Machine Learning for Cyber Security: Intrusion Detection

Springboard DSC:
Capstone Project 2- Final Report

Matthew Wong
June 2019

Section 1: Introduction

Overview:

Cyber-attacks are increasing as more and more data is being processed throughout industries. Many companies are being targeted by detrimental attacks especially in critical business sectors such as banking. It is critical to find solutions to prevent these attacks before any loss can occur. With new improvements in machine learning and data science, data security is becoming a key issue to resolve. Data scientists can apply their knowledge to the cybersecurity field to help protect attacks and identify suspicious behavior. The goal is to identify threats, stop intrusions and attacks, properly identify malware, spam, and prevent fraud. A wide range of problems that fit a data science problem.

For this project, I will be analyzing and using several machine learning techniques that can later become a component of an intrusion detection system. I will dive into supervised learning approaches in order to make predictions on the classification of anomaly data, which will include deep neural networks.

Client: Online company looking for a data science consultant to assist them with their web security problems.

Goal: The main goal is to help build a system for intrusion detection

Dataset:

For this project we are going to utilize the UNSW-NB 15 dataset created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviors. This dataset bridges this gap and covers a variety of modern attacks.

The dataset can be found here:
https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/

Results

- The Random Forest Classifier model with hyper-parameter tuning is the best classifier. It holds a recall score of **100%.** This indicates the model was able to correctly identify all attacks. The model also yields a precision of 75%, an F1-Score of 85%, and an AUC score of 96%.

Section 2: Approach

2.1 Data acquisition and Wrangling

The first step of the data science process is to wrangle the data. Data wrangling is the process of transforming and mapping the data. Before reading the data into my environment, I imported the required packages that will be used to analyze the data. These packages include pandas, numpy, and matplotlib. Pandas is an open source, high-performance library of easy-to-use data structures and data analysis tools for Python. For data wrangling pandas is used in order to fill missing values, correct data types. In summary, this section is designed to create a clean, clear, and presentable dataset.

For this project, the data must be preprocessed into two datasets, a training set and testing set. Using Python and pandas I was able to read in the two datasets into different data frames. Using the info function, I was able to get information on the features, observations, and datatypes of each data frame.

Data Size:

- 49 features
- 175,341 connections in the training dataset
- 82332 connections in the testing dataset

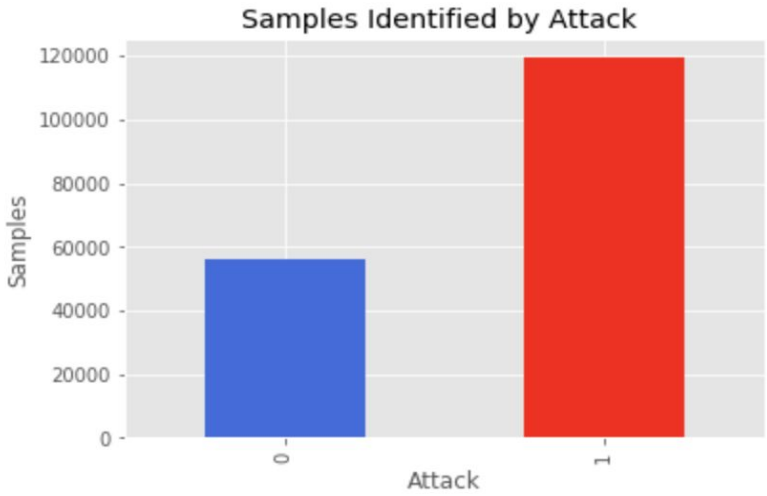
Dataset splitting for learning techniques:

- Splitting a dataset into training and testing sets is essential for the training and validation of its algorithms. A training set is a set of instances for determining the learning classifier parameters, a validation set is a set of instances used to adjust those parameters as much as possible and a testing set is a set of observations for evaluating the classifiers performance.
- The UNSW-NB15 dataset is divided into a roughly 60:40 train/test ratio.

2.2: Exploratory Data Analysis

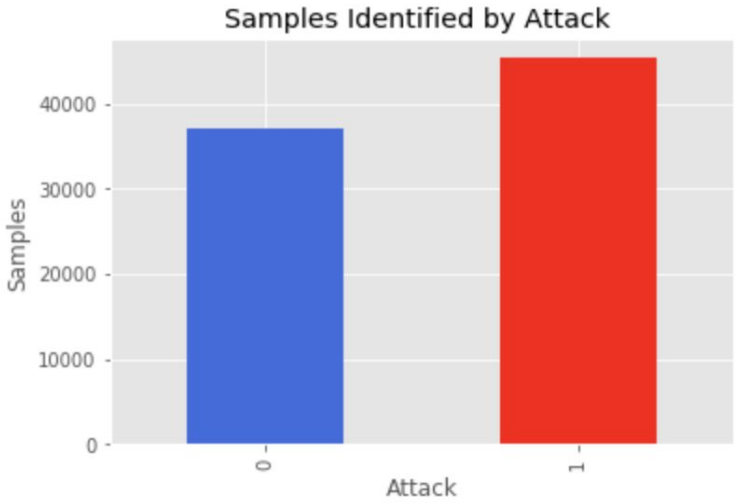
Using Matplotlib library, I checked how much data for each class (normal and anomaly) is contained.

Training set:



- The training set normal connections equals to 32% of the data
- The training set malicious connections equals to 68% of the data

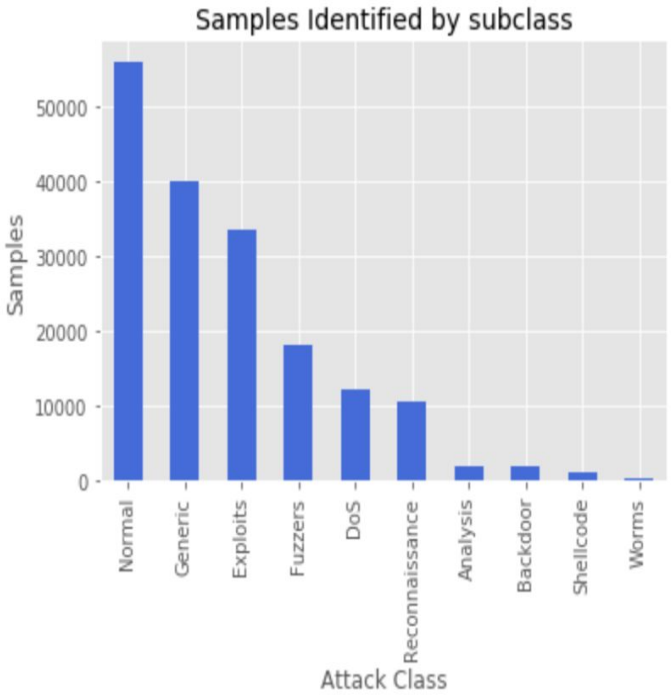
Testing set:



- The testing set normal connections equal to 45% of the data
- The testing set malicious connections equal to 51% of the data

Looking into this distribution further the data set was divided into a 60:40 ratio of training and testing sets. To avoid biasing the decision engine approach and reflecting high FARs (False Alarm Rates), these sets to not contain any redundant observations which ensure the credibility of the evaluations.

Using groupby, I further explored the attack label of the data. There are 9 sub-classes of attacks.

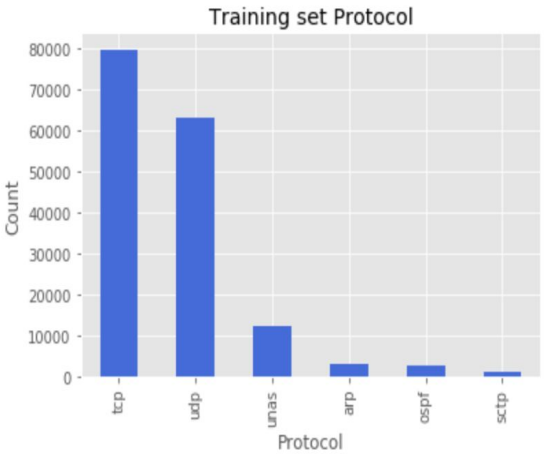


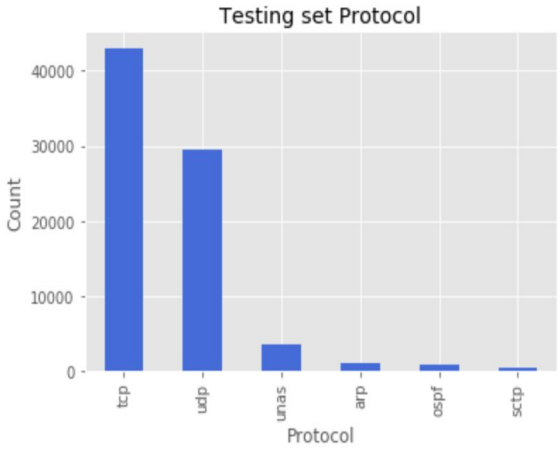
1. Fuzzers
2. Analysis
3. Backdoor
4. DoS
5. Explotis
6. Generic
7. Reconnaissance
8. Shellcode
9. Worms

These attack categories represent the current cyber scenario. It is observed that the distribution of attacks is highly imbalanced which depicts a real-world situation. The majority of the observations are normal followed by generic, exploits and fuzzers attacks.

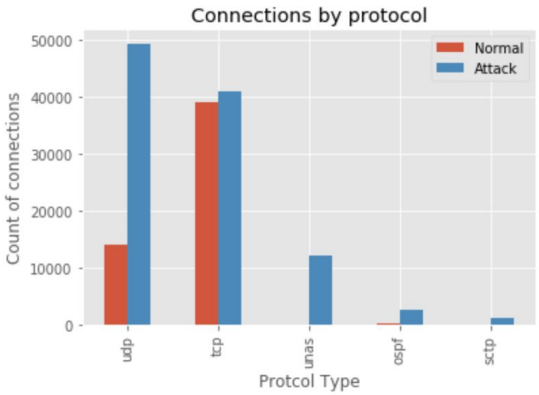
Protocol and attacks

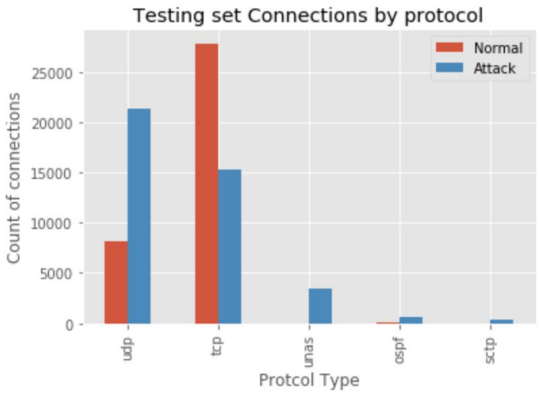
Network protocols are formal standards and policies comprised of rules, procedures and formats that define communication between two or more devices over a network. It governs the end-to-end processes of timely, secure and managed data or network communication.





- Most networks use a TCP or UDP protocol.



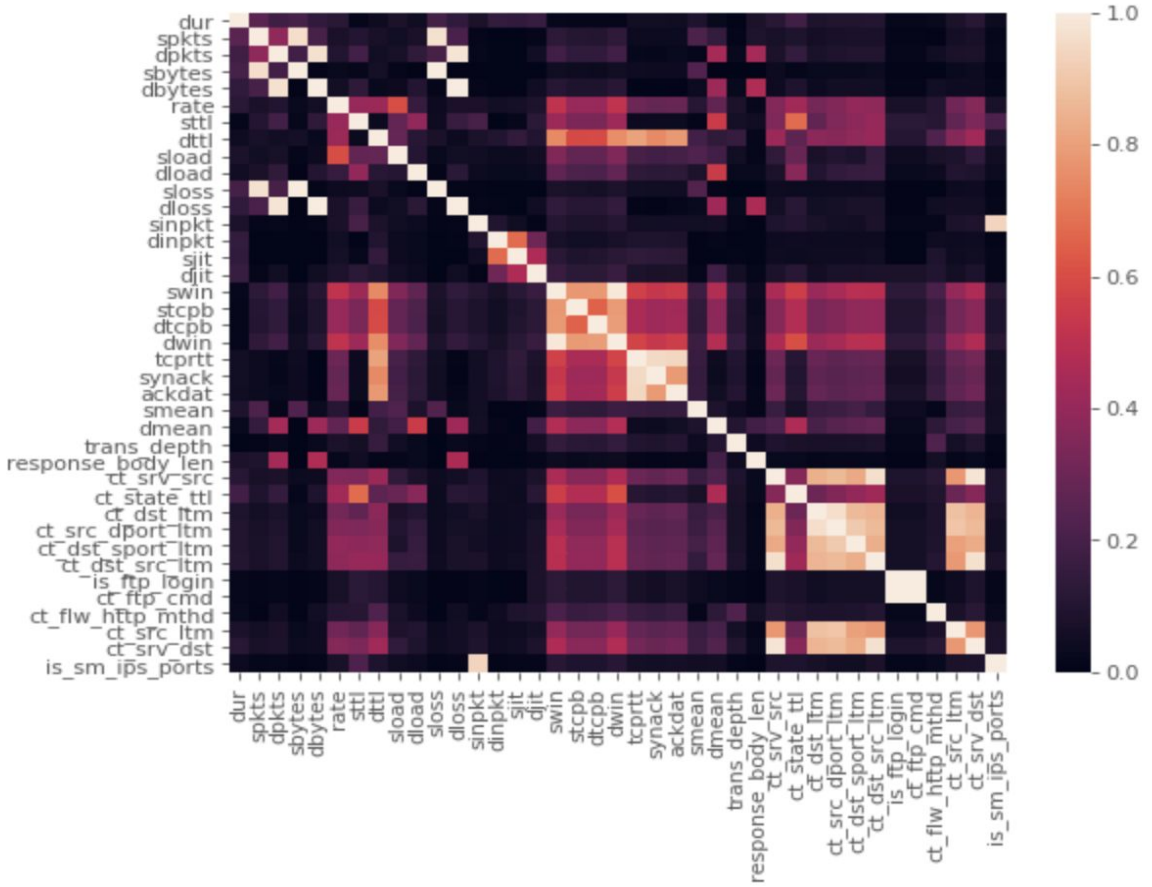


The charts show the top 5 attacks by protocol types. Having most attacks on udp and tcp.

- Percentage of attacks on udp protocols is around 78%
- Percentage of attacks on tcp protocols is around 51%

Feature Correlation

In order to get a better understanding of my features and the correlations between each feature I created a heatmap.



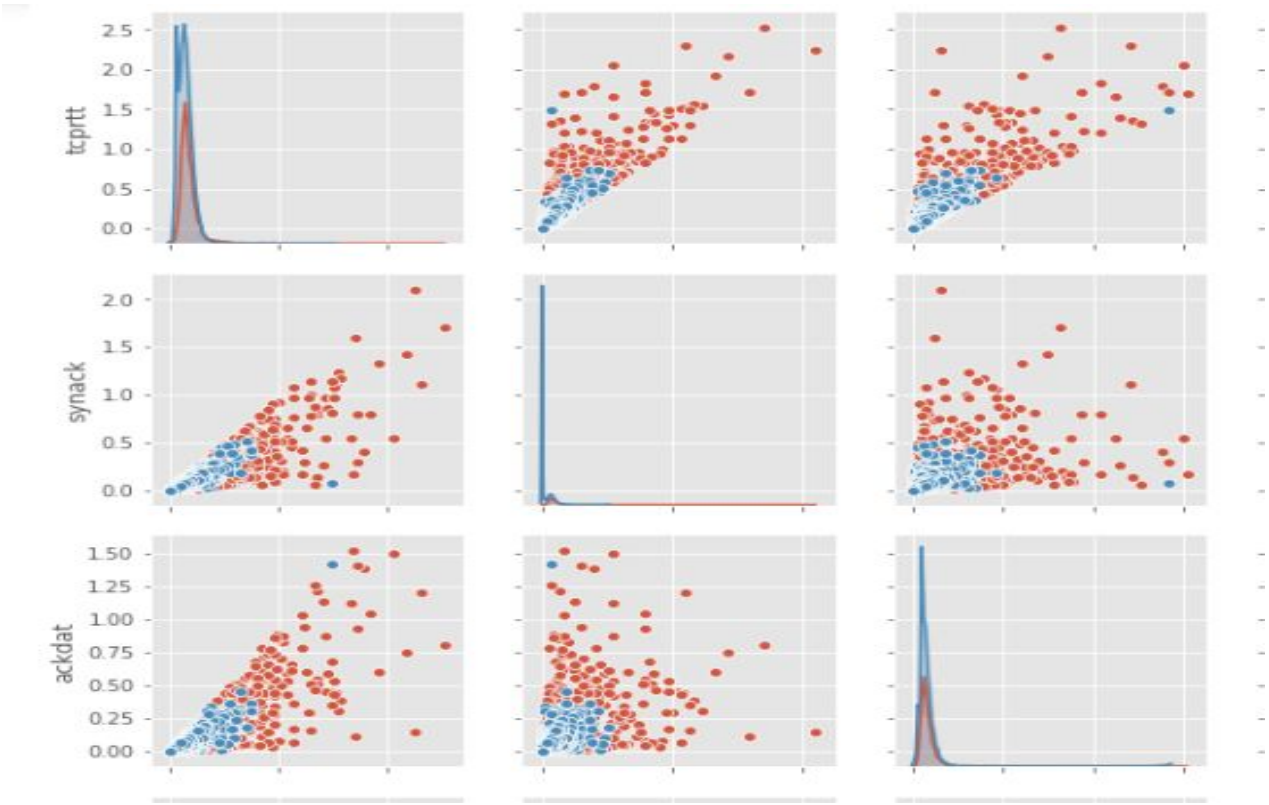
The dataset consists of various network traffic features. Keeping highly correlated features is not ideal for our model. For linear models, like logistic regression, multicollinearity can yield solutions that are widely varying and possibly numerically unstable. The most highly correlated features are

- Sloss and sbytes (actual value of correlation coefficient)
- Dloss and dbytes (actual value of correlation coefficient)
- Is_ftp_login and ct_ftp_cmd (actual value of correlation coefficient)

I removed these features to avoid redundancy. A possibly implementation for later is to use dimensionality reduction techniques to retain only the relevant features.

Pairplots

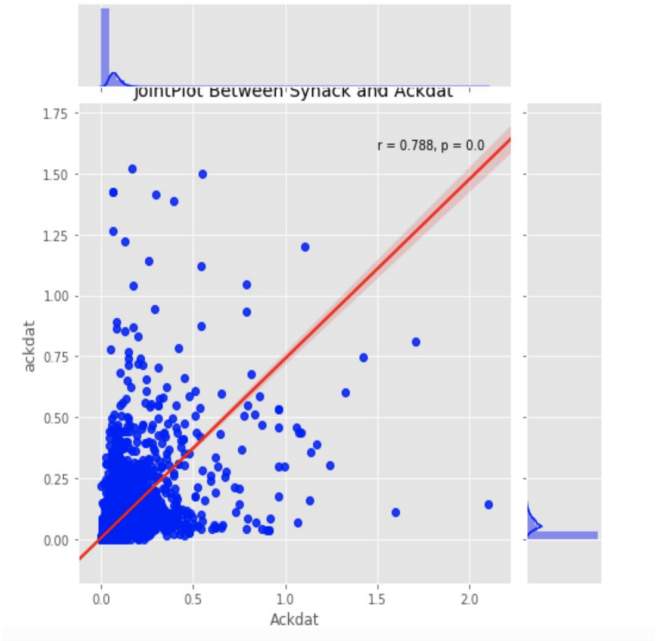
I created pairplots to get a better visual understanding of the relationship between pairs of features. I first sliced the dataset into different sets, for faster processing time and created a pairplot for each set. I then observed each set to determine whether certain features had a positive correlation. Two features in particular had a positive relationship, the features were ‘synack’‘and ‘ackdat’.



Pearson Correlation and p-value

Synack is the TCP connection setup time between the SYN and SYN_ACK packets and Ackdat is the TCP connection setup time between, SYN_ACK and the ACK packets.

A packet is the unit of data that is routed between an origin and a destination on the internet or any other packet-switched network. Network packets are small amounts of data passed over networks. Each packet sent includes information such as the source and destination, the protocols or identification number.



- The correlation coefficient between the two connection times is positive with a score of 0.788. The p-value is close to 0.0, which indicates this relationship is statistically significant.
- From the second chart, that is distributed by label, we can see a clear indication of clusters between attacks and normal. Normal networks are clustered around the 0.0-0.5 synack and 0.0 - 0.5 ackdat connection time.

2.3: Baseline Analysis

Performance Metrics

In order to quantify the performance of the model on training and testing sets, we need a performance metric. However, unlike regression models, classification models have different forms to compute performance metrics. The three metrics we will compute are classification accuracy, precision, recall, and F1-score.

- **Classification Accuracy** is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metrics for classification problems, it is also the most misused. It is really only suitable when there is an equal number of observations in each class and that all predictions and prediction errors are equally important, which is not the case.
- **Classification Report**: Sci-kit learn does provide a convenient report when working on classification problems. This report displays the precision, recall, f1-score and support for each class.
    - **Precision**: the ability of a classifier not to label an instance positive that is actually negative. What percent of your predictions were correct?
    - **Recall** the ability of a classifier to find all positive instances. For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives. What percentage of the positive cases did you catch?
    - **F1 Score**: The weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. What percent of positive predictions were correct?

What to minimize?

- We know that there will be some error associated with every model that we use for predicting the true class of the target variable. This will result in False Positives and False Negatives (model classifying things incorrectly as compared to the actual class) There are no rules that say what should be minimized in situations and depends on the business needs and the context of the problem. Based on these needs we want to minimize either False Positives or False negatives.
    - **False Positives (FP)**: False positives are the cases when the actual class of the data point was 0 (False) and the predicted is 1 (True). False is

because the model has predicted incorrectly and positive because the class predicted was a positive one.
        - A network that is not an attack and the model classifying the network as an attack.
    - **False Negatives (FN)**: False negatives are the cases when the actual class of the data point was 1 (True) and the predicted is 0 (False). False is because the model predicted incorrectly and negative because the class predicted was a negative one (0).
        - A network that is an attack and the model classifying has normal.
- **Minimizing False Negatives**
    - For our network problem minimizing false negatives holds a higher priority than minimizing false positives. If that is the case, in order to capture all attack cases, we might end up making a classification when the network actually does not have an attack. This is less dangerous than the opposite.

Precision and Recall

- **Precision**: is a measure that tells us what proportion of networks we have classified as attacks, actually are attacks. The predicted positives and the network that are attacks are True Positives (TP).
- **Recall and Sensitivity**: Recall is a measure that tells us what proportion of networks actually had an attack that was classified by the algorithm as having attacks. The actual positives (networks with attacks are TP and FN) and the network classified by the model having an attack is TP.
- What to use Precision or Recall?
    - Recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives (how much did we caught)
    - If we focus more on minimizing false negatives, we want our Recall to be as close to 100% as possible without precision being too bad.

Preprocessing

Before implementing algorithms and creating models we first need to preprocess our data. Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Our data is already in a defined data frame however, it is not in a format that a sci-kit machine learning algorithm can read, therefore we need to preprocess our data for modeling.

The first preprocessing step is to create dummy variables from our categorical variables. Machine learning classifiers can only read numeric, hence why we create dummies variables of our categorical columns. The next step was to drop redundant columns. After converting our columns to the correct data types, and dropping the columns so each data frame is of equal size we split the training and testing data frames into X sets and y sets. X sets containing the features and y sets containing our predictors. In total, we have four sets of data, X_train, y_train, X_test, and y_test. Our final step in preprocessing is standardizing our training sets.

Standardizing and normalizing is simply a case of getting all your data on the same scale. If the scales for different features are widely different, this can have a knock-on effect on the ability of the model to learn. Ensuring standardized feature values implicitly weigh all features equally in their representation.

```
#standardize features

sc = StandardScaler()

#fit the scaler to the training data and transform
X_train_std = sc.fit_transform(X_train)

#apply the scaled to the test data
X_test_std = sc.transform(X_test)
```

Baseline model- Logistic Regression

Classification tries to predict which of a small set of classes, an observation belongs to. Our goal is to find y, a label based on knowing a feature of vector x. For this problem, our goal is to predict the label (attack or normal) this is our y-variable.

For our baseline model, we will use a logistic regression model. In short, a Logistic Regression is a Machine Learning algorithm that is used for classification problems, it is a predictive analysis algorithm that is based on the concept of probability. We first import the required sci-kit learn packages and instantiate the logistic regression model. Next, we train on the training sets, then we compute the accuracy of both training sets and testing sets. Finally, we create a classification report.

```
Logistic Regression accuracy training set: 0.9338660096611745
Logistic Regression accuracy testing set: 0.8081183500947384
```

	precision	recall	f1-score	support
0	0.97	0.82	0.89	56000
1	0.92	0.99	0.95	119341
avg / total	0.94	0.93	0.93	175341

	precision	recall	f1-score	support
0	0.95	0.61	0.74	37000
1	0.75	0.97	0.85	45332
avg / total	0.84	0.81	0.80	82332

The results showcase an accuracy score of **93%** for the training data and **81%** for the testing data. For our classification report, we can see the difference between precision and recall on both attack and normal classified data. Our test set indicates a precision of **95%** for classifying normal data, the percentage of the results which are relevant, and a recall of 61% which is the percentage of total relevant results correctly classified. Compared to our attack precision which is **75%** and recall of **97%**.

Depending on the problem you are trying to solve, you could either give a higher priority to maximizing precision or recall. In this case, detecting harmful attacks are our main concern. Another metric that takes into account both precision and recall is the F1-score. This is the harmonic mean of precision and recall. For our baseline model, our F1- Score is **74%** for normal and **85%** for an attack.

Tuning model for better results

Our next goal is to see if we can raise accuracy level, precision, and recall. I used two techniques to compute a higher score. The first technique was to tune my models' hyperparameters. Using a random search, the best parameters computed was an L2 regression, also known as a ridge, and a C value of 0.001. The results did not change, and if anything model accuracy decreased. The second technique was to handle imbalanced data. Using three different techniques to handle imbalanced data (using a weighted parameter on the model, upsampling imbalance data, downsampling imbalance data), I was able to marginally increase the percentage of models accuracy, precision, and recall.

Weighted

```
Logistic Regression accuracy training set: 0.9274556435745205
Logistic Regression accuracy testing set: 0.8306490793373172

              precision    recall  f1-score   support

           0       0.89      0.88      0.89     56000
           1       0.95      0.95      0.95    119341

 avg / total       0.93      0.93      0.93    175341

              precision    recall  f1-score   support

           0       0.89      0.71      0.79     37000
           1       0.80      0.93      0.86     45332

 avg / total       0.84      0.83      0.83     82332
```

Upsampling

```
Logistic Regression accuracy training set: 0.9154607385559028
Logistic Regression accuracy testing set: 0.8261064956517514
```

	precision	recall	f1-score	support
0	0.95	0.88	0.91	119341
1	0.89	0.95	0.92	119341
avg / total	0.92	0.92	0.92	238682

	precision	recall	f1-score	support
0	0.89	0.71	0.79	37000
1	0.80	0.93	0.86	45332
avg / total	0.84	0.83	0.83	82332

Downsampling

```
Logistic Regression accuracy training set: 0.9156071428571428
Logistic Regression accuracy testing set: 0.8303089928581839
```

	precision	recall	f1-score	support
0	0.94	0.88	0.91	56000
1	0.89	0.95	0.92	56000
avg / total	0.92	0.92	0.92	112000

	precision	recall	f1-score	support
0	0.89	0.71	0.79	37000
1	0.80	0.93	0.86	45332
avg / total	0.84	0.83	0.83	82332

From our baseline model and resampling techniques, our best recall score for attacks was **97%**, from our baseline logistic model. Indicating the model successfully predicted 97% label attacks that were actually network attacks. However, our precision is 75% compared to our upsampling and downsampling methods precision's which were both 80%. My next goal is to see if I can increase the precision of attacks while maintaining a high recall percentage.

2.4: Extended Analysis

Ensemble Method- Random Forest

After training the logistic regression models, my next approach is to use other algorithms to see if performance can be improved. Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. We will use two ensemble methods, Random Forest and XGBoost.

- Random Forest model takes advantage of bagging process where it takes different training samples with replacement in order to get predictions for each observation and then averages all the predictions to obtain the estimation.

```
Random Forest accuracy Training set:  0.9948785509378868
Random Forest accuracy Test set:,  0.8659694893844435
```

RF Testing set	precision	recall	f1-score	support
0	0.94	0.75	0.83	37000
1	0.82	0.96	0.89	45332
avg / total	0.88	0.87	0.86	82332

After training both training sets and testing sets our results to show a **99%** accuracy on the training data and 86.5% accuracy on testing data. These are great results, but because of classification, accuracy is not an ideal performance metric. With our classification report, we can see a better distribution of our predicted data. The random forest model yields a precision of **82%** for classifying attacks and **96%** for recalling attacks.

Random Forest - Tuning

We generally see a random forest as a black box which takes in input and gives out predictions. The black box has a few levers we can tune, which affects the performance of the model. Parameters in the random forest are either to increase the predictive power of the model or make it easier to train the model. We will train the parameters of a random forest in order to see if we can optimize the performance. These parameters are n_estimators, max_feattures, max_depth, min_samples_split, min_samples_leaf, and bootstrap.

The method we will use to tune or parameters is a randomized search. Random search is a technique where random combinations of hyperparameters are used to find the best solution for the built model. It tries random combinations of a range of values.

The results are:

```
{'n_estimators': 10,
 'min_samples_split': 5,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 10,
 'bootstrap': True}
```

```
Random Forest accuracy Training set:  0.9349553156420917
Random Forest accuracy Test set:,  0.810268182480688
```

```
Testing Set Classification Report:              precision    recall  f1-s
core   support

          0       1.00      0.58      0.73     37000
          1       0.74      1.00      0.85     45332

avg / total       0.86      0.81      0.80     82332
```

Tuning random forest gave us the worst accuracy compared to the random forest without tuning, however, our precision for classifying normal networks and our recall of attacks are at **100%.**

Ensemble Method- XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.

A quick summary of XGBoost, XGBoost is an ensemble learning method. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The models that the ensemble could be either from the same learning algorithm or different learning algorithms.

The results of using this algorithm are:

```
Random Forest accuracy Training set:  0.9423352210834888
Random Forest accuracy Test set:  0.8309648739250838
```

	precision	recall	f1-score	support
0	0.97	0.85	0.90	56000
1	0.93	0.99	0.96	119341
avg / total	0.94	0.94	0.94	175341

	precision	recall	f1-score	support
0	0.97	0.64	0.77	37000
1	0.77	0.98	0.86	45332
avg / total	0.86	0.83	0.82	82332

This model gives an accuracy score of **83%** and a recall score of **98%**. However, the precision of predicting attack is 77%, slightly higher than precision on random forest with tuning.

XGBoost - resampling

Using resampling techniques on XGBoost, upsampling slightly improves the overall performance of each metric (precision, recall, f1-score) however, it slightly decreased recall at **94%**. Downsampling the data gives us the same performance.

	precision	recall	f1-score	support
0	0.94	0.93	0.94	119341
1	0.93	0.94	0.94	119341
avg / total	0.94	0.94	0.94	238682

	precision	recall	f1-score	support
0	0.91	0.84	0.87	37000
1	0.88	0.94	0.90	45332
avg / total	0.89	0.89	0.89	82332

Deep Neural Network

For this section, I will create a deep neural net using Keras. Deep learning is an increasingly popular subset of machine learning. Deep learning models are built using neural networks. A neural network takes in inputs, which are then processed in hidden layers using weights that are adjusted during the training. Then the model spits out a prediction.

Building the model

The model type we will be using is Sequential. This is the easiest way to build a neural network in Keras. It allows you to build a model layer by layer. Each layer has weights that correspond to the layer that follows it. Using the add function to add layers. Dense is the layer type, the standard layer type that works for most cases. Activation is the activation function of the layer. An activation function allows models to consider nonlinear relationships.

```
#create model
dnn = Sequential()

#get number of columns in training data
n_cols = X_train_std.shape[1]

#add layers to model
dnn.add(Dense(250,activation='relu', input_shape=(n_cols,)))
dnn.add(Dense(250,activation='relu'))
dnn.add(Dense(250,activation='relu'))
dnn.add(Dense(1,activation='sigmoid'))
```

Compiling the model

Compiling the model takes two parameters, optimizer, and loss. The optimizer controls the learning rate. We will use ‘adam’ as our optimizer. The optimizer adjusts the learning rate without training. The learning rate determines how fast the optimal weights for the model is calculated. Smaller may compute more accurate weights however it takes more time. For our loss function, we will use binary cross-entropy.

```
dnn.compile(optimizer = 'adam', loss ='binary_crossentropy',
            metrics=['accuracy'])
```

Training the model

To train our model we will use the fit function, just like for our other machine learning classifiers. However, this function takes in five parameters. Training data, target data, validation split, the number of epochs and callbacks. The validation split will randomly split the data into use for training and testing. The number of epochs is the number of times the model will cycle through the data. The more epochs, the more the model will improve up to a certain point. Early stopping will stop the model from training before the number of epochs is reached.

```
from keras.callbacks import EarlyStopping

#set early stopping monitor so the model stops traing when it won
early_stopping_monitor = EarlyStopping(patience=3)

#train model
dnn_output = dnn.fit(X_train_std, y_train, epochs=30,
        validation_split =0.2,
        callbacks=[early_stopping_monitor])
```

Results:

```
Training Accuracy: 0.9336961658166175
Validation Accuracy:  0.9889849806300232
```

	precision	recall	f1-score	support
0	0.94	0.72	0.81	37000
1	0.81	0.96	0.88	45332
avg / total	0.87	0.85	0.85	82332

Using a deep neural network gave us the best accuracy so far with a **98.8%** accuracy. However, the classification report results while still being good was not an improvement compared to the other models.

Deep Neural Network with resampling

Using the same pipeline as above, however, we will use our upsampled and downsampled data. From computing our performance metrics we can see upsampling data gives us better results than downsampling. Which make sense, the more data we have the better our performance.

Results:

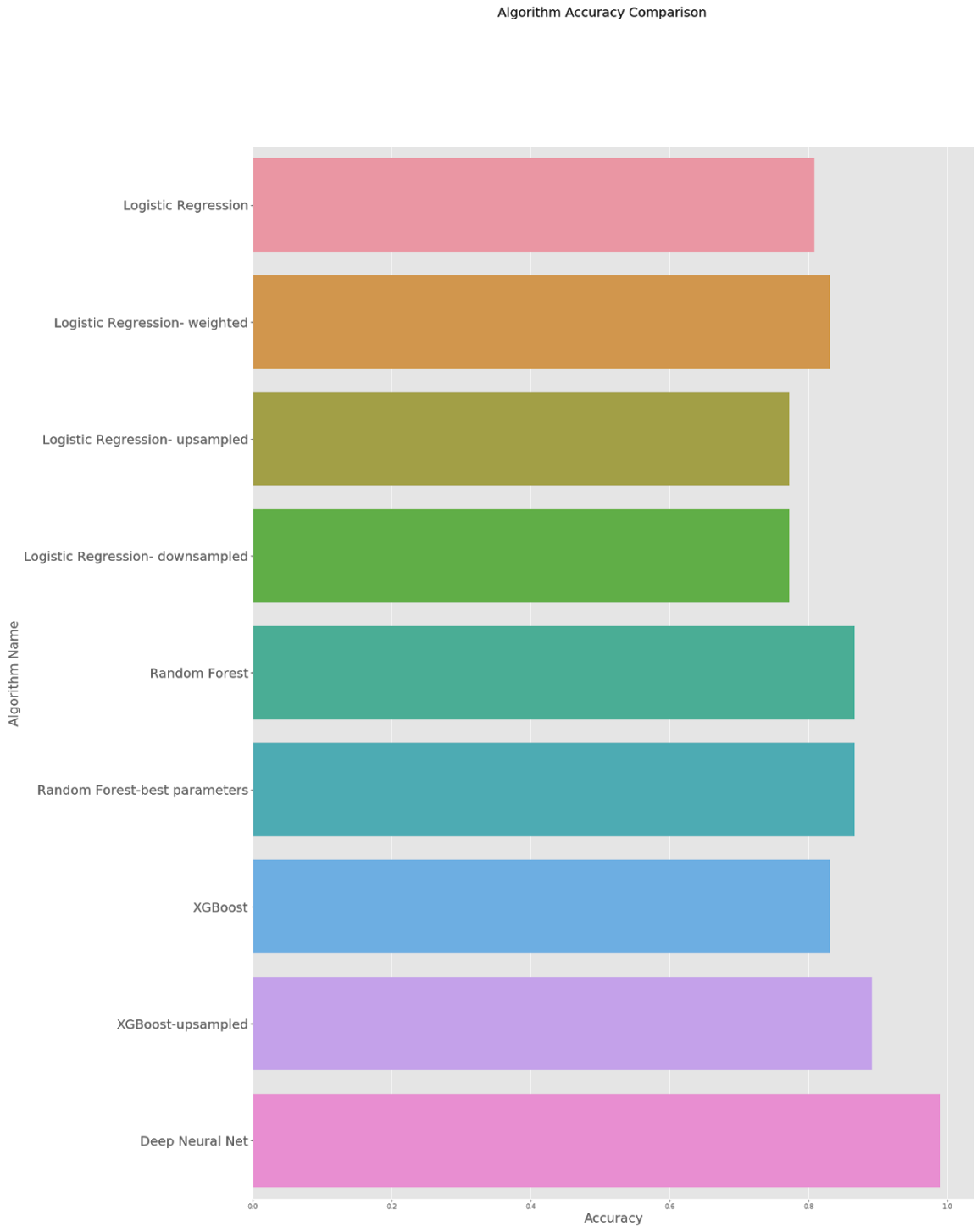
```
Training Accuracy: 0.9438907655200306
Validation Accuracy:  0.9135122302055029
```

	precision	recall	f1-score	support
0	0.96	0.92	0.94	119341
1	0.92	0.97	0.94	119341
avg / total	0.94	0.94	0.94	238682

	precision	recall	f1-score	support
0	0.93	0.75	0.83	37000
1	0.82	0.96	0.88	45332
avg / total	0.87	0.86	0.86	82332

Our accuracy decreased, but our recall increased slightly.

Section 3: Results



Deep Neural Nets computed the highest accuracy, followed by XGBoost on upsampled data and then Random Forest. However, for classification problems, accuracy is not an ideal performance metric.

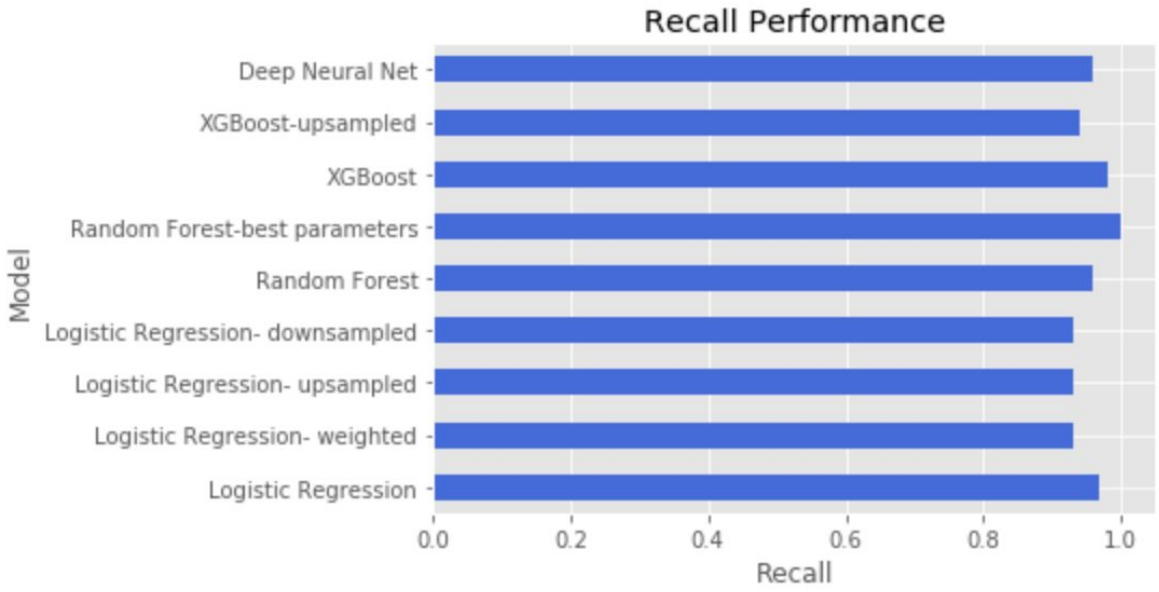
Intrusion detection systems are implemented to detect network attacks. Therefore we are looking to create a system where we detect all attacks from a group of data. We should focus on identifying positive cases. In our problem, it is an attack of labeled data of 1. The metric our intuition tells us we should maximize recall. The ability of a model to find all relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. Which means the number of predicted attacks that are actual attacks divided by the number of all attacks. While recall expresses the ability to find all relevant instances of a dataset, precision expresses the proportion of the data points in our model that says was relevant actually were relevant.

Depending on the problem we might want to maximize one or the other. For network attacks we want a recall near 1.0, we want to find all networks who are label attacks that are actual attacks and we can accept a low precision if the cost of another system is not significant. Another metric to keep in mind is the F1-score, which is the optimal blend of precision and recall.

Model	Accuracy	Precision	Recall	F1-Score	Roc/auc score
Logistic Regression	0.808118	0.75	0.97	0.85	0.953261
Logistic Regression- weighted	0.830649	0.80	0.93	0.86	0.953385
Logistic Regression- upsampled	0.771766	0.80	0.93	0.86	0.947094
Logistic Regression-downsampled	0.771984	0.80	0.93	0.86	0.948824
Random Forest	0.865969	0.82	0.96	0.89	0.958530
Random Forest-best parameters	0.865969	0.75	1.00	0.85	0.966968
XGBoost	0.830965	0.77	0.98	0.86	0.972704
XGBoost-upsampled	0.891112	0.88	0.94	0.90	0.972826
Deep Neural Net	0.988985	0.81	0.96	0.88	0.953169

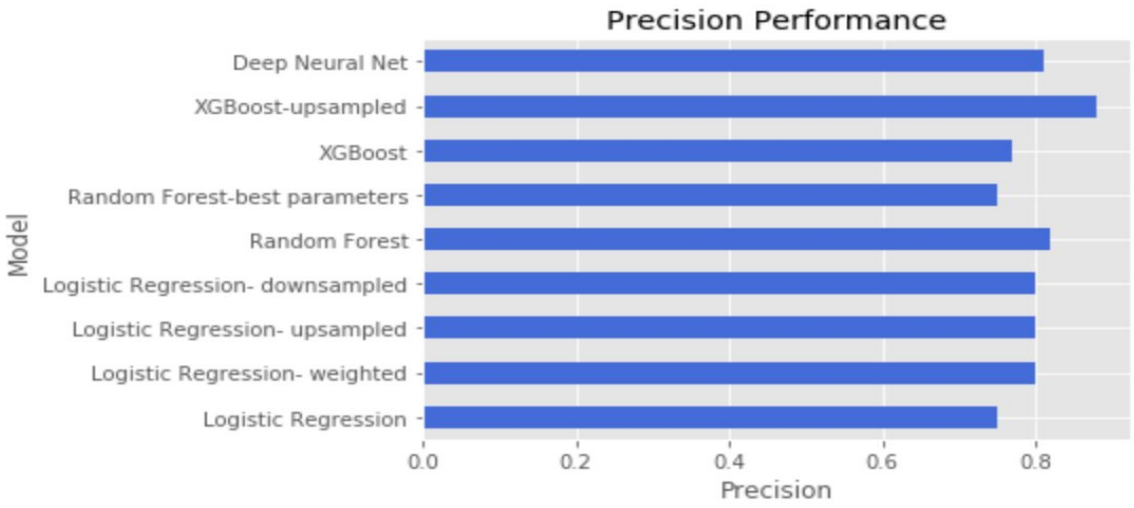
The network detection tasks is an imbalanced classification problem: We have two classes we need to identify- attacks and normal. With this set, one has the majority of the data points. Intuitively, we know that proclaiming all data points as negative in intrusion detection is not helpful, and instead, we should focus on identifying the attacks and maximize recall. But if we label all networks as attacks, then our recall is 1.0, therefore we also need to keep in mind precision and the trade-off between precision and recall. If we hypothetically label ever network as attacks if would not be useful since we would block every network. This model would suffer from low precision, the ability of a classification model to identify only relevant data points. Again, while **Recall** expresses the ability to find all relevant instances in a dataset, **precision** expresses the proportion of the data points in our model says was relevant that were relevant.

Recall Performance Between Models



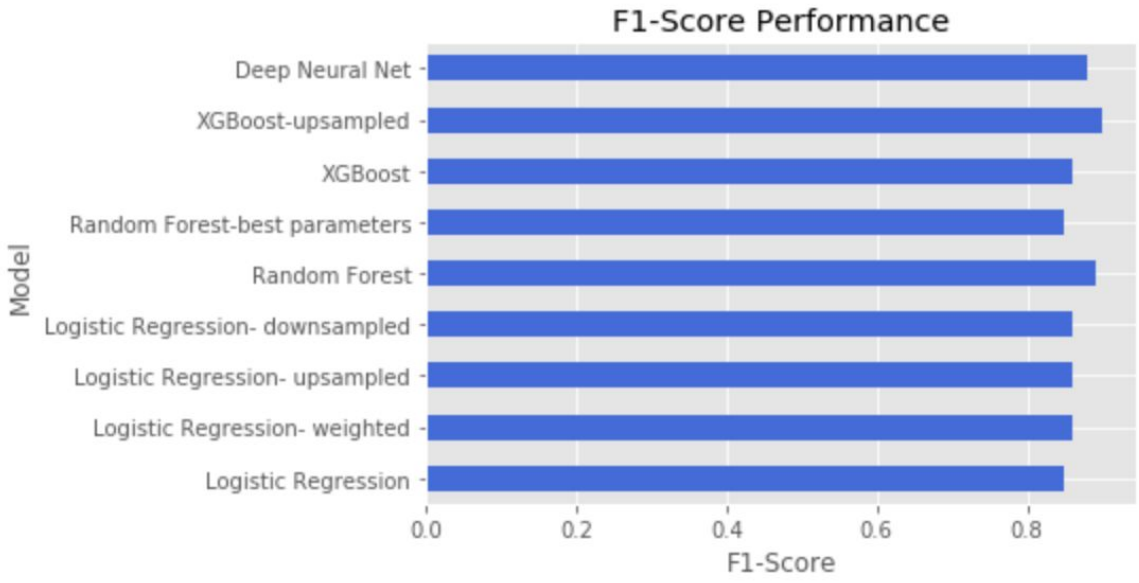
Random Forest with tuning gives us a recall of 1.00. This indicates the model correctly identified **100%** of attacks that were actually attacked. However, the precision is at **75%** indicating that 75% of positive cases were correctly identified.

Precision Performance Between Models



XGBoost with upsampled data gives us the highest precision at **88%** and a recall of **94%**.

F1-Score Performance Between Models



XGBoost with upsampled data gives us the highest F1-score of **90%**, followed by Random Forest at **89%** and Deep Neural Net at **88%.**

Visualizing Precision and Recall

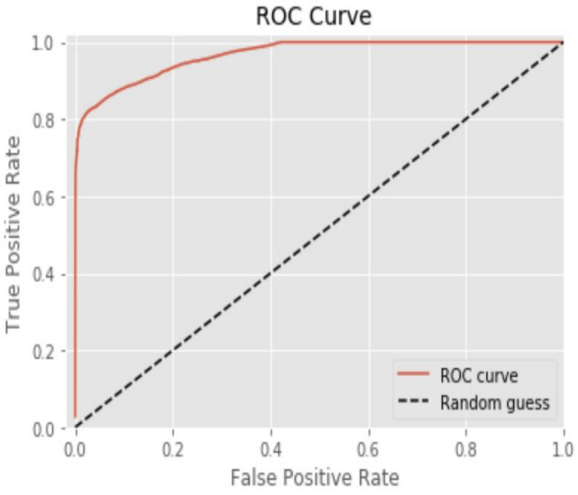
One of the main visualization techniques for showing the performance of a classification model is the Receiver Operating characteristic (ROC) curve. The ROC curve shows how the recall versus precision relationship changes as we vary the threshold for identifying a positive in our model. The threshold represents the value above which a data point is considered in the positive class. If we have a model for identifying an attack, our model might output a score of each network between 0 and 1 and we can set a threshold in this range for labeling a network as an attack. By altering the threshold, we can try to achieve the right precision versus recall balance.

We can quantify a model ROC curve by calculating the total Area Under the Curve (AUC), a metric that falls between 0 and 1 with a higher number indicating better classification performance.

Area Under Curve

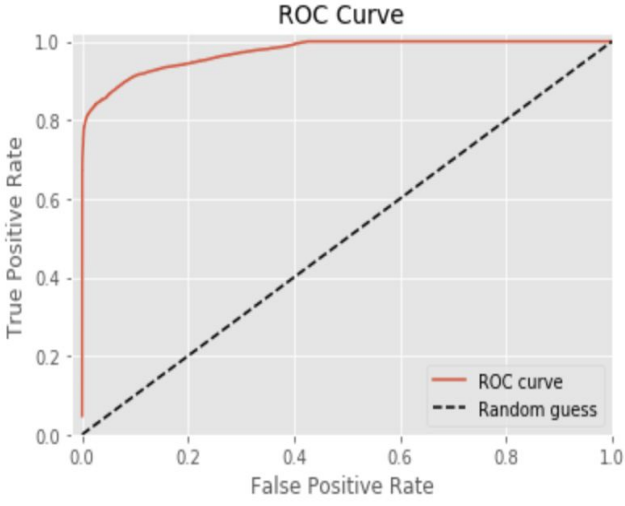
AUC is one of the most widely used metrics for evaluation. It is used for binary classification problems. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example. The greater the value, the better the performance. From the table, we can see the best performance comes from XGboost and random forest. More specifically, XGBoost with upsampled data and Random Forest with tuning.

Random Forest (Best Parameters) ROC curve



AUC Score: **.966**

XGBoost (upsampled) ROC curve



AUC Score: **.9728**

Conclusions

In this project, I have explored the use of different machine learning techniques applied to the problem of network intrusion. I performed data exploration for network data, created data visualizations while doing exploratory data analysis, and build machine learning models using different techniques: including logistic regression, random forest, XGBoost, and deep neural nets. After training and evaluating the models, I computed the performance metrics of each model and identified the best performance. The performance metrics I used were mainly focused on precision and recall and how it affects the problem of intrusion detection. The main goal was to create a model that would be able to classify network attacks given a list of features related to networks. I concluded that the best model was a random forest with tuning. This model gave a recall of **100%**. Indicating it predicted all attacks that were actual attacks from the dataset with a precision of **75%.** Although precision is not the best result from our recall give us the best solution for our problem. With the random forest model we are able to recall 100% of the attacks of network threats at a decent precision. I also computed the AUC score of **96%**, indicating a well performing model.

Future Work

- Study potential collinearity among independent variables to decide which ones not considered in the analysis
- Curate network data from different sources.
- Analyze other machine learning models and how it compares to the ones analyzed.
- Incorporate supervised machine learning
- Find ways to optimize precision for random forest
- Hypertune deep neural network to find optimal score

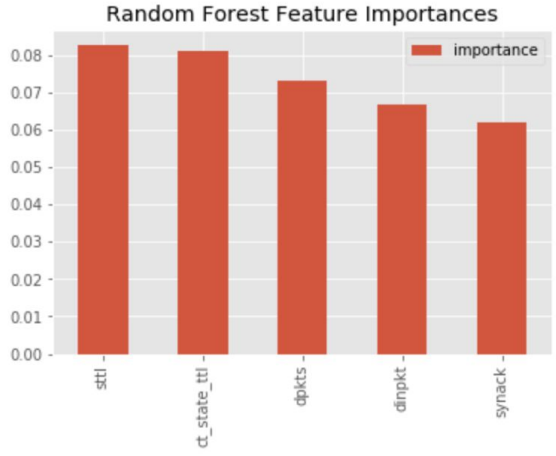
Section 4: Recommendation for the Clients

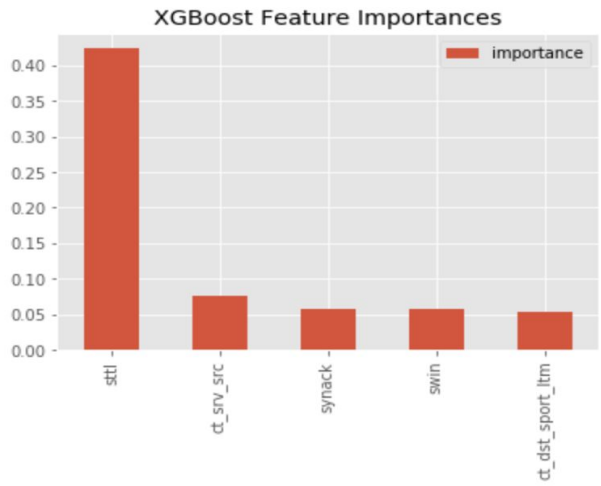
For this particular problem, detecting intrusions, we want to minimize false negatives. We want to correctly classify all the network attacks, even at the cost of accidentally predicting an attack even if it is normal (False Positive) as this is the better alternative than classifying normal even if it is an attack. (False Negative) This might be okay as it is less dangerous than NOT identifying/capturing an attack network since we can always send the network cases that were labeled attacks for further examination and analysis. But missing a network attack will be a huge mistake as no further examination will be done. Therefore, we are looking to optimize recall but also looking to have a decent precision. It is clear that recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about it performance with respect to false positives (how many did we catch). Recall is not about capturing cases correctly but more about capturing the cases that have 'attack'.

Using a random forest model with tuning, we computed a recall of **100%**. This indicates the model was able to correctly predict all of the labeled attacks. The precision for predicting attacks yields **75%**, which means it correctly predicted 75% of the all networks that was labeled attack. With a model that is able to compute a recall of 100%, this indicates a successful intrusion detection. A random forest with tuning correctly classified all network attacks that were indeed attacks. This is a great step toward creating an effective and efficient network detection system. We can now find ways to build around this model, maybe find ways to maximize precision.

Feature Importance

Using the top two machine learning classifiers, I computed its feature importance.





From both classification models ‘sttl’ is the number one indicator that influence model performance. ‘STTL’ stands for Source to destination time to live value. Time-to-live (TTL) is a value in an internet protocol packet that tells a network router whether or not the packet has been in the network too long and should be discarded. This is a great way to view which features are important for our model and may give a great indication to which features corresponds to a potential attack. Another feature to keep an eye for would be synack, as it influences both models as well. ‘Synack’ stands for the TCP connection setup time, the time between the SYN and the SYN_ACK packets.

Features that influence both models

- ‘STTL’ : Source to destination time to live value
- ‘Synack’: TCP connection setup time, the time between the SYN and the SYN_ACK packets

Model’s Applicability

Our model can be used as a backend engine to an intrusion detection system application that can be mounted on the border of any computer network. However, we should perform more tests, retrain the models to see if we can optimize our model even further. This will allow us to further verify our model’s accuracy in detecting anomalies, for it to be used in either an intrusion detection or intrusion prevention system.

- **Model is great at minimizing false negatives**
- **Model captures 100% of attack networks correctly**
- **Model gives important features**