Machine Learning for Cybersecurity Using IoT-23 Dataset

1st Warren Watts
Department of Engineering Technology & Industrial Distribution
Texas A&M University
College Station, United States of America
orangewatts@tamu.edu

***Abstract*—For this project, a number of machine learning models generated from an IoT-23 dataset that could be used for Intrusion Detection Systems were created. These models were created through the programming language *Python*, allowing for ease of configuration and manipulation of said models. By compiling capture packets of both normal and malicious natures into datasets, using Data Conditioning to configure said datasets into a singular equal and concise dataset ready for modelling, and applying both the Decision Tree Classifier and Random Forest Classifier algorithms to the dataset, a Decision Tree and Random Forest model were able to be generated. Through the use of metrics such as accuracy, precision, recall, F1-score, and Confusion matrices, the performance of these models was able to be verified, both models showing extreme similarity in impeccably high metric values. With this it was demonstrated that the models were not only able to fit well with the given data, but would fit well with datasets in general.**

***Index Terms*—Decision Tree, Random Forest, Confusion Matrix, Dataset, Intrusion Detection System**

Introduction

As the threat of cyber criminals and cyber-attacks becomes more prevalent in today's society, the importance of Intrusion Detection Systems (IDS) that can readily prepare and counter newly developed methods of attack cannot be understated. Machine learning algorithms have allowed for the creation of this type of IDS (specifically, algorithms utilizing binary and multiclass classification.) For this project in particular, the goal was to use machine learning models to make an IDS that utilized binary classification (allowing the IDS to flag/differentiate normal packets versus attack packets). This was done by first capturing attack and normal packets in Wireshark to have structured input data. Then, through data conditioning the separate datasets for attack and normal packets are combined and made to have an equal sample size (the number of attack packets used for the data is equal to the number of normal packets used for the data). Unneeded features are removed from the dataset, and the data is split into training and testing sets. Next, a Dummy Classifier is created to give a baseline for the real machine learning model, and finally, the Decision Tree Classifier and Random Forest Classifier models are created and analyzed. With these steps complete, the goal of creating/using machine learning models to make a binary classification IDS will have been achieved.

I. Data Conditioning

A. IoT-23

As mentioned in the Introduction section, structured input data is required to begin the first step towards Data Conditioning. In the case of this project, this structured input data came from multiple IoT-23 datasets (two malicious datasets and one normal dataset). The IoT-23 dataset is a dataset that represents packets on a network for IoT (Internet of Things) mediums [1]. (Hence the reason the dataset is called IoT-23.) This dataset consists of a mix of malicious packets (IoT based malware) and normal packets that IoT devices may receive through a network. This dataset allows for models (like the one being made for this project) to be created and utilized in future Intrusion Detection Systems. There are twenty specific attack packet captures, consisting of Mirai, Torii, Trojan Gafgyt, Kenjiro, Okiru, Hakai, IRCBot, Muhstik, Hajime, and Hide-and-seek attack types. As mentioned prior, there are two malicious datasets (C&C.csv and Torii.csv) and one normal dataset (Normal.csv). (For now, datasets C&C.csv and Normal.csv will be the main focus.) Since the number of captures in each dataset differs (C&C.csv clearly having a much larger number of packet captures), when both datasets are appended to one-another to form a singular dataset, this singular set will be skewed heavily due to the difference is sample size. (See *Figure 1*.) To fix this, the sample size of the larger dataset is made to be equal to the smaller dataset. With this, both datasets can be appended to one-another, as this new singular set is fifty-fifty in containing malicious and normal packet captures. (See *Figure 2*.)

B. Dataset Features

With this complete, unnecessary features (or rather column titles for a dataset) can be removed. Features are removed from datasets because they may serve no use in the creation of a machine learning model, or because they be harmful to the output of the generated model. (Harmful effects include overfitting, where the model fits to well to the given data, but won't be able to generalize to other datasets, or underfitting, where the model fits to poorly to the given data [2].) The features removed included 'Flow_ID', 'Src_IP', 'Src_Port', 'Dst_IP', 'Dst_Port', 'Sub_Cat', and 'Timestamp'. The IP and Port features were removed because they were specific to the

dataset, possibly causing overfitting. The 'Sub_Cat' feature was removed because the instructions directed the user to do so. (The 'Sub_Cat' label was most likely removed because it contains the end identifier of the file, and since all of the rows on the 'Label' column are filled with 'Anomaly' as the value, the file containing the anomalous packets would be identified, possibly construing the data.) The features 'Flow_ID' and 'Timestamp' were also removed due to being unique identifiers for the dataset that are not necessary for the ML Model. After these features are removed, the dataset is left with seventy-eight features. Any values that are Not a Number (NaN) or are equal to infinity will also be removed during this period, as well as changing any string based features that were not removed from the dataset to integers. (This is due to many implementations not allowing string based features/labels.)

C. Training & Testing Data Split

Now, the dataset in split into a training set and a testing set. This allows for a fraction of the data to be allocated to train the machine learning model, and another fraction to be used for testing the machine learning model with various metrics. These split sets also give a view into whether or not the data is overfitting or underfitting. When splitting the dataset into the training and testing sets, a number of parameters can be used, however, the only parameter used was the 'test_size' parameter, since the 'train_size' parameter is set based on this parameter's value. The other parameters such as 'random_state' (which allowed you to get reproducible outputs), 'shuffle', and 'stratify' were either already set by default to values that would be needed for this or were not needed. The 'test_size' parameter was set to a value of thirty-three percent of the data, making the 'train_size' parameter equal to sixty-seven percent of the data. With this split complete, the Data Conditioning section of this project was finished.

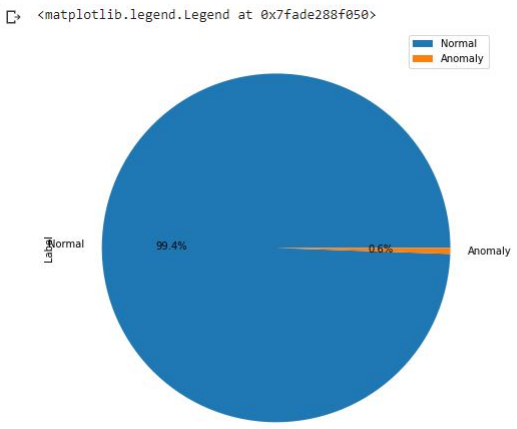

Fig. 1. Dataset Shape Before Re-Sampling

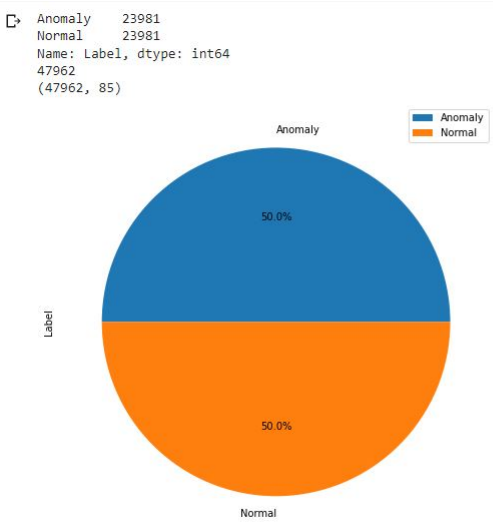

Fig. 2. Dataset Shape After Re-Sampling

II. Algorithms

A. Decision Tree Algorithm

Now that the Data Conditioning portion of this project has been completed, the work towards creating the machine learning models themselves can begin. First a Dummy Classifier is created so that when the real model is generated, it has some sort of foundation to work with. (It should also be noted that the accuracy and precision for the Dummy Classifier should both end up at around fifty-percent, as seen in *Figure 3*.) Now, the Decision Tree can be created. A type of supervised machine learning (i.e. the model is trained and tested for a given dataset), Decision Trees are used to organize and make predictions about a dataset and its outcomes [3], [4]. The concept of a decision tree can be represented through a real-world scenario. On a given day a beach can either be open or closed, and you are wanting to find out whether or not you can go to the beach today. From *Figure 4*, we see that the tree starts with a decision node called the root node. A decision node represents a question that the input variables decide how to answer. For example, in *Figure 4* we see that one decision node asks whether there is dangerous sea life in the water. Based on input variables (such as the number of sharks, jellyfish, etc. present in the waters off the beach), a decision of made, leading either another decision node or a leaf node. A leaf node is an end node of the tree that provides the outcome. In *Figure 4*, there are two types of leaf nodes present after the splitting has finished: open or closed. Splitting is the process of a node dividing into further nodes (either another decision node or a leaf node). (Splitting stops after reaching a leaf node.) For the Decision Tree function in the code, multiple parameters can be used to prevent over and underfitting and speed up the generation of the model. The only parameter used/changed from the default was 'max_depth'. Considering that even without using the

'max_depth' parameter, the decision tree model was neither underfitting nor overfitting, but giving good precision and accuracy instead, the only reason the 'max_depth' parameter was changed was for the purpose of making the execution slightly faster (it takes approximately 0.603 seconds for the decision tree classifier to train) as well as gaining a further understanding of how some of the parameters worked. (The 'max_depth' parameter is the length of the longest path from the tree's root node down to one of its terminal nodes.)

```
y_predict_dummy = model_dummy.predict(X_test)
print('Baseline Accuracy Score: ', accuracy_score(y_test, y_predict_dummy))
print('Baseline Precision Score: ', precision_score(y_test, y_predict_dummy))

Baseline Accuracy Score:  0.5034748546878949
Baseline Precision Score:  0.503107953824686
```

Fig. 3. Dummy Classifier Precision and Accuracy

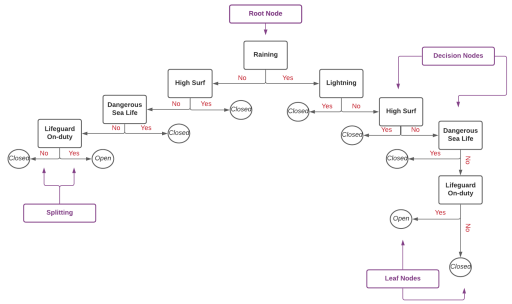

Fig. 4. Decision Tree Real World Example

B. Random Forest Algorithm

With the completion of the first machine learning algorithm, the process for the second algorithm, Random Forest Classifier, begins. To put it simply, Random Forests are a collection of base Decision Tree models that allow for the production of a more optimized model [5], [6]. This is done by creating multiple new training sub-datasets from the original dataset using random sampling with replacement. (This is what is known as bootstrapping.) A Decision Tree model is created for each of these sub-datasets, but not every feature will be used in each subset for the. Once the decision trees are made, when a dataset is given to the Random Forest algorithm, it puts it into each decision tree, and the results of the decision trees are combined using what is called aggregation. This aggregation allows for the creation of the Random Forest model, one that is more optimized than a lone Decision Tree model. Like the Decision Tree function, the Random Forest function in the code has multiple parameters that can be used to prevent over and underfitting and speed up the generation of the model. For the Random Forest model, the only parameters changed from there defaults were the 'n_estimator' and 'max_depth' parameters. Since the 'bootstrap' parameter is already set to 'True' by default, and the fact that when the other parameters for the Random Forest function are changed they only decrease the precision and accuracy or increase the time it takes to generate the model, the two parameters mentioned above were changed solely to allow for the Random Forest Classifier to train at a faster speed. (It takes approximately 0.609 seconds for the Random Forest Classifier to train.) With both machine learning models created, the results for the C&C Attacks can be analyzed.

III. Results for C&C Attacks

A. Metric Equations

Before reviewing the results for the C&C Attacks, the metrics used to analyze the models for this data need to be explained. Accuracy is the average correctness of a classifier, defined as:

$$\frac{(TruePositive + TrueNegative)}{Total} = Accuracy \quad (1)$$

Precision is the average correctness of a classifier when predicting yes, defined as:

$$\frac{TruePositive}{PredictedYes} = Precision \quad (2)$$

Recall is the average correctness of a classifier when it is actually yes, defined as:

$$\frac{TruePositive}{ActualYes} = Recall \quad (3)$$

The F1-score is the weighted average for recall and precision, defined as:

$$2 * (\frac{Precision * Recall}{Precision + Recall}) = F1 - Score \quad (4)$$

The variables 'True Positive' and 'True Negative' can be explained through another metric used called the Confusion Matrix.

B. Confusion Matrices

A Confusion Matrix is a matrix with prediction as the columns and actual as the rows. A 0 on either a row or a column means a no, and a 1 means a yes [7]. The cell that has a column with a no prediction and a no actual represents a true negative (meaning we predicted and got no). The cell that has a no prediction and a yes actual represents a false negative (meaning we predicted no and got yes). The cell that has a yes prediction and a no actual represents a false positive (meaning we predicted yes and got no). The cell that has a yes prediction and a yes actual represents a true positive (meaning we predicted and got yes). The confusion matrix takes a given dataset or a number of samples and classifies them as predictions. Once the model has been created, it can then organize these predictions into one of the aforementioned cells to categorize the data, showing how well the model performed.

C. Results

The performance of the models (both Decision Tree and Random Forest) are shown in *Figure 5* and *Figure 6*. Through these figures it can be seen that for both models, the values acquired for each metric are almost identical in nature, showing that the was little need for the use of a Random Forest algorithm for this dataset, as a lone Decision Tree was able generate a well fitting model. (It also shows the importance of using multiple metrics to evaluate a model, because the model may be able to pass just the accuracy metric, but may not do so well when the F-score or Confusion Matrix metrics are used, providing the knowledge that the model is flawed in some way.) That being said, considering that Random Forests are a conglomeration of Decision Tree models that allow for the output of a more optimized model, the Random Forests model would be chosen for better performance.

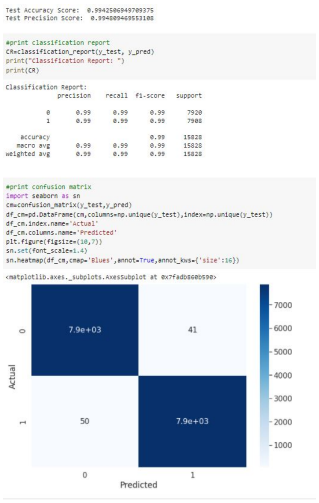

Fig. 5. Decision Tree Model C&C Results

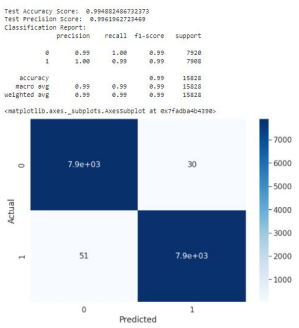

Fig. 6. Random Forest Model C&C Results

IV. Results for Torii Attacks

Using the other malicious dataset mentioned previously in the Data Conditioning section (Torii.csv), another set machine learning models were able to be generated. The performance of the models (both Decision Tree and Random Forest) for this new dataset are shown in *Figure 7* and *Figure 8*. Through these figures it can be seen that for both models, the values acquired for each metric are again almost identical in nature, and for this dataset, slightly closer to one-hundred percent for accuracy and precision metrics. In fact, the Classification Report for both the Decision Tree and Random Forest models showed many values to be one-point-o. Again, this demonstrates how the models generated were able to fit impeccably to not only the given dataset, but datasets in general.

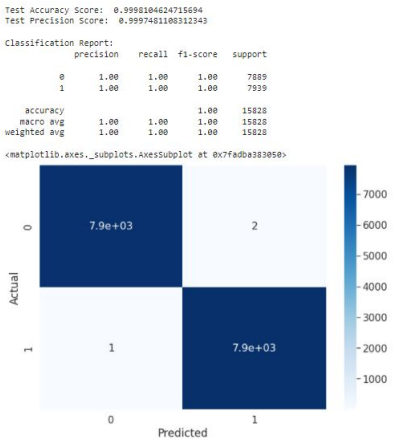

Fig. 7. Decision Tree Model Torii Results

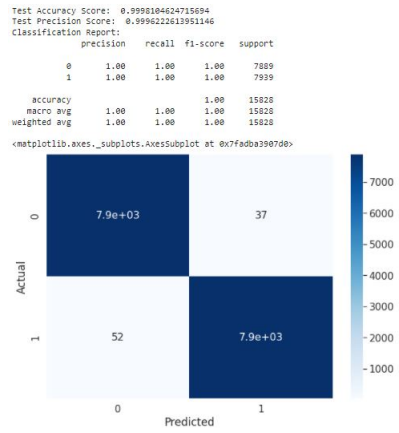

Fig. 8. Random Forest Model Torii Results

References

[1] Sebastian Garcia, Agustin Parmisano, & Maria Jose Erquiaga. (2020). IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. Zenodo. http://doi.org/10.5281/zenodo.4743746

[2] J. Brownlee, "Overfitting and Underfitting With Machine Learning Algorithms," Machine Learning Mastery, 12-Aug-2019. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. [Accessed: 13-Dec-2021].

[3] By: IBM Cloud Education, "What is Supervised Learning?," IBM. [Online]. Available: https://www.ibm.com/cloud/learn/supervised-learning. [Accessed: 13-Dec-2021].

[4] "What Is a Decision Tree?," Master's in Data Science. [Online]. Available: https://www.mastersindatascience.org/learning/introduction-to-machine-learning-algorithms/decision-tree/. [Accessed: 13-Dec-2021].
[5] By: IBM Cloud Education, "What is Random Forest?," IBM. [Online]. Available: https://www.ibm.com/cloud/learn/random-forest. [Accessed: 13-Dec-2021].
[6] N. Donges, "A Complete Guide to the Random Forest Algorithm," Built In. [Online]. Available: https://builtin.com/data-science/random-forest-algorithm. [Accessed: 13-Dec-2021].
[7] K. Markham, "Simple guide to confusion matrix terminology," Data School, [Online]. Available: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/. [Accessed: 13-Dec-2021].

APPENDIX

Google Colab Link: https://colab.research.google.com/drive/1iZ08W1gip3FtZlI56NcJ_HC-lnWH2La9?usp=sharing
Colab Notebooks Link: https://drive.google.com/drive/folders/1kLY4Z0PpG-JzM0lJ0HiNbORK3Jh9RhZ6?usp=sharing