Project Report
WBF-Logs analysis using ML

Williams Tom – Wecker Adam – Labbé Max

We would like to thank the people who worked on the project "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset". Special Thanks to Mr. Benfatma for helping us during the project

Abstract

This project is based on Nickolaos Koroniotis', Nour Moustafa's, Elena Sitnikova's and Benjamin Turnbull's project "*Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset*". Their project's aim is to create a "well-structured and representative" dataset (Bot-IoT) which can be used in different systems as well as in Machine Learning. The dataset's aim is to be able to combat and prevent cyberattacks from third parties. While their project explains how their environment extracts data from signals to create their dataset and its effectiveness in Machine Learning all in an IoT basis which groups does group the data science hierachy of needs, our project focalizes on exploring / transforming the data we have to then aggregate / label it and send the data into multiple machine learning models. Finally, we have to find the most prolific machine learning model to be used with the data.

Table of Contents

TABLE OF CONTENTS 2

I. INTRODUCTION - DATASET ANALYSIS 3

II. DATA MANIPULATION 5

DATASET FILTERING 6

a) Near Miss undersampling 7

b) Condensed Nearest Neighbour (CNN) 7

c) Tomek Links 7

III. MACHINE LEARNING MODELS 8

1. LINEAR SVC 8
2. KNN 9
3. RANDOM FOREST TREE 10
4. DEEP LEARNING / NEURAL NETWORK MODEL 11

a) Keras Models 12

b) Activation Functions 12

c) Losses 13

d) Optimizers 13

IV. RESULTS 14

V. CONCLUSION 16

I. Introduction - Dataset analysis

As said in the abstract, our dataset is based on the works from Canberra University. They have created a testbed consisting of three components: network platforms, IoT services and extracting features & Forensic analytics. From the .pcap files, they used the "Argus client program" to create .argus files from which, by extracting the flow information, created MySQL tables. These tables are then converted into .csv files.

The features extracted from the .argus files are the following :

FEATURE	DESCRITPIOIN	FEATURE	DESCRIPTION
pkSeqID	Row Identifier	**dur**	Record total duration
stime	Record start time	**mean**	Average duration of aggregated records
flgs	Flow state flags seen in transactions	**stddev**	Standard deviation of aggregated records
flgs_number	Numerical representation of feature flags	**sum**	Total duration of aggregated records
proto	Textual representation of transaction protocols present in network flow	**min**	Minimum duration of aggregated records
proto_number	Numerical representation of feature proto	**max**	Maximum duration of aggregated records
Saddr	Source IP address	**spkts**	Source-to-destination packet count
sport	Source port number	**dpkts**	Destination-to-source packet count
daddr	Destination IP address	**sbytes**	Source-to-destination byte count
dport	Destination port number	**dbytes**	Destination-to-source byte count
pkts	Total count of packets in transaction	**rate**	Total packets per second in transaction
bytes	Totan number of bytes in transaction	**srate**	Source-to-destination packets per second
state	Transaction state	**drate**	Destination-to-source packets per second
state number	Numerical representation of feature state	**attack**	Class label: 0 for Normal traffic, 1 for Attack Traffic
ltime	Record last time	**category**	Traffic category
seq	Argus sequence number	**subcategory**	Traffic subcategory

Table 1 Features & description

The dataset is separated into 74 .csv files with each file containing roughly 1 million samples which makes about 15Gb of memory overall. Because we have a certain limit of capacity on our computers, we decided to limit the number of samples we want to work with. Here is how we have decided to extract the dataset:

- first, we extract 2 files of attack and normal traffic which makes about 2 million samples
- then extract all non-attacks contained in the dataset (the reason is explained in Data Manipulation) which groups around 10 thousand samples.

Once we have created our own dataset, we have to manipulate it in order for it to be well fit and organised to be worked on by our Machine Learning models.

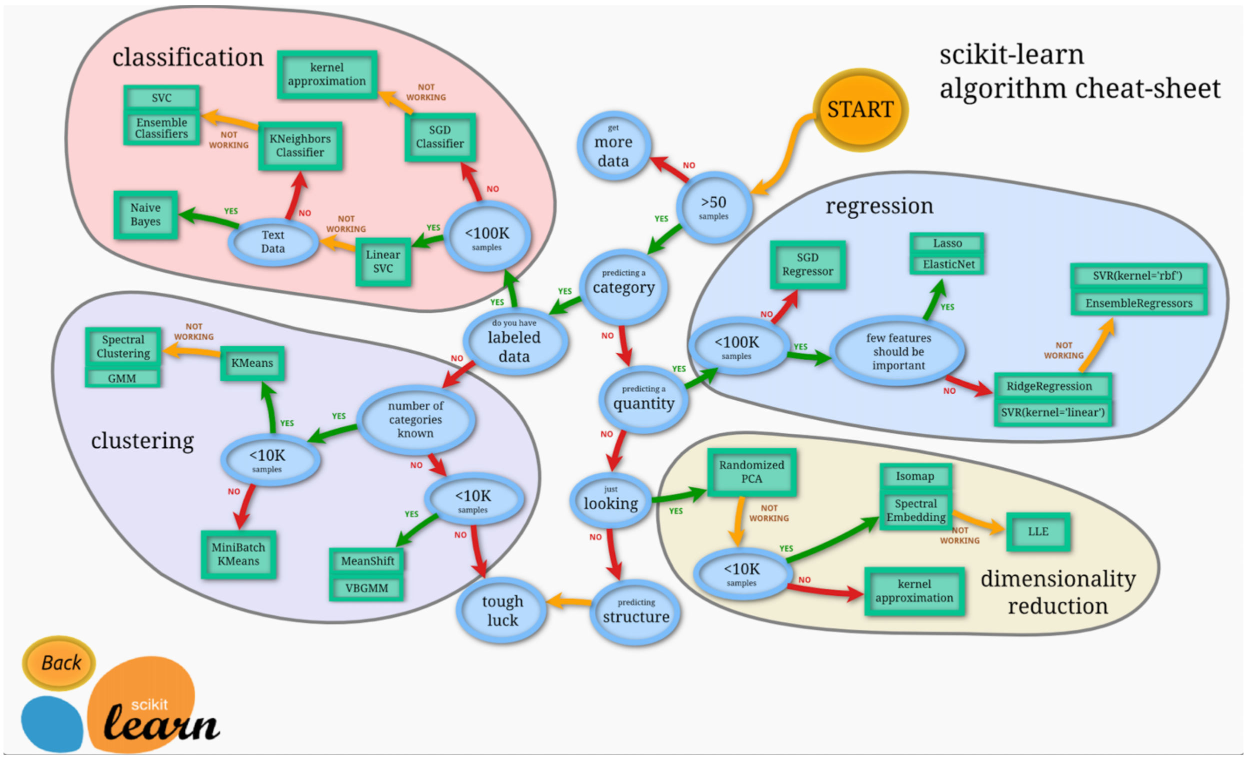


Figure 1- Cheat sheet on how to choose our ML model

Now that we have an idea about the size of our data, we are looking for machine learning models that centres around categorical labels. We can look at Figure 1. As we know, our dataset contains over 50 samples, we have to predict a category (attack or normal traffic), our data is labelled. As for under 100 thousand samples, we do end up with less due to the uneven number of attacks and normal traffic (as explained in Data manipulation) and we do not have any text in our data. We have come to the conclusion to work on LinearSVC, K Nearest Neighbours (KNN). We have also decided to work on RandomForestTree and Neural Networks; RandomForestTree as we have learned during our courses seems to be an effective algorithm and we wanted to experiment Deep Learning.

II. Data Manipulation

Before we start creating our Machine Learning Models, we have to make the data easy to read and use before it can be tested. We are manipulating lots of data (approximately 200MB per file), which means we have to solve problems which can compromise the models' proper performances. These manipulations have been dealt with the help of Scikit-learns functions.

First, we encoded columns that do not contain Int/Float variables which are variable types that cannot be mixed within a model, KNN for instance. Scikit-learn LabelEncoder function was used here. We then Standardised & Centered the data with the help of scikit-learn's pre-processing function StandardScaler(1). StandardScaler takes an array, calculates its mean as well as its standard deviation. The smallest value is either -1 or 0 depending on the parameters used and the highest value is 1. We used StandardScaler to ease calculation when Knn will find the distance between samples.

x_i : value in array	μ : mean of array
σ : standard deviation	x' : new value of x

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)}{N}} \qquad x' = \frac{(x - \mu)}{\sigma}$$

Equation 1 (left) : standard deviation / Equation 2 (right) : new value of x with StandardScaler

Next, we dropped all samples containing empty or NULL values as well as duplicates. Finally, as we will show during the study of the Machine Learning models, we have to perform an undersampling of the majority class in the dataset which are the attack class. Indeed, the number of attacks in the dataframe is greatly superior to the number of normal traffic which creates some form of oversampling of the dataset which prevents the Machine Learning model from being realistic and/or adaptable to new samples.

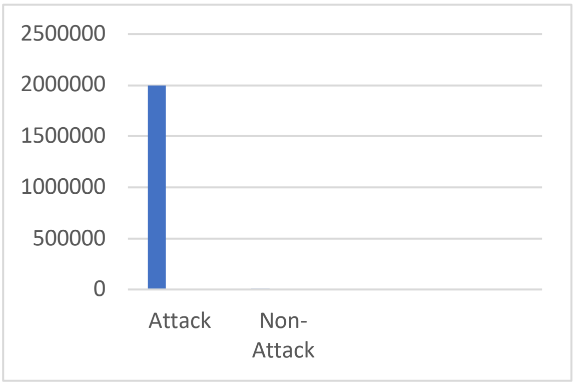
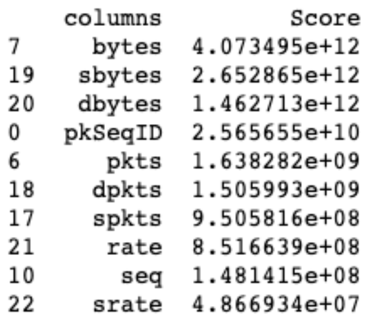


Figure 2: number of attacks extracted from 2 files against all normal traffic

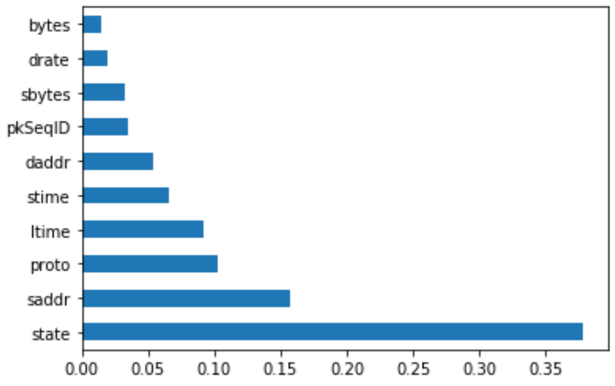
Dataset Filtering

One way to solve this problem, we have used techniques which will permit us to filter features which may not help models predict the correct class of signals. These techniques include Chi-Squared algorithm, ExtraTreesClassifier's feature importance property and Panda's corr() method.

Chi-Squared algorithm finds the features which are more likely to differ depending on the value it is equal to. For instance, the size of a signal may differ whether it is an attack or not, and Chi-Squared will return a score depending on how (un)equal the data is depending on its category. ExtraTreesClassifier's feature importance property is based on the Gini Importance or the Mean Decrease in Impurity. which concists of counting the number of times a feature is used to split a node and divide it by all the nodes that divide the data. Finally, Seaborn's correlation matrix is based on the usage of either Pearson's, kendall's or



```
     columns          Score
7      bytes   4.073495e+12
19    sbytes   2.652865e+12
20    dbytes   1.462713e+12
0    pkSeqID   2.565655e+10
6       pkts   1.638282e+09
18     dpkts   1.505993e+09
17     spkts   9.505816e+08
21      rate   8.516639e+08
10       seq   1.481415e+08
22     srate   4.866934e+07
```



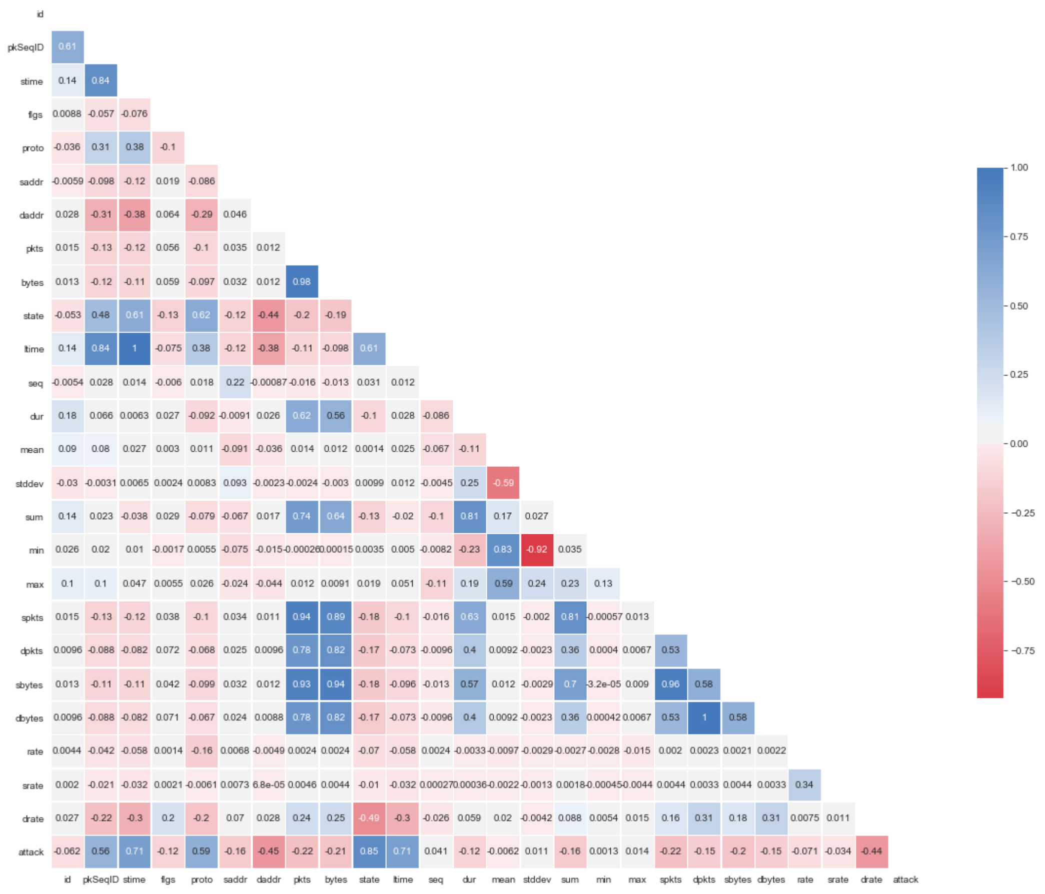


Figure 3: (top left) top 10 Chi-Squared features, (top right) top 10 important features in ExtraTreesClassifier, (bottom) Seaborn Correlation matrix

Spearman’s correlation equation, but in our case, we will be using Pearson’s equation. The results are the following:

By process of elimination (occurrence of feature in overall) we have selected the following features: bytes, sbytes, PkSeqId, spkts, pkts, state, proto, stime, daddr, drate. We will use the 10 features in our algorithm to test its importance.

Another way to solve this problem, we have decided to use the ImbLearn (Imbalanced Learning) library. From the library, we have worked on the following functions: NearMiss, CondensedNearestNeighbour (CNN) and Tomek Links. These algorithm, unlike the previous ones, filter rows instead of features.

a) Near Miss undersampling

Near Miss is an ensemble of algorithms which is based on the average distance of the majority class examples to the minority class examples. There are three different versions to this algorithm:

- Version 1: Majority class examples with minimum average distance to three closest minority class examples.
- Version 2: Majority class examples with minimum average distance to three furthest minority class examples.
- Version 3: Majority class examples with minimum distance to each minority class example.

NearMiss algorithms returns a dataset with as much attacks as normal traffic. Ultimately, we have decided to use the first version of the algorithm.

b) Condensed Nearest Neighbour (CNN)

Condensed Nearest Neighbour is an undersampling algorithm which selects a random point from the majority class and deletes random points around it which are from the majority class depending on the information it returns. For instance, that we choose to compare a point to 1 neighbour, the closest one. If, said neighbour, is from the minority class, then the random point selected must be close to the border between both classes, it is therefore important to keep it in the new dataset as it is important compared to points which are only surrounded by the same class. What can be problematic which CNN is that examples are chosen randomly which can cause redundancies in the examples that are kept in the under sampled class.

c) Tomek Links

Although previous undersampling methods selects examples to keep, Tomek Links selects examples to delete. Tomek-links finds class boundaries and deletes the links by deleting the majority class’ example. It is as if it considers the link as a noise. The main problem with this solution is its time consumption.

We ultimately decided not to use Tomek Links as we would rather not delete the borders between labels.

III. Machine Learning Models

1. Linear SVC

Linear SVC (Support Vector Classifier) is a machine learning model which provides a hyperplane (line for 2 or less features or a plane for 2 or more features) which divides the data you provide into 2 components. In our case, we are working with more than 2 features so we will be looking at how it all works with a plane.

$$W^T \circ X = 0$$

Equation 1: equation of a hyperplane

The algorithm searches for the hyperplane, which is the furthest away from each component, which will reduce risks of making wrong predictions. To do that, it maximises the margin from the hyperplane. The margin is the distance which separates the hyperplane from the closest samples from each component. To find the best margin, it finds the closest distance from each component and creates the hyperplane perpendicular to the distance between both samples as shown below:

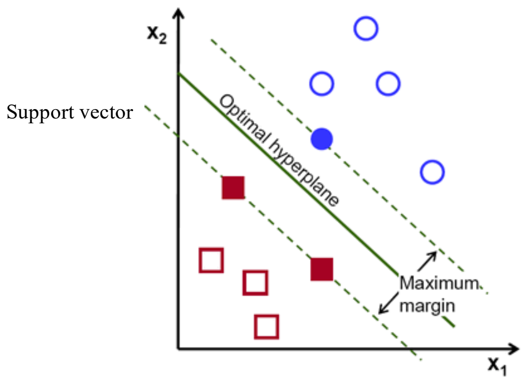


Figure 4: visual of margin, hyperplane and support vector

2. KNN

KNN is a classification algorithm which decides in what class the sample belongs to depending on the number of neighbours it is compared to with k its parameter. The distance used is originally the Minkowski equation, but it depends on a parameter p, which has a final decision on the distance we want to use:

$$d_{minkowski} = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}, \quad d_{manhattan} = \sum_{i=1}^{n} |x_i - y_i| \,, \quad d_{euclidean} = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^{2}}$$

Equation 2: Minkwoski, Manhattan and Euclidean distances

As seen above, the Minkwoski equation has a parameter p which can be used when we create our model. By default, p is equal to 2 which makes it a Euclidian distance, but we can also choose p = 1 to make it a Manhattan equation. Then depending on the number of neighbours k that you compare your sample to, it will calculate the probability of the sample belonging to each component then puts it in the category it is more likely to belong to.

Other parameters used are the algorithms used, there are 3 major algorithms that are offered which are Ball Tree, Kd Tree and Brute Force. Ball. Ball Tree and Kd Tree are tree algorithms that separates data based on their positions but in a different manner:

- Ball tree algorithm divides the data into two circular (2D or 3D) clusters also called Hyperspheres hence the name of the algorithm. It first makes a cluster that gathers all the data (C1) then selects the 2 samples which are the furthest away from its centre which then become the canters of their own cluster (C2 and C3). All other points are attributed a class depending on the closest they are closest to, either C2 or C3) and the process is repeated until there are no more clusters to be made.

- Kd Tree algorithm uses a split criterion in order to divide the data. For instance, it can decide to separate the data by selecting the median of the x or y coordinates of the dataset and separate it accordingly. This method is repeated until the model seams satisfies with the clusters it has made.

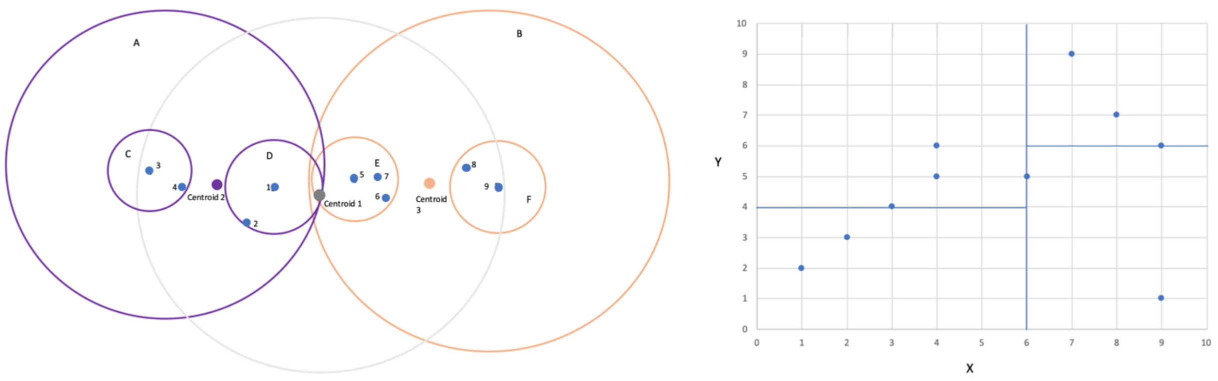
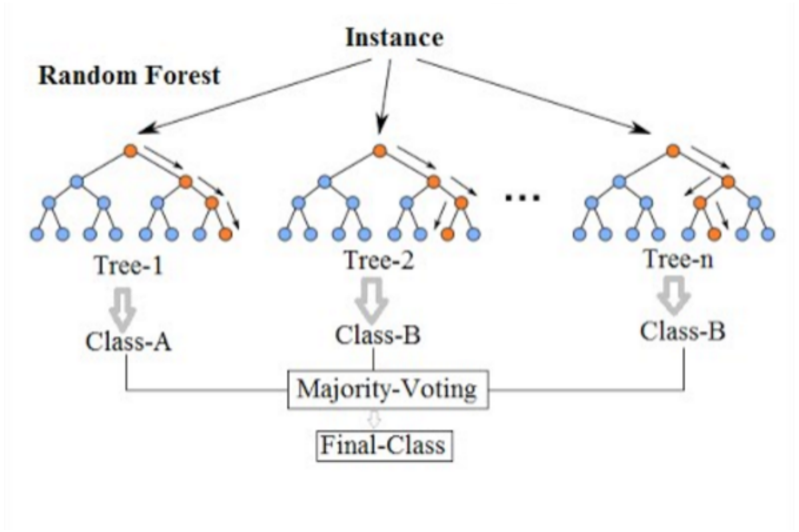


Figure 5: (left) ball tree algorithm (right) Kd tree algorithm

Brute force on the other hand calculates all distance between the sample in question and all the other points, orders the distance then votes accordingly to the k nearest neighbour. However, this algorithm is time consuming.

3. Random Forest Tree



Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. The first step in measuring the variable importance in a data set is to fit a random forest to the data. During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest (errors on an independent test set can be substituted if bagging is not used during training).

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x1, ..., xn$ with responses $Y = y1, ..., yn$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these Xb, Yb.
2. Train a classification or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on *x'*:

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

The number of samples/trees, *B*, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees *B* can be found using cross-validation, or by observing the *out-of-bag error*: the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the *B* trees, causing them to become correlated.

Typically, for a classification problem with *p* features, $\sqrt{p}$ (rounded down) features are used in each split. For regression problems the inventors recommend *p/3* (rounded down) with a minimum node size of 5 as the default. In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

4. Deep Learning / Neural Network Model

We have used Keras / Tensorflow.Keras to create our Deep Learning Model. To explain a simple Deep Learning Model, here are 2 figures:

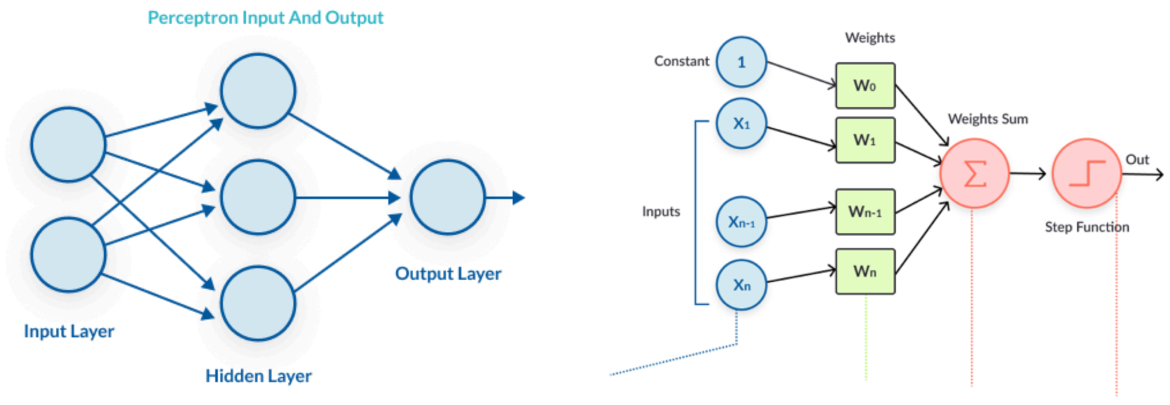


Figure 6: Multilayer Perceptron (left) , Perceptron learning process (right)

Above, we have the representation Multilayer Perceptron. A perceptron is a binary classification algorithm modelled like a brain and so the multilayer perceptron is a group of perceptron organized in multiple layers. On the right we have the functionality of a perceptron; an array of inputs inserted into the input layer are multiplied by the weights of the neurons, added and the previous result is inserted into the function which can be changed as explained in activation function. The result is represented in the output "out" which in the figure is the final output but can then be inserted into another hidden layer by the usage of a different activation function.

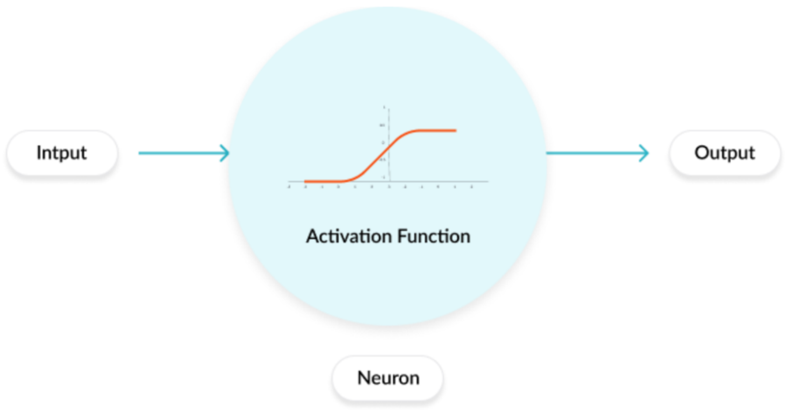
a) Keras Models

In the Keras library, we have used, in the project, two types of Models: The Sequential and Functional API model:

- The Sequential model is the basic way of creating neural networks by stacking layers where each layer has 1 input and 1 output. We manually create the layers with a simple add() function and inside this function we describe the type of layer we want to add and its parameters.
- The Functional API, contrary to the Sequential model, can work with multiple inputs and multiple outputs

b) Activation Functions

Activation functions in Keras are mathematical functions that take in account an input, inserts it into a function and returns an output. Each layer has some sort of activation function. Activation functions determine the output, accuracy and efficiency of the model used.



There are three types of activation functions:

- Binary Step Function is a "threshold-based" activation function which means that if the input value is under of below a certain value, the neuron will be activated and send the same signal to the next layer, but if the condition is not met, then the neuron will return an output of 0. As it is a binary function, it returns 1 or 0, thus it does not support multi-value outputs.

$$y = \begin{Bmatrix} 1 \; if \; x > n \\ 0 \; if \; x < n \end{Bmatrix}$$

- Linear activation function takes multiple inputs and multiplies them by each weights of each neuron and returns the output. Although it allows multiple outputs unlike the binary step function, we cannot use backpropagation (gradient descent) to train the model as the derivative of the activation function is a constant and collapses the neural network into one layer.

$$Y = C * X$$

- Non-Linear Activation Functions like the linear activation function allows multiple inputs and outputs as well as permits the model to work with complex data such as video, images, audio and data sets which are non-linear or have high dimensionality. Unlike the linear activation function, it allows back propagation and allows "stacking" of multiple layers. Some of the most notable nonlinear activation functions are sigmoid, tanH, ReLU, leaky ReLU and Softmax.

c) Losses

The goal of ML models is to reduce the difference between the predicted output and the actual output also known as the Loss or Cost function. As we are classifying data, we will enumerate only loss functions used in this theme. The most notable classification loss functions are Binary/Multi-Class Cross Entropy, Negative Log Likelihood, Mean Squared Error and Soft Margin Classifier.

d) Optimizers

To do this we have to run the model a number of times while attributing to the neurons different weights to find the minimum cost, this method is called Gradient descent. It is here that the optimizer comes into place as its role if to update the weight of each parameters in order to minimize the loss function. Some well-known optimizers might include Momentum, Adagrad, RMSProp, Adam, SGD and Adamax.

IV. Results

We have decided to write a program which is able to group all the conditions we can put our models in such as different undersampling methods and the number of features being used. Here is visual representation of the program:

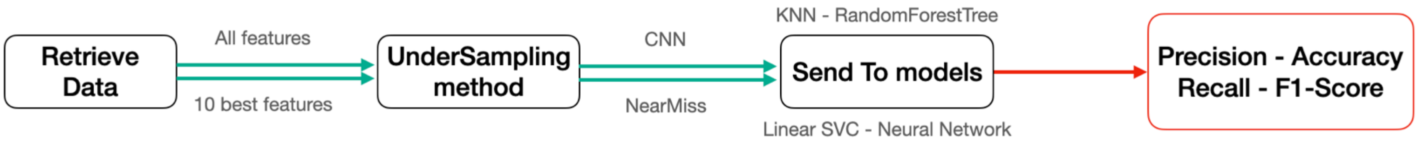
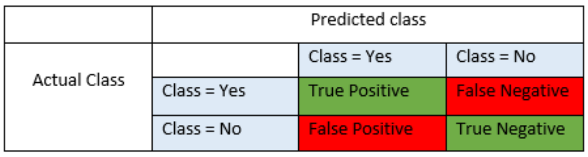


Figure 7: Illustration of our program

As illustrated above, first, we have to retrieve our data which consists of 2 random files each containing 1 million samples as well as all normal traffic in the entire dataframe (roughly 2 million attacks and 10 thousand normal traffic signals). We then use 2 for loops, on for the number of features we will use (10 or 24) and the other for the type of undersampling method we will use (CNN or NearMiss). Nearmiss makes the data balanced (20 thousand samples) whereas with CNN we have less attacks than normal traffic (9600 normal traffic and around 20 attack signals). We end up with 4 datasets to test in our models. With each model, we retrieve its precision, accuracy, recall and f1-score. These 4 variables will determine how our models have performed.

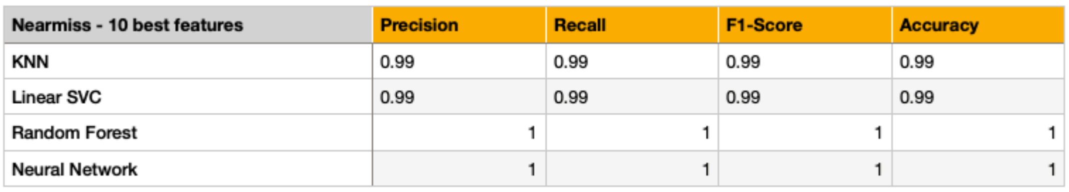


		Predicted class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

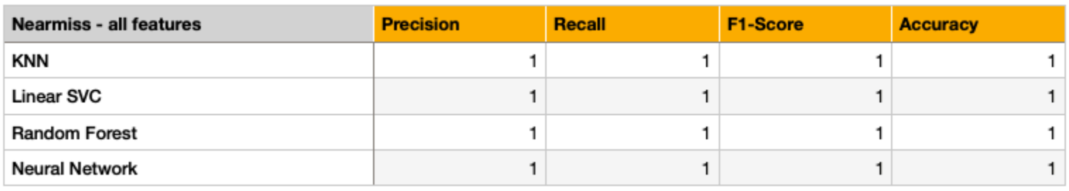
$$Accuracy = \frac{TP + TN}{all} \qquad precision = \frac{TP}{all\ positives}$$

$$recall = \frac{TP}{TP + FN} \qquad F1 - score = \frac{2 * (recall * precision)}{recall + precision}$$

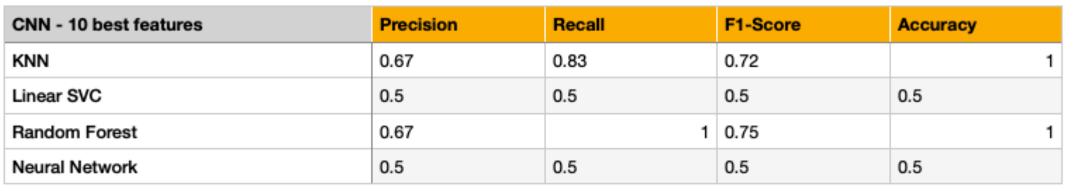
Thanks to scikit-learn's classification report, we can have these variables appear at the end of each model's predictions. Because we can be looking for either attacks or normal traffic, we are using the macro average as each model's final results. The results for each condition and model are the following:



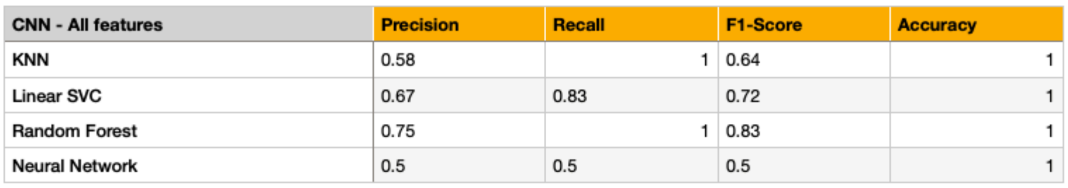
Nearmiss - 10 best features	Precision	Recall	F1-Score	Accuracy
KNN	0.99	0.99	0.99	0.99
Linear SVC	0.99	0.99	0.99	0.99
Random Forest	1	1	1	1
Neural Network	1	1	1	1



Nearmiss - all features	Precision	Recall	F1-Score	Accuracy
KNN	1	1	1	1
Linear SVC	1	1	1	1
Random Forest	1	1	1	1
Neural Network	1	1	1	1



CNN - 10 best features	Precision	Recall	F1-Score	Accuracy
KNN	0.67	0.83	0.72	1
Linear SVC	0.5	0.5	0.5	0.5
Random Forest	0.67	1	0.75	1
Neural Network	0.5	0.5	0.5	0.5



CNN - All features	Precision	Recall	F1-Score	Accuracy
KNN	0.58	1	0.64	1
Linear SVC	0.67	0.83	0.72	1
Random Forest	0.75	1	0.83	1
Neural Network	0.5	0.5	0.5	1

We can observe that all models have performed well while using Nearmiss whether we are using the 10 best features or all features from the dataset. Since they all have about the same results, we must focus on their results while using CNN algorithm. Now, if we look at the models' results int the CNN tables, we can observe that the results very much differ from their NearMiss counterparts which helps us see which model seems to perform best. In these cases, we can see that RandomForest Classifier is the model that has performed best followed by Linear SVC. We can conclude that overall, RandomForest is the model that should be used.

Ultimately, we cannot truly compare our results with the original work done on this project due to the different conditions onto which the data was send to the models. Indeed, no undersampling methods were used previously, which does hinge the proper functionality of the models in practice. But if we do not consider the conditions we have put in place, we do end up with a worse score while using CNN but with better scores using NearMiss.

V. Conclusion

This project presents a new approach in data science especially in in transforming, aggregating and learning the dataset. Our aim on this project was to predict if samples could be predicted as attack or normal traffic. We had to use numerous data manipulation algorithms such as standardizing, scaling, encode and undersample in order to make the data realistic for our models. The results above show promising results overall especially RandomForestTree which has the best scores as well in NearMiss as in CNN. We can go further in this project by trying to find all subclasses of the samples that can be used and optimize the models.

Link to original dataset : https://cloudstor.aarnet.edu.au/plus/s/umT99TnxvbpkkoE?path=%2F