DATS6301 - Project Report

William Arliss, warliss@gwu.edu

June 2022

1 Introduction

This project is focused on detecting network intrusions in server telemetry data. This is a classification task where each observation is a collection of measurements from a given network flow. Each flow is either identified to be "malicious" or "benign". This problem was selected out of personal interest.

The rest of this report is structured as follows. Section 2 describes the data, section 3 describes the machine learning model used, section 4 describes the experimental setup, and section 5 describes the results. Section 6 summarises the project and concludes the findings.

2 Data

The data used in this project come from the "LUFlow Network Intrusion Detection Data Set" (Lancaster-University, 2022). This is a publically available dataset created by Lancaster University for the purpose of researching "detection mechanisms suitable for emerging threats". The data are available through Kaggle or a GitHub repository. The dataset is composed of 9 months of network flow data from 2020-2021 amounting to roughly 150 million observations. Due

to limited computing resources and time restrictions, only a small sample (1%) of the data are used —this amounts to 1,441,018 observations used for training and 337,186 held out for testing.

The data come as a CSV file with 16 columns as described in table 1. Each row of the data is labeled as "benign", "malicious", or "outlier". For the purposes of this project, all observations labeled as "outlier" were dropped outright. Some additional feature engineering is done to derive "dayofweek" and "timeofday" variables corresponding to the day of the week and the time of the day from the "time_start" column.

Table 1: Raw Data Fields

"avg_ipt"	Average inter-packet arrival time
"bytes_in"	Number of bytes from source to destination
"bytes_out"	Number of bytes from destination to source
"dest_ip"	Destination IP address
"dest_port"	Destination port
"entropy"	Entropy in bits per byte of each data field
"num_pkts_out"	Number of packets from destination to source
"num_pkts_in"	Number of packets from source to destination
"proto"	Protocol number associated with the flow
"src_ip"	Source IP address
"src_port"	Source port
"time_end"	Epoch time of end of the flow
"time_start"	Epoch time of start of the flow
"total_entropy"	Entropy in bytes of all data fields
"duration"	Duration of the flow
"label"	"Benign", "malicious", or "outlier"

The "src_ip" and "src_port" features can be concatenated into a new "src" feature. Similarly, the "dest_ip" and "dest_port" features can be concatenated into a new "dst" feature. Using these two features, a graph can be constructed such that each edge in the graph represents a flow from a "src" node to a "dst" node. Because each edge would represent a row in the raw data, they can carry weights in correspondence to the columns of the raw data. A sample of such a graph is displayed in figure 1. The graph structure of the data is not exploited

in the modeling process described here, but is intended for use in future work.

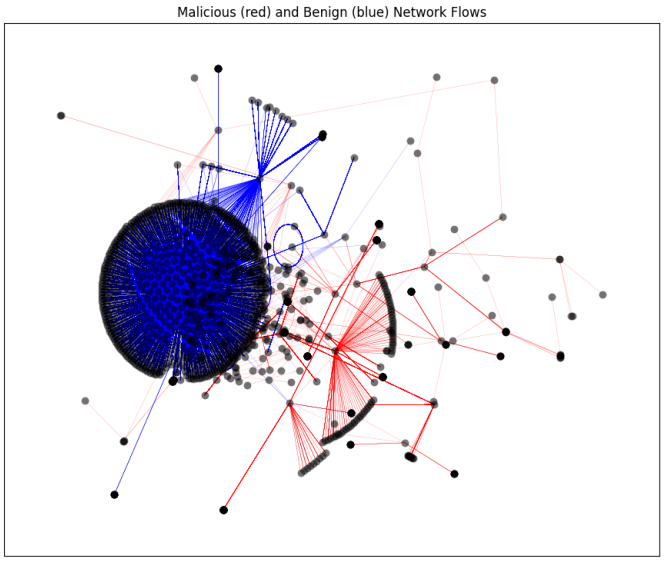

Figure 1: Sample of graph layout

3 Modeling

The modeling process for this project leverages decision trees and ensembling. The decision trees are fit using the CART algorithm (Breiman et al., 1984). CART works by greedily partitioning the input data in a recursive fashion, such that at each split a cost function is minimized. Splits are conducted by identifying the feature in the data with the most information gain. Information gain is defined as the entropy of the parent node (split) less the weighted average of the child nodes (resulting splits).

$$\text{information gain} = \text{entropy}([\text{parent}]) - \sum_i w_i \ \text{entropy}([\text{child}]_i) \tag{1}$$

Entropy is computed from class probabilities. Here, probability p_c is defined as the number of of samples in a split corresponding to a given class c divided by the total number of samples in the split.

$$\text{entropy} = \sum_{c \in \text{classes}} -p_c \log p_c \tag{2}$$

Alternatively, information gain can be computed by substituting Gini for entropy. Gini is also computed from class probabilities.

$$\text{gini} = 1 - \sum_{c \in \text{classes}} p_c^2 \tag{3}$$

CART recursively divides the feature space by maximizing information gain at each binary split until a minimum number of samples (1 for example) remain in each child node.

Decision trees are used as component models for larger ensembles in this project. One type of ensemble used is bagging. Bagging entails fitting many models on different subsets of the training data in order to create a more robust predictor. The other type of ensemble used is boosting. Boosting entails sequentially fitting models such that successive models depend on the performance of previous models.

One technique for boosting —gradient boosting —involves fitting successive models to the negative loss gradient of their predecessors. The full model F is the recursive sum of the component models h_m. The total number of component models is a user-defined hyper-parameter. At each step m in the algorithm, the model is defined as

$$F_m = F_{m-1}(x) + h_m(x). \tag{4}$$

Here, h_m is found by minimizing a loss function L.

$$h_m = \underset{h}{\operatorname{argmin}}\ L(y, F_{m-1}(x) + h(x)) \tag{5}$$

It can be shown that this loss can be approximated as

$$L(y, F_{m-1}(x) + h(x)) \approx L(y, F_{m-1}(x)) + h_m(x) \left[\frac{\partial L(y, F_{m-1}(x))}{\partial F_{m-1}(x)} \right] \tag{6}$$

and thus the minimizer h_m can be approximated as

$$h_m \approx \underset{h}{\operatorname{argmin}}\ h(x) \left[\frac{\partial L(y, F_{m-1}(x))}{\partial F_{m-1}(x)} \right] \tag{7}$$

which is solved by fitting the model h_m (a decision tree in this case) to the negative gradient of the loss of the preceding model F_{m-1}.

Another technique for boosting —Adaboost —involves sequentially training "weak" models with instance weights that are proportional to errors made previously. Here, the weights being applied to observations/instances are taken from a distribution of weights that is updated at each iteration so that observations that are "hardest" to predict have higher weights in training. The full model F is the weighted sum of the component models h_m. The number of component models is user-defined as the hyper-parameter T.

$$F_m = \sum_{m=1}^{T} \alpha_m h_m(x) \tag{8}$$

Here, α_m is proportional to weighted prediction errors and defined as

$$\alpha_m = \frac{1}{2}\left(\frac{1-\epsilon_m}{\epsilon_m}\right) \tag{9}$$

where the error rate ϵ_m weighted by distribution D is

$$\epsilon_m = D(i)\, I(y_i \neq h_m(x_i)). \tag{10}$$

After an estimator/model is fit, the weight distribution is updated following

$$D_{m+1} = \frac{D_m(i)\, e^{-\alpha_m y_i h_m(x_i)}}{Z_m} \tag{11}$$

where Z_m is a constant for normalizing the distribution and $y_i \in \{-1, 1\}$. By successively fitting "weak" estimators and updating the instance weight distribution, the full model F adapts to the "difficult to predict" observations.

This project uses decision trees as component models for boosting and bagging, as described in section 4.

4 Experimental Setup

Three different ensembling techniques for decision trees are tested during the modeling process in this project: random forest (Breiman, 2001), Adaboost (Freund and Schapire, 1997), and gradient boosting (Friedman, 2001). Additionally, soft voting is used to combine models for a final estimator. The Scikit-Learn library in Python is used to implement each algorithm.

Model performance was measured by Receiver Operating Characteristic (ROC) —specifically the area under the curve (ROC-AUC) —and recall. These metrics are chosen instead of simple accuracy because they are better suited for tasks where class imbalance is a problem. The data used in this project contain roughly 62% "benign" samples and 38% "malicious samples". Recall (true-

positive rate) is a metric of interest because failing to catch positives (malicious flows) is more dangerous in the context of network intrusion than failing to pass negatives.

Three hyper-parameters are optimized through cross-validated grid-search: learning rate (Adaboost and gradient boost), maximum tree depth (random forest, Adaboost, and gradient boost), and instance weights (random forest and Adaboost). Each ensemble technique uses 100 "weak" estimators. Learning rate is meant to control the magnitude of the update taken when fitting sequential estimators for boosting models. Updates must be large enough to make progress toward a local minima of the loss function, but not so large that the minima is overshot. Maximum tree depth is meant to prevent overfitting of any estimator. Trees must be allowed to grow enough that they extract meaningful partitions, but not enough that they fit too precisely to the training data. Instance weights are meant to combat the class imbalance problem. Giving more weight to the minority class and less weight to the majority class can help to balance the learning objective.

The data used in grid-search is a subset of the features described in section 2. This subset includes "avg_ipt", "bytes_in", "bytes_out", "entropy", "num_pkts_in", "num_pkts_out", "proto", "total_entropy", "duration", "dayofweek", and "timeofday". The "proto" and "dayofweek" variables are one-hot encoded. The non-categorical features are centered and scaled as part of the processing for the gradient boosting model.

Grid-search is used to identify the best model/hyper-parameter combination among the 3 candidate algorithms and the hyper-parameter grid. The model with the highest score (either ROC-AUC or recall) is then refit on the entire training dataset using the corresponding hyper-parameters.

5 Results

There are three different models evaluated on the holdout data in this project. The first was tuned according to the ROC-AUC metric and achieves better performance on the benign flows, the second was tuned according to the recall metric and achieves better performance on the malicious flows, and the third combines the previous two to achieve a slightly more balanced performance.

As in training, the models are evaluated on ROC-AUC and recall. Additionally, accuracy is measured, confusion matrices are shown, and the ROC is plotted. In the context of this problem, recall (true-positive rate) represents the percentage of malicious network flows that were correctly identified by the model.

The confusion matrices show the true-positive rate and true-negative rate along the diagonal as well as the false-positive and false-negative along the off-diagonal. Each cell is shaded according to a color gradient commensurate to the rate. The ROC plots show the false-positive rate plotted against the true-positive rate evaluated at different thresholds on the probability prediction. A perfect score would be plotted as a right angle pointed toward the upper left corner of the figure. A diagonal line would correspond roughly to random guessing. Summing the area under this curve yields the ROC-AUC score. The optimal probability threshold for prediction is marked as a red dot in the plots —this is the threshold with the shortest Euclidean distance to the point $(0, 1)$, or a perfect score.

Table 2: Evaluation metrics

	Model 1	Model 2	Model 3
Accuracy	0.934	0.894	0.928
ROC-AUC	0.991	0.971	0.987
Recall	0.905	0.980	0.905

The first round of grid-search was conducted using ROC-AUC as the valida-

tion metric. The Adaboost model with a maximum tree depth of 6, learning rate of 0.1, and no weighting achieved the highest score. The results of this model on the holdout set are shown in the first row of figure 2. It can be seen that the true-positive rate (recall) is 88% and the true-negative rate is 99%. This corresponds to an accuracy of 93% as seen in the first column of table 2. Having the true-negative rate be so much higher than the true-positive rate is not ideal for the use-case of network intrusion detection. The model's true-positive rate —i.e. the proportion of malicious flows correctly identified —is likely of more interest to the security analyst.

The next round of grid-search was conducted using recall score as the validation metric. The Adaboost model once again achieved the highest score, this time with a maximum tree depth of 4, learning rate of 0.001, and balanced weighting. The results of this model on the holdout set are shown in the second row of figure 2. It can be seen that the true-positive rate is now 98% and the true-negative rate is 81%. This corresponds to an accuracy of 89% as seen in the second column table 2. The recall of this model was indeed much higher, but it came at the cost of a lower true-negative rate and consequently lower accuracy and ROC-AUC. The true-positive rate being so much higher than the true-negative rate here could be caused by an over-correction from the balanced sample weighting.

Combining these two models proved to somewhat level out the true-negative and true-positive rates. A soft-voting ensemble was used to combine the predicted probabilities of both models. The results of this combined model on the holdout set are shown in the third row of figure 2. It can be seen that the true-positive rate is now 91% and the true-negative rate is 95%. This corresponds to an accuracy of 93% as seen in the third column of table 2. Combining the two previous models resulted in more balanced true-positives and true-negatives.

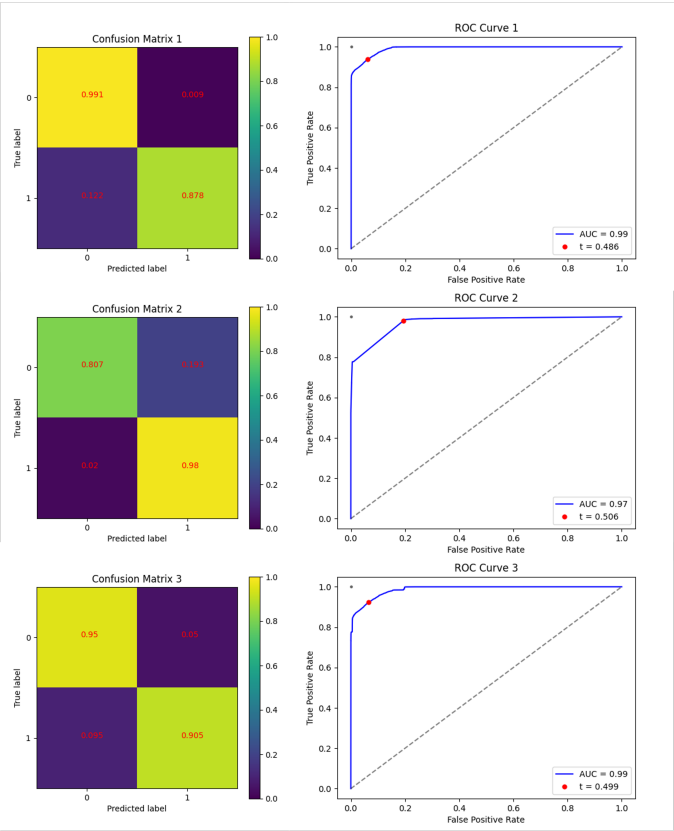

Figure 2: Modeling results - confusion matrix and ROC curve

There are trade-offs to each of the three models evaluated here. The second model appears to catch the most malicious network flows, but also seems to incorrectly classify many benign flows as malicious. The first model correctly classifies most of the benign flows, but does not perform as well on the malicious flows. The third model does decently well on both malicious and benign flows, but leaves much room for improvement.

6 Conclusion

In summary, boosted decision trees seem to have done well at identifying malicious network flows (or network intrusions) in the LUFlow dataset. The full robustness of this performance is yet to be observed, but can but can be tested with a larger sample of data —as stated in section 2, only 1% of the available data were used for training and testing.

There are several improvements that can be done to the modeling, including training at a larger scale, using a more exhaustive hyper-parameter tuning process, and exploiting the network structure of the data.

The network structure of the data can be used in a number of ways. Distributional features of the nodes and edges can be used to enrich the data, the problem can be reframed as one of link prediction, flows can be encoded as embedded vectors using Node2Vec (Grover and Leskovec, 2016), or graph neural networks can be applied.

This project opens the door to many learning opportunities in machine learning and graph analysis. In future work, better modeling performance and more insights on the structure of the data may be gained.

References

Breiman, L. (2001, 10). Random forests. *Machine Learning 45*, 5–32.

Breiman, L., J. Friedman, C. Stone, and R. Olshen (1984). *Classification and Regression Trees*. Chapman and Hall/CRC.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*(1), 119–139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics 29*(5), 1189 – 1232.

Grover, A. and J. Leskovec (2016). node2vec: Scalable feature learning for networks.

Lancaster-University (2022). Luflow network intrusion detection data set. Technical report, Lancaster University.