	187	527
	mailbomb	293	0	0	0	0	0
	processtable	685	0	125	0	0	0
	udpstorm	2	0	0	0	0	0
Probe	mscan	996	755	217	170	49	572
	saint	319	319	306	310	310	313
U2R	httptunnel	133	0	0	0	0	0
	named	17	0	0	0	0	0
	sendmail	14	0	0	0	0	0
	snmpgetattack	178	0	0	0	0	0
	snmpguess	331	0	0	0	0	0
	worm	2	0	0	0	0	0
	xlock	9	0	0	0	0	0
	xsnoop	4	0	0	0	0	0
R2L	ps	15	0	1	0	0	0
	sqlattack	2	0	2	0	0	0
	xterm	13	0	1	2	0	0
	Total/ detected	3750	**1221**	**732**	**1120**	**546**	**1412**
	% Detected		**32.56**	**19.52**	**29.87**	**14.56**	**37.65**

Table 6.17: Detected novel attacks multi class scenario

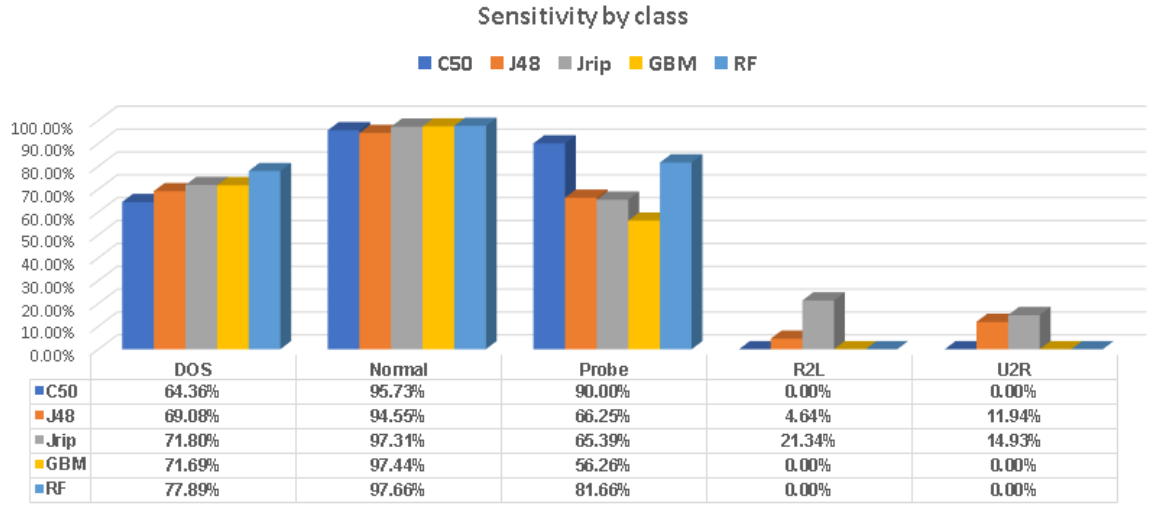
6.2.7 Summary

This section will aggregate most relevant metrics for all the algorithms tested and a evaluation with the aim of decide which one has better performance will be done.

Figure 6.11 shows sensitivity per class,for each class it shows the percentage of samples correctly classified for that specific class. The most relevant data to look is for class Normal. This class specific metric shows for normal traffic what percentage is detected as normal, a higher value results in a low number or wrong alerts irrespective of the class. Performance is good for all algorithms with values between 94% to 97%.

It can be observed as well that algorithms C50, GBM and Random Forest are unable to detect any single sample for attacks type R2L and U2R. Remaining algorithms J48 and JRip can detect some of the attacks but performance is low anyway. This is caused by the fact that train data is unbalance for those classes.

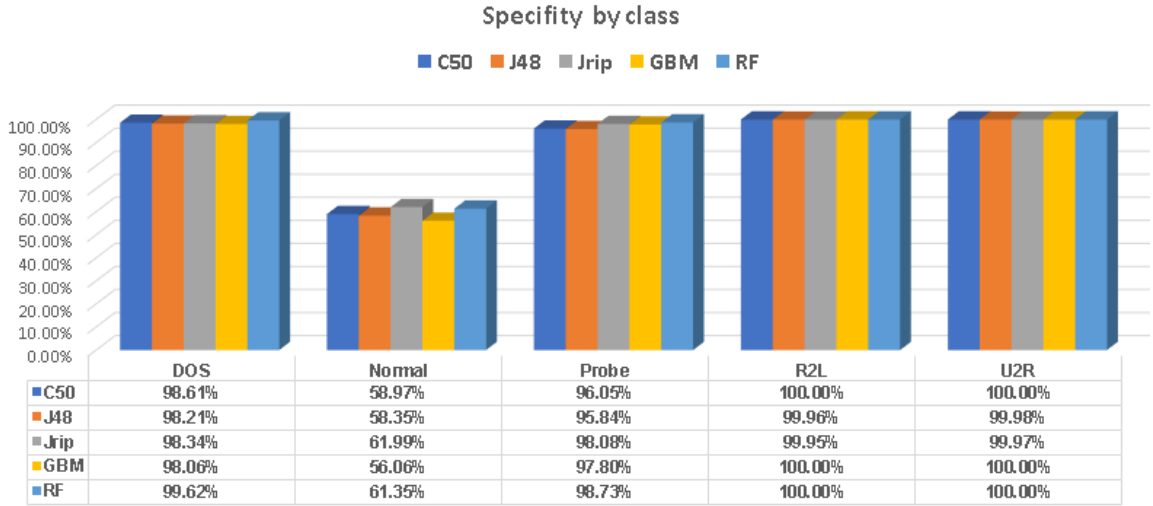
For remaining classes Dos and Probe best results are obtained from algorithm Random Forest with a detection of 77.69% and 91.86% respectively. It is also the algorithm with highest sensitivity for normal class.



	DOS	Normal	Probe	R2L	U2R
C50	64.36%	95.73%	90.00%	0.00%	0.00%
J48	69.08%	94.55%	66.25%	4.64%	11.94%
Jrip	71.80%	97.31%	65.39%	21.34%	14.93%
GBM	71.69%	97.44%	56.26%	0.00%	0.00%
RF	77.89%	97.66%	81.66%	0.00%	0.00%

Figure 6.11: Sensitivity by class Multi class scenario

Similarly Figure 6.12 shows specificity obtained for each class and algorithm. In this case metric for class Normal shows that around 60% of the other classes, i.e attacks, are not detected as normal class meaning that the attacks are detected even they fall into a different category attack.



	DOS	Normal	Probe	R2L	U2R
C50	98.61%	58.97%	96.05%	100.00%	100.00%
J48	98.21%	58.35%	95.84%	99.96%	99.98%
Jrip	98.34%	61.99%	98.08%	99.95%	99.97%
GBM	98.06%	56.06%	97.80%	100.00%	100.00%
RF	99.62%	61.35%	98.73%	100.00%	100.00%

Figure 6.12: Specificity by class Multi class scenario

Next overall accuracy is showed for all algorithms in Figure 6.13. Best performing algorithm is Jrip followed by J48 with very similar performance. All models archive a decent accuracy of around 75%.

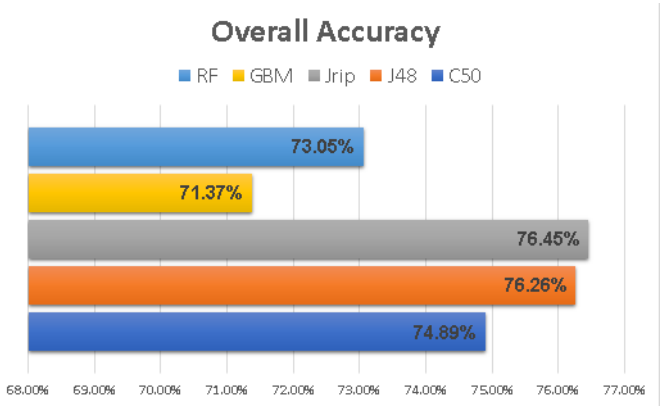


Figure 6.13: Overall accuracy multi class scenario

Figure 6.14 highlights the number of new attacks that have been correctly detected as its specific class. Best performing model is in this case RandomForest with 37.65% followed by C50 with 32.56%. In general results are low, this is partially caused by the fact that in the novel attacks there are significant number of attacks falling into U2R where performance is quite low for all algorithms due to data unbalance.

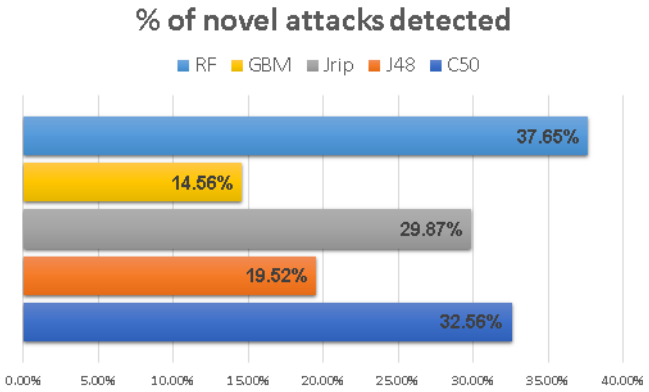


Figure 6.14: Percentage of novel attacks with class correctly detected

Still presented information indicates that novel attacks can be detected by learning from previous known attacks, this is possible because in some cases new attacks are variations of other attacks.

From all the information presented algorithm that has overall best performance would be JRip, it has the highest overall accuracy, 97.31% of the normal samples are correctly categorized so wrong attack numbers would be low. Detection of attack classes is fair, 71.80% of Dos attacks, 65.39% of Probe are detected. For classes R2L and U2R which have worst performance by far this algorithm is the best with 21.34% of R2L samples and 14.93% U2R samples classified correctly.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

In this chapter we present the final conclusion of this project, taking a look to overall results for both types of experiments to finally propose several possible improvements for future work.

7.1 Conclusion

As it has been presented in this final project security on communications over networks is an important concern for many organizations and users. Illegitimates attacks have increased during all these years, increasing costs on companies, and approach taken by attackers keeps evolving over the time. Rise in encrypted traffic as an enhancement in security, used as a secure way of transferring sensitive information such passwords, credit card details has also make more difficult for telecom operators and security organizations to detect potential attacks.

On this project binary (normal or anomaly) and multi class (Dos, probe, R2L, U2R and normal) classification algorithms have been evaluated. Results on binary experiments on Section 5.2.7 have highlighted that binary classification provides a better overall accuracy with algorithms archiving around 80% of accuracy, detecting around of 70% of the anomalies while having a low number of wrong attacks, measured with a specificity of 94-98%.

Algorithms in multi class scenario have showed a decent performance with an overall accuracy of 70-75% even lower than binary scenario. However it has also showed that some algorithms for some specific classes can have a very good performance, for example C50 algorithm detected 90% and random forest 81.66% of the probe attacks. Approximately for Dos and probe classes typical performance was around 70%. Main problem was the very poor performance on R2L and U2R classes with 3 of the 5 algorithm not detecting a single attacks and the others from 5% to 20% of correctly detected attacks. This was possible caused by the low number of samples for those classes in the train dataset. Detection on novel attacks has been lower compared to binary scenario.

Finally on this project it has been showed from the experiments executed that novel attacks can be detected from learning on previous attacks which have similarities that machine learning algorithms are able to synthesize. On Section 5.2.6 and Section 6.2.6 it has been proven that new attacks can be detecting archiving a 74.03% of new attacks detected on test dataset by C50 algorithm in binary classification. Other algorithms archive lower performance but some algorithms reach a fair detection rate of 57.25% for random forest ad 51.52% for GBM in binary scenario.

All these results suggest that addition of machine learning capabilities in a NIDS can enhance their detection capabilities and reduce time taken to detect new attacks. The proposed approach can directly produce rules to be included to Snort which can also feedback system to update detection capabilities over the time.

7.2 Future works

One of the limitations on this project related to algorithm selection was the condition of models been able to produce directly rules or trees that can be transformed to rules in order that they can be directly included in the Snort workflow.

Further work would be to evaluate other algorithms which may provide better performance and consider the approach to include them into a NIDS not necessarily SNORT.

In addition also would be interesting to evaluate techniques that minimize impact of unbalance dataset to improve detection of low recurring attacks types such R2L or U2R in case of NSL KDD dataset is used.

Bibliography

[1] Cisco, "Cisco 2018 annual security report." https://www.cisco.com/c/en/us/products/security/security-reports.html. visited on 2017-08-20.

[2] R. Tewatia and A. Mishra, "Introduction to intrusion detection system: Review," *INTERNATIONAL JOURNAL OF SCIENTIFIC AND TECHNOLOGY RESEARCH*, vol. 4, 2015.

[3] P. Innella and O. McMillan, "An introduction to intrusion detection systems." https://www.symantec.com/connect/articles/introduction-ids. visited on 2017-12-02.

[4] "Bro ids." https://www.bro.org. visited on 2016-10-20.

[5] "Martin roesch." https://en.wikipedia.org/wiki/Martin_ Roesch. visited on 2017-11-25.

[6] O. I. S. F. (OISF), "Suricata ids." https://suricata-ids.org/. visited on 2017-11-28.

[7] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Towards generating real-life datasets for network intrusion detection," *I. J. Network Security*, vol. 17, pp. 683–701, 2015.

[8] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers and Security, Volume 31, Issue 3*, pp. 357–374, 2012. visited on 2016-09-15.

[9] C. i. f. C. University of New Brunswick, "Unb iscx 2012 ids datasets." http://www.unb.ca/cic/datasets/ids.html. visited on 2016-09-20.

[10] Information and I. Computer Science University of California, "Knowledge discovery in databases darpa archive." https://archive.ics.uci.edu/ml/databases/kddcup99/.

[11] M. I. of technology: Linconln Laboratory, "Darpa intrusion detection evaluation : Detections list file." https://www.ll.mit.edu/ideval/docs/detections_ 1999.html. visited on 2017-11-20.

[12] M. Tavallaee, E. Bagheri W. Lu, , and A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," 2009. visited on 2017-09-15.

[13] U. of New Brunswick, "Nsl-kdd dataset." http://www.unb.ca/cic/datasets/nsl.html. visited on 2017-10-15.

[14] T. Radcliffe, "Python versus r for machine learning and data analysis." https://opensource.com/article/16/11/python-vs-r-machine-learning-data-analysis. visited on 2017-12-05.

[15] S. Hettich and S. D. Bay, "The uci kdd archive: Kdd 99 task." http://kdd.ics.uci.edu/databases/kddcup99/task.html. visited on 2017-11-10.

[16] Snort, "Snort manual: how to create rules." http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node27.html. visited on 2017-11-28.

[17] C. Mazzierello, C. Sansone, and F. Olivero, "Snort preprocessor and detection engine." https://sourceforge.net/projects/s-predator/.

[18] M. Kuhn, "The caret package." https://topepo.github.io/caret/index.html. visited on 2017-11-20.

[19] J. Brownlee, "How to estimate model accuracy in r using the caret package." https://machinelearningmastery.com/how-to-estimate-model-accuracy-in-r-using-the-caret-package/. visited on 2018-04-10.

[20] J. Brownlee, "Machine learning evaluation metrics in r." https://machinelearningmastery.com/machine-learning-evaluation-metrics-in-r/. visited on 2018-04-10.

[21] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[22] M. Kuhn, "Classification using c5.0 user 2013." http://appliedpredictivemodeling.com/s/user_C50.pdf. visited on 2018-01-08.

[23] W. W. Cohen, "Fast effective rule induction," in *Twelfth International Conference on Machine Learning*, pp. 115–123, Morgan Kaufmann, 1995.

[24] U. of Waikato, "Jrip weka class documentation." http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/JRip.html. visited on 2018-05-28.

[25] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," pp. 119–139, 1997.

[26] J. Friedman, "Greedy function approximation: A gradient boosting machine," pp. 1189–1232, 2001.

[27] B. Gorman, "A kaggle master explains gradient boosting." http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/. visited on 2018-0-30.

[28] G. Ridgeway, "gbm: Generalized boosted regression models." https://CRAN.R-project.org/package=gbm. visited on 2018-0-30.

[29] L. Breiman and A. Cutle, "Classification and regression by randomforest," visited on 2018-01-15.

[30] V. K. Houtao Deng, Xin Guan, "intrees: Interpret tree ensembles." https://CRAN.R-project.org/package=inTrees. visited on 2018-03-20.