CES 632 Project 2:

Data Mining for Classification of Web Traffic

Chrisina Giagnoni

February 17, 2021

Abstract

We will explore a type of cyber attack, called a distributed denial of service attack (DDoS), through research of current literature, and then with a given data set, we will create models using data mining algorithms for classification of DDoS or benign web traffic. The data set has multiple features, and a classifying column with binary labels "ddos" or "Benign." The data will require preprocessing, feature selection and dimensionality reduction before model building. We will take suggestions from the literature review, and use a decision tree for feature selection of categorical variables. For dimensionality reduction of numerical data we will use PCA. After data cleaning, we separate the data into a test set and a training set. Then we used three different data mining algorithms: random forest, logistic regression and support vector machine, to build three different models with the training data set. Then we test each of the models with the test data and summarize their performance with a variety of accuracy measures and ROC curve. We use the performance summaries to compare all three models and offer a comparative analysis of their strengths and weaknesses. Finally, we offer our conclusions, limitations, and a discussion on possible future work and open questions.

3.1 Problem description

The internet is now an integral part of retail, industry, and communication, so cyber security is of vital importance in modern society. Monshizadeh and Yan explain cyber attacks as follows "Cyber-attacks are also called as intrusions that stand for any types of actions that threaten network security and stability." There are multiple methods of cyber attack. The one we will focus on is a distributed denial of service attack (DDoS).

A DoS attack is a cyber attack where a single perpetrator floods a website or server with superfluous service requests. This causes an online "traffic jam" which prevents legitimate service requests to be filled slowly, denied, or cause a system crash (CISA). In a DDoS attack, multiple sources send the fake service requests to cause a system crash or disruption of service requests (Garg, Chawla, 2010). A DDoS attack includes two steps. The first step is a "recruiting phase" where the master computer finds "zombies" to run the attack commands. This recruitment of zombies can be done without a user's knowledge through installation of malware. In the "action phase", the master computer tells the zombie computers to run the installed malware, which opens the flood of fake service requests.

In both DoS and DDoS attacks, the motivation to carry out such an attack varies from political to personal. Recently, the "hacktivist" group Anonymous was credited with a suspected DDoS attack on the Minneapolis Police department in relation to the George Floyd protests in 2020, which took the department's website offline for a few days (Molloy, D. and Tidy, J., 2020).

It can be tricky to determine a real flood of service requests - e.g. a retail site crashes due to too many people trying to buy a high-demand item at the same - versus a DDoS attack (Molloy, D. and Tidy, J., 2020). However, data mining can be used to detect DDoS attacks in an attempt to stop or prevent them. So we will begin with a review of some current literature on using data mining to combat network security issues, and in particular, DDoS attacks. Then, we will be using various data mining techniques to classify whether web traffic is benign or a DDoS attack. Our training data set has 760,427 observations and 84 various features as well as the binary labels "Benign" or "ddos", which we will use for model training. After building some models, we will use a test set to evaluate the accuracy of our models and decide which model is "best" for classification of web traffic. In our conclusion, we will discuss the strengths and weaknesses of our models and model building process that can not be quantified by error estimation.

3.2 Literature review

The paper titled *Security Related Data Mining* by Monshizadeh and Yan provided an overview of what data mining is, as well as common data mining techniques, then related how the techniques could be applied

in cyber security. The data mining techniques were divided into two sections: classification and clustering. The classification algorithms covered were Bayesian networks, decision tree, artificial neural network, support vector machines, and k-nearest neighbor. The clustering techniques covered were fuzzy clustering, partitioning, k-means, genetic clustering algorithm, hierarchical CURE clustering, and swarm intelligence. After explaining each technique, the authors organize the techniques in a table providing advantages and disadvantages of each technique. Another table makes comparisons of the attributes of efficiency, robustness, insensitivity, simplicity, unknown pattern detection and adaptability. The general conclusion they draw is that linear methods - such as decision trees - are good for feature selection since they are fast and easy to understand. They also point out that more complex algorithms - such as neural nets - are good for unknown pattern recognition so it is good for learning new types of attacks. After discussing the benefits and drawbacks of different algorithms, the paper explains how they use these data mining algorithms for attack detection. They broke the method down into four main steps: classification/clustering, anomaly detection, link analysis, and association rules. Each step required a different learning algorithm. The authors then survey previous research discussing different techniques used and compare their accuracy for the given cyber security problem. They conclude that the algorithms that perform the best are combinations of more than one algorithm. For example, a combination of decision tree (for feature selection) and neural net (for labeling) for DoS attacks performed with a 97.1% accuracy rate.

The paper *DDoS Detection System Based on Data Mining* by Zhong and Yue provides a brief explanation of DDoS attacks, including three characteristics of a DDoS attack: abnormal traffic, send a request to a server but never completes the handshake, and uses a characteristic of TCCP/IP protocol on which non-compliant packets could be used for DDoS attacks. They also include a brief review of common detection methods for DDoS attacks: protocol analysis, cluster, and network traffic statistics. The authors explain that all three methods have some problems, and claim that a possible solution would be to create combinations of these typical methods to make up for deficiencies of the individual methods. They propose a six-step detection procedure and include a flow chart. To summarize the protocol, the machine captures packets and uses two different algorithms for two different features: network packet protocol, and network traffic. The proposed detection procedure is then tested with a LAN, and finds that the model detection is at least 97%.

The last article we reviewed titled *Detecting Distributed Denial of Service Attacks Using Data Mining Techniques* by Alkasassbeh, Hassanat, Al-Naymat, and Almseidin introduces DDoS attacks and presents a relatively straight forward model for classification of web traffic into five categories: normal, or a type of DDoS attack (Smurf, UDP-Flood, HTTP-Flood, SIDDOS). The authors collected their own data set of 27 features - since they argued that available data sets were unrealistic and outdated - and used it to build their classification model. The authors give a table with all 27 features which includes the feature type. They test three data mining techniques: multilayer perceptron, naive Bayes and random forest. To evaluate all three models, the authors compare the precision and recall of all three models, and conclude that the MLP outperforms the random forest and naieve Bayes algorithms.

All reviewed papers evaluated different models and model building techniques. The first two papers argued that in order to achieve the most robust and accurate results, the model must include more than one algorithm. The third article, however, tested three models - each built by one algorithm - and determined that a MLP gave the best results. As we move forward, we will consider the results of all three articles in our process of feature selection, parameter selection, finally model construction, and model testing.

3.3 Software tools used for predictive data mining

R is open source statistical software commonly used in academia and industry. The base R comes with many commonly used functions, but "packages" can be installed and used for functions that do not come with the base software. We used R for preprocessing, dimensionality reduction, model building and model testing. The following table summarizes all R packages used.

Package	Description	Usage
dplyr	Tools for working with data frames	Preprocessing
factoextra	Functions for visualizing multivariate data analyses	Visualization
rpart	Recursive partitioning and regression trees	Model Building
rpart.plot	plots rpart models	Visualization
caret	functions for model training	Model summarization
randomForest	Classification and regression based on random forest	Model Building
nnet	feed-forward neural nets and multinomial log-linear models	Model Building
e1071	Misclelanious statistical functions	Model Building
pROC	For plotting ROC curves	Model evaluation

Many of the R functions used were also built-in. All annotated code is included in the appendix, and all packages are cited in the works cited section.

3.4 Preprocessing of data and dimensionality reduction

General Preprocessing

Based on the literature review, it is clear that not all 84 features are necessary to build an effective model. Thus our first task is to examine the features of the data, then use that information and other techniques to reduce the dimensionality of the data set. Before looking closely at the individual features, we gathered some initial information about the training data set. There were 84 features, a Label column labeling each entry as "ddos" or "Benign", and 760,427 observations. We then checked for missing values, and found that only one feature contained NA values: Flow.Byts.s had 2,996 NA values. Removing the 2996 rows would still leave 757,431 observations, so we decided that would be the best procedure to follow. Next, we checked for columns with only one entry type, and removed them since they would not provide us with any useful information. This reduced our number of features to 76.

After removing rows with NAs and non-informative features, we will look at the remaining 76 features. Most features appear to be numeric, however, we will need to take a closer look at some to be sure that some of the numbers are not catergorical. We created a table, which can be found in the appendix, based on the file given with the data set that explains most of the features. While Alkasassbeh et.al. found time to be an important factor, our data set was inconsistent in labeling times "AM" or "PM" rendering the feature "timestamp" essentially useless, so we discarded it. R read in binary factors as numeric, so we changed them to factors, so that R would treat them as binary factors rather than numerical values. We also changed the labels from ddos and Benign to 0 and 1, respectively, so that they are easier for the software to work with. Our hardware was limited and could not work with the entire given training set (757,431 observations), so for all following processes in R, we only used a randomly selected portion of the data.

Dimensionality Reduction

So now our task is to reduce dimensionality of the numeric and categorical features. We will use principal component analysis (PCA) to reduce dimensionality of numeric factors. PCA reduces dimensionality by transforming numerical data into new vector samples, then by checking the amount of information "explained" by the principal components via cumulative variability. Then the numerical vectors are replaced by a smaller selection of the new vectors (principal components) based on the desired amount of variability explanation. Then a model can be built by the new data of smaller dimension. PCA is sensitive to unit differences, so the data should be standardized (Kantardzic). It is important to note that PCA depends on the distribution of the data, that is, for PCA to be justified we need to check that the data is approximately multivariate normal. This is a very strong condition, and when we checked for normality, by attempting to create a Q-Q plot and check skewness and kurtosis, we found that the data matrix was singular, so while

PCA is not justified will proceed anyway. It appears that it is common practice to use PCA without checking normality, as this condition is not mentioned in the textbook *Data Mining Concepts, Models, Methods and Algorithms* by Mehmed Karardzic. There were 59 numeric features with all finite entries (two features, Flow.Pkts.s and Flow.Byts.s, had infinite values, so we removed them from the PCA), and we subsetted the data into 100,000 values for PCA computation. We loosely followed M. Lopes' guide to using PCA for dimensionality reduction to find the principal components with the built in R function `prcomp`. We found that 18 principal components explain 95% of the variance, so we will replace the 59 numeric features with the 18 principal components. We included a scree plot of the PCA made with the `factoextra` package. We noted the "elbow" of the scree plot then we would indicate 8 principal components may be enough, but this only explains 77-80% of the variance. We decided it would be best to train and test our models twice, once with 18 principal components and once with 8 principal components to see if there was a big difference in misclassification.

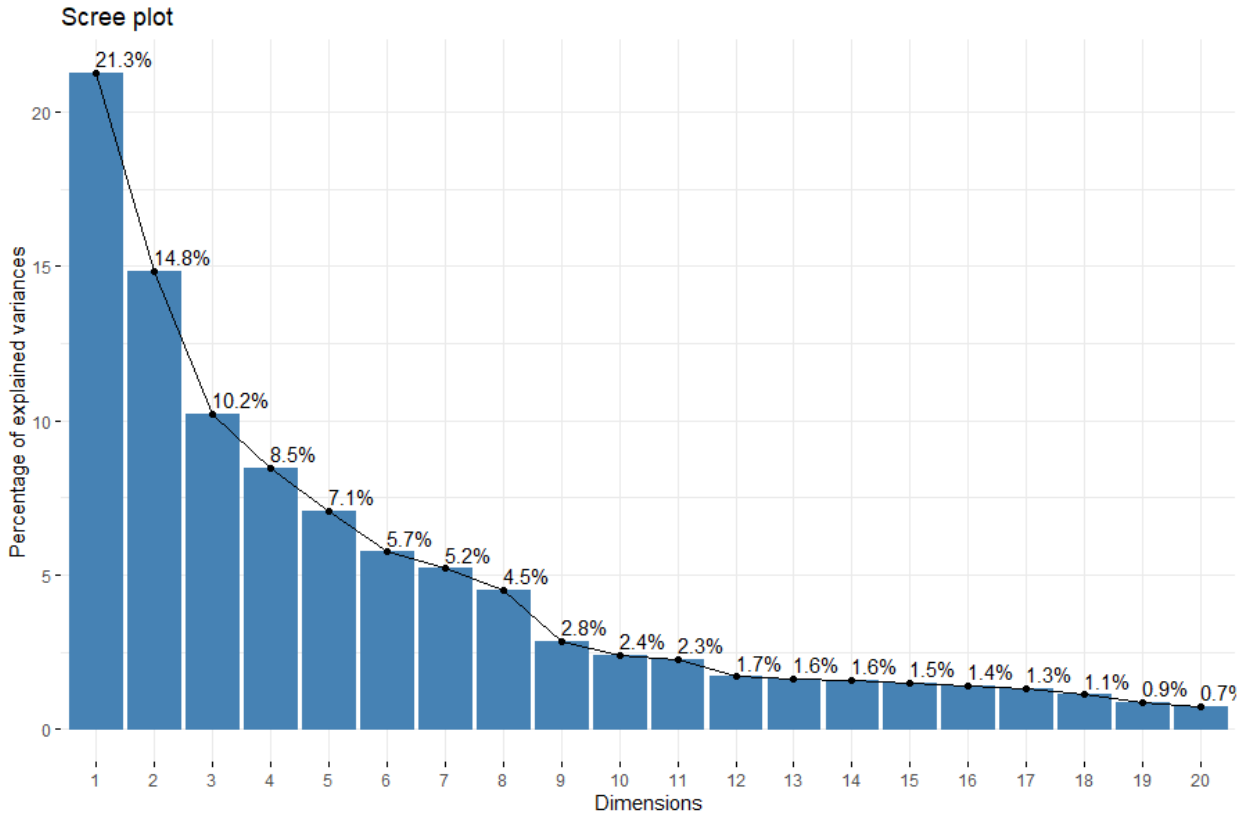


Figure 1: Scree Plot for PCA

Feature Selection

Monshizadeh and Yan mentioned that decision trees were good for feature selection, so we decided to use a tree from the `rpart` package to select categorical features, and we included the two numerical features removed from the PCA: Flow.Pkts.s and Flow.Byts.s. Decision trees are simple to understand, and use recursive splits to predict or, in our case, classify. We subsetted the training data into 300,000 observations. From this tree we see that important features were ACK.Flag.Cnt, CWE.Flag.Count, SYN.Flag.Cnt, Flow.Pkts.s and Flow.Byts.s.

When we went to build the models, we found that none of the algorithms could handle the infinite values, so we decided to remove the two features entirely. Then we repeated the tree feature selection excluding the

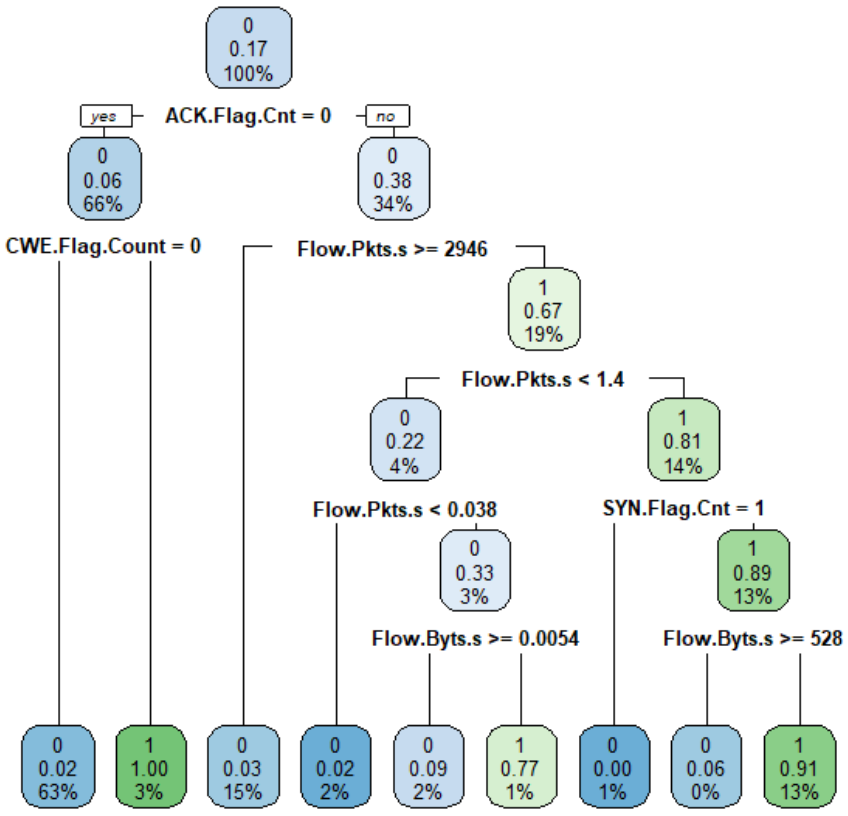


Figure 2: Decision tree for feature selection

numerical variables. Again, we used 300,000 observations, and got an even more simplified tree that had only two selected features: ACK.Flag.Cnt and CWE.Flag.Count.

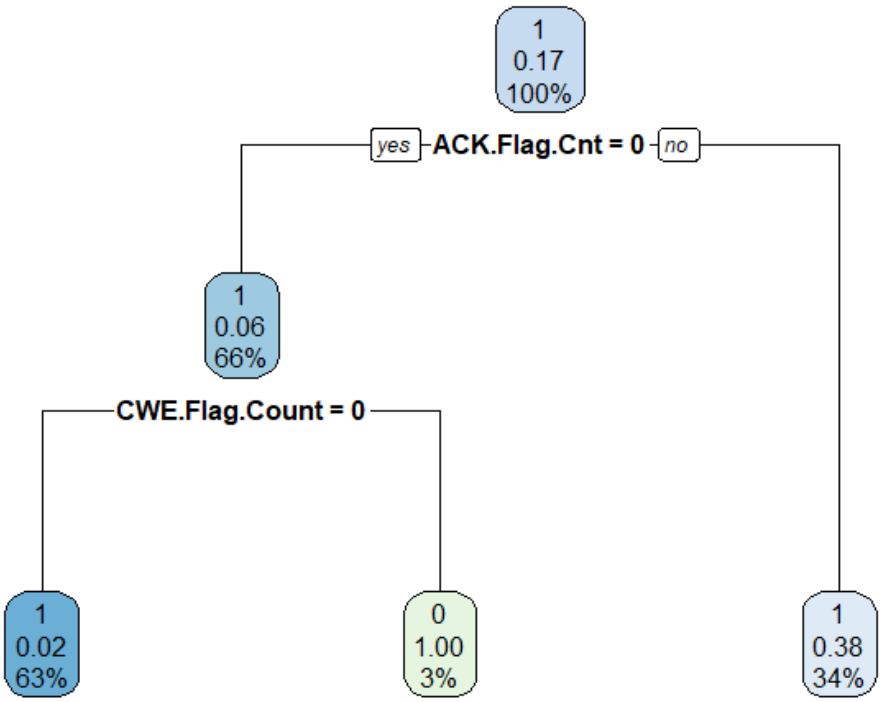


Figure 3: Decision tree for feature selection for only categorical variables

After preprocessing, dimensionality reduction of numerical features by PCA, and categorical feature selection via decision tree, our data set had 757,431 observations, 18 principal components, two binary features, and a column of labels transformed to 0 for a ddos attack, and 1 for benign.

3.5 Data mining process

We initially tried to use the 300,000 observations in the training set from preprocessing, but found that in some cases the hardware could not handle a data set of that size. Thus we reduced the training set to 100,000 observations, and put the other 657,431 observations in a test set. We note that this is not a normal sized split, since normally the training set would be the larger of the two, however the large test set should give good error estimation (Shao, 1993). We found the correct principal components for the training and testing sets and created new sets with 18 principal components, ACK.Flag.Cnt, CWE.Flag.Count, and Label. This gave us a total of 20 explanatory variables and a column of labels.

After testing each model we output some basic summaries including a confusion matrix; a summary table which gives accuracy, no information rate, sensitivity, specificity, balanced accuracy, F1, area under the curve (AUC); and plotted a ROC curve using packages `caret` and `pROC` (Kuhn, 2020, Robin, 2011). The confusion matrix gives the predicted labels on the left and correct labels on the top. So the confusion matrix also shows and organizes the TP, FP, TN, and FN values, where TP=number of correctly classified positives, FP=number of samples incorrectly classified as positive, TN=number of correctly classified negatives, and FN=number of samples incorrectly classified as negative. Accuracy tells us how well our model correctly classifies the data, with the formula

$$AC = (TP + TN)/(TP + FP + TN + FN).$$

The no information rate simply tells us that if we classified all values as 0 what our accuracy rate would be. This is particularly useful in an unbalanced data set like this one, because it gives us a lower bound on how well our model should do.

Sensitivity is the true positive rate and specificity is 1-false positive rate (Kantardzic, 2020). The balanced accuracy is (specificity+sensitivity)/2 and gives a more accurate representation of accuracy in unbalanced data sets.

The F1 measure is

$$F1 = 2(precision \times recall)/(precision + recall)$$

where

$$precision = TP/(TP + FP) \text{ and } recall = TP/(TP + FN).$$

Precision tells us how good the model is at labeling the points correctly classify true positives, and recall tells us how good the model is at catching all positive cases. The F1 score tells gives us a balanced representation of precision and recall (Koehrsen, 2018).

The receiver operating characteristic curve (ROC) and area under the curve (AUC) are related as AUC is the area percentage under the curve. The ROC curve plots sensitivity against 1-specificity and assesses the classification model's performance at different threshold values, and we are able to analyze both characteristics at the same time (Kantardzic, 2020).

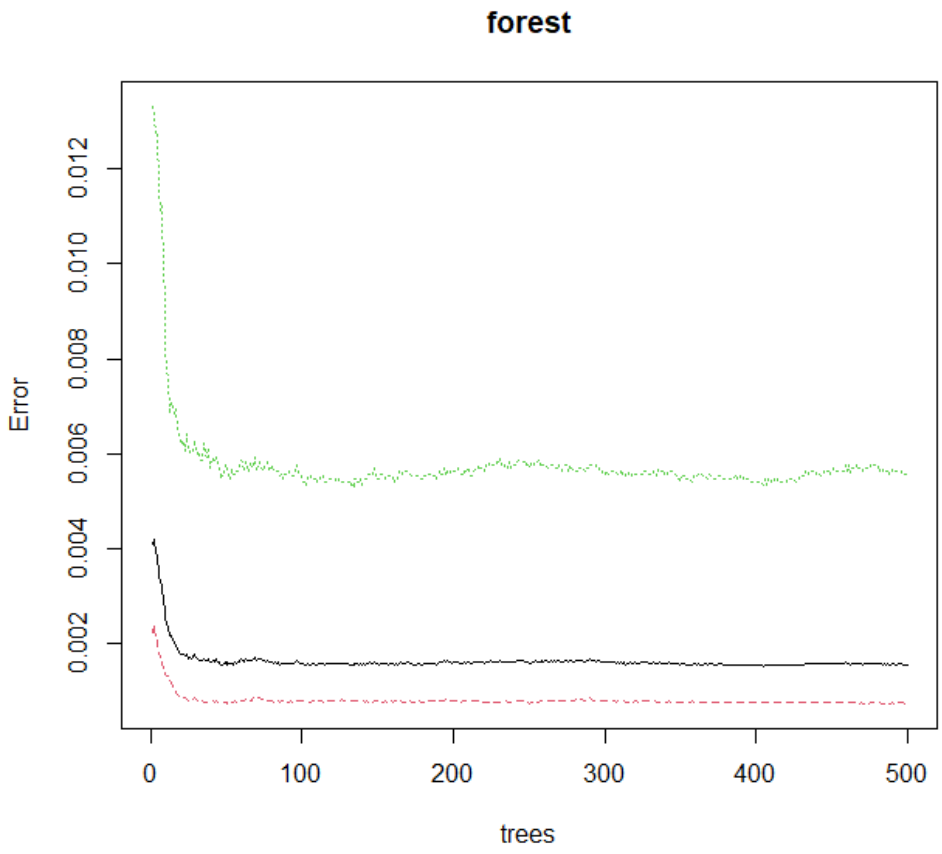
Random Forest

The first algorithm we implemented was a random forest. Random forests are an ensemble method which constructs multiple trees based on subsets of the given data that each "vote" on the classification output. We used the `randomForest` package to create our model, it has multiple parameters which are all listed in the R documentation (Liaw, Wiener, 2002). Some of the most notable parameters are the formula to follow, which just tells R what features are independent and dependent variables, and the training data must also be specified. Other parameters this algorithm takes are the number of trees to create; a cutoff value which gives the algorithm a threshold for voting a certain class; whether sampling should be done with replacement; and there are further parameters for the trees, like defining the number of nodes on the trees. For our purposes, we simply used the default parameters of 500 trees, no cutoff, with replacement, and no maximum number of nodes. We only specified the data and formula.

Random forest outputs a confusion matrix of the training data with the class error. We see that the rate of misclassification of ddos attacks is higher than that of benign attacks on the training data.

	Benign	ddos	class error
Benign	83092	61	0.0007335875
ddos	94	16753	0.0055796284

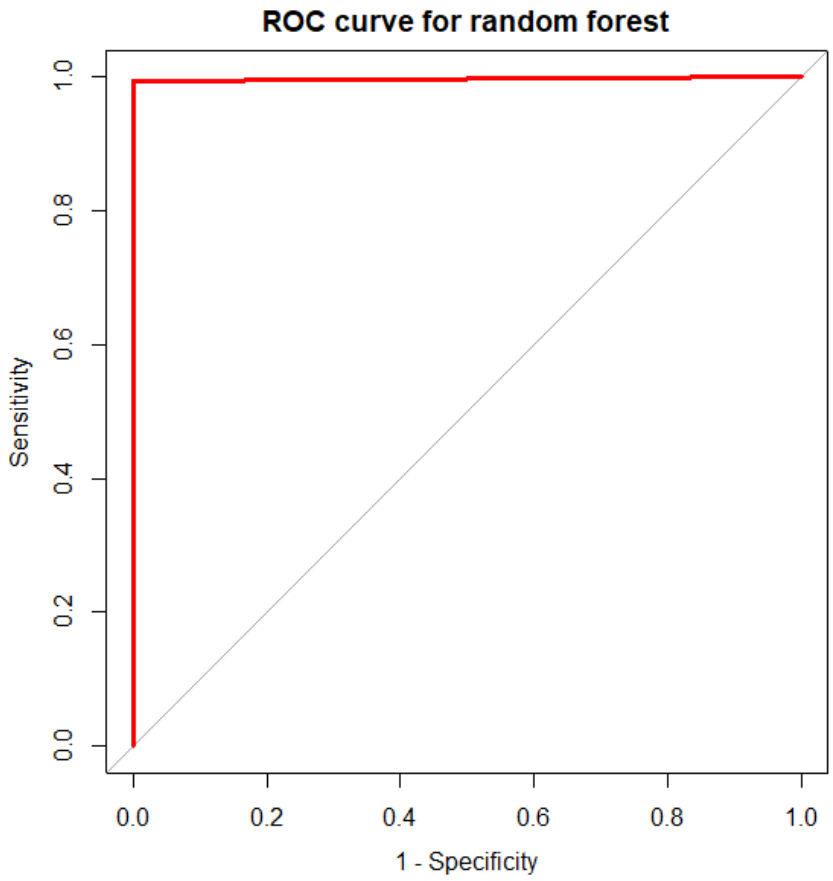
The random forest outputs a graph of the error (mean squared error) and number of trees. The black line shows the overall error rate, and the green and red lines represent the error rates of ddos and benign, respectively. We see that the error declines sharply from one to fifty trees, and then the error rate stays about the same from 50 to 100 trees. This tells us that we could set our number of trees parameter to any number greater than 50 and have model with a low error rate.



Then we used the model to make predictions on the test set. We output the confusion matrix, summary of accuracy measures, and a ROC curve. The tables and ROC curve for the random forest are given below. We see that all values are around 99%, which tells us that our model performs overall quite well. The shape of the ROC curve also tells us that at almost any threshold the model performs well.

		Reference	
		0	1
Prediction	0	544793	651
	1	333	111654

Accuracy	0.9985
No Information Rate	0.8292
Sensitivity	0.9994
Specificity	0.9942
Balanced Accuracy	0.9968
AUC	0.9968
F1	0.9990977



Logistic Regression

The second algorithm we used was logistic regression using the `nnet` package (Venables and Ripley, 2002). Logisic regression is a supervised learning algorithm that estimates the probability that an observation has the label 0 or 1 (Kantardzic, 2020). The model itself is a logit equation, and the goal of the algorithm is to find the best coefficients using linear regression. The model itself would be a logistic equation with 20 explanatory variables with the given coefficients and an intercept. A portion of the logit equation, and complete table of the coefficients is given below.

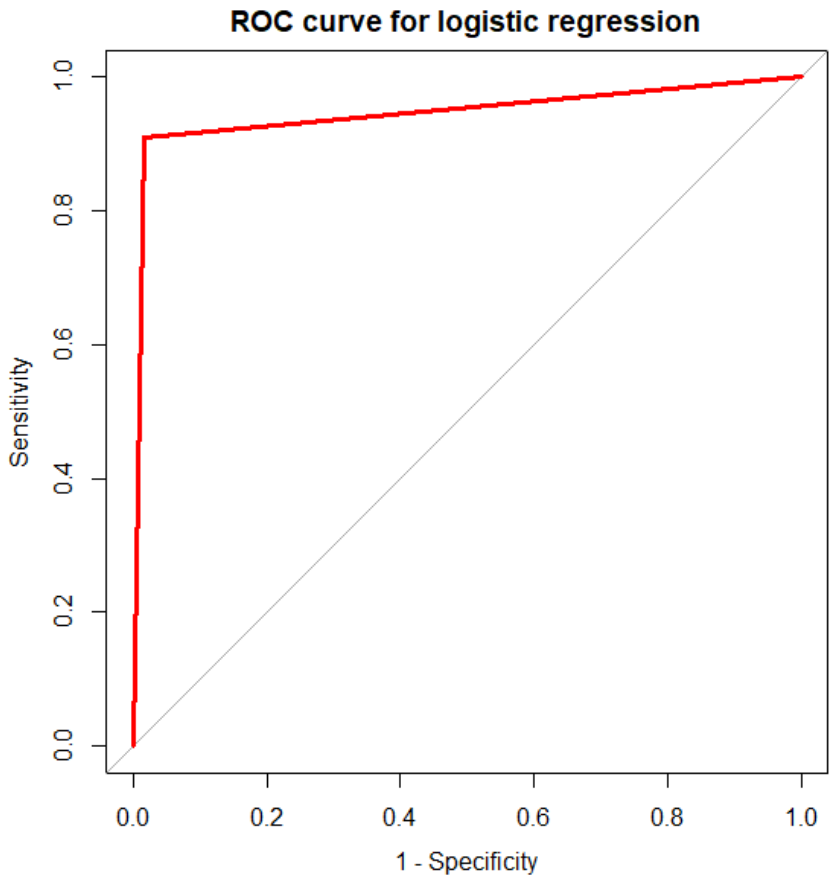
$$\ln\left(\frac{p}{1-p}\right) = -26.6 + 0.5x_1 + 0.3x_2 + 1.8x_3 + 11.3x_4 + 0.6x_5 + 0.2x_6 - 0.8x_7 + 1.4x_8 - 0.3x_9 + ... + 13.6x_{20}$$

Coefficients:	
	Values
(Intercept)	-26.6531432
PC1	0.4813795
PC2	0.3321207
PC3	1.7655373
PC4	11.3221316
PC5	0.5773341
PC6	0.1623438
PC7	-0.8236036
PC8	1.3809639
PC9	-0.2558553
PC10	4.7087423
PC11	-3.8840765
PC13	2.4532421
PC14	-3.5863263
PC15	-1.5461308
PC16	-5.5382115
PC17	1.4693123
PC18	-1.1066039
ACK.Flag.Cnt	2.9601895
CWE.Flag.Count	13.5802046

We then used the model to classify the entries in the testing data set. The confusion matrix, general statistics, and ROC curve are given below. We can see immediately from the ROC curve and the table that the logistic regression model struggles with specificity. The ROC curve also tells us that the model does a better job of catching true positives than true negatives. That is, the model did not do as well at classifying true negatives, but is good at catching positive cases, so if we were more concerned with catching ddos attacks than we were with preventing legitimate web traffic, this model would be sufficient.The rest of the values were 98% and above, so the model would be fine to use as long as we are not too concerned with misclassifying benign web traffic as a ddos attack. The F1 score was surprisingly high when compared with all of the other error measures, which tells us that overall the logistic regression model is not a terribly inaccurate model.

		Reference	
		0	1
Prediction	0	536358	10209
	1	8768	102096

Accuracy	0.9711
No Information Rate	0.8292
Sensitivity	0.9839
Specificity	0.9091
Balanced Accuracy	0.9465
AUC	0.9465
F1	0.9826169



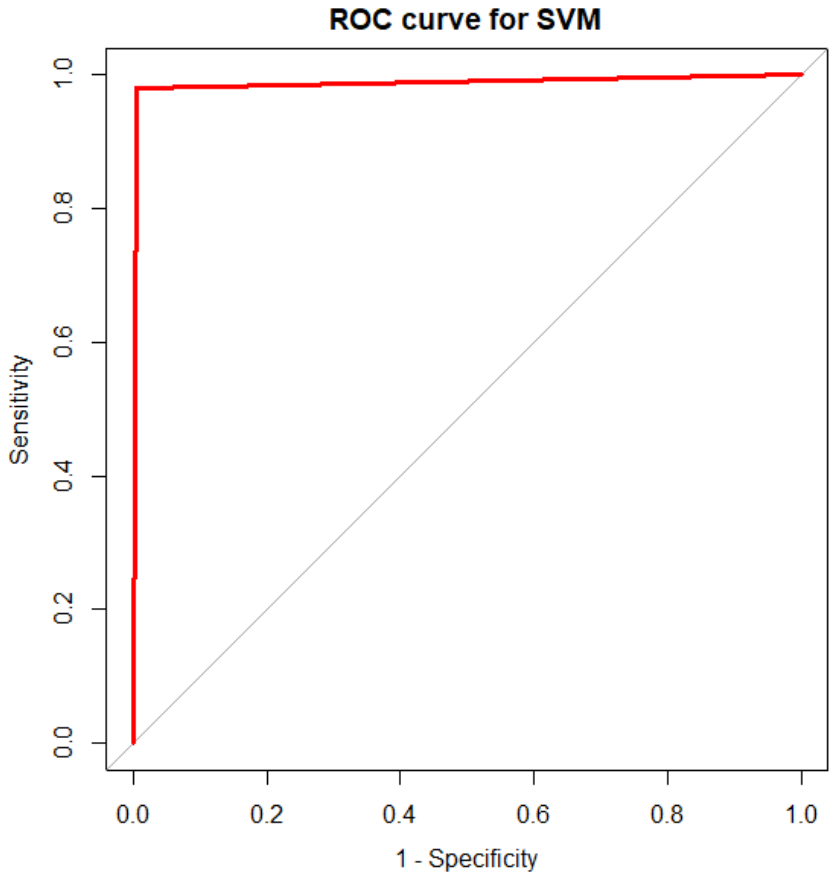
SVM

The third algorithm we used was support vector machine (SVM), which is a supervised learning algorithm which creates a decision boundary between the classes. We wil use the `svm` function from the `e1071` package (Meyer et al., 2020). The algorithm seeks to set the decision boundary as far as possible from both classes simultaneously. Out of the three algorithms used, SVM was the most computationally expensive. Similar to the previous two algorithms, we needed to input a formula and data set to the function in R. There were additional adjustable parameters, like scaling (which we already did during PCA), or defining a kernel function, but we found that the defaults worked well. The defaults for our model can be called with the summary function. So we see that the model is a classification model, with a radial SVM kernel. The final model used 2,518 support vectors. Support vectors are the data points that lie closest to the decision boundary, and are the hardest to classify. Support vectors are then used to define the decision boundary, and are included in the solution, i.e. defined hyperplane that divides the classes (Kantardzic, 2020).

Like the other two models, we tested the model on the testing data set and summarized the results in a confusion matrix, a table of relevant statistics, and a ROC curve given below. The SVM had a high accuracy, and relatively high balanced accuracy, and the sensitivity and specificity tell us that the model is good at catching ddos attacks, and also relatively good at not labeling benign attacks as ddos. The ROC curve gives us a visualization of the balanced sensitivity and specificity. The F1 score of 99.5% tells us that it is a good classifier from what we can tell based on the testing data.

		Reference	
		0	1
Prediction	0	541959	2111
	1	3167	110194

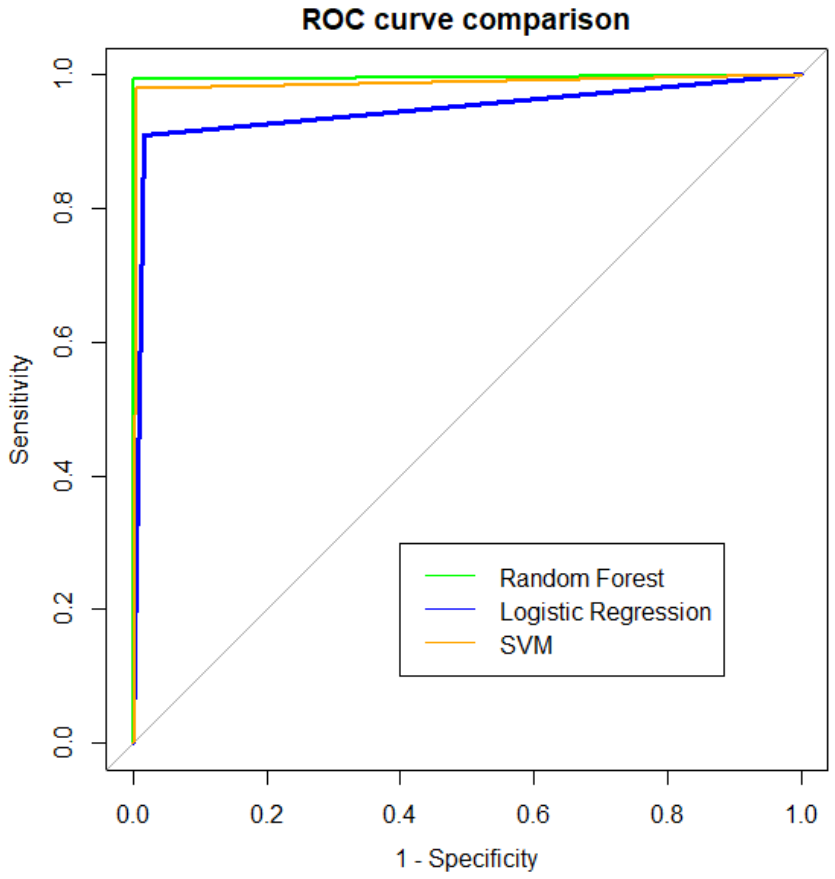
Accuracy	0.992
No Information Rate	0.8292
Sensitivity	0.9942
Specificity	0.9812
Balanced Accuracy	0.9877
AUC	0.9877
F1	0.9951542



Model comparison

Now that we have looked at each model individually, we would like to compare the three against each other. To do this we created a summary table for all three models which includes accuracy, no information rate, sensitivity, specificity, balanced accuracy, AUC and F1 score. We also included a plot of all three ROC curves in the same place for easy comparison. On every measure, there is a clear winner. The model created through the random forest algorithm out performs the models created by logistic regression and SVM. SVM came in a close second, whereas logistic regression performed the worst.

	Random Forest	Logistic Regression	SVM
Accuracy	0.9985	0.9711	0.992
No Information Rate	0.8292	0.8292	0.8292
Sensitivity	0.9994	0.9839	0.9942
Specificity	0.9942	0.9091	0.9812
Balanced Accuracy	0.9968	0.9465	0.9877
AUC	0.9968	0.9465	0.9877
F1	0.9990977	0.9826169	0.9951542



In the ROC curve compilation we see that the green line for random forest was above, SVM (orange), and logistic regression (blue). This indicates that over any given threshold the model created by the random forest algorithm is the best classifier.

Out of the three algorithms, random forests are the most flexible since they are based on decision trees, and was the only one that did not require coercing binary factors into numeric variables. This could have contributed to its overall better performance. SVM was competitive in its results, but at a time consumption trade off - out of the three algorithms, SVM took the longest to run. Logistic regression did not do as well as the other two algorithms, but it had the least amount of flexibility since we based it on linear regression, rather than choosing a polynomial kernel which may have been more appropriate and given better results. However, logistic regression was the fastest algorithm, and gave a model output in seconds rather than several minutes.

It is clear from the summarized statistics and compared ROC curves that random forest offers the best results based on every measure we looked at, despite its computational complexity, so we will use it in the competition aspect of the project on the unlabeled test data set.

3.6 Results

In section 3.5 we used three different data mining algorithms: random forest, logistic regression, and SVM, to create three different classification models. We used a test set to test the models and help evaluate their performance. To quantify and visualize the performance of each model we summarized the results in a confusion matrix; a ROC curve; a table including the accuracy, sensitivity, specificity, balanced accuracy, AUC, F1, and compared these values to the no information rate. All models performed better than simply labeling every new observation as "benign". By every presented measure, we noted that the random forest model performed best, the SVM model was a close second, and logistic regression performed the worst. We do acknowledge that our accuracy and other evaluation rates were exceptionally high, and this could be a sign of overfitting. However, the large test set and variety of error measures should give a generally accurate picture of how the models would perform on unseen data from the same distribution.

As we discussed, the random forest was the most flexible in that it was the only algorithm that did not require coercing binary factors into numeric variables. Since the random forest algorithm is based on a variety of trees created with subsets of the training data it has a large variety of models which each give a "vote" as to which class it thinks the sample belongs in. The support vector machine and logistic regression models are not ensemble methods so the classification "vote" is based on one model, so overall the random forest had more strengths, and it is not surprising that it performed well. While the SVM method produced a good classifier, it was a more computationally expensive than the random forest, which was its greatest weakness. Logistic regression was the fastest algorithm, but provided the least accurate results, and overall performance was acceptable, but had a particularly poor specificity rate. The biggest drawback of using `randomForest` in R was that there was not enough memory available for the algorithm to run on a data set larger than 100,000 values. So while the other algorithms could handle larger data sets, our model building was limited by `randomForest`. We decided it was best to use the same training and test sets for all three algorithms in order to provide the most controlled comparison. With a data set of this size, it would have been ideal to perform cross validation for model selection. This way we would be able to make better use of the entire data set. This would be an area for improvement in future work.

We also made a note that our training set was significantly smaller than our testing set, which is unusual in practice. We did subset the testing set into a smaller set of 100,000 samples to see if there was a significant difference, and found that (as we suspected might be the case) all measures showed that the models were more accurate an performed better. This could be for several reasons. One, the difference could simply be due to chance, and performing a hypothesis test or permutation test would shed some light on if this was the case. Second, the entire given data set was quite unbalanced with the vast majority of samples being labeled "benign". Thus a smaller test set would likely reflect this with a small amount of "ddos" entries making the misclassification error appear smaller, and accuracy appear higher. So we believe the larger test set should give a better measure of the true accuracy of our classification models.

In preliminary attempts at model building we tested the `neuralnet` package in R, since our research papers claimed it to be an excellent algorithm for classification of web traffic (Fritsch et. al., 2019). Unfortunately, it was so computationally expensive, that it would not run on data sets larger than 10,000 samples, and we thought it was better to train other algorithms on a larger training set. In this case, an ensemble method with neural nets, similar to the random forest, might have circumvented the memory issue. Meaning, we could have subsetted the training data into smaller sets, and trained multiple neural nets on the smaller sets and averaged their results for classification. However, this type of programming is beyond my skill level, so while it might be a good idea in theory, I could not execute it. In the future, this would be something worth revisiting.

We saw in the previous section that using 18 principal components produced relatively good results. However we noted in section 3.4 that a clear elbow appeared in the scree plot for 8 principal components. We did decide to run the algorithms again with only 8 principal components, and found that for random forest and SVM there was only a slight difference in accuracy, but a larger difference in accuracy for logistic regression. We did not include these findings in section 3.5 because we would not use these results for model selection. For each algorithm there may be an optimal number of principal components, which balances dimensionality and model accuracy. Thus, fine-tuning the principal components would be another area for improvement.

Finally, we want to note some issues brought up in the preprocessing phase. First, two variables, Flow.Pkts.s and Flow.Byts.s., were deemed important by our decision tree that we considered in feature selection. However, both contained infinite values, so we ended up removing the features from our training data entirely. Ideally we would have found some direction on working with the infinite values so these two numeric features could have been included in the PCA. Second, the use of PCA was not entirely justified by the theory, i.e. we did not show that normality was present in the data, we would hope that the size of the data would compensate, but there are no guaruntees. Thus, we should have either used PCA on features that followed a multivariate normal distribution, or found a different way of selecting numerical variables, or used a different method of reducing dimensionality.

Overall, there are significant areas for improvement in all aspects of model creation. We had some areas in feature selection and dimensionality reduction that could have been better executed. Improvements on the software packages and hardware and coding skills would have also allowed for better model building techniques, including making better use of the entire given data set either through combining algorithms or cross validation techniques. Aside from issues with model building, our model evaluation was executed well and looked at different areas of evaluation. We are interested to see if out evaluation measures are reflected in the results of the competition test set.

4. Appendix

Table of features

Feature	Description	Type
"X"	Not sure	Numeric
"Flow.ID"	Not sure	character
"Src.IP"	source internet protocol address	character
"Src.Port"	source port	Numeric
"Dst.IP"	destination internet protocol address	character
"Dst.Port"	destination port	Numeric
"Protocol"	protocol 3 levels	factor
"Timestamp"	date and time of observation	Time
"Flow.Duration"	flow duration	Numeric
"Tot.Fwd.Pkts"	Total packets in the forward direction	Numeric
"Tot.Bwd.Pkts"	Total packets in the backward direction	Numeric
"TotLen.Fwd.Pkts"	Total size of packet in forward direction	Numeric
"TotLen.Bwd.Pkts"	Total size of packet in backward direction	Numeric
"Fwd.Pkt.Len.Max"	Maximum size of packet in forward direction	Numeric
"Fwd.Pkt.Len.Min"	Minimum size of packet in forward direction	Numeric
"Fwd.Pkt.Len.Mean"	Average size of packet in forward direction	Numeric
"Fwd.Pkt.Len.Std"	Standard deviation size of packet in forward direction	Numeric
"Bwd.Pkt.Len.Max"	Maximum size of packet in backward direction	Numeric
"Bwd.Pkt.Len.Min"	Minimum size of packet in backward direction	Numeric
"Bwd.Pkt.Len.Mean"	Mean size of packet in backward direction	Numeric
"Bwd.Pkt.Len.Std"	Standard deviation size of packet in backward direction	Numeric
"Flow.Byts.s"	Flow byte rate that is number of packets transferred per second	Numeric
"Flow.Pkts.s"	flow packets rate that is number of packets transferred per second	Numeric
"Flow.IAT.Mean"	Average time between two flows	Numeric
"Flow.IAT.Std"	Standard deviation time two flows	Numeric
"Flow.IAT.Max"	Maximum time between two flows	Numeric
"Flow.IAT.Min"	Minimum time between two flows	Numeric
"Fwd.IAT.Tot"	Total time between two packets sent in the forward direction	Numeric
"Fwd.IAT.Mean"	Mean time between two packets sent in the forward direction	Numeric
"Fwd.IAT.Std"	Standard deviation time between two packets sent in the forward direction	Numeric
"Fwd.IAT.Max"	Maximum time between two packets sent in the forward direction	Numeric
"Fwd.IAT.Min"	Minimum time between two packets sent in the forward direction	Numeric
"Bwd.IAT.Tot"	Total time between two packets sent in the backward direction	Numeric
"Bwd.IAT.Mean"	Mean time between two packets sent in the backward direction	Numeric
"Bwd.IAT.Std"	Standard deviation time between two packets sent in the backward direction	Numeric
"Bwd.IAT.Max"	Maximum time between two packets sent in the backward direction	Numeric
"Bwd.IAT.Min"	Minimum time between two packets sent in the backward direction	Numeric
"Fwd.PSH.Flags"	# of times PSH flag was set in packets in fwd dir(0 for UDP)	Binary Factor
"Bwd.PSH.Flags"	# of times PSH flag was set in packets in bkw dir (0 for UDP)	Binary Factor
"Fwd.Header.Len"	Total bytes used for headers in the forward direction	Numeric
"Bwd.Header.Len"	Total bytes used for headers in the backward direction	Numeric
"Fwd.Pkts.s"	Number of forward packets per second	Numeric
"Bwd.Pkts.s"	Number of backward packets per second	Numeric
"Pkt.Len.Min"	Minimum length of a flow	Numeric
"Pkt.Len.Max"	Maximum length of a flow	Numeric
"Pkt.Len.Mean"	Mean length of a flow	Numeric
"Pkt.Len.Std"	Standard deviation length of a flow	Numeric
"Pkt.Len.Var"	Minimum inter-arrival time of packet	Numeric

Feature	Description	Type
"FIN.Flag.Cnt"	# of packets with FIN	Binary Factor
"SYN.Flag.Cnt"	# of packets with SYN	Binary Factor
"RST.Flag.Cnt"	# of packets with RST	Binary Factor
"PSH.Flag.Cnt"	# of packets with PUSH	Binary Factor
"ACK.Flag.Cnt"	# of packets with ACK	Binary Factor
"URG.Flag.Cnt"	# of packets with URG	Binary Factor
"CWE.Flag.Count"	# of packets with CWE	Binary Factor
"ECE.Flag.Cnt"	# of packets with ECE	Binary Factor
"Down.Up.Ratio"	Download and upload ratio	Numeric
"Pkt.Size.Avg"	Average size of packet	Numeric
"Fwd.Seg.Size.Avg"	Average size observed in the forward direction	Numeric
"Bwd.Seg.Size.Avg"	Average size observed in the backward direction	Numeric
"Subflow.Fwd.Pkts"	ave # of packets in a sub flow in the forward direction	Numeric
"Subflow.Fwd.Byts"	ave # of bytes in a sub flow in the forward direction	Numeric
"Subflow.Bwd.Pkts"	ave # of packets in a sub flow in the backward direction	Numeric
"Subflow.Bwd.Byts"	ave # of bytes in a sub flow in the backward direction	Numeric
"Init.Fwd.Win.Byts"	# of bytes sent in initial window in the forward direction	Numeric
"Init.Bwd.Win.Byts"	# of bytes sent in initial window in the backward direction	Numeric
"Fwd.Act.Data.Pkts"	# of packets with at least 1 byte of TCP data payload in the forward direction	Numeric
"Fwd.Seg.Size.Min"	Minimum segment size observed in the forward direction	Numeric
"Active.Mean"	Mean time a flow was active before becoming idle	Numeric
"Active.Std"	Standard deviation time a flow was active before becoming idle	Numeric
"Active.Max"	Maximum time a flow was active before becoming idle	Numeric
"Active.Min"	Minimum time a flow was active before becoming idle	Numeric
"Idle.Mean"	Mean time a flow was idle before becoming active	Numeric
"Idle.Std"	Standard deviation time a flow was idle before becoming active	Numeric
"Idle.Max"	Maximum time a flow was idle before becoming active	Numeric
"Idle.Min"	Minimum time a flow was idle before becoming active	Numeric
"Label"	ddos=1 or Benign=0	factor

References

A. Liaw and M. Wiener (2002). *Classification and Regression by randomForest.* R News 2(3), 18–22.

Alboukadel Kassambara and Fabian Mundt (2020). *factoextra: Extract and Visualize the Results of Multivariate Data Analyses.* R package version 1.0.7. https://CRAN.R-project.org/package=factoextra

Alkasassbeh, M., Hassanat, Al-Naymat, and Almseidin, M. (2016). "Detecting Distributed Denial of Service Attacks Using Data Mining Techniques" *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 1.

CISA. (2009, Nov 4)."Security Tip (ST04-015) Understanding Denial-of-Service Attacks." *Cybersecurity and Infrastructure Security Agency.* zttps://us-cert.cisa.gov/ncas/tips/ST04-015.

David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel and Friedrich Leisch (2020). *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071)*, TU Wien. R package version 1.7-4. https://CRAN.R-project.org/package=e1071

Garg, K. and Chawla, R. (2011). "Detection of DDOS Attacks Using Data Mining." *International Journal of Computing and Business Research.* Vol 2, Issue 1.

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2020). *dplyr: A Grammar of Data Manipulation.* R package version 1.0.2. https://CRAN.R-project.org/package=dplyr

Koehrsen, W. (2018, Mar 3). "Beyond Accuracy: Precision and Recall". *Towards Data Science* https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

Lopes, M. (2017, Jan 9). “Dimensionality Reduction - Does PCA Really Improve Classification Outcome?” *Towards Data Science.* https://towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32

M. Monshizadeh and Z. Yan, “Security Related Data Mining,” *2014 IEEE International Conference on Computer and Information Technology*, Xi’an, 2014, pp. 775-782, doi: 10.1109/CIT.2014.130.

Max Kuhn (2020). *caret: Classification and Regression Training.* R package version 6.0-86. https://CRAN.R-project.org/package=caret

Molloy, D. and Tidy, J. (2020, June 1). “George Floyd: Anonymous hackers re-emerge amid US unrest”. *BBC News.* https://www.bbc.com/news/technology-52879000.

Shao, J. (1993). "Linear model selection by cross-validationa'. *Journal of the American Statistical Association*, vol 88, 486-494.

Stefan Fritsch, Frauke Guenther and Marvin N. Wright (2019). *neuralnet: Training of Neural Networks.* R package version 1.44.2. https://CRAN.R-project.org/package=neuralnet

Terry Therneau and Beth Atkinson (2019). *rpart: Recursive Partitioning and Regression Trees.* R package version 4.1-15. https://CRAN.R-project.org/package=rpart

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

Xavier Robin, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez and Markus Müller (2011). *pROC: an open-source package for R and S+ to analyze and compare ROC curves.* BMC Bioinformatics, 12, p. 77. DOI: 10.1186/1471-2105-12-77 http://www.biomedcentral.com/1471-2105/12/77/

Zhong, R. and Yue, G. (2010, April). “DDoS Detection System Based on Data Mining.” *Proceedings of the Second International Symposium on Networking and Network Security.*