VIRTUAL PRIVATE NETWORK DETECTION THROUGH MACHINE LEARNING

Submitted
to
Dr. Miguel Grazon, Professor
University of Ottawa
Ottawa, Ontario

by
Jonathan Alexander Emile Boerger
300098639

July 10, 2020

This report examines the development of machine learning models for automated VPN detection. Directed towards web-based service providers, and data scientist, this report draws conclusion as to the ideal model as well as key model characteristics for VPN detection.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

GITHUB PROJECT REPOSITORY

https://github.com/Jonathan-RCN/BOERGER-CSI4900-VPN-Detection.git

ABSTRACT

This report discusses the methodology and results of the 4th honors project aiming to develop an ideal machine learning model to automate virtual private network (VPN). Automated VPN detection would relieve service providers of the need to manually maintain a blacklist with VPN server IP addresses.

The raw dataset used in this project was the ISCXVPN2016 dataset from the University of New Brunswick's Canadian Institute for Cybersecurity (UNB CIC). The dataset consisted of 34GB of .PCAP files containing network captures of various VPN and non-VPN traffic.

To allow further processing of the data, the raw data was preprocessed using the UNB CIC developed CICFlowMeter application to convert the PCAP data into netflows store as CSV files.

Feature engineering was conducted using a rolling window approach given the time-series nature of the netflow data. Specifically, time and connection-based rolling windows were implemented through a dynamic programing approach to generate bi-directional features. The engineered features were derived from the netflow IP addresses, timestamp, and the number of packets and bytes contained in a netflow.

The engineered features were subsequently used to train Random Forest and Gradient Boost machine learning models. The models were implemented through the sklean module. The models were first evaluated with baseline (non-engineered) features using the default sklearn model hyperparameters. Next the models were evaluated using the engineered features, including various subsets of engineered features. Subsequently, the models underwent hyperparameter optimization. The improved models were then reevaluated with the engineered features and feature subsets. Lastly, the models were assessed using different rolling windows sizes.

Through the various evaluation stages, performance metrics were collected and analyzed with particular importance given to the minimization of false negatives and false positives of VPN traffic.

Ultimately, the ideal model was determined to be a Gradient Boost model (tuned hyperparameters) trained from the connection forward engineering feature subset calculated from a rolling window with size of 5000 connections / 5 minutes.

1.0 INTRODUCTION

A virtual private network (VPN) is the creation of a private network across public network infrastructure using encapsulation (encryption) of the individual's data. VPNs are used both by individuals to increase their internet privacy, and by corporations to make their cooperate networks available through wide area networks. Moreover, a VPN can be used to mask the individual's geolocation, allowing the user to access geographically locked content (ex: accessing Twitter in China).

Although there are legitimate uses for VPNs, nefarious actors also use VPN. Therefore, service providers may wish to block VPN traffic from accessing their services (ex: preventing a connection to American Netflix from Canada with a VPN). Traditionally, the blocking of VPN was done through manually constructed IP blacklist for the various VPN services. However, this approach is insufficient to reliably block VPN traffic because maintaining a blacklist is resource intensive, and service providers are routinely changing and creating new VPN service nodes (changing IP address). Thus, an improved approach is to use machine learning to automate the detection process.

This project aims to develop an ideal machine learning model that could reliably detect VPN network traffic. Furthermore, it aims to develop a model that can be integrated in a near real-time network traffic evaluation system. The ideal model needs balance predictive capability with model efficiency.

This model will be developed through the traditional data science pipeline including data acquisition, data preprocessing, feature engineering, machine learning model implementation, and result generation.

Specifically, this report will discuss the raw data used in the project, the data-preprocessing applied to make the data feature engineering ready, the framework used for feature engineering, the implementation of the Random Forest and Gradient Boost machine learning models and the results generated by these models. Finally, the results will be analyzed to determine the ideal machine learning model for VPN detection.

2. DATA

The VPN - non VPN (ISCXVPN2016) dataset from the University of New Brunswick's Canadian Institute for Cybersecurity comprised the raw data for this project. The dataset, created by researchers G. Gil, A. Lashkari, M. Mamun, and A. Ghorbani for their research *Characterization of Encrypted and VPN Traffic Using Time-Related Features* [1] presents a cross section of real-world network traffic. The dataset compromises 7 different traffic categories (browsing, email, chat, streaming, file transfer, VOIP & TraP2P) in both non-VPN and VPN

[1] (Draper-Gil, Lashkari, Mamun, & Ghorbani, 2016)

form. Within each category, network traffic was collected using various applications and services (example: gmail chat, facebook chat, hangout chat).

The dataset contained 150 .PCAP files containing between 280 to 5 743 493 individual packets. Figure 1 shows sample PCAP data from the dataset. As discussed in the background section, a packet is the means by which information is transmitted over the internet. Along with the information being transmitted (the payload), packets have also underlying characteristic information (headers).

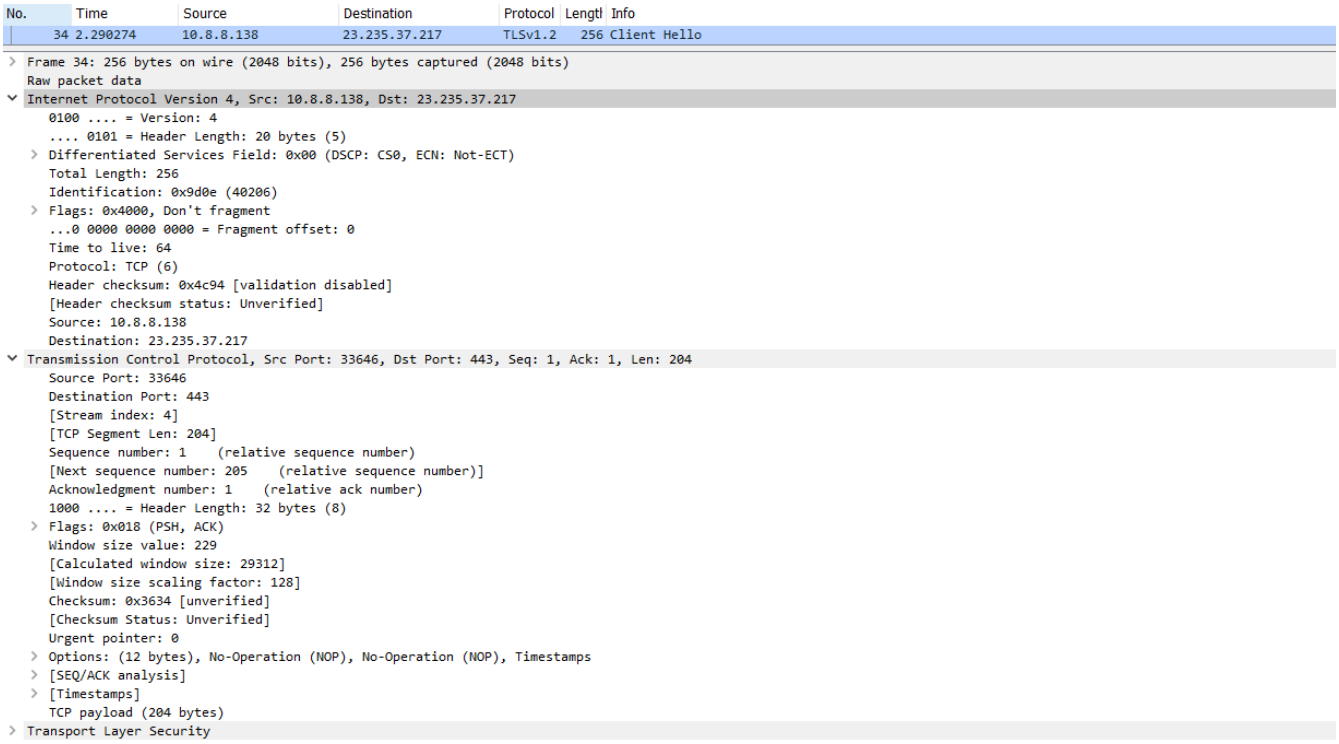


Figure 1: Packet Capture with Header Information from Vpn_Vimeo_A.Pcap of ISCXVPN2016 Dataset

The next step in the data science pipeline is the pre-processing of the raw data to prepare it for the feature engineering.

2.1 Data Pre-Processing

The information required for VPN detection is contained within the packet headers and it is therefore necessary to separate the headers from the payloads. Moreover, only basic pieces of header information are useful for the facilitation of VPN detection, such as the source and destination IP address and port number, the size of the packet, and the time of transmission. To isolate and extract the required information from the raw PCAP files, the Canadian Institute of Cybersecurity *CICFlowMeter* application was employed. CICFlowMeter convert PCAP files into network traffic flows.

A traffic flow (alternatively netflow or simply flow) is the compilation of all packets being sent between a given source and destination over a specific time period. It can be considered as "an artificial logical equivalent to call or connection"[2]. Like packets, a traffic flow also possesses characteristic information (compiled from the packets in the flow) such as the source and destination IP addresses, a port number, timestamps, the total number of packets transmitted in the flow and the total number of bytes transmitted in the flow. Figure 2 shows a traffic flow generated by CICFlowMeter based of a PCAP file from the VPN dataset.

Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestam	Flow Dura	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts	Fwd Pkt Le	Fwd Pkt Le	Fwd Pkt Le	Fwd Pkt Le	Bwd Pkt Le	Bwd Pkt Le	Bwd Pkt Le	Bwd Pkt Le	Flow Byts/	Flow Pkts/
131.202.24	131.202.24	56669	224.0.0.25	5355	17	25/05/201	410466	2	0	44	0	22	22	22	0	0	0	0	0	107.1952	4.872511
131.202.24	131.202.24	57058	224.0.0.25	5355	17	25/05/201	410825	2	0	44	0	22	22	22	0	0	0	0	0	107.1016	4.868253
131.202.24	131.202.24	54240	224.0.0.25	5355	17	25/05/201	410627	2	0	44	0	22	22	22	0	0	0	0	0	107.1532	4.8706
131.202.24	131.202.24	60710	224.0.0.25	5355	17	25/05/201	410594	2	0	44	0	22	22	22	0	0	0	0	0	107.1618	4.870992
173.194.15	173.194.15	465	131.202.24	5516	6	25/05/201	78	2	0	0	0	0	0	0	0	0	0	0	0	0	25641.03
131.202.24	131.202.24	54481	224.0.0.25	5355	17	25/05/201	420507	2	0	44	0	22	22	22	0	0	0	0	0	104.6356	4.756163
131.202.24	131.202.24	58902	224.0.0.25	5355	17	25/05/201	420501	2	0	44	0	22	22	22	0	0	0	0	0	104.6371	4.756231
131.202.24	131.202.24	137	131.202.24	137	17	25/05/201	750839	2	0	100	0	50	50	50	0	0	0	0	0	133.1843	2.663687
131.202.24	131.202.24	62695	224.0.0.25	5355	17	25/05/201	410930	2	0	44	0	22	22	22	0	0	0	0	0	107.0742	4.867009
131.202.24	131.202.24	56046	224.0.0.25	5355	17	25/05/201	422958	2	0	44	0	22	22	22	0	0	0	0	0	104.0292	4.728602
131.202.24	131.202.24	50946	224.0.0.25	5355	17	25/05/201	409762	2	0	44	0	22	22	22	0	0	0	0	0	107.3794	4.880882
131.202.24	131.202.24	54154	224.0.0.25	5355	17	25/05/201	410175	2	0	44	0	22	22	22	0	0	0	0	0	107.2713	4.875968
131.202.24	131.202.24	61669	224.0.0.25	5355	17	25/05/201	411072	2	0	44	0	22	22	22	0	0	0	0	0	107.0372	4.865328
131.202.24	131.202.24	51481	224.0.0.25	5355	17	25/05/201	410653	2	0	44	0	22	22	22	0	0	0	0	0	107.1464	4.870292
131.202.24	131.202.24	54361	131.202.24	32414	17	25/05/201	1.1E+08	23	0	483	0	21	21	21	0	0	0	0	0	4.390923	0.209092
131.202.24	131.202.24	58605	224.0.0.25	5355	17	25/05/201	411451	2	0	44	0	22	22	22	0	0	0	0	0	106.9386	4.860846
131.202.24	131.202.24	52555	224.0.0.25	5355	17	25/05/201	412412	2	0	44	0	22	22	22	0	0	0	0	0	106.6894	4.849519
131.202.24	131.202.24	55697	224.0.0.25	5355	17	25/05/201	410513	2	0	44	0	22	22	22	0	0	0	0	0	107.183	4.871953
131.202.24	131.202.24	51042	224.0.0.25	5355	17	25/05/201	414368	2	0	44	0	22	22	22	0	0	0	0	0	106.1858	4.826628
131.202.24	131.202.24	56495	224.0.0.25	5355	17	25/05/201	410445	2	0	44	0	22	22	22	0	0	0	0	0	107.2007	4.87276
131.202.24	131.202.24	55311	224.0.0.25	5355	17	25/05/201	410705	2	0	44	0	22	22	22	0	0	0	0	0	107.1329	4.869675
131.202.24	131.202.24	52119	224.0.0.25	5355	17	25/05/201	411199	2	0	44	0	22	22	22	0	0	0	0	0	107.0042	4.863825
131.202.24	131.202.24	64755	224.0.0.25	5355	17	25/05/201	431392	2	0	44	0	22	22	22	0	0	0	0	0	101.9954	4.636155
131.202.24	131.202.24	62065	224.0.0.25	5355	17	25/05/201	415032	2	0	44	0	22	22	22	0	0	0	0	0	106.0159	4.818906
131.202.24	131.202.24	60391	224.0.0.25	5355	17	25/05/201	411067	2	0	44	0	22	22	22	0	0	0	0	0	107.0385	4.865387
131.202.24	131.202.24	53122	224.0.0.25	5355	17	25/05/201	410736	2	0	44	0	22	22	22	0	0	0	0	0	107.1248	4.869308
131.202.24	131.202.24	59804	224.0.0.25	5355	17	25/05/201	419255	2	0	44	0	22	22	22	0	0	0	0	0	104.9481	4.770366
131.202.24	131.202.24	49256	224.0.0.25	5355	17	25/05/201	423431	2	0	44	0	22	22	22	0	0	0	0	0	103.913	4.72332
131.202.24	131.202.24	64804	224.0.0.25	5355	17	25/05/201	411163	2	0	44	0	22	22	22	0	0	0	0	0	107.0135	4.864251
131.202.24	131.202.24	57772	224.0.0.25	5355	17	25/05/201	411420	2	0	44	0	22	22	22	0	0	0	0	0	106.9467	4.861212
131.202.24	131.202.24	51491	224.0.0.25	5355	17	25/05/201	414378	2	0	44	0	22	22	22	0	0	0	0	0	106.1832	4.826511
131.202.24	131.202.24	63463	224.0.0.25	5355	17	25/05/201	410721	2	0	44	0	22	22	22	0	0	0	0	0	107.1287	4.869486
131.202.24	131.202.24	50181	224.0.0.25	5355	17	25/05/201	414606	2	0	44	0	22	22	22	0	0	0	0	0	106.1249	4.823857
131.202.24	131.202.24	56373	224.0.0.25	5355	17	25/05/201	449247	2	0	44	0	22	22	22	0	0	0	0	0	97.94167	4.451894
131.202.24	131.202.24	58328	224.0.0.25	5355	17	25/05/201	412041	2	0	44	0	22	22	22	0	0	0	0	0	106.7855	4.853886

Figure 2: Netflow CSV Generated by CICFlowMeter from email1a.pcap

For the purpose of this project, the conversion from PCAP file to netflow was done manually. Given that CICFlowMeter is developed in Java, it is not easily integrable into a Python project file. Furthermore, since there is no functional difference between using raw PCAPs and netflows for the eventual classification and since the operation is only conducted once, the PCAP files will not be included in the project files and the netflow csv files will be considered as the 'raw data' for the project.

2.1.1 Data filtration and consolidation

In addition to the basic flow features, CICFlowMeter also generates over 80 additional analytic features, some of which are visible in figure 2. However, given the nature of this project, these additional features are not required. Therefore, the next two steps in the data processing pipeline are to filter out the unnecessary features and consolidate all 150 netflow csv files into one csv that can be subsequently used for feature engineering.

[2] (Brownlee, Mills, & Ruth, 1999)

The desired baseline netflow features are the source IP address, the source port, the destination IP address, the destination port, the total number of packets, and the total number of bytes transmitted (length). In addition, a label was added to each flow to indicate if the flow was of type VPN or non-VPN. All but the packet number, the bytes transmitted, and the VPN label already existed in the CICFlowMeter generated flows. Therefore, these columns were simply copied over to the filtered version of the traffic flow.

It is important to note that CICFlowMeter generated features for a bi-directional flow, thus - as shown in Figure 3 -, it calculated the total number of packets and bytes transmitted in the forward direction (source to destination) as well as the reverse (destination to source). The filtered netflow requires the total number of packets/ bytes. Consequently, the forward and reverse directions of the CICFlowMeter were simply combined and added to the filtered version of the flow. Lastly, the VPN label was determined in accordance with the file name, since all PCAP files (and by extension CICFlowMeter netflows) of type VPN in the dataset start with 'vpn_'. An example of a filtered netflow is shown in Figure 4.

Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts
2	0	44	0
2	0	44	0
2	0	44	0
2	0	44	0
2	0	0	0
2	0	44	0
2	0	44	0
2	0	100	0
2	0	44	0
2	0	44	0
2	0	44	0
2	0	44	0
2	0	44	0
2	0	44	0
23	0	483	0
2	0	44	0
2	0	44	0
2	0	44	0
2	0	44	0

Figure 3: Packet Numbers & Packet Length Metric of Netflow Generated by CICFlowMeter from email1a.pcap

Src IP	Src Port	Dst IP	Dst Port	Timestamp	Tot Pkts	TotLen	VPN
131.202.240.83	56669	224.0.0.252	5355	25/05/2015 12:53:05 p.m.	2	44	0
131.202.240.102	57058	224.0.0.252	5355	25/05/2015 12:59:34 p.m.	2	44	0
131.202.240.102	54240	224.0.0.252	5355	25/05/2015 12:59:50 p.m.	2	44	0
131.202.240.102	60710	224.0.0.252	5355	25/05/2015 01:00:40 p.m.	2	44	0
173.194.196.16	465	131.202.240.242	5516	25/05/2015 12:58:27 p.m.	2	0	0
131.202.243.49	54481	224.0.0.252	5355	25/05/2015 12:58:19 p.m.	2	44	0
131.202.243.49	58902	224.0.0.252	5355	25/05/2015 12:55:43 p.m.	2	44	0
131.202.240.102	137	131.202.243.255	137	25/05/2015 01:02:47 p.m.	2	100	0
131.202.240.70	62695	224.0.0.252	5355	25/05/2015 12:59:13 p.m.	2	44	0
131.202.240.102	56046	224.0.0.252	5355	25/05/2015 12:54:35 p.m.	2	44	0
131.202.240.242	50946	224.0.0.252	5355	25/05/2015 12:55:13 p.m.	2	44	0
131.202.240.70	54154	224.0.0.252	5355	25/05/2015 12:58:47 p.m.	2	44	0
131.202.242.93	61669	224.0.0.252	5355	25/05/2015 01:01:55 p.m.	2	44	0
131.202.242.93	51481	224.0.0.252	5355	25/05/2015 01:00:48 p.m.	2	44	0
131.202.240.183	54361	131.202.243.255	32414	25/05/2015 01:00:59 p.m.	23	483	0
131.202.242.93	58605	224.0.0.252	5355	25/05/2015 01:02:00 p.m.	2	44	0
131.202.242.93	52555	224.0.0.252	5355	25/05/2015 12:55:42 p.m.	2	44	0
131.202.242.93	55697	224.0.0.252	5355	25/05/2015 12:54:21 p.m.	2	44	0
131.202.242.93	51042	224.0.0.252	5355	25/05/2015 12:55:18 p.m.	2	44	0

Figure 4: Filtered Netflow CSV (data derived from email1a.pcap)

The netflow filtering was implemented by using NumPy arrays and Pandas to allow for efficient coping of entire columns from the CICFlowMeter netflow CSV file to the filtered netflow CSV file. Additionally, all 150 filtered netflow csv files were consolidated into a singular CSV file using Panda's concatenation function file with all netflows sorted chronologically.

Overall, with the use of NumPy arrays, the filtering and consolidation of the raw data were very efficient, taking ~5 seconds to filter and consolidate. The total number of netflows in the consolidated CSV file is ~215 000 netflows of which ~23 000 are VPN netflows and the data is now ready for feature engineering.

3. FEATURE ENGINEERING

Feature engineering aims to extract additional information from the raw data so that this extra information can be passed to machine learning algorithms to maximize their predictive capability. This project uses a rolling window framework to enable the feature engineering from pre-processed data.

3.1 Rolling Window

A rolling window (RW) is an analytical framework for time-series data, where whole dataset is analyzed through fixed sized subsamples of the dataset. The RW initially starts with the oldest data, and on each iteration, it progresses to the next time-series data which is 'younger/newer' than the previous iteration. The analysis comprises the target of the RW (the youngest of newest

data entry) + the RW size – 1 trailing data entry which fall in the window. Figure 5 is a visual representation of a RW where *m* is the time-series dataset and *h* is the RW.:

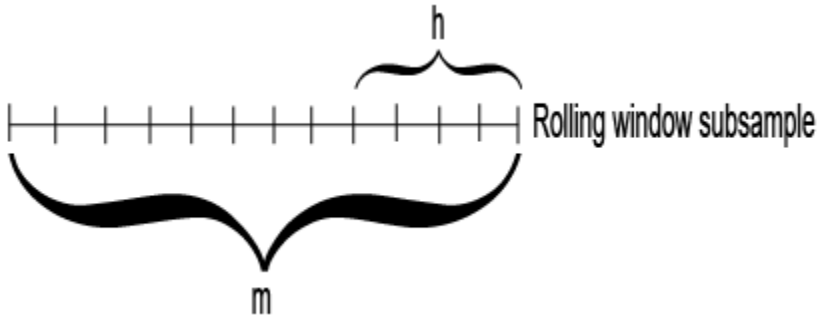


Figure 5: Rolling Window[3]

Figure 6 illustrates how a RW iterates across a time-series dataset where each subsequent RW frame is one data entry closer to reaching the end of the data set.

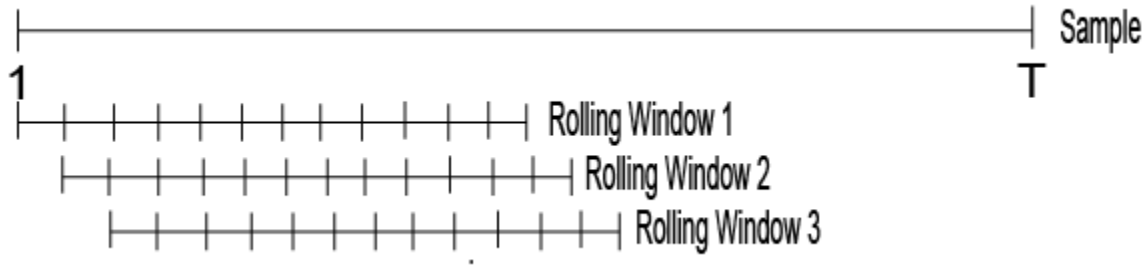


Figure 6: Rolling Window Iteration[4]

The RW framework was selected to enable the feature engineering since the netflow data represents a time-series given that netflows occur at a given time and that they occur sequentially.

Since the netflow data represents two time-series, two approaches can be employed to engineer features: a connection based RW where the size is the number of connections; and a time based RW where the size is the fixed time delta between the target netflow and the trailing netflows in the RW (for example connection size of 10 000 netflows and a time size of 10 minutes).

Additionally, given the bi-directional nature of internet communication, both forwards (source -> destination) and reverse or backward (destination -> source) features can be calculated for each target netflow through the RW approach.

For each target netflow, Table 1 shows the four categories of engineered features generated by the RW approach as applied to this project.

[3] (MathWorks Help Center)
[4] *Ibid.*

Table 1: Categories of Engineered Features

RW Type / Direction	Connection Based	Time Based
Forward	Connection - Forward	Time - Forward
Reverse	Connection - Reverse	Time - Reverse

The next section will explore methodology used to get the features for different categories of engineered features.

3.2 Features to be Engineered

Engineered features are generated for each netflow contained in the consolidated filtered netflow file. Each target netflow (the netflow for which the features are being engineered) has a given source/destination IP address tag (SDIPT); then, using this SDIPT, all netflows within the RW are checked for the same SDIPT. For all netflows with a matching SDIPT to the target netflows, the values of packet number, packet length and time data are used the generate the features for the target netflow.

Specifically, of the matching flow (including the target flow) within the RW, the minimum number of packets transmitted in one flow, the maximum packets in one flow, the total number of packets transmitted across all matching flow and the mean number of packets transmitted across the matching flows are calculated. The same minimum, maximum, total and mean values are calculated for packets length (bytes transmitted) in the netflows.

Next, with the matching flows (including the target flow) within the RW, the minimum time differential (delta) between any two of the matching flows, the maximum time delta between any two flows, and the mean time delta between matching flows are calculated.

Lastly, the number of matching flows within the RW is also recorded as engineered feature.

These engineered features are generated both for a connection based and time based RW. Furthermore, these features are forward direction features as they are being generated based on source -> destination SDIPT.

The same set of features are also generated in the reverse direction by swapping the source IP address with the destination IP address in the SDIPT. With this new (reversed SDIPT), the RW is checked again to find all matches, which are used to generate the reverse features for both the time and connection based RW.

Figure 7 shows a sample filtered data, where all data entries are in the rolling window. The target netflow for feature engineering is the last one (highlighted in dark green), matching flows are highlighted in light green, and reverse flows are highlighted in blue.

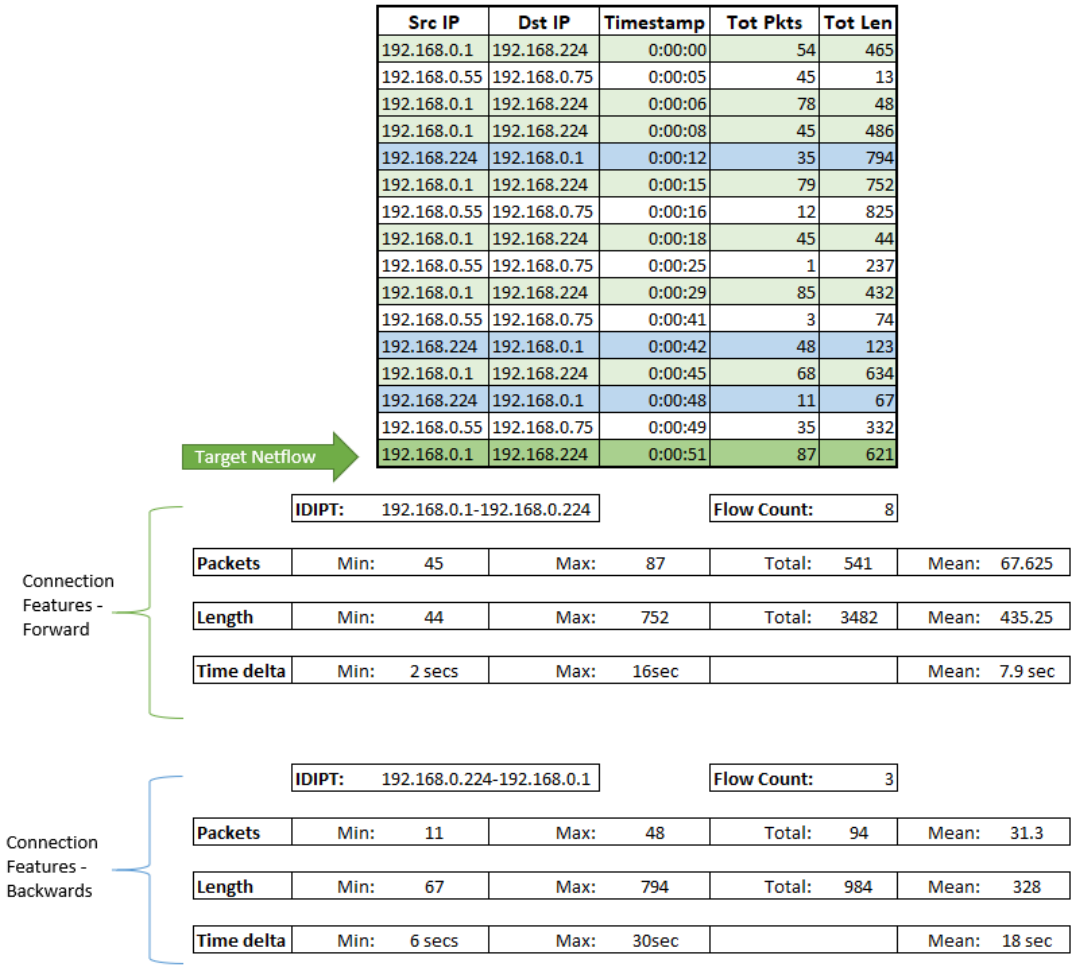


Figure 7: Feature Engineering Via Rolling Window in Forward (Green) and Reverse (Blue) Direction Example

Below the sample RW in Figure 7, the engineered features (connection), in both the forward and reverse directions are shown. These engineered features will also be calculated for a time based RW in the project implementation.

The following section will discuss how this rolling window framework was applied to this project.

3.3 Implementation

The rolling window framework to feature engineering is implemented in the project using a dynamic programing approach, so that there is only one iteration through the list of netflows (linear big-O // O(n)).

Netflows are represented through the RW_NETFLOW class, where an instance of the class is created for each unique source/destination IP tag (SDIPT). The class attributes are shown in Figure 8.

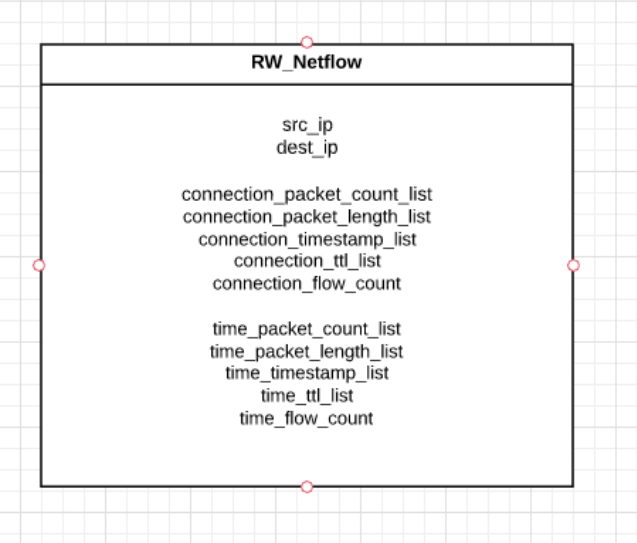


Figure 8: RW_NETFLOW Class with attributes

The class contains two categories of similar attributes: the first contains netflow feature values for connection based RWs, and the seconds contains the equivalent values expect for time based RWs. The time to live (TTL) list is used to determine whether or not a target netflow fall within the rolling window. This will be explained in further detail later in this section.

The set of all RW_NETFLOW objects are held in a dictionary, as shown if Figure 9, which uses the SDIPT as the key.

```
{'104.156.81.217-131.202.240.150': <__main__.RW_NETFLOW object at 0x000001D93148C588>,
 '108.175.39.202-131.202.240.150': <__main__.RW_NETFLOW object at 0x000001D9314937C8>,
 '108.175.42.181-131.202.240.150': <__main__.RW_NETFLOW object at 0x000001D931AE6548>,
 '119.81.4.50-131.202.240.150': <__main__.RW_NETFLOW object at 0x000001D93198DE48>,
 '131.202.240.150-104.156.81.217': <__main__.RW_NETFLOW object at 0x000001D9318EC988>,
 '131.202.240.150-104.156.85.217': <__main__.RW_NETFLOW object at 0x000001D93198AEC8>,
 '131.202.240.150-107.21.242.214': <__main__.RW_NETFLOW object at 0x000001D93147CD08>,
 '131.202.240.150-108.175.42.181': <__main__.RW_NETFLOW object at 0x000001D931493788>,
 '131.202.240.150-129.33.205.80': <__main__.RW_NETFLOW object at 0x000001D92F32AA48>,
 '131.202.240.150-131.202.243.255': <__main__.RW_NETFLOW object at 0x000001D9319900C8>,
 '131.202.240.150-131.202.244.3': <__main__.RW_NETFLOW object at 0x000001D9318EEA48>,
 '131.202.240.150-131.202.244.5': <__main__.RW_NETFLOW object at 0x000001D9318E83C8>,
 '131.202.240.150-131.202.6.3': <__main__.RW_NETFLOW object at 0x000001D9318EC208>,
 '131.202.240.150-131.253.40.59': <__main__.RW_NETFLOW object at 0x000001D931986F48>,
 '131.202.240.150-152.163.13.6': <__main__.RW_NETFLOW object at 0x000001D931478BC8>,
 '131.202.240.150-162.243.234.174': <__main__.RW_NETFLOW object at 0x000001D931982F48>,
 '131.202.240.150-173.194.123.109': <__main__.RW_NETFLOW object at 0x000001D93148F548>,
 '131.202.240.150-173.194.123.115': <__main__.RW_NETFLOW object at 0x000001D931478508>,
 '131.202.240.150-173.194.123.121': <__main__.RW_NETFLOW object at 0x000001D931973E48>,
 '131.202.240.150-173.194.123.122': <__main__.RW_NETFLOW object at 0x000001D93148F308>,
```

Figure 9: Example of dictionary containing RW_NETFLOW objects

When the RW is iterating through the list of netflows, the project first determines the SDIPT of the target netflow, then it checks the netflow dictionary to verify if a RW_NETFLOW object already exists for that SDIPT.

If there is no object for the SDIPT, then the project creates a new RW_NETFLOW object and initializes it with the network metric of the target netflow. However, if an object exists, then it is retrieved and the network metrics of the target netflow are appended to their respective lists. The metrics are added to both the time-based list and connection-based list. For the TTL list, the target netflow index in the CSV file is used as the TTL value to be appended.

In a near real-time system, there would not be a csv file index; therefore, an arbitrary number could be used for the TTL value. However, the arbitrary number must be incremented by one after each netflow is processed.

It is important to note that the RW_NETFLOW object does not directly represent a singular netflow. As shown in Figure 10, the object is effectively a data structure that is used in the project to maintain and organize all netflows sharing the same SIDPT. The individual netflows are represented by the values placed in the attribute list (with a netflow sharing the same index between the different attribute lists).

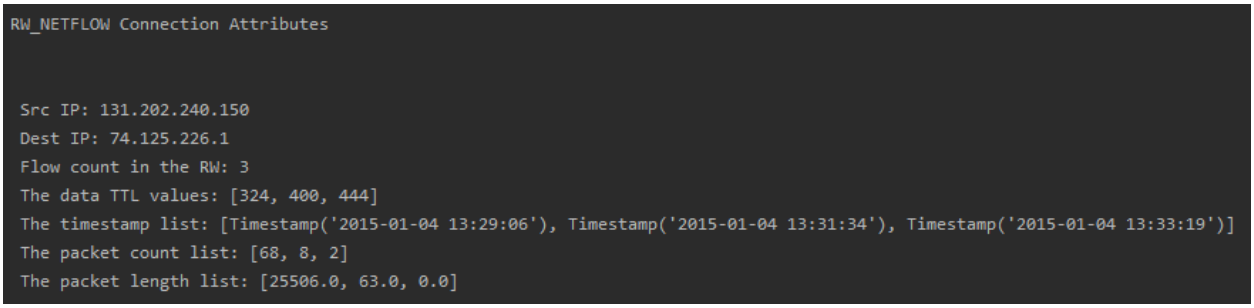


Figure 10: Example of the Connection Attributes of a RW_NETFLOW Object (RW Size of 500)

This self-contained organization of the pre-requisite data for the feature engineering enables easy generation of the engineered features for the target netflow. The first step of generating the engineered features is determining which of the values in the attribute list fall within the RW. This is determined by the TTL list.

For connection based RW, the number of connections is determined by taking the TTL value of the target netflow and comparing it to the values contained in the TTL list. If the difference exceeds the proscribed RW size, then the associated netflow characteristics for that index position are not considered to be in the RW. In order to ensure that the RW_NETFLOW object is not trailing huge attribute list, if a netflow is found to be outside the RW (the TTL value difference exceeds the RW size), all values associated with the netflow are removed (popped) from their respective list. Therefore, the attribute list act as queue where the lowest index (0) indicates the oldest netflow, and the highest index indicated the newest netflow for a given SDIPT. This limits the maximum number of attributes in any one list to be equal to the size of the connection based RW.

For time based RW, the concept is effectively the same. However, instead of using TTL values, the determination of whether a netflow is in the RW occurs through the timestamps. An elapsed time value is determined by calculating the difference between the target netflow's timestamp and the timestamps in the timestamp_list. If this difference exceeds the time based RW size, then

the value is not considered to be in the RW and all associated netflow characteristics are popped from their respective list. Since the number of netflows with a given SDIPT varies within a specified timeframe, there is no implicit cap to the number of netflow contained within the time-based attribute list of the RW_NETFLOW object.

To improve efficiency, the initial validation check is done on the youngest element of the list and if that check fails, all other values in the TTL list are also outside of the RW. Otherwise an interval-based checking method is used to find the first value in the TTL list that is within the RW.

Lastly, once the self-contained netflow values have been validated to be in the RW, the engineered features are easily calculable by applying the min, max, sum and mean functions to the list of values.
For the time delta features, the additional step of converting timestamps into time deltas is required prior to calculating the min, max and mean values. An example of the calculated engineered features for forward connections is shown below in Figure 11.

```
[11] [0.0, 144.0, 28.3] [0.0, 89769.0, 11076.90909090909, 121846.0] [2, 74, 16.363636363636363, 180]
```

Figure 11:Engineered Features (Connection Forward). [Format: flow count, time delta features (min, max, mean), packet length features (min, max, mean, total), packet count features (min, max, mean, total)]

With regards to the calculation of reverse features, the process is very similar. Since the traffic is still occurring between the same two entities, the source IP and destination IP addresses are swapped in the SDIPT. The modified SDIPT value is then checked against RW_NETFLOW object dictionary. If there is no match, the reverse features are all zeros. If there is a match, then the attribute lists for the reverse RW_NETFLOW object are validated to ensure that the values are within the RW and lastly the engineered features are calculated. It is important to note that no new values are added to the attribute list for the reverse RW_NETFLOW objects since the reverse object is not the target netflow, thus there is no network metrics to be added.

Ultimately, the four categories of engineered features are added to the CSV file containing the engineered feature data and the RW moves onto the next netflow in the list.

This implementation is overall quite efficient. However, efficiency does decrease with the number of RW_NETFLOW objects in the dictionary. As a result, larger RW sizes take longer to generate. The implementation is also compatible with a near real-time system as it does not require the netflow to be all in one CSV file, instead it can generate the engineered features for the individual netflow immediately upon receiving the basic network metrics (ip addresses, timestamp, etc…) and then pass it along to the classifier. The history of previous netflows that have been processed is maintained in RW_NETFLOW object.

Figure 12 present a summary of the rolling window approach to feature engineering.

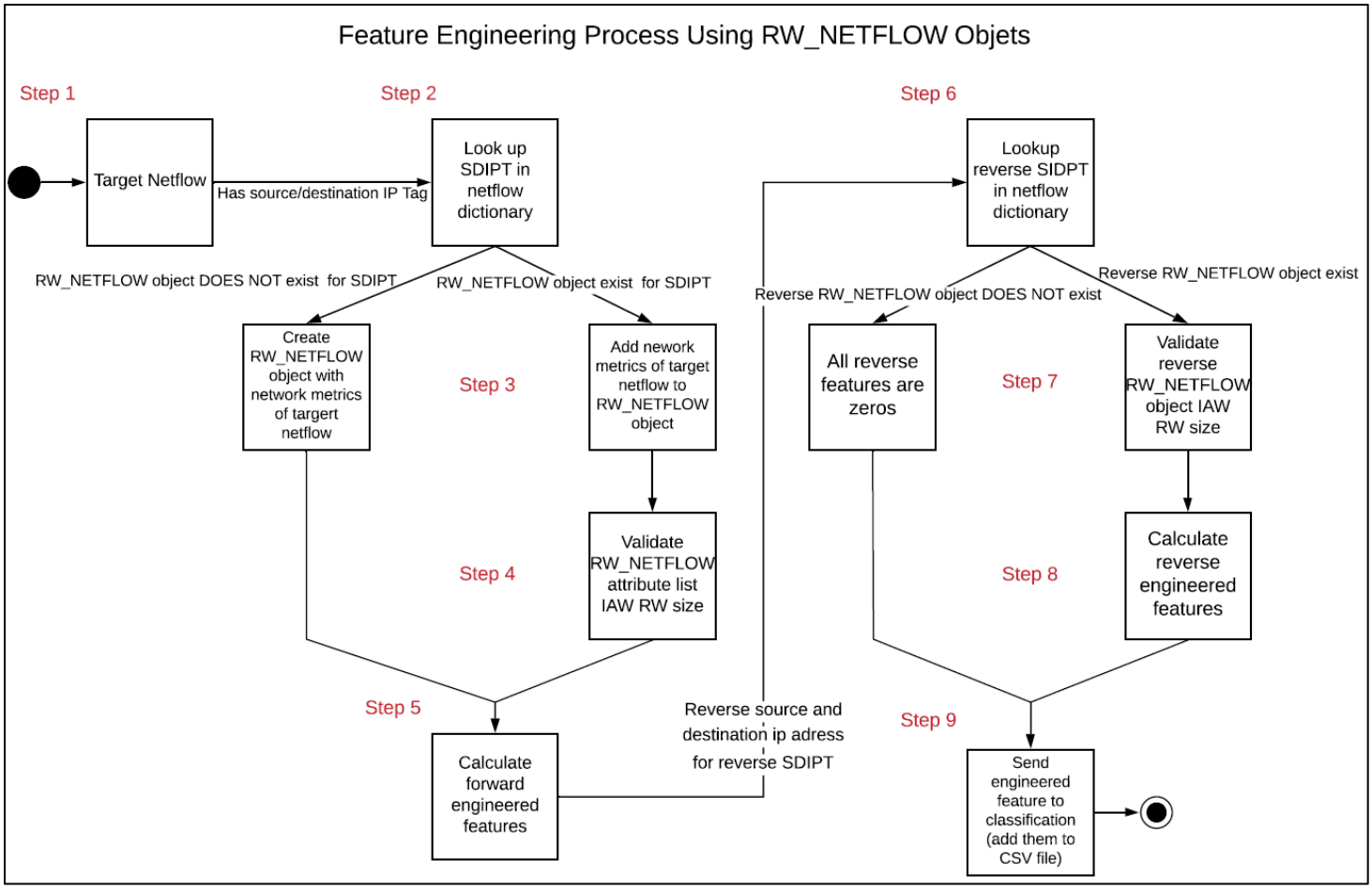


Figure 12: Summary of Feature Engineering Process via Rolling Windows and RW_NETFLOW Class

The dynamic programming implementation of the rolling window framework to feature engineering has resulted in the necessary data to implement the machine learning models.

4. MACHINE LEARNING MODELS

Given that this project aims to classify network traffic into VPN or non-VPN, a binary machine learning (ML) classifier model is required. The two overarching categories for classifiers are lazy learners and eager learners. Lazy learners use the entire training dataset at prediction time by comparing the target data and matching it with the most related data entry in the data set. On the other hand, eager learners construct a predictive model from the training data prior to predictions. At classification time, the predictive model is the only source for the target data classification. Lazy learners are fast learners and slow predictors, whereas eager learners are slow learners and fast predictors[5]. When considering the intent of implementing the classification

[5] (Asiri, 2018)

system in a near real-time system, prediction time should be minimized. Therefore, eager learning classification model will be used. Popular eager learning model include the Decision Tree, Logistic Regression, Naïve Bayes, artificial neural networks.

Prior to select specific ML models, it is pertinent to review the characteristics of the engineered features. The engineered features constitute a large dataset and have relatively high dimensionality (~50 engineered features). The data is not normalized, the data is not scaled, and the dataset is unbalanced (23 000 VPN netflow out of 215 000 netflows).

Given these characteristics (with particular importance to the fact that the data is not normalized) and the need for an eager learner, the Decision Tree is the favorable model for classification as other popular models require normalization. However, the Decision Tree model require special consideration for the algorithm used to determine the best choice at each node. As a result, decision trees are prone to overfitting and have a high degree error due to variance and bias[6].

Ultimately, to overcome the drawbacks of the Decision Tree model while preserving its advantages, this project combines multiple decision trees into an optimal model, which constitutes an ensemble model in machine learning. The project implements two ensemble models, each using a specific ensemble technique to include bagging (Bootstrap AGGregatING) through the Random Forest model and boosting through the Gradient Boost model.

The two models and their respective ensemble technique will be further discussed in the following sections of the report.

4.1 Random Forest

The Random Forest (RF) model is an implementation of the bagging ensemble technique, which combines bootstrapping and aggregation. It combines multiple weak models (the decision tree) in parallel to create the strong model, the random forest. All decisions trees are independent from each other and will each generate a prediction as to how to classify the target data. The overall prediction generated by the random forest is determined by majority voting of the constituent trees. Figure 13 shows an example of majority voting for a multiclass classification problem; however, the sample principle applies to binary class classification.

[6](Liberman, 2020)

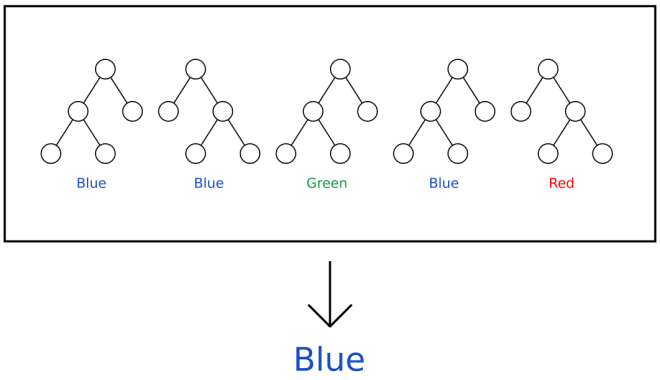


Figure 13: Example of Majority Voting from Multiple Independent Decision Trees in a Random Forest Model[7]

The bootstrap portion of the bagging technique is the variation of the seed data from which each tree is generated through bootstrap sampling. Bootstrap sampling is a "resampling method by independently sampling with replacement from an existing sample data with same sample size n[8]". Thus, the decision tree will be trained on a dataset, which is the same size as the training data, but will have duplicate entries.

Given that decision trees are very sensitive to the data they are trained on; a small change to the training set can result in a significantly different tree structure and corresponding prediction[9]. The net result is that the decision trees within the random forest will be loosely correlated.

The resulting advantage of the bagging technique is that by having multiple independent and relatively uncorrelated decision trees operating as a committee, the results will always outperform any one individual constituent. The increase in performance is guaranteed by the fact that the multiple decision trees protect each other from their individual errors, thus several individual trees can make an incorrect prediction, yet the overall prediction will still be correct[10]. Figure 14 shows the general structure of a RF model with the bootstrapping of training data and the aggregation of the individual tree to produce a final result.

[7] (Zhou, 2019)
[8] (Yen, 2019)
[9] (Yiu, 2019)
[10] *Ibid*

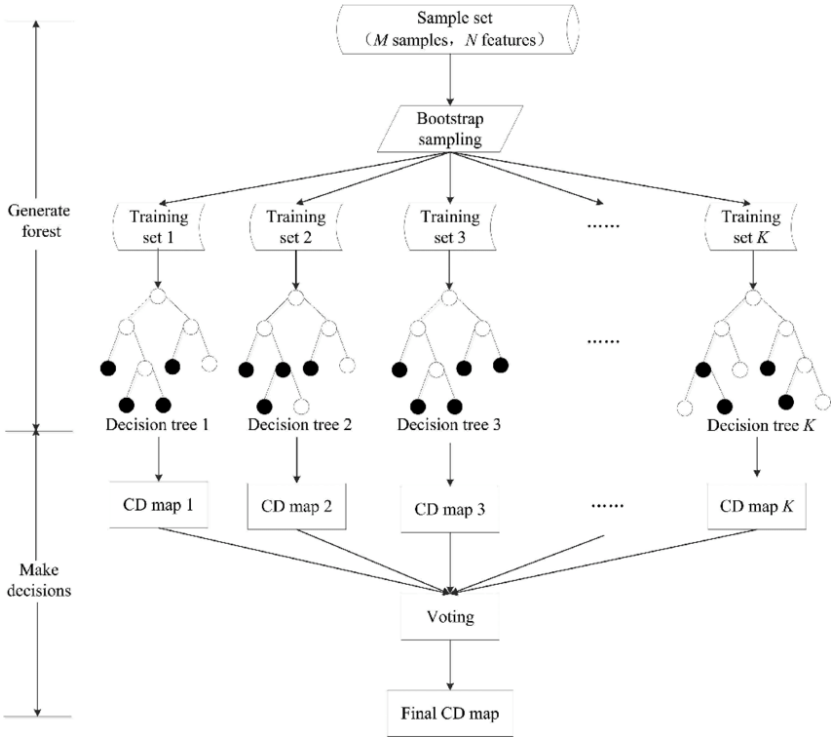


Figure 14: Random Forest Structure Highlighting Bootstrapping and Aggregation (Note Cd Map is Equivalent to a Prediction)[11]

The overall advantages of Random Forest classifiers are the following: prediction speed is faster than training speed (relevant for near real-time systems); it is robust to outlier and non-linear data (relevant for netflows); it is not impacted by unbalanced datasets (relevant to the project's dataset); it can use high dimensionality data; and it is resistant to overfitting.

4.2 Gradient Boost

The Gradient Boost (GB) model is an implementation of the boosting ensemble technique. It combines multiple weak models (the decision tree) sequentially to create a strong model. The process of training the GB model to a dataset is iterative (sequential), where in each subsequent iteration a new tree is added to the model. This new tree aims to correct any error or misclassification in the previous trees in the model. The original tree is based on an equal weighting of all observations, but subsequent trees are 'grown' based off a decreased weighting of correctly classified objects and increased weighting of incorrectly classified objects. Each subsequent iteration (new tree) aims to improve the predictions of the previous trees in the model. The overall prediction of the GB model is determined by taking a weighted average of the predicted value generated by each tree in the sequence. The weighting is with respect to the individual tree success at optimizing the prediction vs the other trees in the sequence. Figure 15 shows the overall structured sequence of gradient boosting.

[11](Feng et al., 2018)

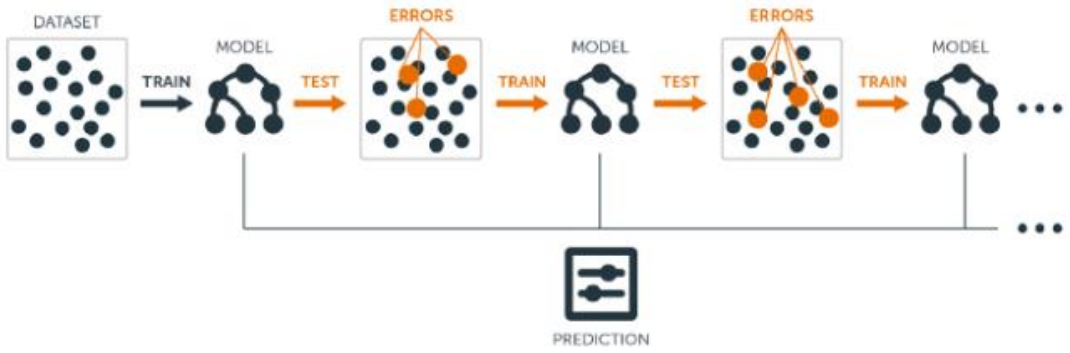


Figure 15: Gradient Boost Process Where Each Iteration of The Model is the Sequential Interaction of a New Tree Which Aims Correct the Errors in the Previous Iteration[12]

Although the GB model shares similar advantages to the Random Forest model, one notable issue with the GB model is that due to the sequential nature, it is quite vulnerable to overfitting. Therefore, it relies on hyperparameters to fine tune the model to the given dataset to maximize the predictive capability through the reduction of bias and variances. In particular, the learning rate hyperparameter is a critical hyperparameter since it controls the degree to which the weighting is being applied to each iteration. Generally, it is better to have a lower learning rate with a higher number of iterations (trees). Overall, when the Gradient Boost model is properly tuned, it is an incredibly strong predictor.

4.3 Model Implementation

Both model were implemented through the use of scikit-learn specifically sklearn.ensemble.RandomForestClassifier method for the random forest model and sklearn.ensemble.GradientBoostingClassifier method for the gradient boost model. The default test-train split used in generating the model was a 70-30 test-train split.

[12] (Ramzai, 2019)

Figures 16 (a, b, c) show the structure of a single decision tree generated by the Random Forest model at various max depth model. This represent only one of many decision trees which comprise the random forest.

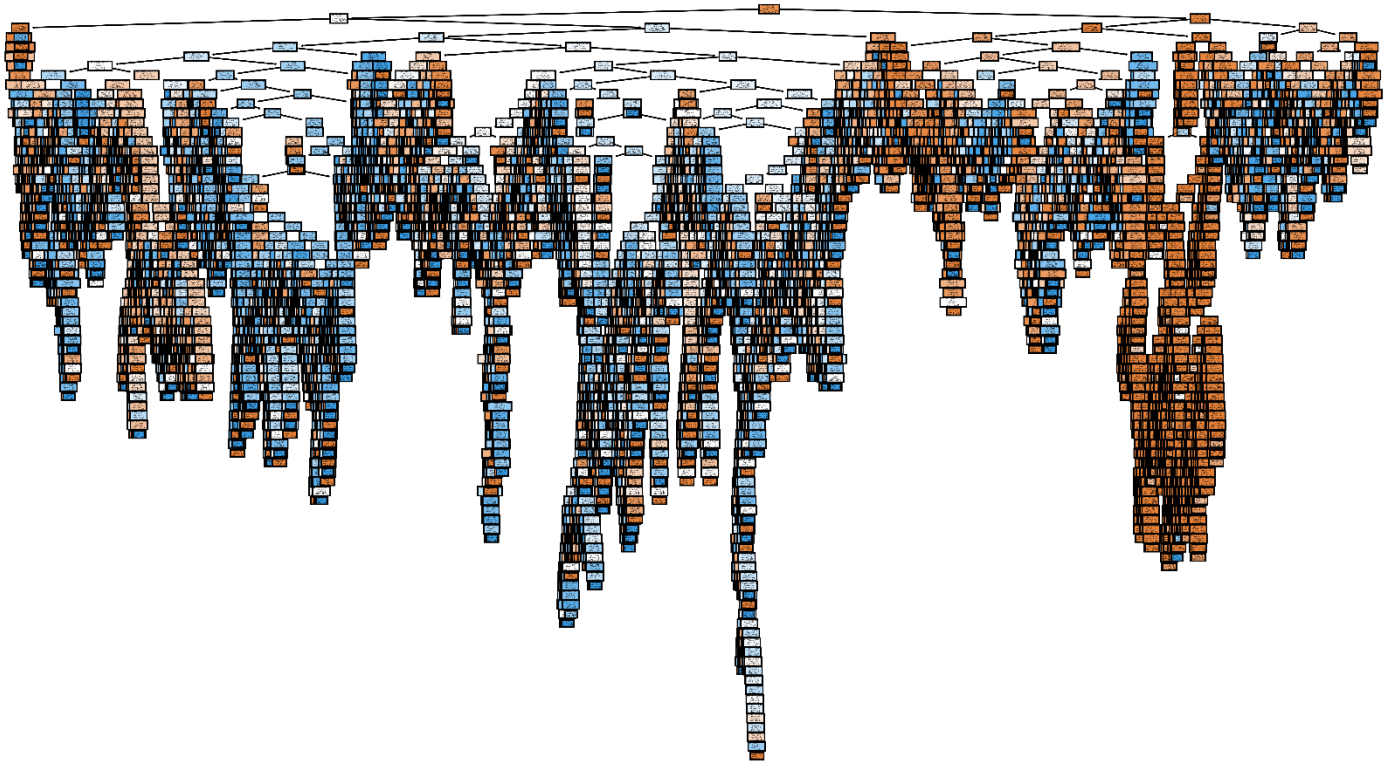


Figure 16 (a): Singular Decision Tree from the Implemented Random Forest (Max_Depth: None)

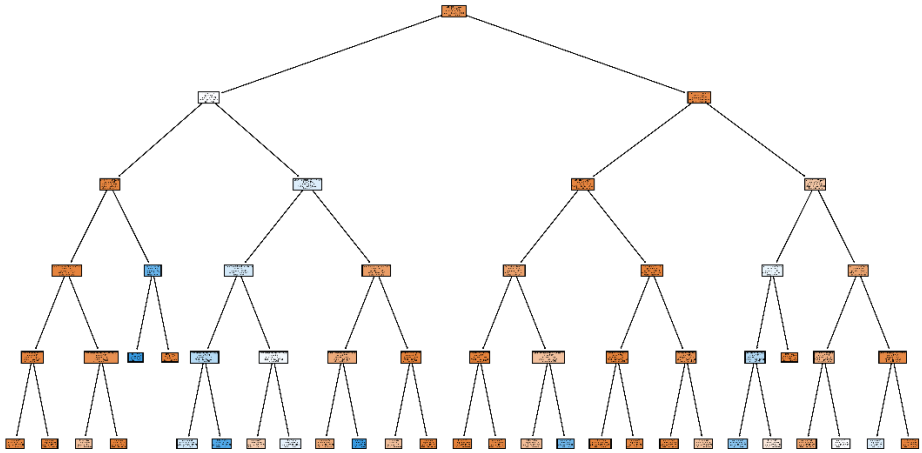


Figure 16 (b): Singular Decision Tree from the Implemented Random Forest (max_depth: 5 layers)

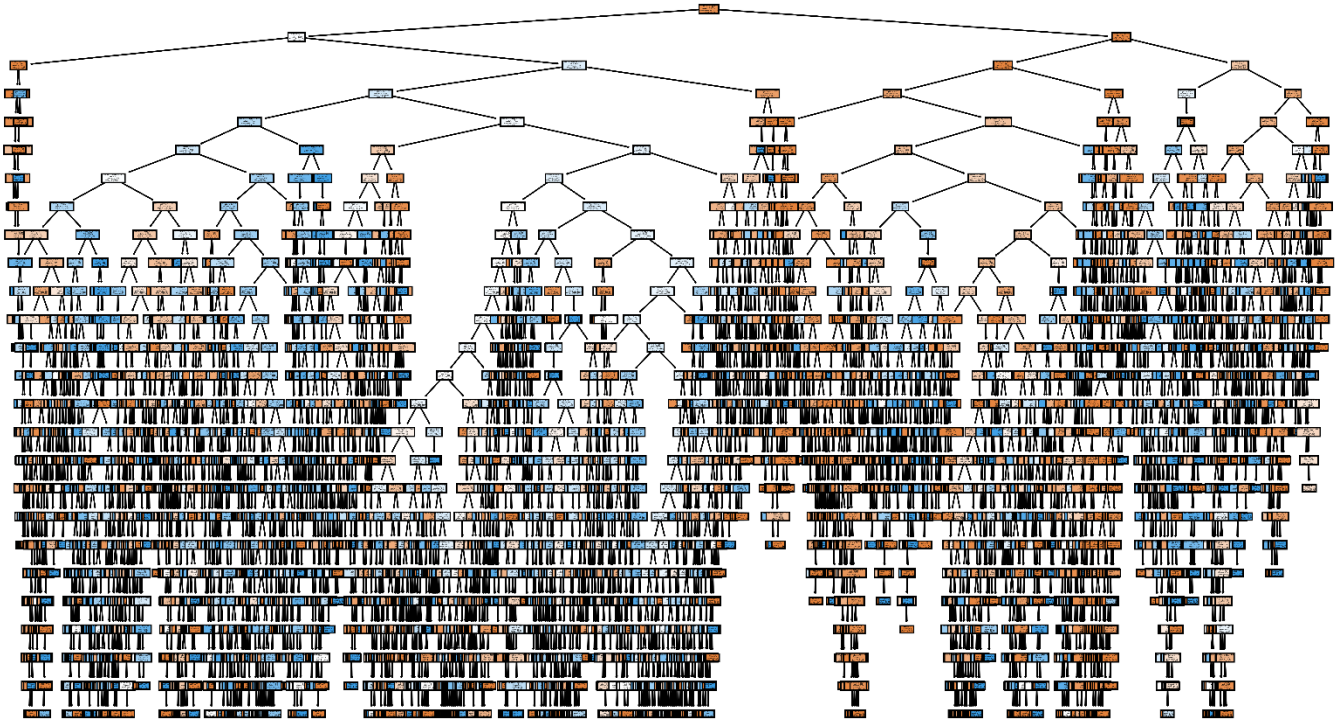


Figure 16 (c): Singular Decision Tree from the Implemented Random Forest (max_depth: 25 layers)

The decision trees generated by the Random Forest model are similar in structure to those generated by the Gradient Boost model. The main difference between the models is how these decision trees are assembled together to create the overall model. In the next section of the report, metrics for the evaluation of the two models will be discussed.

4.4 Evaluation Metrics

Given that this project aims to implement the ideal machine learning model for automated VPN detection, and that two separate machine learning models are being tested, evaluation metrics are necessary to determine which model is in fact the ideal model.

In an ideal VPN detection system, the minimization of the rate at which the system erroneously classifies traffic as VPN traffic (i.e. blocking a user who has a normal connection) should be prioritized over the degree to which the system catches all VPN traffic (i.e. VPN traffic not blocked when it should have been). In other words, although the system should aim to minimize all false negatives and false positives, more importance is placed on minimizing false negatives than false positives.

Given the established context, precision and recall with regards to the VPN class will be the key evaluation metrics for this project.

Additionally, although accuracy results will be presented, they are not a robust evaluation metric due to the unbalanced dataset which favors normal (non-VPN) traffic. Therefore, a high accuracy result could mask suboptimal precision and recall values for the VPN class. Also, since the project is analyzing precision and recall results separately the f-1 score will not be used as a metric. On the other hand, the receiver operating characteristics (ROC) - area under curve (AUC) metric will be considered. Lastly, only prediction time will be considered as training time is marginally relevant since the model is trained prior to deployment in the detection system.

These evaluation metrics will assist with the model optimization process discussed in the next section of this report.

4.5 Hyperparameter Optimization

As discussed in the gradient boost section of this report, the Gradient Boost model requires hyperparameter tuning to have peak predictive capability. This section of the report will discuss the hyperparameters that were optimized and the resulting hyperparameter values. The parameter optimization was performed through the sklearn.model_selection.GridSearchCV method from the sicikit-learn module. Additionally, the Random Forest model also underwent hyperparameter tuning. However, the improvements to the RF predictions engendered by the tuning were minimal.

For both models, the hyperparameters were tuned by running GridSearchCV multiple time, where at each iteration a different hyperparameter was tested. GridSearchCV was configured to do five-fold cross validation and scored based on accuracy, using random-state 42 and a test train split of 70-30. The engineered feature subset used for the hyperparameter optimization was the connection feature subset.

Upon GridSearchCV returning the ideal hyperparameter value, the model being evaluated was changed to reflect the newly determined ideal value. Thus, each subsequent iteration of GridSearchCV determined the ideal hyperparameter from a partially optimized model.

The following tables (Table 2 is Gradient Boost model and Table 3 is Random Forest model) show the tuned hyperparameters, their default value, and the order they were tuned in.

Table 2: Gradient Boost Optimized Hyperparameters

Hyperparameter	Optimized Value	Default Value	Order Tuned
learning_rate	0.6	0.1	1
n_estimators	500	100	2
max_depth	12	3	3
min_samples_split	2	2	4
min_samples_leaf	1	1	5
subsample	1.0	1.0	6
max_features	None	None	7

Table 3: Random Forest Optimized Hyperparameters

Hyperparameter	Optimized Value	Default Value	Order Tuned
n_estimators	300	100	1
max_depth	25	None	2
max_features	None	sqrt	3
min_samples_leaf	2	1	4
min_samples_split	2	2	5

The next section of the report will discuss the technique of using subsets of the engineered features to further improve the model performance.

4.6 Feature Subsets

In any machine learning model, certain features will have a greater impact on the classification of the target object. Figures 17 show the feature importance in decrementing order of the Random Forest model with using different feature sets.

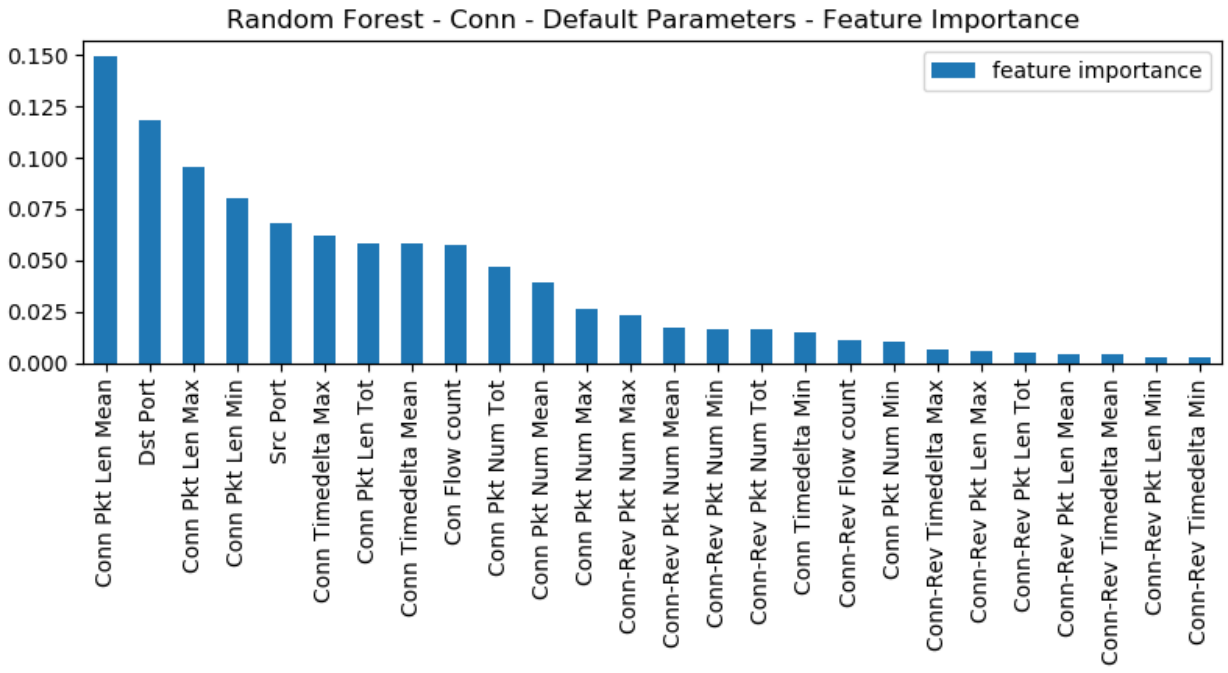


Figure 17 (a): Feature Importance for Random Forest Model (Connection Feature Subset).

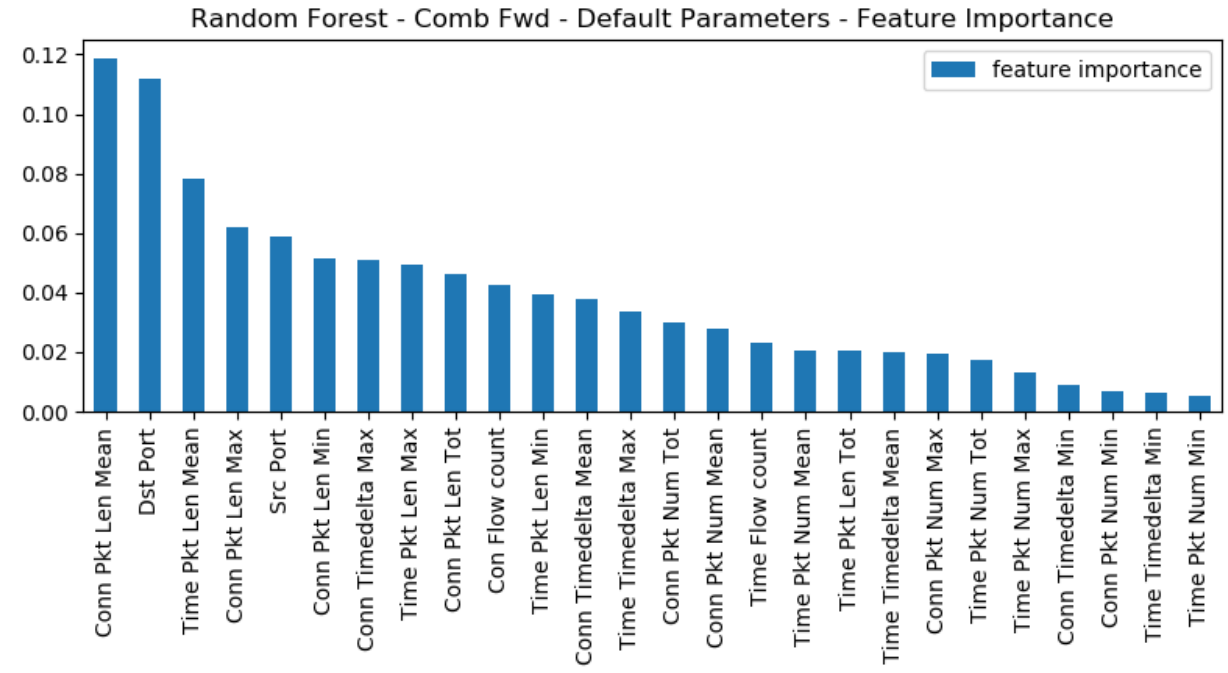


Figure 17 (b): Feature Importance for Random Forest Model (Connection Forward Feature Subset).

These two figures highlight that of the four feature engineering categories in Table 1 (discussed in rolling window section of feature engineering), certain feature categories play an overall

greater role in the classification. Specifically in Figure 17(a) forward features play a significantly larger role than reverse features (the most important reverse feature is the 12th most important), and in Figure 17(b) connection features play a more important role than time features (a connection feature is the most important feature and of the top 10 features, time features are only three).

To further exploit the varying impact of the engineered feature categories, the engineered features were divided into nine feature subset groups. Table 4 shows these nine subsets, which are generated by first dividing the engineered features into either time features, connection features, or combination of time and connection features (simply referred to as "combined"), then each initial group can be further divided by only considering forward features, reverse features or the bidirectional features.

Table 4: Engineered Feature Subsets

RW Type / Feature Direction	Combined (Time and Conn)	Time	Connection
Combined (Fwd and Bwd)	Combined All features	Time ½ of all Engineered Ft	Connection ½ of all Engineered Ft
Forward	Combined – Fwd ½ of all Engineered Ft	Time – Fwd ¼ of all Engineered Ft	Connection – Fwd ¼ of all Engineered Ft
Backwards	Combined – Bwd ½ of all Engineered Ft	Time – Bwd ¼ of all Engineered Ft	Connection – Bwd ¼ of all Engineered Ft

The potential advantage associated with employing features subsets for training and testing the models is that it allows for greater efficiency in both the feature engineering and the employment of the model, which is important in a real time system. A smaller subset reduces the number of attributes that need to be maintained, validated and calculated in the feature engineering phase and reduces the dimensionally in the training/prediction phase. Thus, given equivalent metrics between two feature subsets, the smaller subset will be considered the better performing feature subset.

The final factor that has the potential of impacting the models predictive capabilities is discussed in the following section.

4.7 Rolling Windows Sizes

For any machine learning model, a change in the training data has the possibility to impact the subsequent models' predictive capabilities. One such method of changing the training data in this project is to change the size of the rolling window. Since the size of the rolling window directly impacts the values of the engineered features and since those engineered features constitute the training data, there is the possibility that a change in rolling window size could modify the overall results of the model. Therefore, this project examined multiple different rolling window sizes to ascertain the relative importance rolling window size in the model's overall performance. If two models with different rolling window sizes have the same results, the one with the

smallest rolling window will be favored as a small rolling windows leads to a more efficient feature engineering process.

Overall, the two models along with the variations of the training data and model optimization generated results. The next section of this report showcased these results.

5. RESULTS

The following section of the report will present the various evaluations metrics generated by the differencing iterations of the machine learning models and the various feature subsets. Although short descriptions will accompany the figures and table a more substantial discussion of the results can be found in the analysis section. Precision and recall metrics will be shown for both the non-VPN class and VPN class. Unless otherwise indicated, all results are generated with random state 42, a 70-30 test-train split, and a rolling window size of 10 000 connections / 10 minutes.

5.1 Baseline

Baseline results are generated through non-engineered features (port numbers, total number of packets, total number of bytes). These are the results with no feature engineering and no hyperparameter optimization. Figure 18 is the confusion matrix for the random forest baseline. Figure 19 is the confusion matrix for the gradient boost baseline. Figure 20 presents a graphical comparison of the evaluation metrics for the baseline of the two models. Finally, Table 5 showcases a tabular representation of the baseline results.

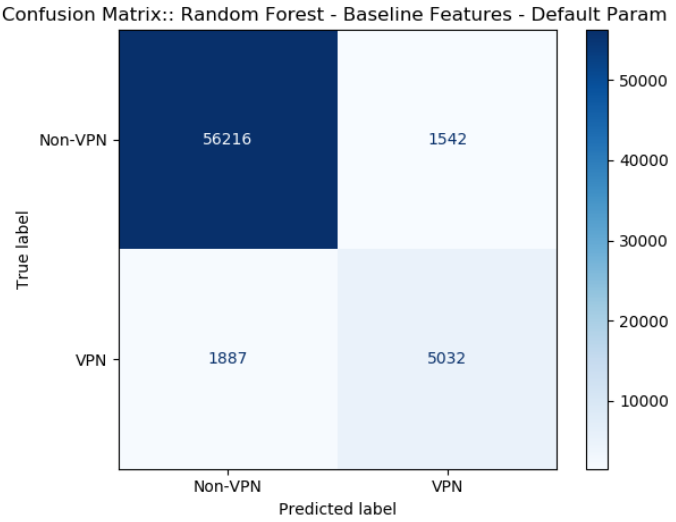


Figure 18: Confusion Matrix for Random Forest Baseline Features with Default Parameters

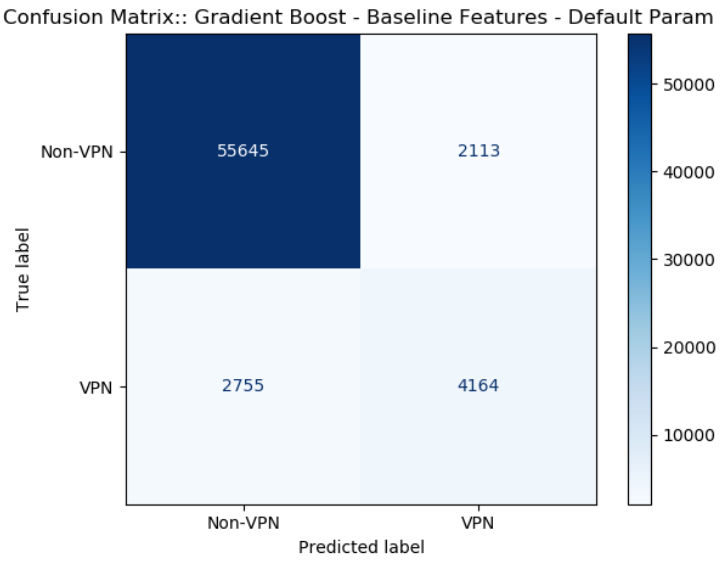


Figure 19: Confusion Matrix for Gradient Boost Baseline Features with Default Parameters

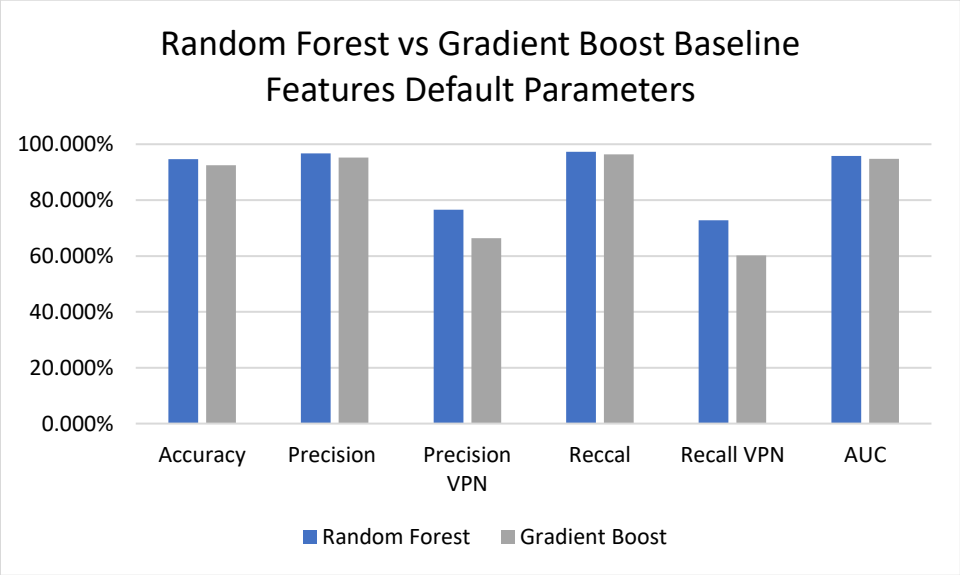


Figure 170: Graphical Comparisons of Random Forest and Gradient Boost Evaluation Metrics for Baseline Features with Default Parameters

Table 5:Evaluation Metrics of Baseline Models with Default Parameters

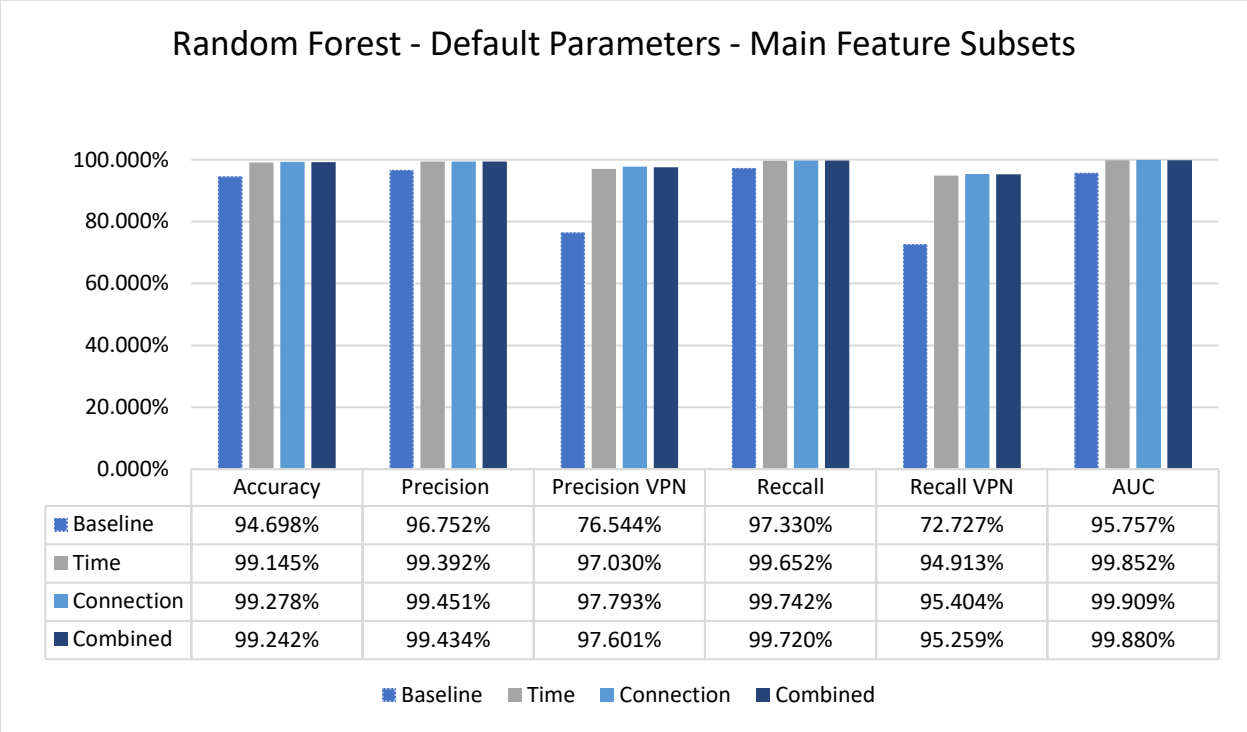
Feature Subset	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Random Forest	94.698%	96.752%	76.544%	97.330%	72.727%	95.757%
Gradient Boost	92.473%	95.283%	66.337%	96.342%	60.182%	94.774%

The key takeaways for the baseline results are two-fold. Firstly, the random forest has better results than the gradient boost model across all metrics. Secondly, although the accuracy values are high, the recall and precision metrics for the VPN class are significantly lower.

5.2 Default Parameters

Default parameter results are the results generated by the nine different feature subsets using the stock hyperparameters used in the respective sklearn implementation of the model. The first set of figures and tables presented below relate to the Random Forest model. Figure 21 is a graphical representation of the results for main feature subsets (time, connection and combined) for the RF model. Figure 22 is a vertical chart representation of the confusion matrix for these 3 feature subsets. Figure 23 is a graphical representation of the three connection-based feature subsets. The connection features were selected since they are the best performing of the main three subsets. Table 6 is a tabular representation of the results for all 9 features subsets for the RF model with default parameters. Figures 24 to 26 and Table 7 are the equivalent figures and table

for the Gradient Boost model. Finally, Figure 27 presents a graphical comparison between the best performing RF feature subset and the best performing GB feature subset.



	Accuracy	Precision	Precision VPN	Reccall	Recall VPN	AUC
Baseline	94.698%	96.752%	76.544%	97.330%	72.727%	95.757%
Time	99.145%	99.392%	97.030%	99.652%	94.913%	99.852%
Connection	99.278%	99.451%	97.793%	99.742%	95.404%	99.909%
Combined	99.242%	99.434%	97.601%	99.720%	95.259%	99.880%

Figure 21: Results – Random Forest Default Parameters – Main Three Feature Subsets

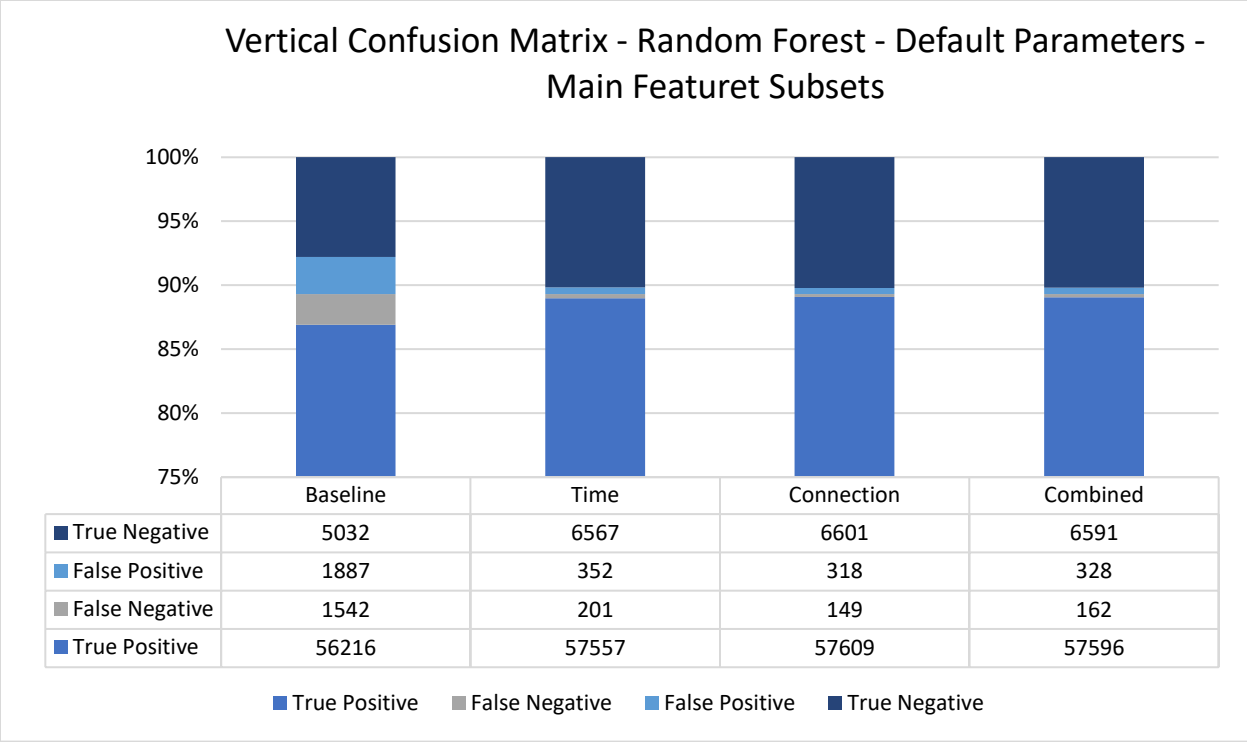
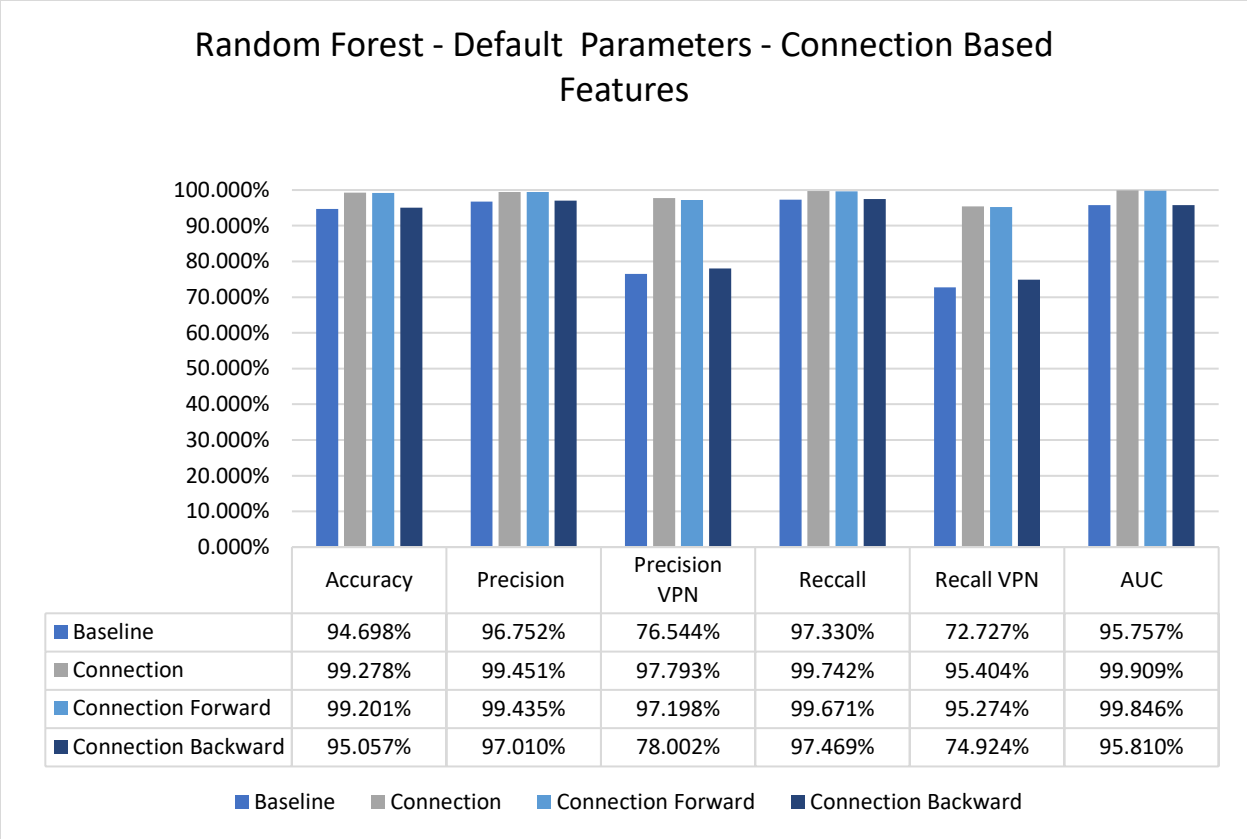


Figure 182: Confusion Matrix (Normalized Representation) – Random Forest Default Parameters – Main Three Feature Subsets

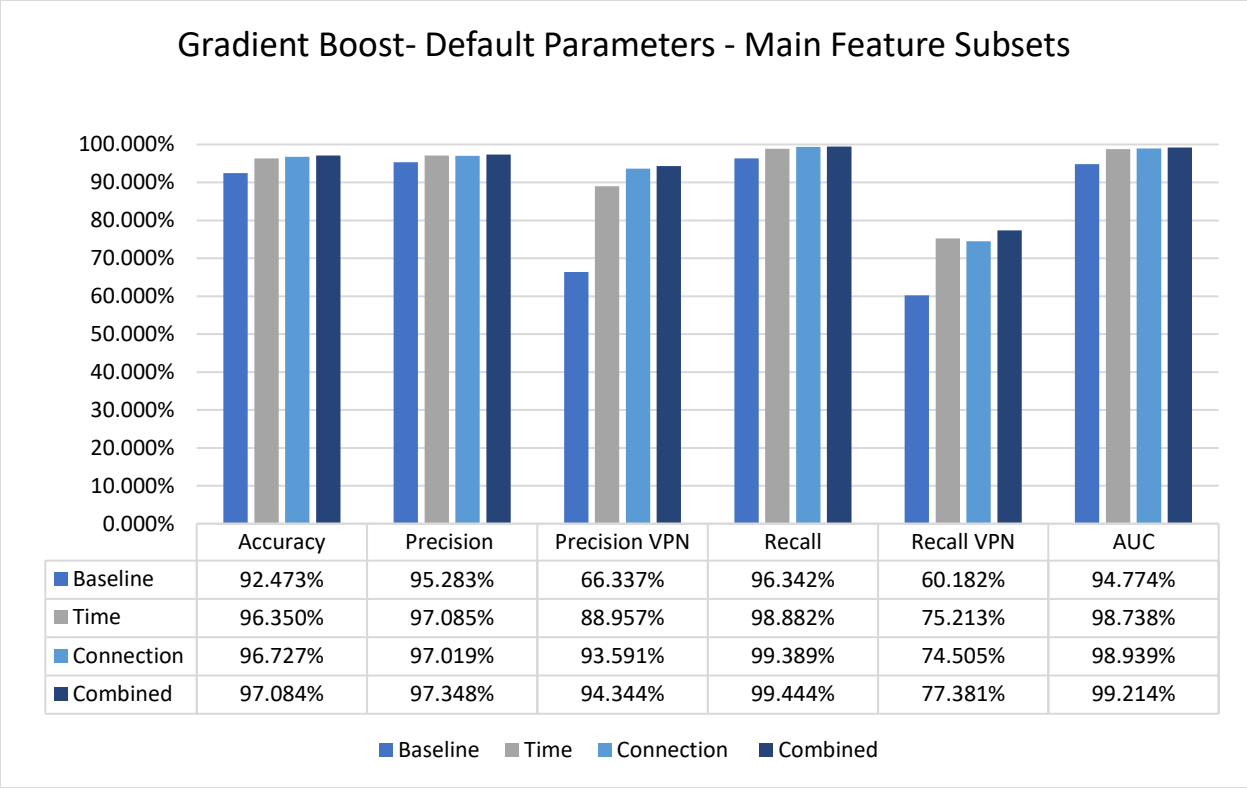


	Accuracy	Precision	Precision VPN	Reccall	Recall VPN	AUC
Baseline	94.698%	96.752%	76.544%	97.330%	72.727%	95.757%
Connection	99.278%	99.451%	97.793%	99.742%	95.404%	99.909%
Connection Forward	99.201%	99.435%	97.198%	99.671%	95.274%	99.846%
Connection Backward	95.057%	97.010%	78.002%	97.469%	74.924%	95.810%

Figure 23: Results – Random Forest Default Parameters – Connection-Based Feature Subsets

Table 6: Results – Random Forest Default Parameters – All Features Subsets

Feature Subset	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline	94.698%	96.752%	76.544%	97.330 %	72.727%	95.757 %
Connection	99.278%	99.451%	97.793%	99.742 %	95.404%	99.909 %
Connection Forward	99.201%	99.435%	97.198%	99.671 %	95.274%	99.846 %
Connection Backward	95.057%	97.010%	78.002%	97.469 %	74.924%	95.810 %
Time	99.145%	99.392%	97.030%	99.652 %	94.913%	99.852 %
Time Forward	99.041%	99.349%	96.419%	99.579 %	94.551%	99.780 %
Time Backward	95.007%	96.975%	77.803%	97.450 %	74.621%	95.740 %
Combined	99.242%	99.434%	97.601%	99.720 %	95.259%	99.880 %
Combined Forward	99.156%	99.414%	96.950%	99.642 %	95.100%	99.848 %
Combined Backwards	95.042%	97.003%	77.918%	97.458 %	74.866%	95.753 %



	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline	92.473%	95.283%	66.337%	96.342%	60.182%	94.774%
Time	96.350%	97.085%	88.957%	98.882%	75.213%	98.738%
Connection	96.727%	97.019%	93.591%	99.389%	74.505%	98.939%
Combined	97.084%	97.348%	94.344%	99.444%	77.381%	99.214%

Figure 194: Results – Gradient Boost Default Parameters – Main Three Feature Subsets

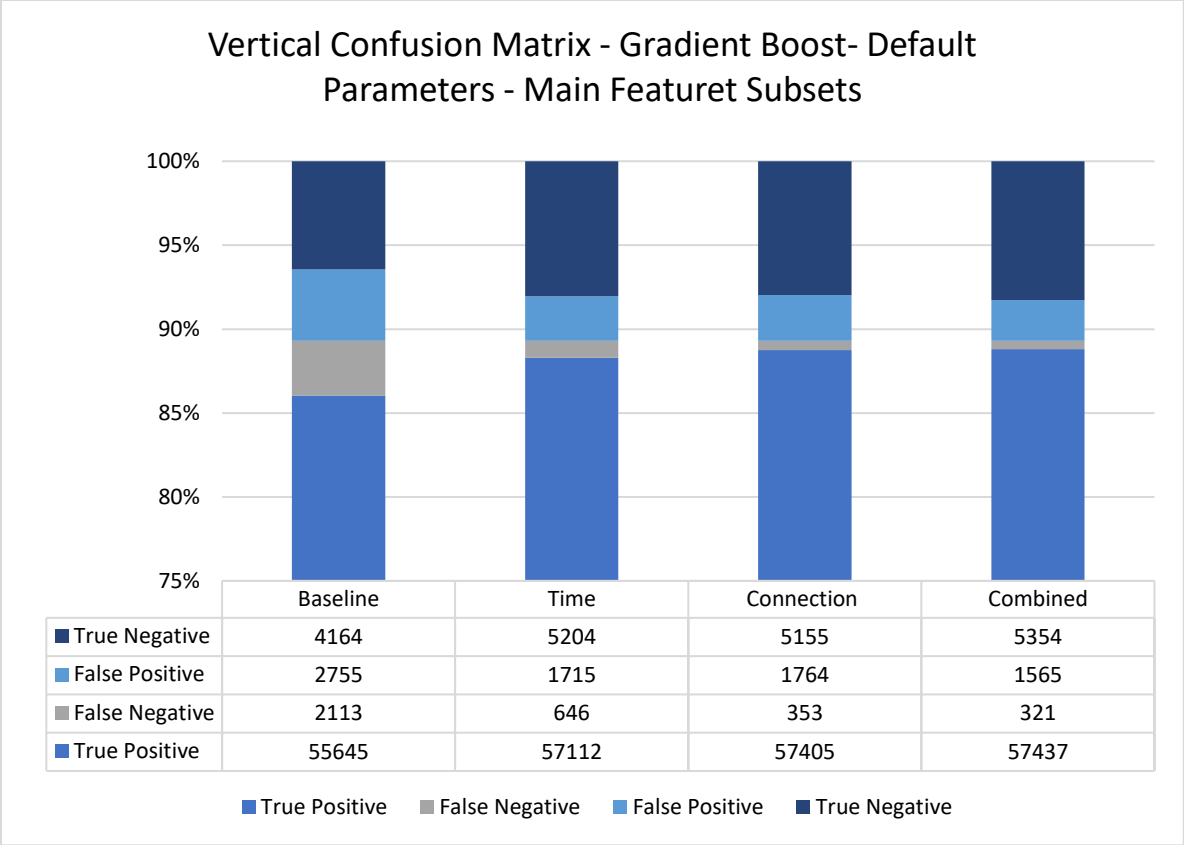
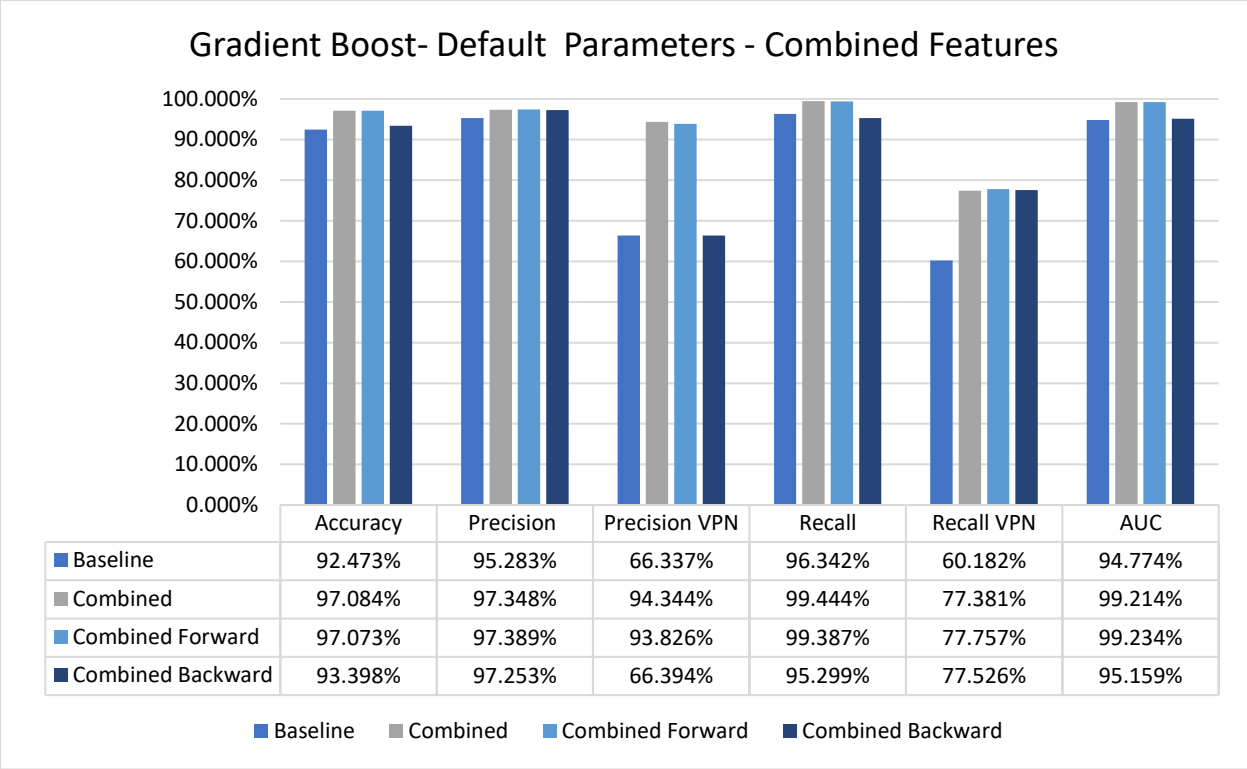


Figure 25: Confusion Matrix (Normalized Representation) – Gradient Boost Default Parameters – Main Three Feature Subsets

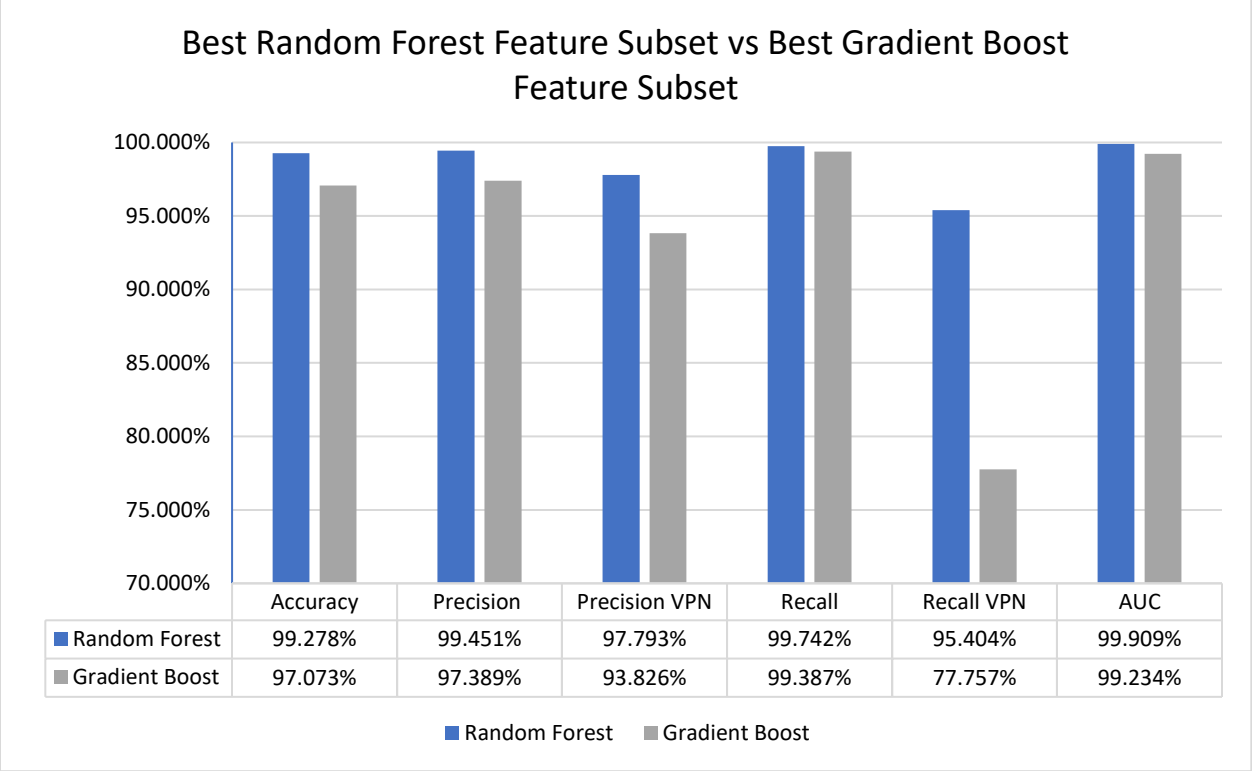


	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline	92.473%	95.283%	66.337%	96.342%	60.182%	94.774%
Combined	97.084%	97.348%	94.344%	99.444%	77.381%	99.214%
Combined Forward	97.073%	97.389%	93.826%	99.387%	77.757%	99.234%
Combined Backward	93.398%	97.253%	66.394%	95.299%	77.526%	95.159%

Figure 26: Results – Gradient Boost Default Parameters – Combined Feature Subsets

Table 7: Results – Gradient Boost Default Parameters – All Features Subsets

Feature Subset	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline	92.473%	95.283%	66.337%	96.342 %	60.182%	94.774 %
Connection	96.727%	97.019%	93.591%	99.389 %	74.505%	98.939 %
Connection Forward	96.841%	97.129%	93.786%	99.401 %	75.473%	99.046 %
Connection Backward	93.410%	97.265%	66.432%	95.301 %	77.627%	95.127 %
Time	96.350%	97.085%	88.957%	98.882 %	75.213%	98.738 %
Time forward	96.275%	96.934%	89.478%	98.959 %	73.869%	98.663 %
Time backwards	93.095%	96.920%	65.551%	95.296 %	74.722%	94.995 %
Combined	97.084%	97.348%	94.344%	99.444 %	77.381%	99.214 %
Combined Forward	97.073%	97.389%	93.826%	99.387 %	77.757%	99.234 %
Combined Backward	93.398%	97.253%	66.394%	95.299 %	77.526%	95.159 %



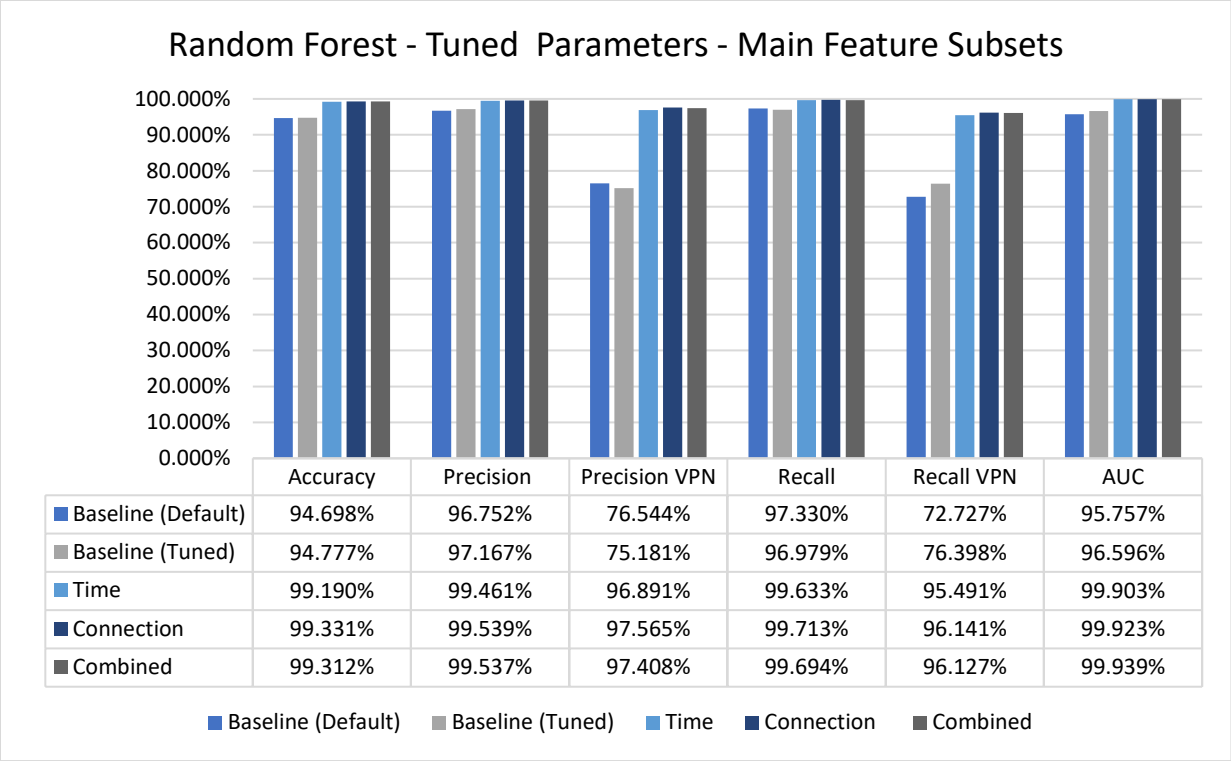
	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Random Forest	99.278%	99.451%	97.793%	99.742%	95.404%	99.909%
Gradient Boost	97.073%	97.389%	93.826%	99.387%	77.757%	99.234%

Figure 27: Comparison – Top Performing Gradient Boost Feature Subset Vs Top Performing Random Forest Feature Subset (N.B.: Vertical Axis Starts At 70%)

The key observations for the default parameter results are that engineered features improve all metrics, especially the key metrics where there is a ~20% improvement over the baseline results. Secondly, the random forest still has overall better results, especially for the VPN recall metric. The best performing Random Forest feature subset was the connection subset and combined forward feature subset for the Gradient Boost.

5.3 Optimized Hyperparameters

Optimized hyperparameter results are the results generated by the improved models where the parameters have been tuned in accordance with the values discussed in the hyperparameter optimization section. The figures and tables will follow the same structure as the default parameter results presented above. Figures 28 to 30 and Table 8 pertain to the Random Forest model, Figures 31 to 33 and Table 9 relate to the Gradient Boost model, and Figure 34 presents the comparison of the top performing feature subset of each model.



	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline (Default)	94.698%	96.752%	76.544%	97.330%	72.727%	95.757%
Baseline (Tuned)	94.777%	97.167%	75.181%	96.979%	76.398%	96.596%
Time	99.190%	99.461%	96.891%	99.633%	95.491%	99.903%
Connection	99.331%	99.539%	97.565%	99.713%	96.141%	99.923%
Combined	99.312%	99.537%	97.408%	99.694%	96.127%	99.939%

Figure 28: Results – Random Forest Tuned Parameters – Main Three Feature Subsets

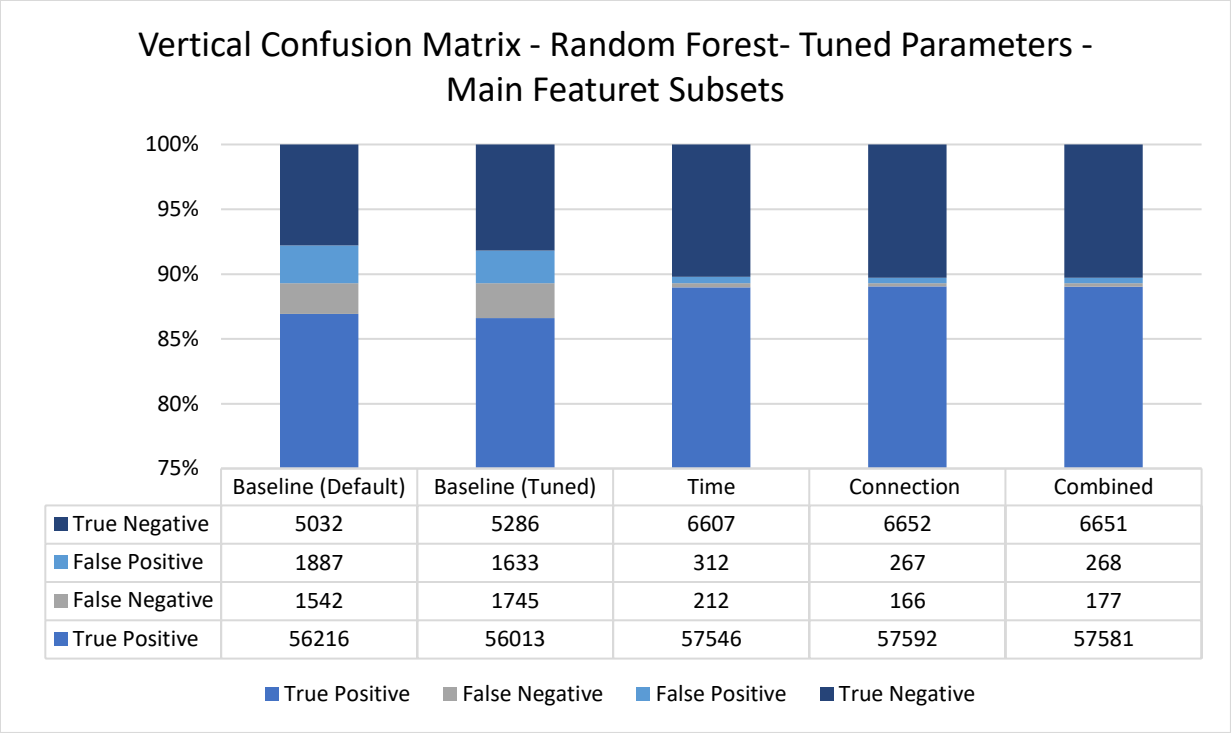


Figure 29: Confusion Matrix (Stacked Column Representation) – Random Forest Tuned Parameters – Main Three Feature Subsets

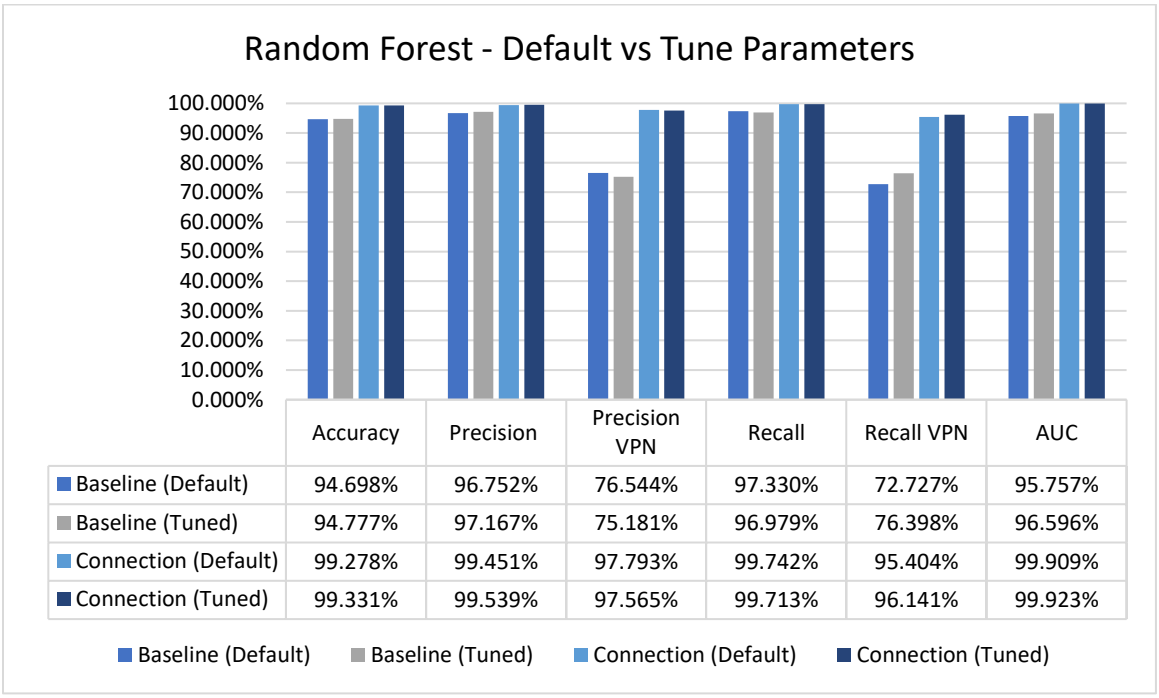
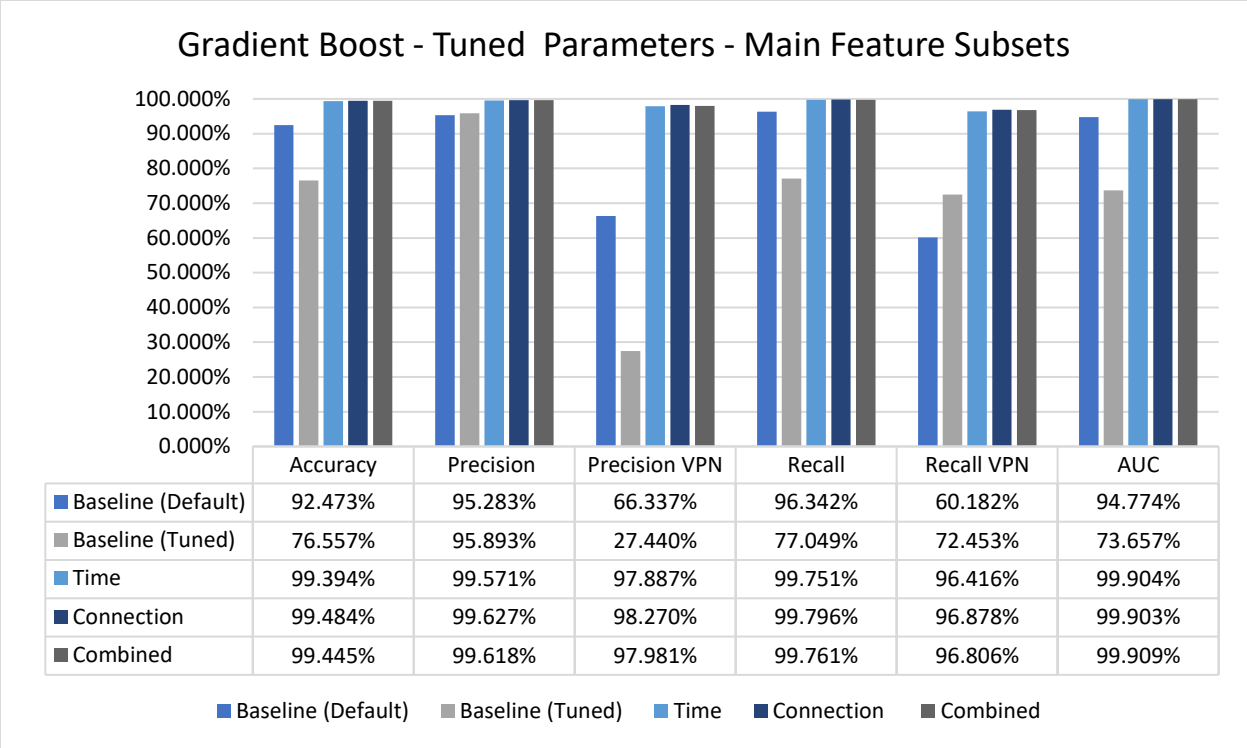


Figure 30: Results – Random Forest – Default Vs Tuned Parameters – Connection Feature Subset

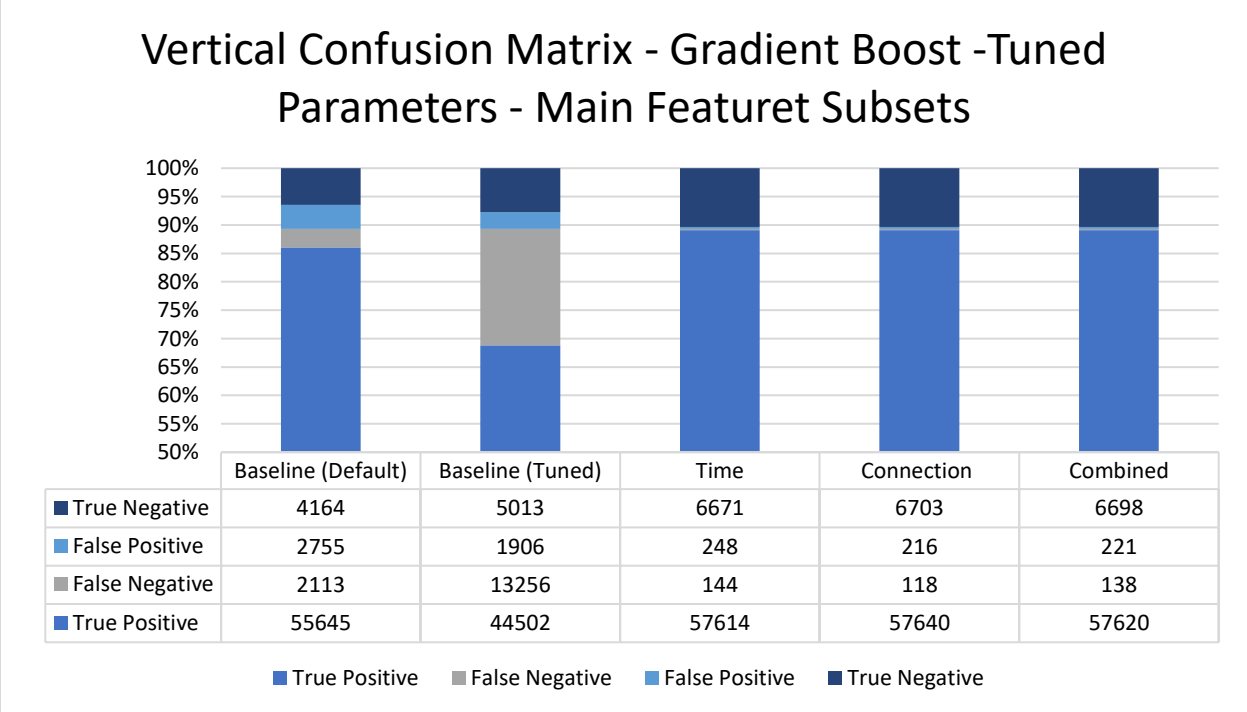
Table 8: Results – Random Forest Tuned Parameters – All Features Subsets

Feature Subset	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline (Default)	94.698%	96.752%	76.544%	97.330 %	72.727%	95.757 %
Baseline (Tuned)	94.777%	97.167%	75.181%	96.979 %	76.398%	96.596 %
Connection	99.331%	99.539%	97.565%	99.713 %	96.141%	99.923 %
Connection Forward	99.303%	99.528%	97.392%	99.692 %	96.054%	99.879 %
Connection Backward	95.290%	98.113%	74.767%	96.584 %	84.492%	96.670 %
Time	99.190%	99.461%	96.891%	99.633 %	95.491%	99.903 %
Time forward	99.123%	99.428%	96.542%	99.591 %	95.216%	99.834 %
Time backwards	95.258%	98.105%	74.591%	96.555 %	84.434%	96.636 %
Combined	99.312%	99.537%	97.408%	99.694 %	96.127%	99.939 %
Combined Forward	99.270%	99.518%	97.175%	99.666 %	95.968%	99.900 %
Combined Backward	95.289%	98.106%	74.782%	96.589 %	84.434%	96.661 %



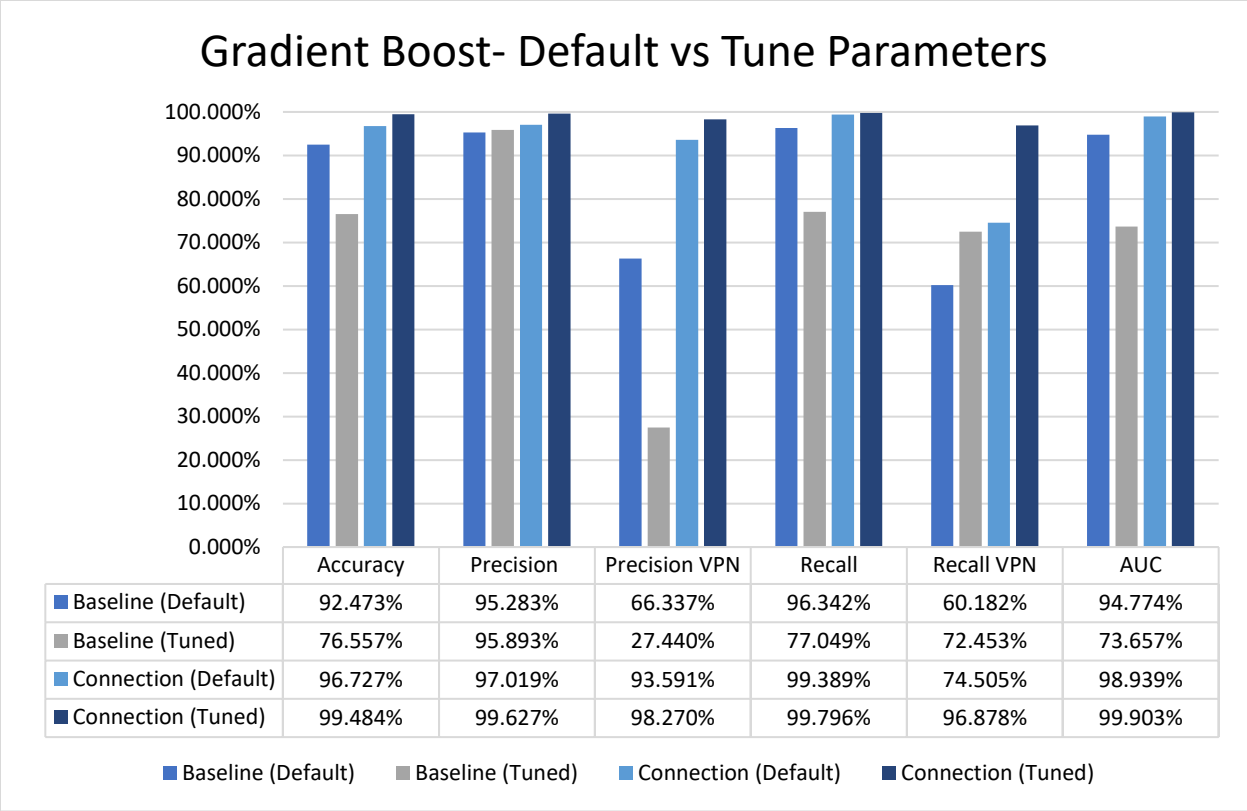
	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline (Default)	92.473%	95.283%	66.337%	96.342%	60.182%	94.774%
Baseline (Tuned)	76.557%	95.893%	27.440%	77.049%	72.453%	73.657%
Time	99.394%	99.571%	97.887%	99.751%	96.416%	99.904%
Connection	99.484%	99.627%	98.270%	99.796%	96.878%	99.903%
Combined	99.445%	99.618%	97.981%	99.761%	96.806%	99.909%

Figure 31: Results – Gradient Boost Tuned Parameters – Main Three Feature Subsets



	Baseline (Default)	Baseline (Tuned)	Time	Connection	Combined
True Negative	4164	5013	6671	6703	6698
False Positive	2755	1906	248	216	221
False Negative	2113	13256	144	118	138
True Positive	55645	44502	57614	57640	57620

Figure 32: Confusion Matrix (Stacked Column Representation) – Gradient Boost Tuned Parameters – Main Three Feature Subsets (N.B. Vertical Axis Starts At 50%)



	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline (Default)	92.473%	95.283%	66.337%	96.342%	60.182%	94.774%
Baseline (Tuned)	76.557%	95.893%	27.440%	77.049%	72.453%	73.657%
Connection (Default)	96.727%	97.019%	93.591%	99.389%	74.505%	98.939%
Connection (Tuned)	99.484%	99.627%	98.270%	99.796%	96.878%	99.903%

Figure 33: Results – Gradient Boost – Default Vs Tuned Parameters – Connection Feature Subset

Table 9: Results – Gradient Boost Tuned Parameters – All Features Subsets

Feature Subset	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Baseline (Default)	92.473%	95.283%	66.337%	96.342 %	60.182%	94.774 %
Baseline (Tuned)	76.557%	95.893%	27.440%	77.049 %	72.453%	73.657 %
Connection	99.484%	99.627%	98.270%	99.796 %	96.878%	99.903 %
Connection Forward	99.439%	99.609%	97.994%	99.763 %	96.734%	99.875 %
Connection Backward	83.595%	96.611%	36.912%	84.598 %	75.228%	82.610 %
Time	99.394%	99.571%	97.887%	99.751 %	96.416%	99.904 %
Time forward	99.340%	99.557%	97.498%	99.704 %	96.300%	99.799 %
Time backwards	69.989%	95.823%	22.647%	69.421 %	74.736%	67.762 %
Combined	99.445%	99.618%	97.981%	99.761 %	96.806%	99.909 %
Combined Forward	99.446%	99.621%	97.968%	99.759 %	96.835%	99.907 %
Combined Backward	75.538%	96.153%	26.871%	75.635 %	74.736%	73.381 %

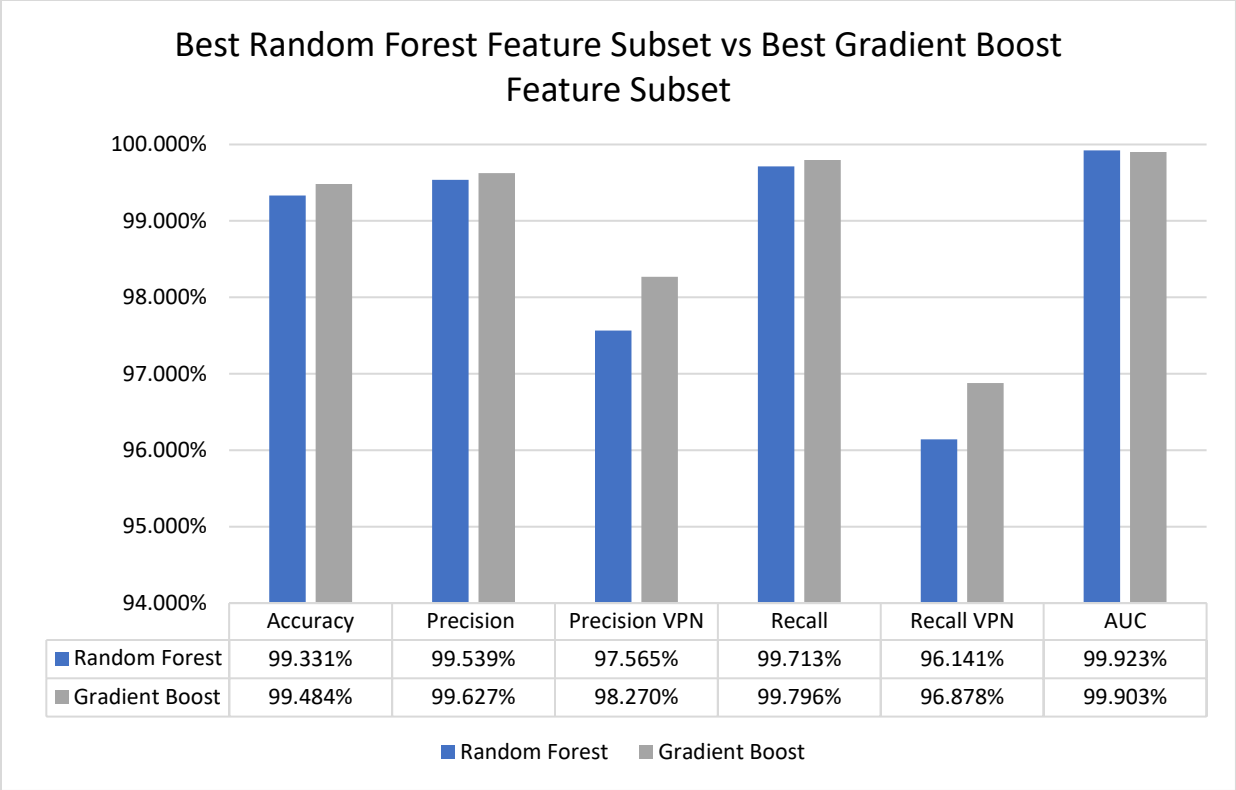


Figure 34: Comparison – Top Performing Gradient Boost Feature Subset Vs Top Performing Random Forest Feature Subset (N.B.: Vertical Axis Starts At 94%)

The main takeaway from these results is that parameter tuning enabled the Gradient Boost model to increase its predictive capability to surpass the Random Forest model. Parameter tuning only had a minor impact on the RF model. Lastly, the best performing feature subset was the connection feature subset for both models.

5.4 Rolling Windows Sizes

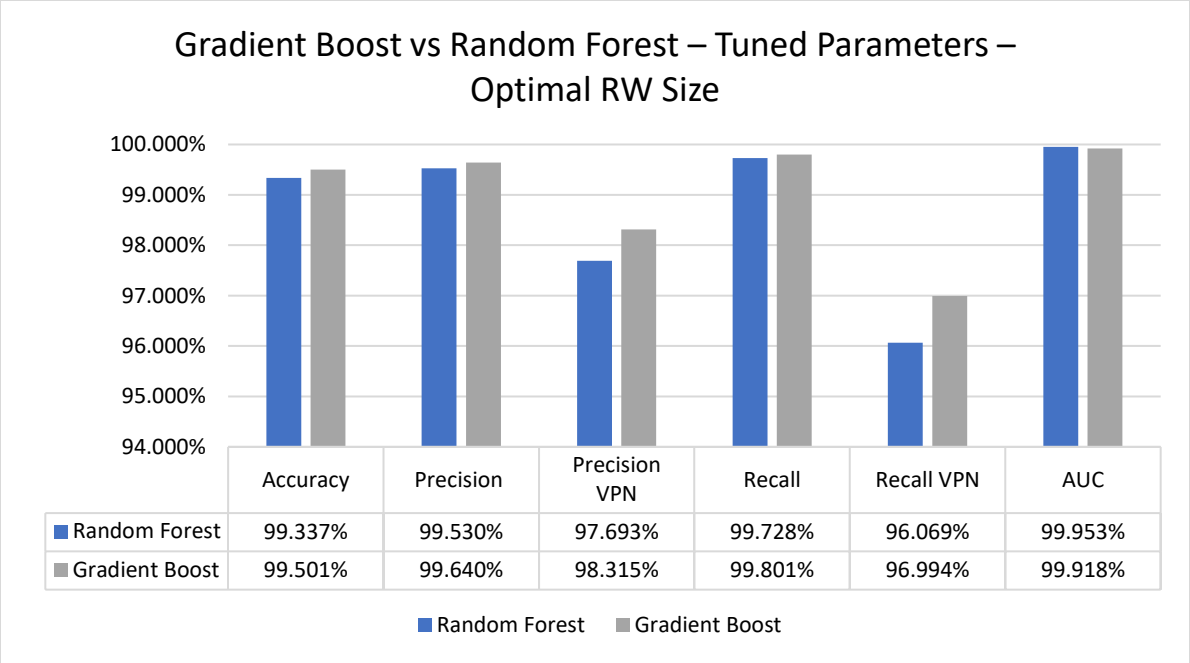
This last section of results will present the impact of changing the rolling window size for the model's predictive capability. All models are trained using the connection feature subset as it was the best performing subset with the tuned models. Table 10 displays the results with the tuned Random Forest model and Table 11 showcases the results for the Gradient Boost model. Figure 35 offers a comparison of the best performing random forest rolling window size vs the best performing gradient boost size.

Table 10: Results – Random Forest – Tuned Parameters – Connection Feature Subset – Various RW Sizes

Rolling Window Size	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
1000 Connection - 1 min	99.116%	99.373%	96.911%	99.638%	94.754%	99.882%
2000 Connection - 2 min	99.211%	99.456%	97.132%	99.662%	95.447%	99.887%
5000 Connection - 5 min	99.259%	99.483%	97.353%	99.688%	95.679%	99.912%
10000 Connection - 10 min	99.331%	99.539%	97.565%	99.713%	96.141%	99.923%
15000 Connection - 15 min	99.327%	99.523%	97.662%	99.725%	96.011%	99.931%
20000 Connection - 20 min	99.337%	99.530%	97.693%	99.728%	96.069%	99.953%

Table 11: Results – Gradient Boost – Tuned Parameters – Connection Feature Subset – Various RW Sizes

Rolling Window Size	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
1000 Connection - 1 min	99.420%	99.577%	98.089%	99.775%	96.459%	99.819%
2000 Connection - 2 min	99.361%	99.516%	98.036%	99.770%	95.953%	99.240%
5000 Connection - 5 min	99.474%	99.633%	98.127%	99.778%	96.936%	99.877%
10000 Connection - 10 min	99.484%	99.627%	98.270%	99.796%	96.878%	99.903%
15000 Connection - 15 min	99.453%	99.601%	98.194%	99.787%	96.661%	99.692%
20000 Connection - 20 min	99.501%	99.640%	98.315%	99.801%	96.994%	99.918%



	Accuracy	Precision	Precision VPN	Recall	Recall VPN	AUC
Random Forest	99.337%	99.530%	97.693%	99.728%	96.069%	99.953%
Gradient Boost	99.501%	99.640%	98.315%	99.801%	96.994%	99.918%

Figure 35: Results – Gradient Boost Vs Random Forest – Tuned Parameters – Optimal RW Size (N.B.: Vertical Axis Starts At 94%)

The main takeaway from these results is that the size of the rolling window has a minimal impact on the model's predictive capabilities.

6.0 ANALYSIS OF RESULTS

The aim of this project is to determine the ideal machine learning model for automated VPN. Based on the results, the model with the highest raw performance is the ***Gradient Boost model, with hyperparameter optimization, with connection feature subset, with rolling window size of 20 000 connections/20-minutes***. However, an ideal model encompasses more than just raw predictive capability, it must also consider efficiency. When considering the optimized parameter results in Table 8 (random forest) and Table 9 (gradient boost), there is a very small spread between the results of the various features subset (reverse feature subsets are not considered given their poor performance). For example, for GB (tuned parameters) the highest precision value was 98.270% and the lowest was 97.498%, resulting in a difference of only 0.772%. The same trend is found in the RF (tuned parameters) results which produced a difference of 1.023% between the top precision value of 97.565% and the lowest of 96.542%. In fact, the difference between the top GB precision value and the lowest RF value is only 1.728%. Moreover, this tight clustering of results is also seen in the RW size comparison results tables (Tables 10 and 11). Consequently, an ideal result can sacrifice a small amount of performance in exchange for gains in efficiency.

As discussed in the feature subset section of the report, the forward feature subset has a lower dimensionality and is thus more efficient with both the feature engineering and the

training/prediction of the model. Consequently, the selection of a forward feature subset for the ideal model is justifiable given the efficiency increase vs the marginal performance decrease. Specifically, the **connection forward subset is the ideal feature subset** for both models since it offers the best results with the lowest dimensionality.

Next, the choice of rolling window size should balance the size of the RW with the its results. The smaller the RW, the more efficient the feature generation. Therefore, the **5000 connection/ 5-minute rolling window is the ideal rolling window size** since it balances efficiency and predictive capability.

Lastly, the choice of model. Given that the predictive capabilities of both models are very similar, they could both be hypothetically used in a detection system. Yet, since the Gradient Boost model is the more robust model and prediction time between the two models are similar, **Gradient Boost is the ideal machine learning model**. Moreover, with respect to ideal hypermeters, given the considerable performance difference between the default parameters of GB and the optimized parameters, the **optimized parameters (learning rate 0.6, max depth 12, number of estimators 500) are the ideal hyperparameters**. However, if one were to choose the RF as the ideal model for an evaluation system, the system implementer would need to evaluate within the context of the system if the minimal increase in predictive capability justifies the increase in computation time of the optimized hyperparameters.

Ultimately, based on the results generated by this project, the ideal model for automated VPN detection is a Gradient Boost model with hyperparameter optimization using the connection forward feature subset generated by a rolling window of size 5000-5. Figure 36 shows the confusion matrix for this model applied to the entire dataset (random state none).

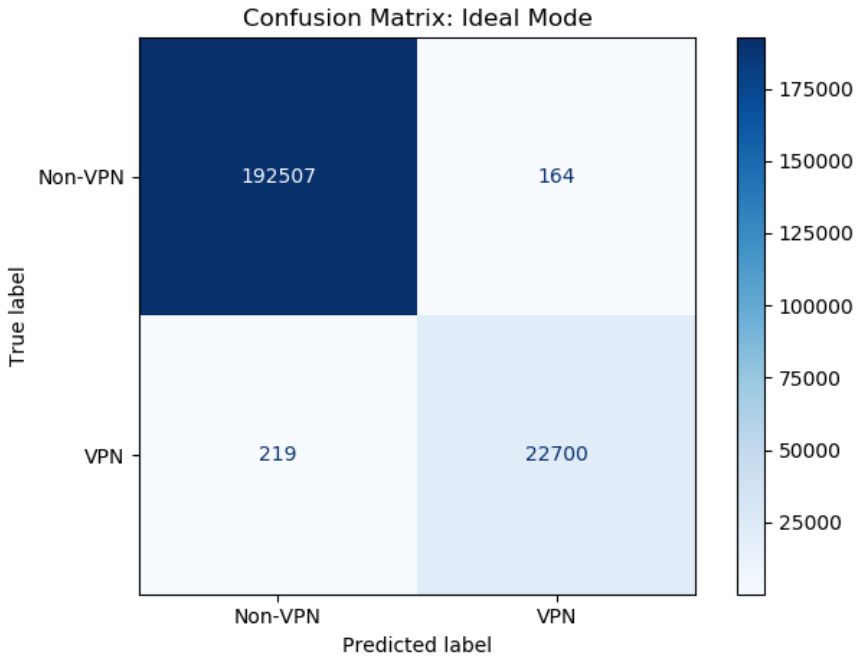


Figure 20: Confusion Matrix - Ideal Model - Entire Dataset

7.0 CONCLUSION

This project aimed to determine the ideal machine learning model for automated VPN detection. This was achieved by using the UNB CIC VPN traffic dataset and initially pre-processing the data through the CICFlowMeter application. Then, a rolling window approach for feature engineering was applied to the dataset to develop tailored engineered features based on connection and time-based rolling windows bi-directionally. These engineered features were then applied to Gradient Boost and Random Forest machine learning models. These models were evaluated with different features subsets, with different hyperparameters, and with different variations of the training data (rolling window size variation).

The ideal model was determined to be a Gradient Boost machine learning model with a learning rate of 0.6, a max depth of 12 layers and a maximum number of estimators of 300 decision trees, trained on the connection forward feature subset generated with a rolling window size of 5000 connections / 5 minutes. This model resulted in a predictive capability with an overall accuracy of 99.8%, a VPN classification precision rate of 99.4% and VPN detection (recall) rate of 99.0%.

REFERENCES

Asiri, Sidath. “Machine Learning Classifiers,” June 11, 2018. https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623.

Draper-Gil, Gerard, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. “Characterization of Encrypted and VPN Traffic Using Time-Related Features.” *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*, 2016. https://doi.org/10.5220/0005740704070414.

Feng, Wenqing, Haigang Sui, Jihui Tu, Weiming Huang, Chuan Xu, and Kaimin Sun. “A Novel Change Detection Approach for Multi-Temporal High-Resolution Remote Sensing Images Based on Rotation Forest and Coarse-to-Fine Uncertainty Analyses.” *International Journal of Remote Sensing* 10, no. 7 (2018). https://doi.org/10.3390/rs10071015.

Liberman, Neil. “Decision Trees and Random Forests,” May 21, 2020. https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991.

MathWorks Help Center. “Rolling-Window Analysis of Time-Series Models.” Accessed July 10, 2020. https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html.

Network Working Group , N. Brownlee, C. Mills, and G. Ruth, RFC 2722 - Traffic Flow Measurement: Architecture § (1999). https://www.ietf.org/rfc/rfc2722.txt.

Ramzai, Juhi. “Simple Guide for Ensemble Learning Methods,” March 6, 2019. https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2.

Yen, Lorna. “An Introduction to the Bootstrap Method,” January 28, 2019. https://towardsdatascience.com/an-introduction-to-the-bootstrap-method-58bcb51b4d60.

Yiu, Tony. “Understanding Random Forest,” August 14, 2019. https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

Zhou, Victor. “Random Forests for Complete Beginners,” April 10, 2019. https://victorzhou.com/blog/intro-to-random-forests/.