Network Traffic Classification with Machine Learning

Joseph Coble
Department of Computer Science
Naval Postgraduate School
Monterey, California
joseph.coble@nps.edu

Alexander Huang
Department of Computer Science
Naval Postgraduate School
Monterey, California
alexander.huang@nps.edu

Seth Kyler
Department of Computer Science
Naval Postgraduate School
Monterey, California
seth.kyler@nps.edu

***Abstract*—The internet is inundated with malicious traffic that can cripple an organization that is unprepared for it. The U.S. Navy spends a lot of time investigating potentially malicious traffic on their networks everyday. Machine learning (ML) provides an opportunity to create a computationally based filter that learns daily what malicious data looks like. The ability of this technology enables the reduction of work the human capital of the U.S. Navy by effectively catching only the truly malicious traffic. This paper looks at multiple types of ML to find the one most effective at reducing the malicious traffic on a network while reducing the amount of human oversight.**

***Index Terms*—machine learning, supervised learning, unsupervised learning, classification, network traffic, anomaly detection**

I. Introduction and Background

The Navy operates large, sprawling enterprise computer networks with hundreds of thousands of users, and millions of hosts, servers, mobile devices, and other tactical-level endpoints and nodes. Network operators and network defenders often face the challenge of monitoring these networks for events including simple misconfigurations and policy violations, insider threat behavior, and up to malicious behavior from nation-state advanced persistent threats. With such a large number of devices across vast areas of responsibility, individual operators are often overwhelmed with the sheer number of incidents to investigate and triage from alerts generated by rule-based intrusion detection and prevention systems, a problem often termed "alert fatigue." Our motivation for this project was to examine if ML models could be developed to assist with alert fatigue, provide additional insight to operators about network and user behavior, and be a useful tool to enhance network operations and network defense.

The data set used in this project is the LUFlow data set from Lancaster University [1]. The university had placed multiple honeypots inside their address space to capture the data traversing through their networks [1]. This particular data set was started in June 2020 and is being updated daily [1]. It is open to the public for public use. We focused on the feature of predicting the label for each data packet, which indicates that each packet is either malicious, benign, or an outlier [1]. Malicious packets are characterized as those with ill intent; benign as those characterized as normal; and outlier as those characterized as indeterminable. These labels are assigned based on a Lancaster University program called "Tangerine," which conducts cluster analysis to label collected traffic.

II. Goals and Experiment Objective

Our primary experiment objective was to determine if ML methods would be able to produce a suitable aid for network operators and security personnel to identify malicious and outlier network traffic from benign network traffic. Analysis along this vein may also provide additional insight about network traffic. Our hypothesis was that machine learning models would be able to be utilized as an effective prediction tool to classify traffic and provide useful characteristics to assist with investigation. We had two primary goals to support the objective: 1) use supervised learning to train a model on the internet traffic of that day and apply that model to accurately predict the traffic of the next day, and 2) take network traffic labeled by an unsupervised learning model and create a model to predict the classification of new network traffic.

A. Supervised Learning

For supervised learning we trained and tested several multi-class models: a k-nearest-neighbors (k-NN) classifier, a logistic regression classifier, a random forest classifier, and a multi-layer perception neural network.

B. Unsupervised Learning

For unsupervised learning methods we explored using k-means clustering to determine patterns in the data, as well as isolation forest for anomaly detection.

III. Data Exploration and Cleaning

We primarily worked with two data sets, one from June 19, 2020, which we mainly used to train the data. The second data set, from June 20, 2020, was mainly used to test the data once a model had successfully trained on the June 19, 2020 data set. Using two successive days of data captures a more realistic deployment situation for the ML system in which the system would be trained for any given day using data collected from the previous day. Thus, maximizing performance on the test dataset for June 20, 2020 was prioritized over performance on the validation dataset split from the June 19, 2020 collection.

The June 19, 2020 data set includes 765,360 entries and the June 20, 2020 data set contains 770,853 entries. We commenced data exploration by generating a correlation matrix and the scatter plot matrix depicted in Figure 1. The scatter plot matrix indicated that there was not any particular linear correlation between the features.d The data set was mostly

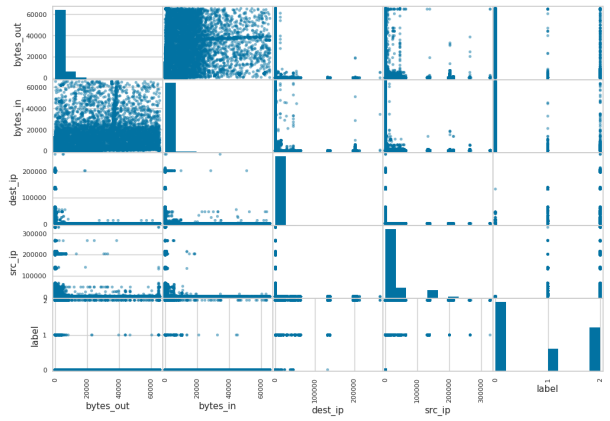

Fig. 1. Scatterplot matrix with selected features.

clean, with the exception of some instances that were missing entries for the source or destination port number feature. Because these instances were relatively few, we suspected missing data fields may have improperly influenced the "Tangerine" model from which the data set was produced. These entries were removed from the data set. To prepare for training and validation with the classification models, we used an ordinal encoder to encode the labels of "benign", "outlier", and "malicious" as numerical values 0, 1, and 2, respectively.

IV. Methodology and Experiments

We developed our models and conducted our experiments using the scikit-learn machine learning library. Tensorflow and Keras were also used for the multi-layer perceptron neural network. For all experiments, we used a standard 80%/20% split of the June 19, 2020 dataset into training and validation sets.

A. Supervised Learning

The goal of supervised learning was to train multi-class classification models to predict the label of "benign", "outlier", or "malicious." We considered binary classification, but decided a model that would only predict between "benign" and "non-benign" traffic would not be particularly useful in assisting with incident triage and reducing alert fatigue. A binary model in this case would conceivably generate alerts for both malicious and outlier traffic, and still require significant human operator analysis to properly triage malicious events from outlier traffic.

B. k-NN Classifier

The k-NN Classifier is a supervised learning algorithm that uses the closeness of a data point in relation to other data points to predict the classification of that point. K-NN stands for k-nearest neighbors, and the k is the placeholder for however many comparing data points are used. The model takes the distance from the neighboring points, which all vote on the point in question, and the majority vote wins. This model was also selected due to its ease of implementation. It would allow us to test the data from the repository and identify any issues that need to be cleaned. The hyperparameters we focused on with this model were the n-neighbors, weights, algorithm, and p-value. N_neighbors tells the classifier how many surrounding data points to use in determining the classification of the data point being classified [3]. When these data points have been identified, the distance between them and the point is calculated and passed back to the classifier. The weights parameter allows the scientist to manipulate the influence of the surrounding data points. Weights in scikit-learn are divided into "uniform," "distance," and a user-defined function [3]. The difference between "uniform" and "distance" is that uniform keeps all data points equal in terms of voting strength, whereas "distance" will give more voting power to data points closer to the test point [5]. The algorithm hyperparameter is how the nearest neighbors are calculated. The three algorithms in scikit-learn are "brute," "K-D Tree," and "Ball Tree [3]." The "brute" algorithm calculates the distance between all the data set points and the new point. This algorithm works better with smaller data sets and can be overwhelmed if the data set is too large [3]. The "K-D Tree" and "Ball" are similar in the way that they take generalized distances from the test point and data set points based on previous nearby data set point distances [5]. The "Ball" algorithm can deal with higher dimensional data sets by taking the data and putting them into nodes based on a centroid with a radius [5].

The P value is used to choose the distance formula desired, 1 for Manhattan distance and 2 for Euclidean distance. For this model, we started with the default hyperparameters and received very high accuracy (98.3%) on the test data from the same day. We then tested this trained model on the data from the following day and returned a 13% drop (84.7%). The model had a hard time identifying the outliers label correctly, generating an F-1 score of 29%. We then took this model through a hyper-tuner using gridsearchcv [6]. The hyper tuner returned the result "algorithm: brute, p=2, weights = distance, and n_neighbors= 3." This contradicted what we thought because we assumed that the "K-D Tree" algorithm would have scored better. The hyper-tuned model provided us a better same-day score (98.53%) but still gave us a next-day score in the mid-eighties (85%). This model had performed well compared to some of the other models we ran but was still outperformed by the Random Forest Classifier.

C. Logistic Regression Classifier

The second model we tested was the Logistic Regression Classifier which uses predictive analysis to classify what the data will be labeled. The predictive analysis uses the natural logarithm of odds to determine the probability of an event given independent variables in the data set [4]. The method provided by the scikit-learn library had multiple

	precision	recall	f1-score	support
0	0.99	0.99	0.99	366310
1	0.26	0.32	0.29	69389
2	0.84	0.81	0.83	335154
accuracy			0.85	770853
macro avg	0.70	0.71	0.70	770853
weighted avg	0.86	0.85	0.86	770853

Fig. 2. k-NN classification report for June 19.

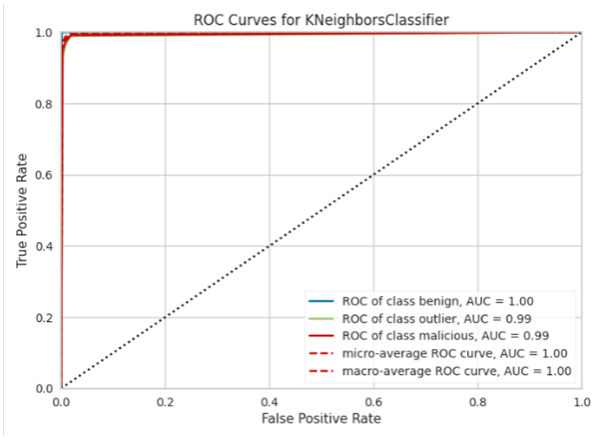

Fig. 3. k-NN ROC for June 19.

hyperparameters, but we focused on testing "penalty," "C," "solver," and "max_iter [5]." The penalty is used to impose a negative hit to the model's score if the model has too many variables that effectively regularize the model. We used two parameters, L1 and L2, which were two different formulas for the penalty calculation [5]. The C parameter is also used to impose regularization on the model, where the smaller the value, the more regularized the model. The solver had multiple parameters, "liblinear," "Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (LBFGS)," "sag," "saga," and "newton-cg [4]." Sag and saga use stochastic average gradient descent [4]. Liblinear uses coordinate descent that is not well suited for multi-class modeling and can lead to extensive run times [2]. Newton-CG and LBFGS are similar algorithms where LBFGS is better for smaller data sets.

The final parameter was max iterations, which limits the maximum number of times the model could iterate before forced to pick a label. If this number is too low, it could fail to converge, and the accuracy could suffer. It it is too large, it can take too much time and too many resources to finish. The scoring also indicates that dramatically increasing the iterations does not always provide better accuracy; it can plateau after a certain point. We tested the model on both the same day data as well as data of the next day, and both scores were worse than the k-NN model tested on data from the next day data (83% and 53% respectively). This was after the hyper tuner, which used the "gridsearchcv" class, returned that the best model ran with the parameters 'C': 100, 'max_iter': 2000, 'penalty': 'l2', 'solver': 'lbfgs' [6]. This model had minimal

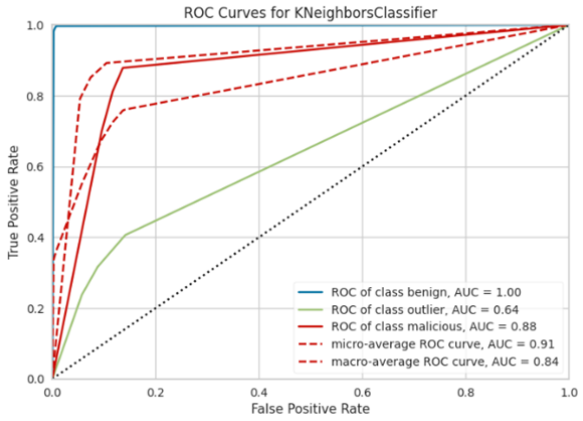

Fig. 4. k-NN ROC for June 20.

changes when exposed to a wide range of C and max iteration values. In the end, we decided that this model was not suited for our data.

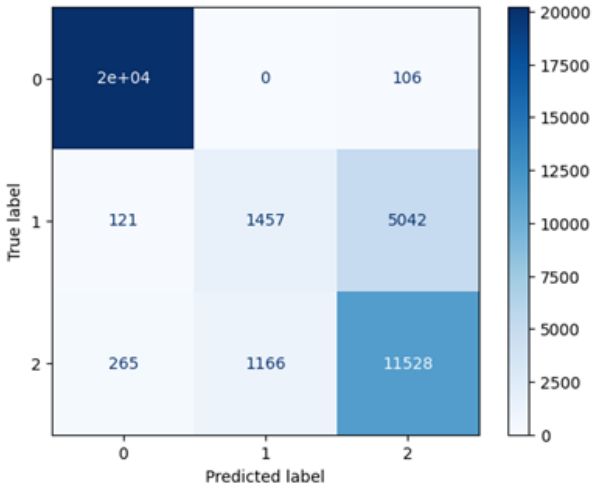

Fig. 5. Logistic regression model confusion matrix for June 19.

D. Random Forest Classifiers

The third model tested was a Random Forest Classifier. Our hypothesis was that this model might have superior performance to the other models tested due to the ensemble model nature of the random forest classifier [10].

Initially, the model was provided only limited constraints to find how well or poorly the model would overfit. The maximum depth was set to 30, minimum samples per split was set to 4, and minimum samples per leaf was set to 4. The resulting trees severely overfit the training data, and the graphical representation produced by the graphviz library was vast enough to be intractable. After tuning the hyperparameters, including adding maximum leaf nodes as an additional hyperparameter to regularize the model, we found a far better performing model. The maximum depth remained 30, minimum samples per split and samples per leaf were

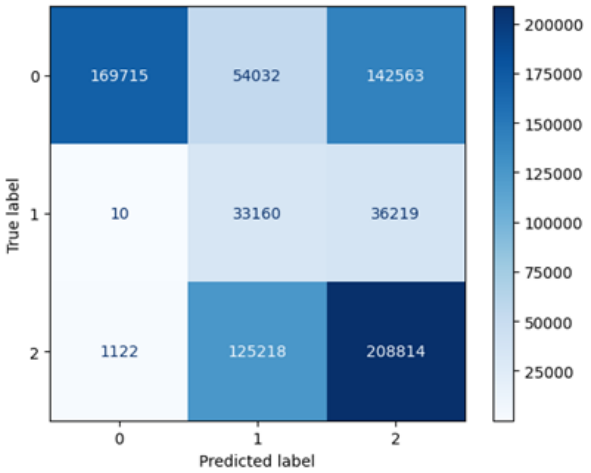

Fig. 6. Logistic regression model confusion matrix for June 20.

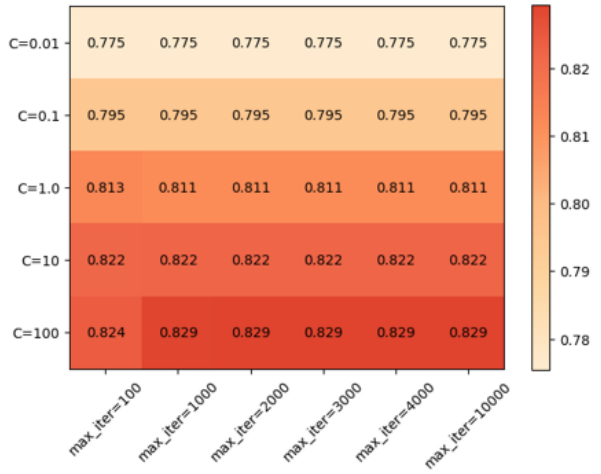

Fig. 7. Logistic regression model max iterations v.s. C value.

100, and maximum leaf nodes was set to 16. An example tree is depicted in Figure 8. Values higher or lower than 16 for maximum nodes began to exhibit lesser performance by accuracy score. The resulting model had an accuracy score of 0.90 on the validation data set for June 19, 2020 and an accuracy score of 0.89 on the next-day data set for June 20, 2020. Overall performance for true positives against false positives is represented in a receiver operating characteristic (ROC) curve in Figure 9.

E. Multi-Layer Perceptron

A multi-layer perceptron (MLP) is a type of artificial neural network that uses layers of neurons, which activate weighted input values with a specified activation function, to produce an output either to the next layer of neurons, or to produce a prediction [10] [11]. We also attempted training a multi-layer perceptron, with the intuition that a neural network approach may have better performance due to the possible non-linear

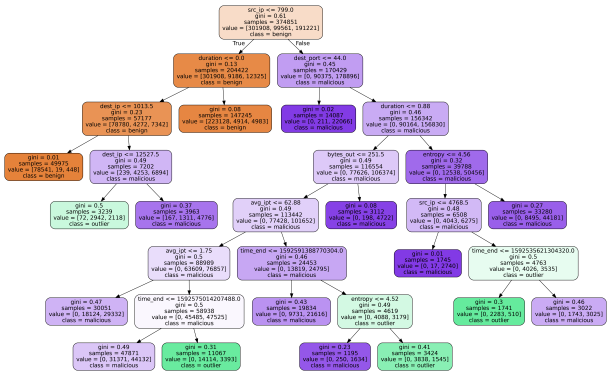

Fig. 8. Sample decision tree from random forest model.

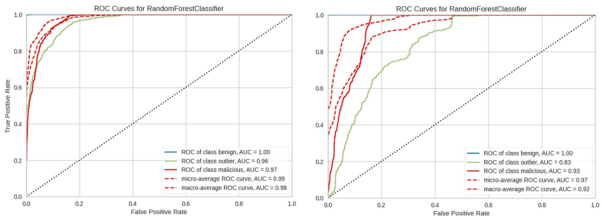

Fig. 9. Random forest classifier ROC curve for June 19 and 20.

correlations observed during data exploration for many of the features.

Initially, we tried a deeper model with several layers. We found immediately that this model significantly overfit the training data. It performed well on the June 19, 2020 data, including the 20% validation data set split from the training set, but extremely poorly on the June 20, 2020 data. Due to the expected use case and our goals for this model to be effective on next-day predictions based on training from the previous day, we decided this approach was not appropriate to meet our goals. To fight overfitting, we decreased the number of

```
15
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 300)               4800

 dropout (Dropout)           (None, 300)               0

 dense_1 (Dense)             (None, 200)               60200

 dense_2 (Dense)             (None, 3)                 603

=================================================================
Total params: 65,603
Trainable params: 65,603
Non-trainable params: 0
```

Fig. 10. MLP model summary.

layers to two dense layers and the number of neurons per layer. The initial MLP without dropout regularization, with learning rate of 0.001, and with the stochastic gradient descent (SGD) optimizer performed better than the deep model, but we found further improvements in performance by using the Adam optimizer and adding a dropout layer of 0.5 between

two dense layers. The model summary is depicted in Figure 10. The model had an accuracy score of 0.95 on the June 19, 2020 validation data set, and an accuracy score of 0.86 on the June 20, 2020 data set, which we found to be much more acceptable than our results form the k-nearest neighbors classification model and the logistic regression model.

F. Unsupervised Learning

The goal of using unsupervised learning was to see what pattern, if any, the algorithm found that could help identify malicious packets. Ideally, the unsupervised algorithm would find a different pattern than used by any of the supervised learning algorithms, providing both an additional method of analysis for malicious packets and an effective method to scrutinize unlabeled data.

G. Clustering and K-means

K-means, as a general unsupervised learning algorithm, seemed a good place to start. Just as with the supervised learning algorithm data, we took out the null values and scaled the data using Standard Scaler. One Hot Encoding was used to label the packets as "benign," "outlier,", or "malicious." This category will hereafter be referred to as "Labels." By plotting Labels against first the Source IP Address and then the Destination IP Address, we hoped to find a pattern for which types of packets (benign, outlier, or malicious), tended to come from or go to which IP Addresses.

Plotting the data using K-means did not produce even clusters, which is ideal for K-means [10]. In fact, no clusters really existed, just three horizontal lines running across the screen, which correspond to the three Label categories (benign, outlier, or malicious), Figures 11 and 12. K-means does not separate the plotted data along those lines as one might expect. Rather, it drew the lines between the data points closest to the y-axis since the majority of the IP addresses used lower IP address numbers. The Source IP and Labels K-means plot illustrates this more distinctly because Source IP only had a small sample towards the y-axis on the lowest horizontal stream of plotted data. Regardless, k-means did not find a suitable method to separate the plotted data, even with the adjustments to the value of k discussed below. Running inertia values on the feature combinations showed both plots placed the elbow at k = 3, Figure 13 [7]. Although inertia and k are inversely proportional and thus inertia cannot be used to select the value of k, the inertia did show that increasing k beyond 3 would return only marginally better results [10]. Upon analyzing all values for k = 1 – 99, the processor returned a best value of k = 99. This further illustrates that k-means could not find distinct clusters in the plotted data. Successfully calculating the silhouette score and graph would have shown the best value for k, however, Jupyter Notebook was still processing the silhouette score after 7.5 hours. Given that k-means clustering was not producing the results hoped for, we did not pursue the silhouette score and graph further. It is unlikely receiving a silhouette score and graph would have given us the desired clustering anyway. We did try

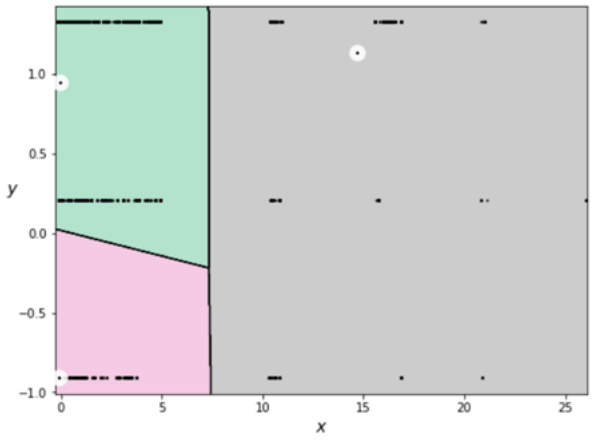

Fig. 11. Destination IP and Labels with k = 3

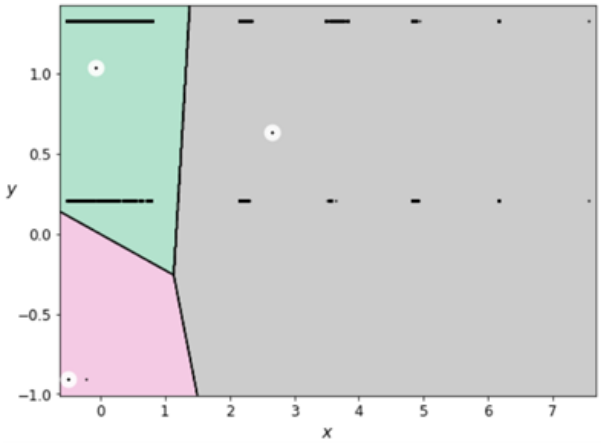

Fig. 12. Source IP and Labels with k = 3

different values for k, notably 4-6, and while these values returned tighter clusters, they did so closer to the y-axis. Thus, increasing k did not cluster the data in a way that would help identify malicious packets. The poor results produced by k-means are not surprising considering k-means is best suited for data with even cluster sizes, and our data does not have even clusters [10].

H. Anomaly Detection and Isolation Forests

Isolation Forest seemed more promising as it gives each cluster an anomaly score, which can help detect anomalies [8]. If malicious packets came from or went to unusual IP addresses, Isolation Forest stands a chance of identifying that pattern. Once plotted, however, and like the k-means plots, the isolation forest plots show three streams of data running parallel to the x-axis, representing the three different Labels (benign, outlier, or malicious), Figures 14 and 15. Most of the data being plotted near the y-axis is reflected in the isolation forest graphs by the shades of blue grower darker as the x-axis increases [9]. This plot also does not provide a good

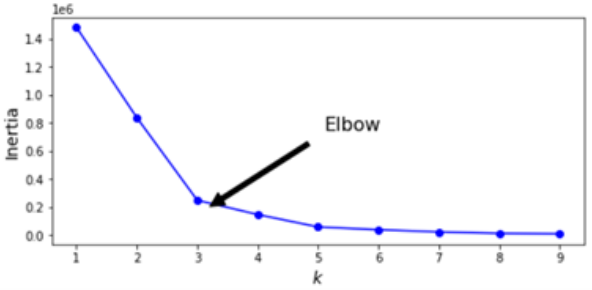

Fig. 13. Inertia graph for Destination IP and Labels. Inertia graph of Source IP and Labels also had the elbow at k = 3.

indication of malicious packets, so isolation forest does not provide a much better analysis of the data than k-means did.

Interestingly though, on the Source IP plot, the data points in the bottom left of the graph, which k-means plots as a very small cluster, isolation forest expands in a distinct horizontal line. Although this horizontal line looks to be a similar length to the two above it, the isolation forest plot shows the distinction through the top right of the plot being shades darker than the bottom right. Our hypothesis is that the lowest horizontal line of plotted data, which represents benign packets, is more centrally clustered than the packets of outlier or malicious packets because the same IP address, or IP addresses, consistently sent benign packets. Other, perhaps less trusted, IP addresses produced more of the outlier and malicious packets.

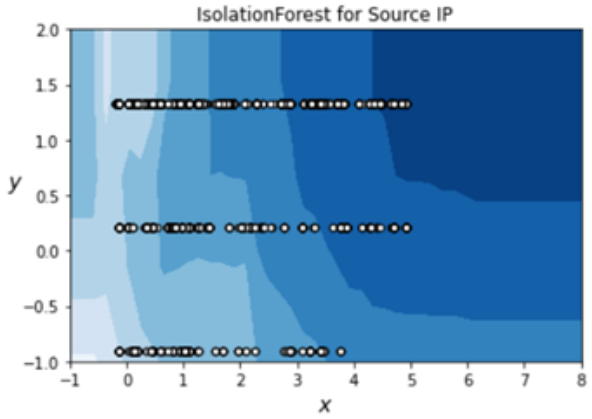

Fig. 14. Isolation Forest plot for Destination IP and Labels.

I. Other Unsupervised Learning Algorithms

Other unsupervised learning algorithms were tried but did not run to completion due to a RAM limitation on Jupyter Notebook. Those algorithms included affinity propagation, spectral clustering, agglomerative clustering, and DBSCAN. Affinity propagation and DBSCAN in particular looked promising because they can analyze uneven cluster sizes [10].

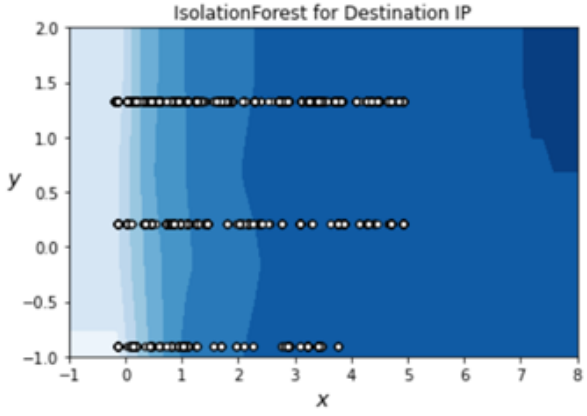

Fig. 15. Isolation Forest plots for Source IP and Labels.

V. Conclusions

Of the supervised learning methods tested, the random forest classifier had the best performance on the June 20, 2020 data set, representing next-day performance from a model trained with previous-day data. We believe this result supports the acceptance of our original hypothesis that supervised classification with ML models can assist operators with incident triage and reduce alert fatigue.

Neither of the unsupervised learning models tested supported the hope that an unsupervised learning model would find an alternative method to detect and classify data packets. However, as the realm of unsupervised learning is extraordinarily vast and research has barely scratched its surface, this conclusion only applies to this experiment and is far from conclusive.

VI. Future Work

Due to time and scope constraints, further analysis of the LUFlow data set may be warranted. Some specific areas for future investigation are as follows. Further research into suspected bias in the Tangerine Classifier that was used. The removal of their model and labeling altogether might create an opportunity to build an in-house model that provides more insight into how the labels were decided. A deeper dive into feature engineering might increase the amount of outlier class data points in relation to malicious and benign which could help with future model accuracy as more training data would be available for the most critical label.

It would be exciting if one of the untested unsupervised learning algorithms could identify a pattern to help identify malicious packets. Due to memory constraints, this future work will need to be undertaken by someone with more powerful tools at their disposal.

A. Similarity with Thesis Work

This project was similar to Alexander Huang's thesis work with regard to classification of network traffic data. The data set used for this project was publicly available and will not be the same data set used for thesis work. Additionally,

the nature of the classification labels differ, as the thesis work seeks to classify firewall alllow and block decisions instead of the labels derived from a network security monitor, and incorporates directionality (inbound or outbound) as an additional consideration.

REFERENCES

[1] R. Mills, LUFlow: Data Traffic Capture, Lancaster University. 2020 [Online]. Available: https://github.com/ruzzzzz/LUFlow

[2] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. 2008. [Online]https://www.csie.ntu.edu.tw/ cjlin/papers/liblinear.pdf

[3] "KNN Classifiers" Scikit learn. Accessed December 9, 2022. [Online].Available: https://scikitlearn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[4] "Logistic Regression Classifiers" Scikit learn. Accessed December 9, 2022. [Online]. https://scikit-learn.org/stable/modules/ generated/sklearn.linear_model.LogisticRegression.html

[5] "Supervised Learning" Scikit learn. Accessed December 9, 2022. [Online].https://scikit-learn.org/stable/supervised_learning.html

[6] "Tune hyperparameters for Classification Machine Learning Algorithms," J. Brownlee, Accessed December 9, 2022. [Online]. Available: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

[7] "KMeans," Scikit learn. Accessed December 9, 2022. [Online]. Available: https://scikit-learn.org/stable /modules/generated/sklearn.cluster.KMeans.html

[8] "IsolationForest," Scikit learn. Accessed December 9, 2022. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[9] "IsolationForest example," Scikit learn. Accessed December 9, 2022. [Online]. Available: https://scikit-learn.org/stable/ auto-examples/ensemble/plot-isolation-forest.html

[10] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow. Sebastopol, CA: O'Reilly Media Inc., 2017.

[11] A. Abraham, "Artificial Neural Networks," Handbook of Measuring System Design, P. H. Sydenham and R. Thorn ed. New York, NY:John Wiley and Sons, Ltd, 2005.