## Introduction

## Background

## 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]](#footnote-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 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).

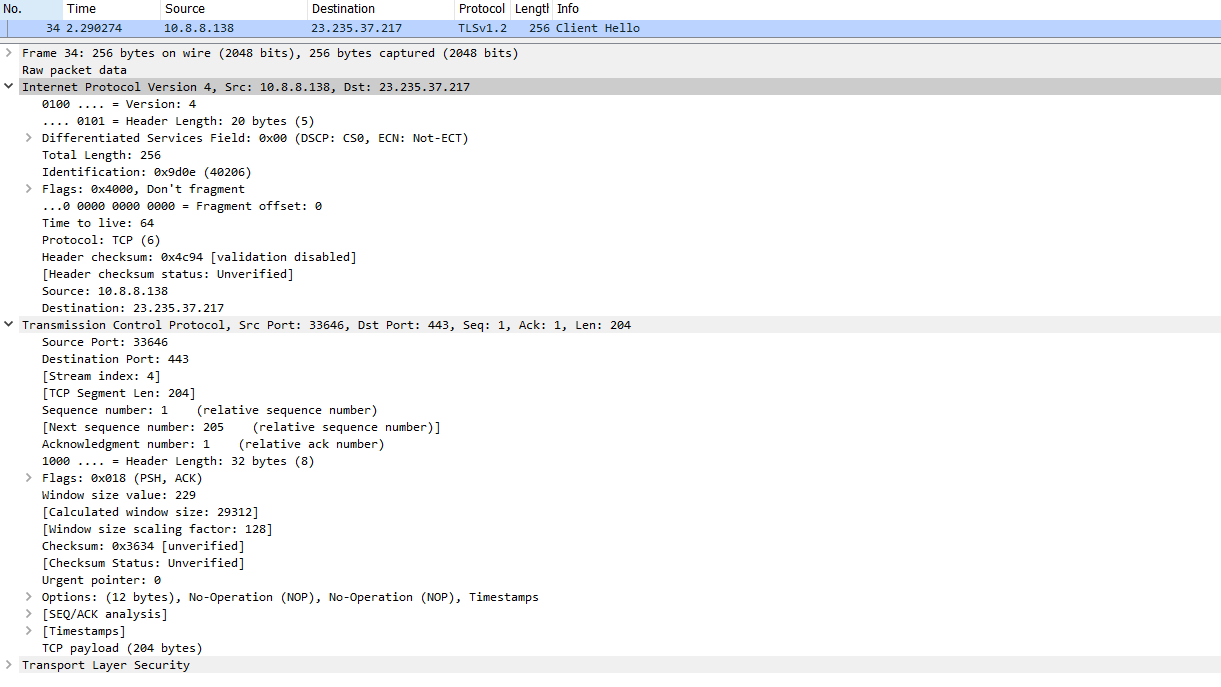


Figure 1: Packet Capture with Header Information from vpn\_vimeo\_A.pcap of ISCXVPN2016 Dataset[[2]](#footnote-2)

Transition

### 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 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”[[3]](#footnote-3). 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.

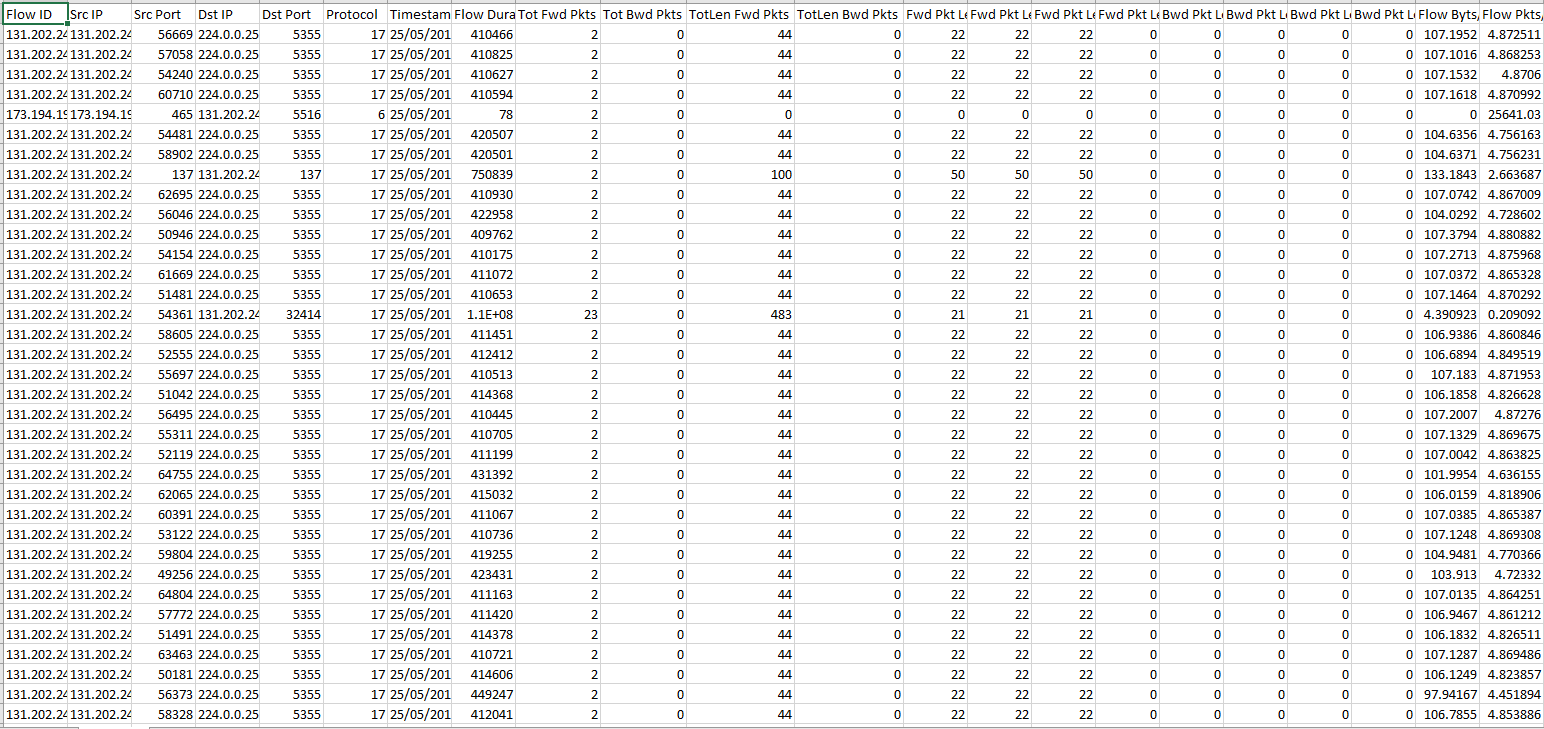


Figure 2: Netflow CSV Generated by CICFlowMeter from email1a.pcap

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.

Transition

#### 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, in the context 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.

The desired baseline netflow features are source the 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.

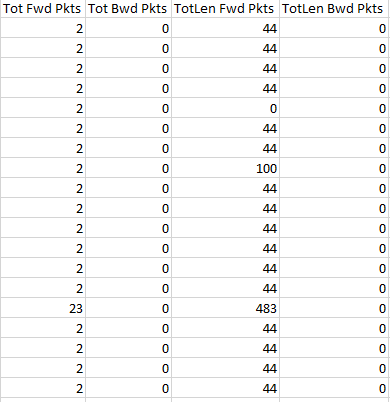


Figure 3: Packet Numbers & Packet Length Metric of Netflow Generated by CICFlowMeter from email1a.pcap

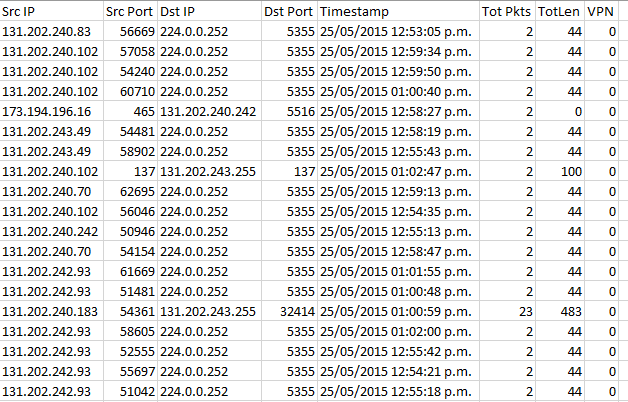


Figure 4: Filtered Netflow CSV (data derived from email1a.pcap)

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.

Transition/conclusion data

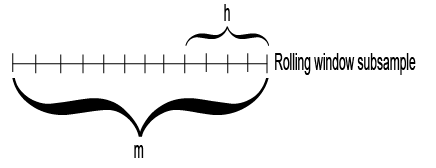
## Feature Engineering

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.

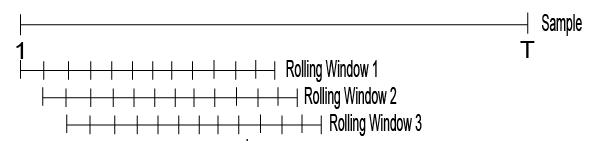
### Rolling Window

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 :



<https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html>

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. :



The RW framework was selected to enable the feature engineering since the netflow data represents a time-series given that they occur at a given time and that the connections occur sequentially (one connection opens, then a second connections opens, then the first closes, then a third connection opens, etc…).

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 bidirectional nature of internet communication, both forwards (source -> destination) and reverse (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.

Table 1: Categories of Engineered Features

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

Transition

#### 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, calculate 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. 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, calculate 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.

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.

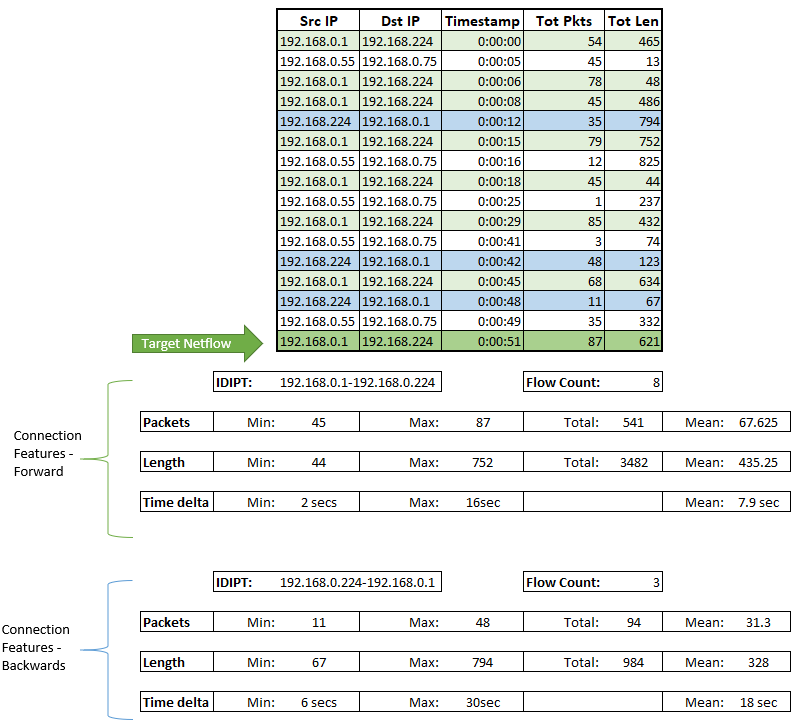


Figure 7: Feature Engineering Via Rolling Window in Forward (Green) and Reverse (Blue) Direction Example

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.

Transition

#### Implementation

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

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.

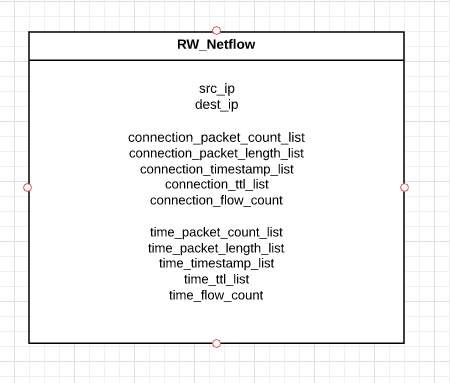


Figure 8: RW\_NETFLOW Class with attributes

The class contains two categories of similar attributes: the first contains netflow feature (ex: timestamp, packet lengths, etc…) 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.

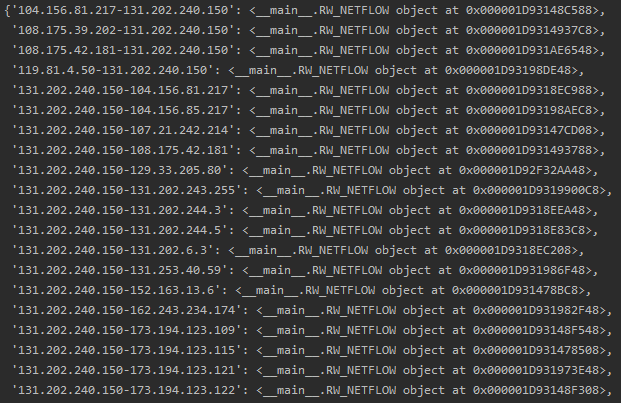


Figure 9: Example of dictionary containing RW\_NETFLOW objects

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 index of the netflows 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 struct 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).

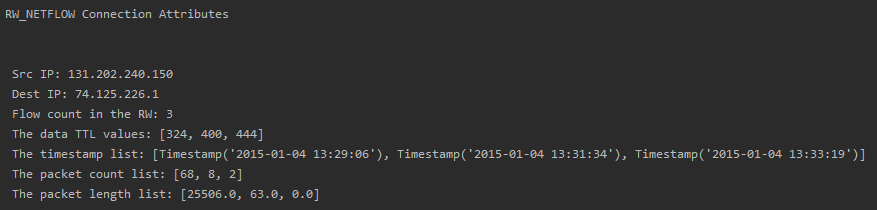


Figure 10: Example of the connection attributes of a RW\_NETFLOW object (RW size of 500)

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 network metrics for that index position (i.e. the netflow) 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 whethera netflow is in the RW occurs through the timestamps. By calculating the time delta between the target netflow’s timestamp and the oldest timestamp in the timestamp\_list, if the difference exceeds the time based RW size then the value is not considered to be in the RW and all associated network metrics are popped from the respective list. Although the lists are acting as a queue, there is no maximum limit to the numbers of values in the list since the number of encountered netflows with a given SDIPT in any given timeframe varies.

To improve efficiency, the end of the list is the initial check and if that check fails, all other values in the TTL list are also outside of the RW (the last item is the newest). 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 bellow in Figure 11.



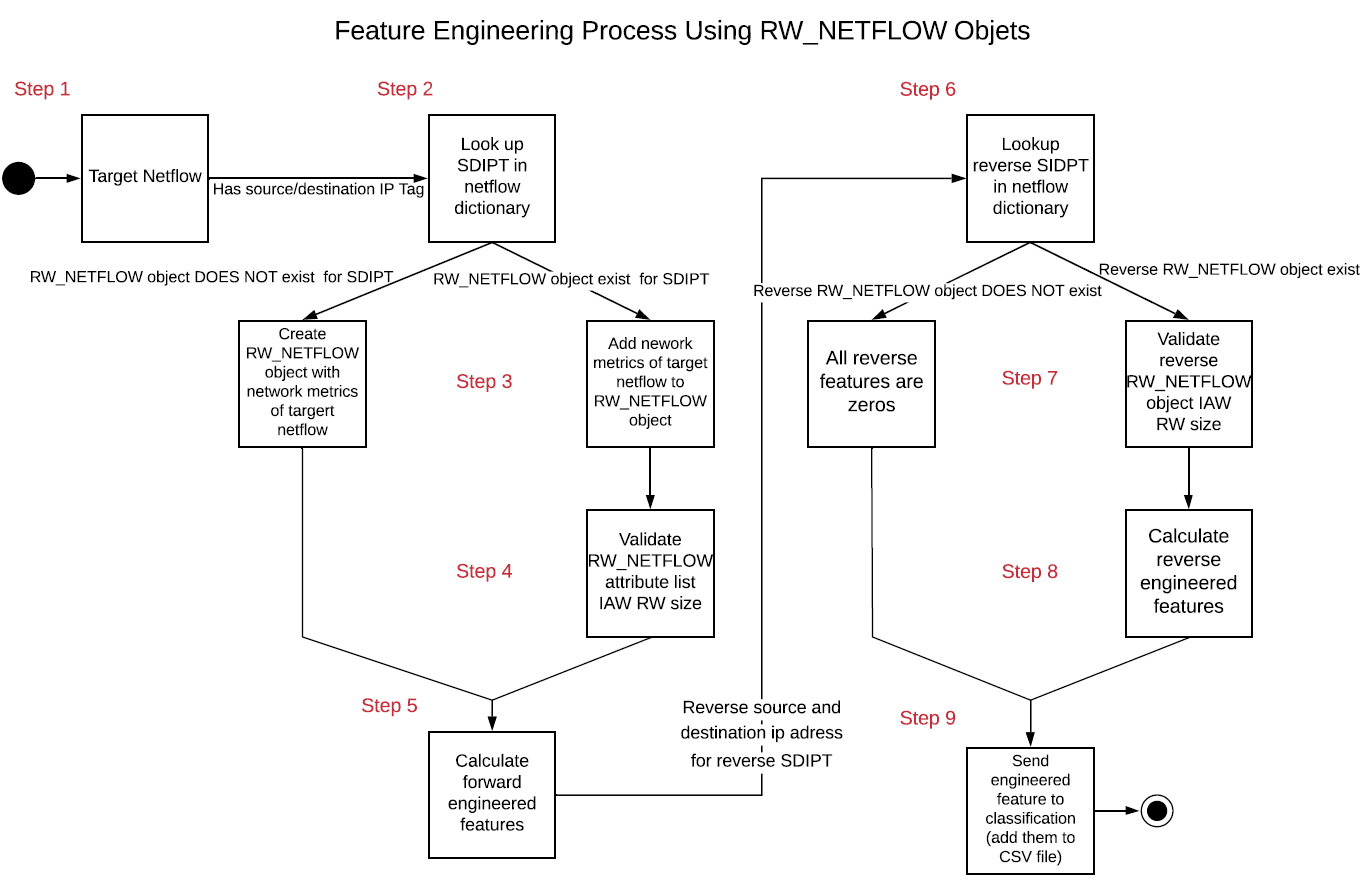
Figure 11: Engineered feature (connection forward). [Format: flow count, time delta features (min, max, mean), packet length features (min, max, mean, total), packet count features (min, max, mean, total)]

With regards to the calculation of reverse features, the process is very similar. Since it is the reverse direction, 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 well as how much information each RW\_NETFLOW object is ‘holding’. As a result, larger RW sizes take longer to generate. It 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 can pass it along to the classifier. While the history of previous netflows is maintained in RW\_NETFLOW object.

The average time to generate the engineered features for a single netflow is xx.

Recap

Figure 12: Summary of Feature Engineering Process via Rolling Windows and RW\_NETFLOW Class

Transition /Conclusion for FT eng

## 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 classifier 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 prediction, the predictive model is the only source for the target data classification. Lazy learners are fast learners and slow predictors while eager learners are slow learners and fast predictors[[4]](#footnote-4). When considering the intent of implementing the classification system in a near real-time system perdition time should be minimized, therefore eager learning classification model will be used.

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

Given these characteristics, in particular the fact that the data is not normalized, and the need for an eager learner, the decision tree is favorable model for classification. However, decision trees require special consideration of 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[[5]](#footnote-5).

Ultimately, to overcome the drawbacks of the decision tree 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.

### Random Forest

The random forest 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 show an example of majority voting for a multiclass classification problem; however, the sample principle applies to binary class classification.

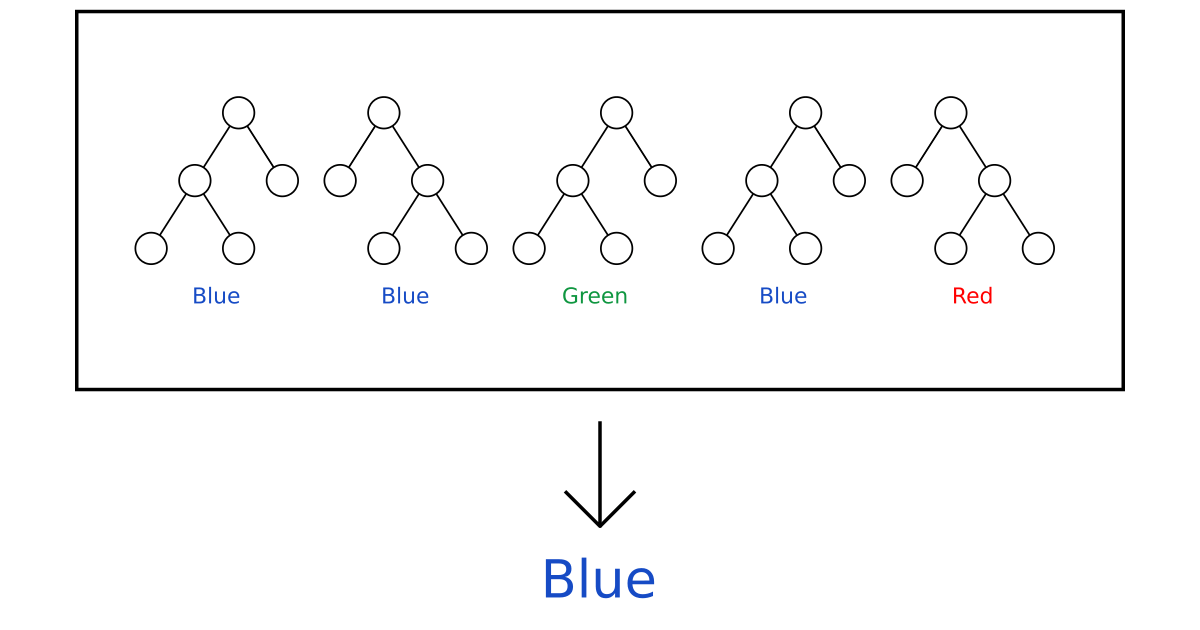


Figure 13: Example of majority voting from multiple independent decision trees in a random forest model.*[[6]](#footnote-6)*

The bootstrap portion of the bagging technique is that seed data for 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”[[7]](#footnote-7). 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[[8]](#footnote-8). 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[[9]](#footnote-9). Figure 14 show the general struct of a random forest with the bootstrapping of training data and the aggregation of the individual tree to produce a final result.

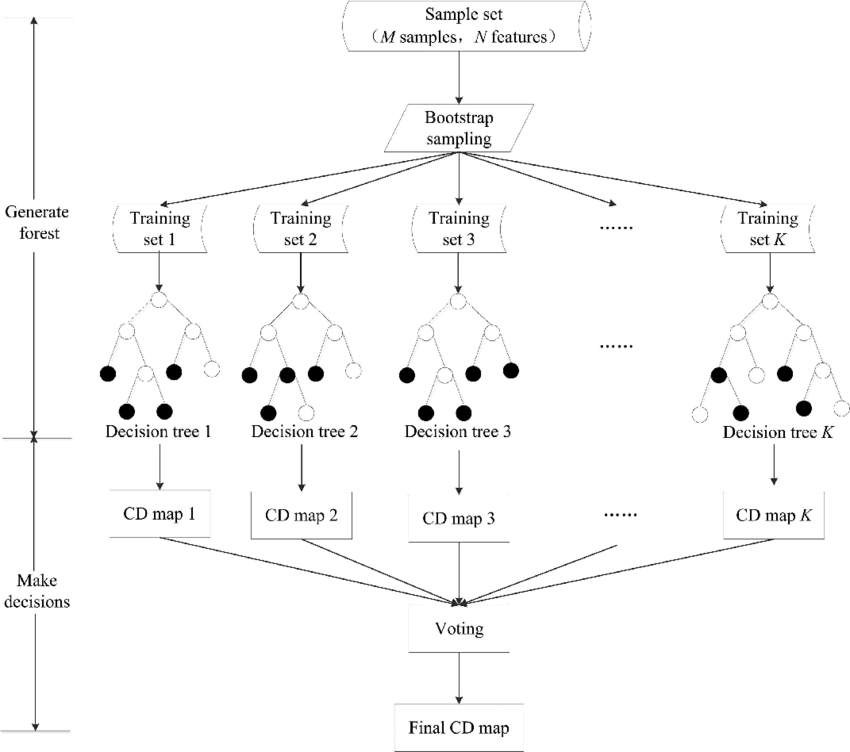


Figure 14: Random forest structure highlighting bootstrapping and aggregation (note cd map is equivalent to a prediction)*[[10]](#footnote-10)*

The overall advantages of random forest classifiers are 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.

//Transition

### Gradient Boost

The gradient boost model is an implementation of the boosting ensemble technique; it combines multiple weak models (the decision tree) sequentially to create the strong model. The process of training the gradient boost model to a dataset is iteratively (sequentially) where in each subsequent a new tree is added to the model that aims to correct any errors or misclassification in the previous tree. 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 improve the predictions of the previous tree. The overall prediction of the gradient boost 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.

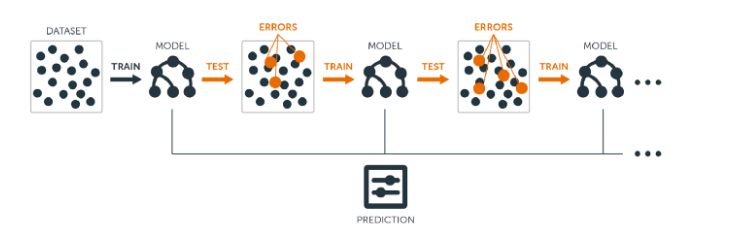


Figure 15: Gradient boost process where each iteration of the model is the sequential interaction of a new tree which aims correct the errors in the previous iteration*[[11]](#footnote-11)*

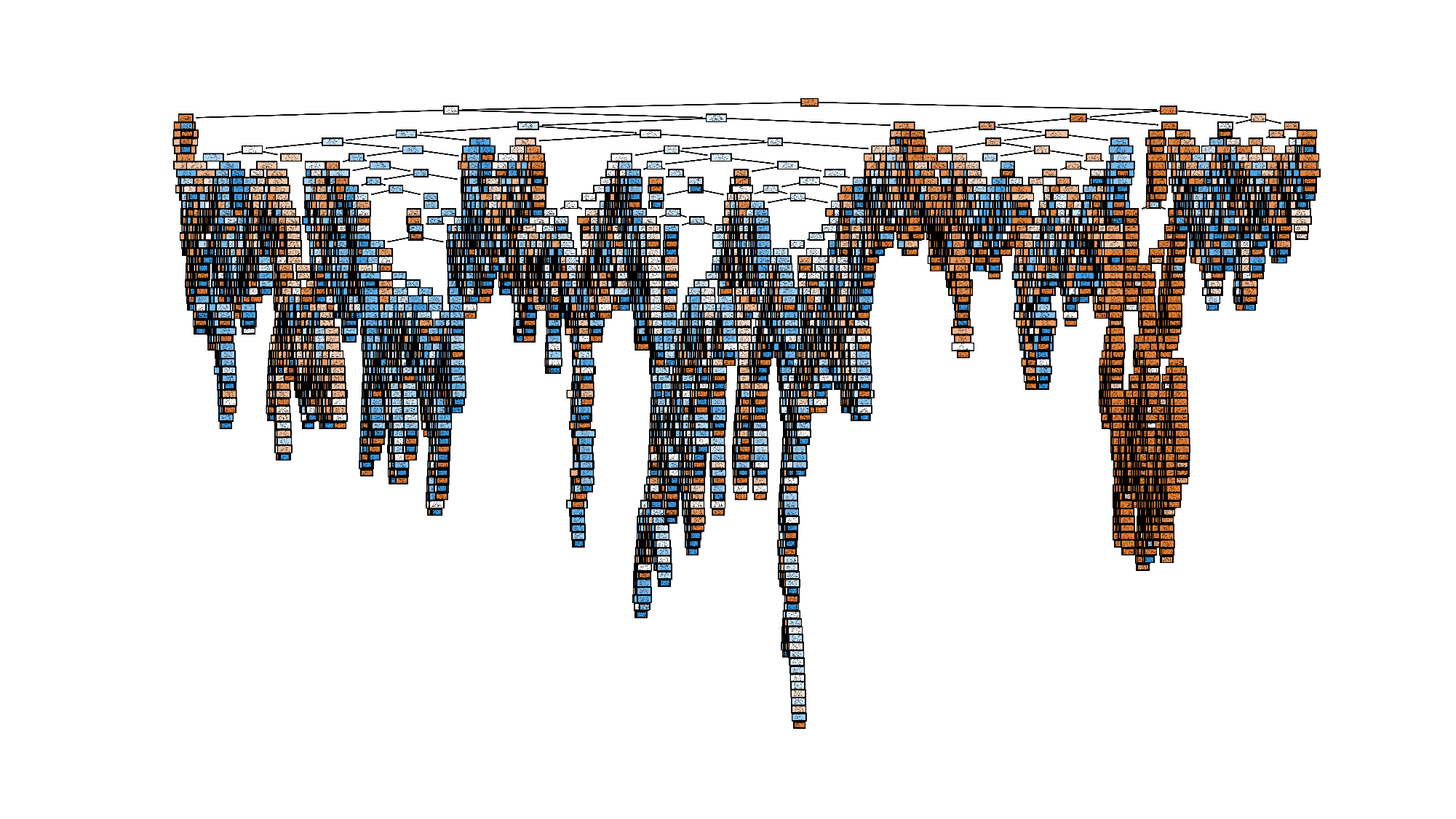
Although the gradient boost model shares similar advantages to the random forest model, one notable issue with the gradient boost model is that due to the sequential nature, it is quite vulnerable to overfitting. Therefore, the model relies on hyperparameters to fine tune the model to the given dataset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

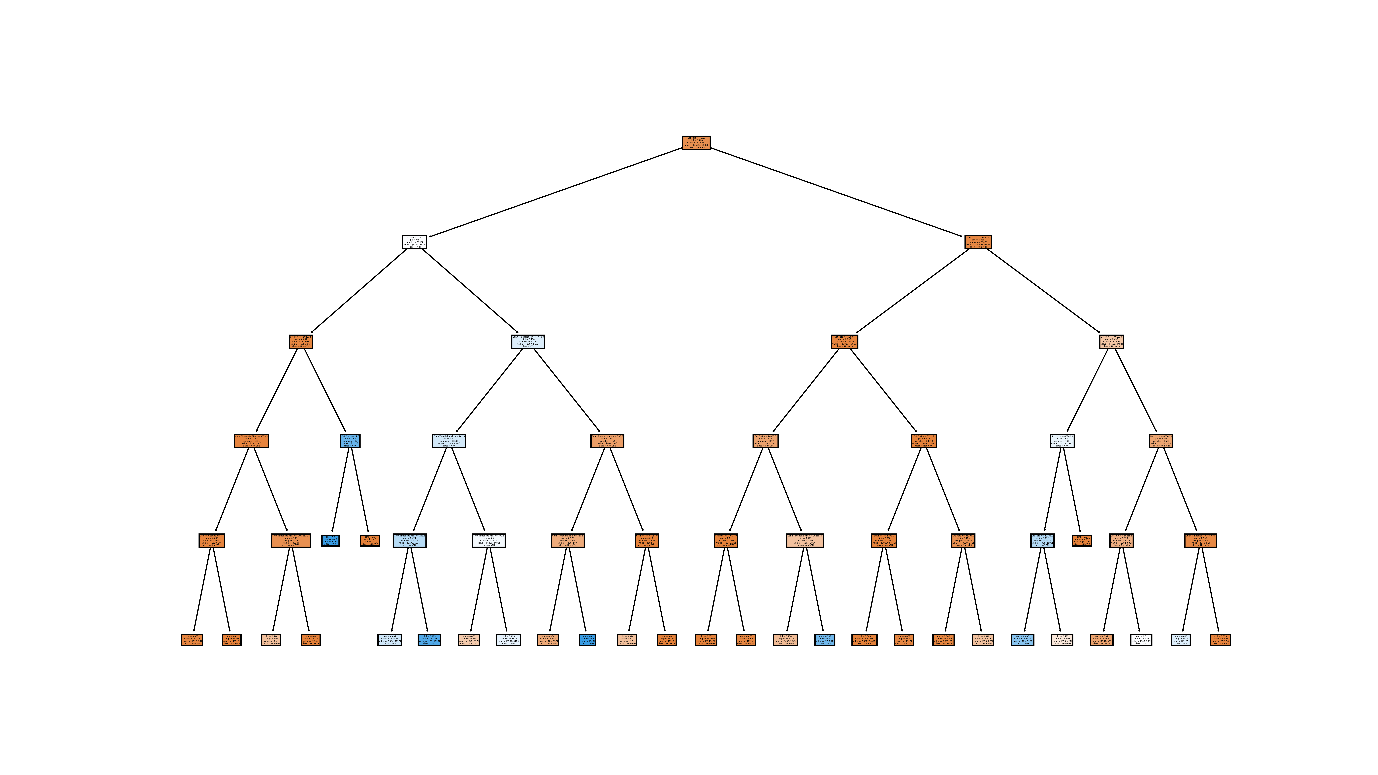
//Transition

### 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.

Figure 16 shows 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)



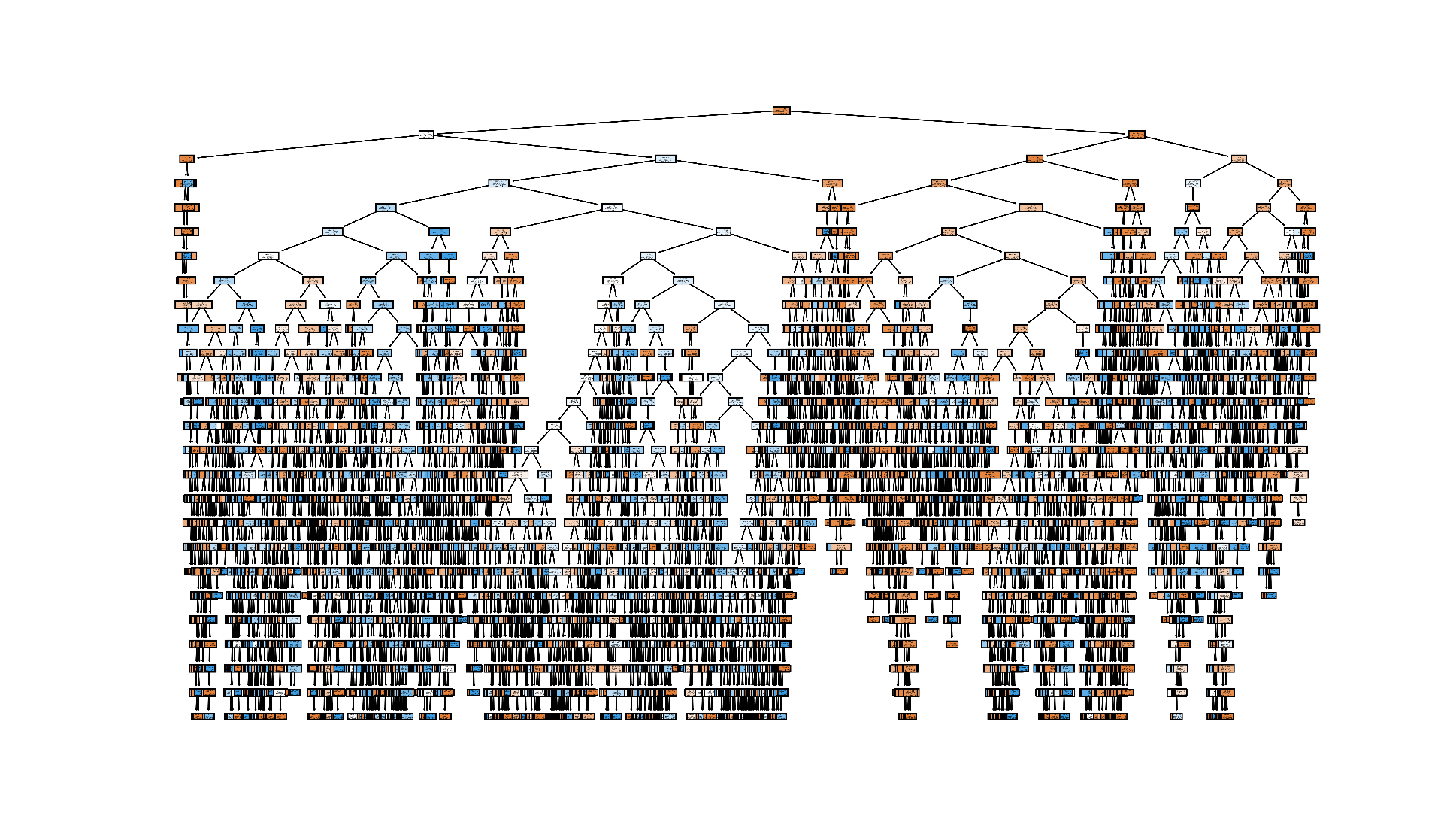
Figure 16 (b): Singular decision tree from the implemented random forest (max\_depth: 5 layers)

Figure 16 (c): Singular decision tree from the implemented random forest (max\_depth: 25 layers)

The decision tree in generated by the random forest model (Figure 16) are similar in structure to those generated by the gradient boost model. The main difference between the model 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.

### 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 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 will 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 given 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.

// Transition

### Hyperparameter Optimization

As discussed in the gradient boost section of this report, the gradient boost model requires hyperparameter tuning to have peak predictive capacity. This section of the report will discuss the hyperparameters that were optimized and what 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 also underwent hyperparameter tuning, however the improvements an engendered by the tuning were minimal (will be further discussed in report section.

For both models, the hyperparameters were tune by running GridSearchCV multiple time, where 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 feature subset which had the best metrics with default parameters.

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

The following tables shows the tuned hyperparameters, (Table 2 is gradient boost and Table 3 is random forest), their default value and the order they were tuned.

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 |

// Transition

### Feature Subsets

In any machine learning model, certain features will have a greater impact on the classification of the target object. Figure 17 shows the feature importance in decrementing order of the random forest model with using different feature sets.

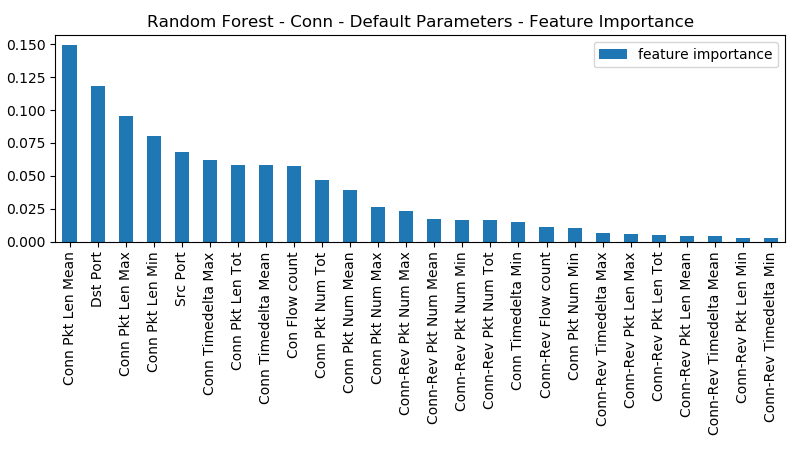


Figure 17 (a): Feature importance for random forest model (connection feature subset).

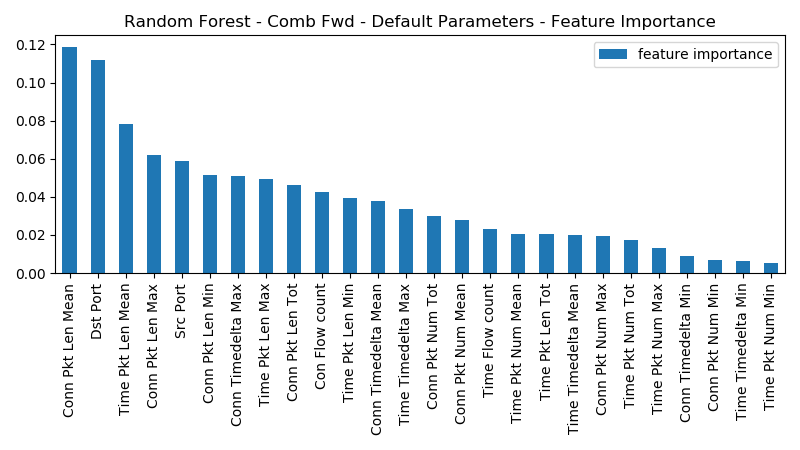


Figure 17 (b): Feature importance for random forest model (connection forward feature subset).

These two figures highlight that of the four feature engineering categories in Table 1 (discussed in rolling window section of feature engineering) certain features 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 features is the 12th most important) and in Figure 17(b) connection features play a slightly more important role than time features (a connection feature is the most important feature and of the top 10 features time feature are constitute 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 combine), 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 Engineered Ft | Connection  ½ Engineered Ft |
| Forward | Combined – Fwd  ½ Engineered Ft | Time – Fwd  ¼ Engineered Ft | Connection – Fwd  ¼ Engineered Ft |
| Backwards | Combined – Bwd  ½ Engineered Ft | Time – Bwd  ¼ Engineered Ft | Connection – Bwd  ¼ Engineered Ft |

The potential advantages 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.

// Transition

### 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 those engineered features constitute the training data, there is the possibility that a change in rolling window size could change 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 difference 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.

// section conclusion

## Results

### Baseline

### Default Parameters

### Optimized Hyperparameters

### Rolling Windows Sizes

## Analysis

## Conclusion

1. <https://www.unb.ca/cic/datasets/vpn.html> [↑](#footnote-ref-1)
2. Ibid [↑](#footnote-ref-2)
3. <https://www.ietf.org/rfc/rfc2722.txt> [↑](#footnote-ref-3)
4. <https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623> [↑](#footnote-ref-4)
5. <https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991> [↑](#footnote-ref-5)
6. <https://victorzhou.com/blog/intro-to-random-forests/> [↑](#footnote-ref-6)
7. <https://towardsdatascience.com/an-introduction-to-the-bootstrap-method-58bcb51b4d60> [↑](#footnote-ref-7)
8. <https://towardsdatascience.com/understanding-random-forest-58381e0602d2> [↑](#footnote-ref-8)
9. ibid [↑](#footnote-ref-9)
10. <https://www.researchgate.net/publication/326436981_A_novel_change_detection_approach_based_on_visual_saliency_and_random_forest_from_multi-temporal_high-resolution_remote-sensing_images> [↑](#footnote-ref-10)
11. <https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2> [↑](#footnote-ref-11)