Cyber Security Anomaly Detection Model Training: Intrusion Detection

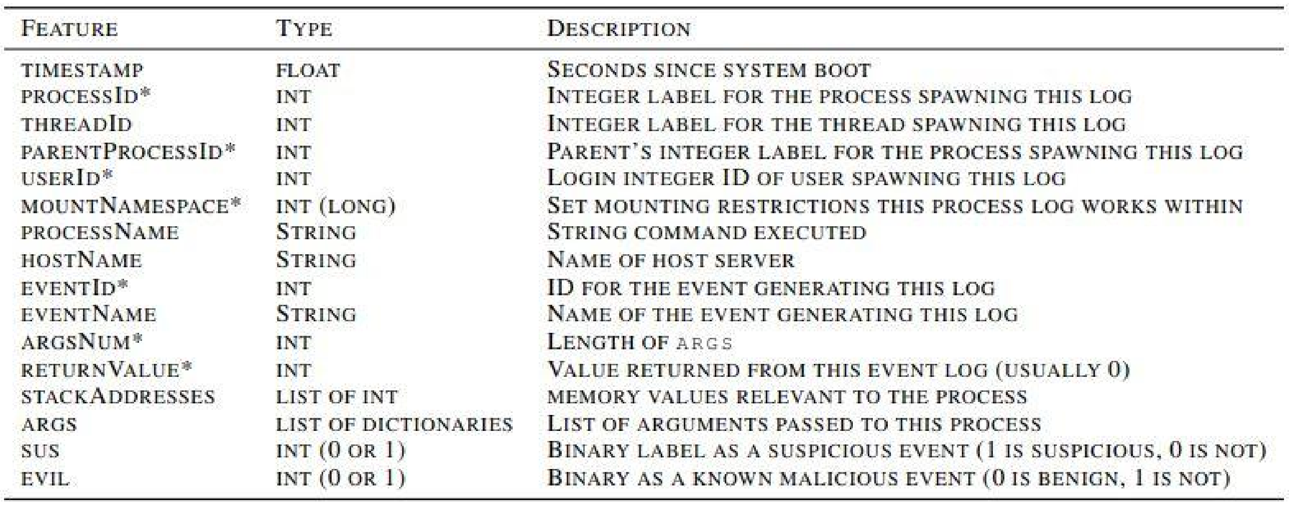
James Grandin

Computer Science - Predictive Analytics, Austin Peay State University
E-mail: jgrandin1@my.apsu.edu

Abstract

Cyber Security has long been an unbalanced battle between security professionals and a seemingly never-ending stream of faceless attackers. As technology continues to grow exponentially, so do the methodologies created to exploit vulnerabilities within these technologies. It's virtually impossible to eliminate all computer system vulnerabilities as they often go unnoticed until they are exploited in the wild (these are known as "zero-day vulnerabilities"), allowing attackers to gain command and control of your computer system. Instead, it may be more useful to focus on recognizing when an attack is underway and preventing bad actors from accomplishing their goals. This can be accomplished by training anomaly detection models to classify and alert on unusual activity in a network or on a host. In the industry, these models are marketed as Intrusion Detection Systems. In this report, we used the BETH Cybersecurity dataset to train various models using some of the most common anomaly detection algorithms (Local Outlier Factor, One-Class SVM, and Isolation Forest) and investigated their performance. After comparing model performance, we determined that the Isolation Forest algorithm produced the best model at the lowest computational cost. After hyper-parameter tuning, we achieved a Precision score of 0.997, a Recall Score of 0.850, and an F1 Score of 0.918. While the Recall Score and False Negative rate of 13.2% left something to be desired, the performance of this model is very promising. It shows that anomaly detection models can be a powerful tool in mitigating the risk of Cyber Attacks on organizations when paired with security best practices.

Table 1. The type and description of each feature in the Beth Dataset



Feature	Type	Description
timestamp	float	Seconds since system boot
processId*	int	Integer label for the process spawning this log
threadId	int	Integer label for the thread spawning this log
parentProcessId*	int	Parent's integer label for the process spawning this log
userId*	int	Login integer ID of user spawning this log
mountNamespace*	int (long)	Set mounting restrictions this process log works within
processName	String	String command executed
hostName	String	Name of host server
eventId*	int	ID for the event generating this log
eventName	String	Name of the event generating this log
argsNum*	int	Length of args
returnValue*	int	Value returned from this event log (usually 0)
stackAddresses	list of int	memory values relevant to the process
args	list of dictionaries	List of arguments passed to this process
sus	int (0 or 1)	Binary label as a suspicious event (1 is suspicious, 0 is not)
evil	int (0 or 1)	Binary as a known malicious event (0 is benign, 1 is not)

Note. Reprinted from "BETH Dataset: Real Cybersecurity Data for Anomaly Detection Research" by Highnam, Arulkumaran, Hanif, and Jennings, 2021. *Gatsby Computational Neuroscience Unit*, p. 3

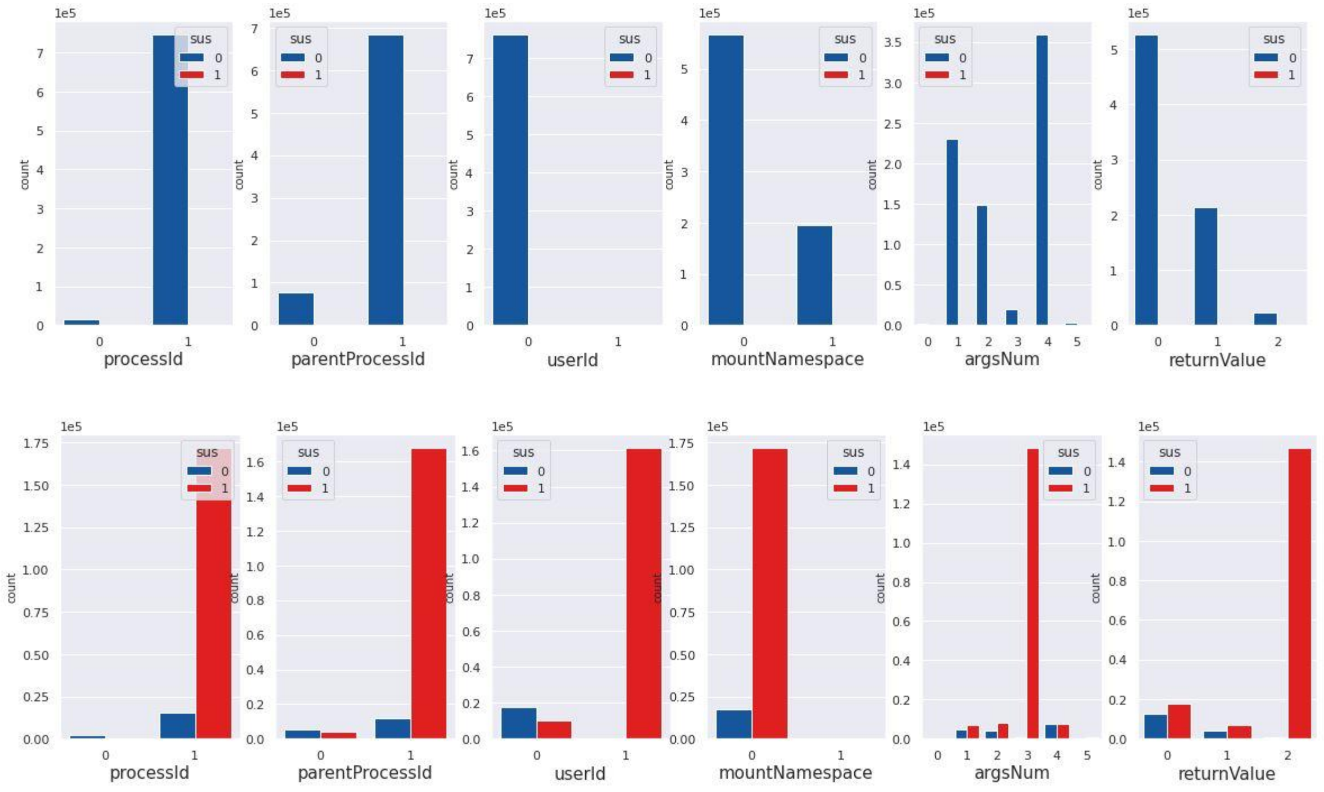
1. Introduction

As technology continues its ever-moving march towards progress, nefarious cyber criminals follow suit. Advancements in malicious techniques to exploit vulnerabilities in a network or on a host far outpace the progress made on the defensive side. Instead of relying on outdated heuristics and manpower to stem the flow of attacks, some organizations have turned to machine learning to provide a solution. While ML techniques can be used to combat a broad scope of problems in the cyber security industry, one of the most critical use cases is in anomaly detection.

In the cyber security industry, anomaly detection systems are often known as Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS) if response capabilities are included in its functionality. Historically, these systems relied on predefined signatures of well-known network threats to determine if an attack was underway. More modern versions can use machine learning to identify when novel attack techniques and patterns are being deployed. This can allow cyber security professionals to stay ahead of the curve and respond to active threats. With machine learning, the anomaly detection model will learn what activities on a system are normal and what activities are anomalous. If anomalous network or system activity patterns are detected, the system can alert or prevent the activity, giving security professionals the ability to intervene before irreparable damage is done.

There are many supervised and unsupervised learning techniques that can be used to train a model to accomplish this goal. This report will focus on comparing three common unsupervised learning techniques (Local Outlier Factor, One-Class SVM, and Isolation Forest) to find a best-fit anomaly detection model trained on the BETH Cyber Security Dataset. The BETH dataset is a fully labeled, structured dataset with a record of over 8 million real events collected across 23 honeypots (purposely vulnerable computer systems). This dataset is one of the most robust and recently produced datasets (2021) of its nature, which should allow for the creation of a high-performing model.

Figure 1. Count Plots of each feature after processing grouped by "Sus" label for Training (top) and Testing (bottom) datasets



2. Exploratory Data Analysis

The training and testing datasets used for analysis and fitting models are actually 10 percent subsamples of the entire dataset specifically formed for anomaly detection research. The training set has a minimal ratio of positive labels (761875 negative and 1269 positive) to simulate a baseline environment with few anomalies. Conversely, the testing dataset has a higher percentage of positive labels, though still imbalanced (171459 negative and 17508 positive), to better test the performance of fitted models.

In each dataset, two possible labels are available: Sus and Evil. The Sus feature is a binary variable that researchers involved with the creation of the dataset hand labeled as suspicious activity that is worthy of investigation (1) or normal activity (0). The Evil feature is a binary variable that signals fully verified attacks on the network (1) and normal activity (0). In industry, organizations are often interested in being notified of abnormal activity by users, even if the actions end up being benign, so we will use the "Sus" label to test the performance of our models.

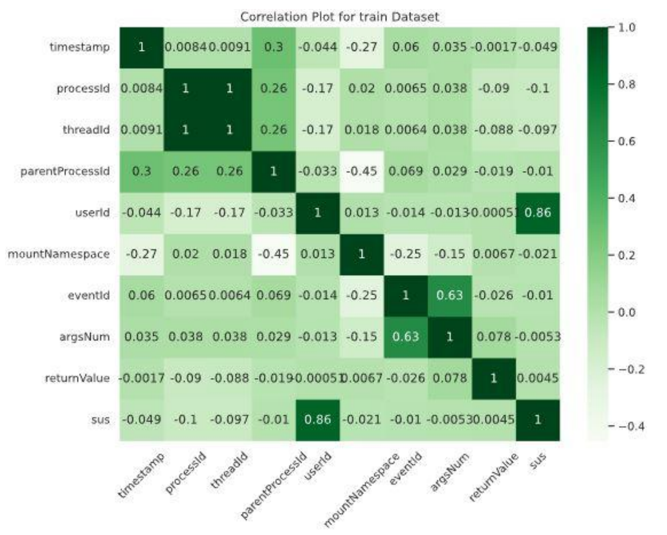
During the analysis phase, we used different techniques to gain key insights into the dataset that will help improve our understanding of the data and the performance of the trained anomaly detection models. Because this dataset was curated, not much cleaning was necessary. None of the observations were seen to have missing values for any of the features, so data imputation was also unnecessary.

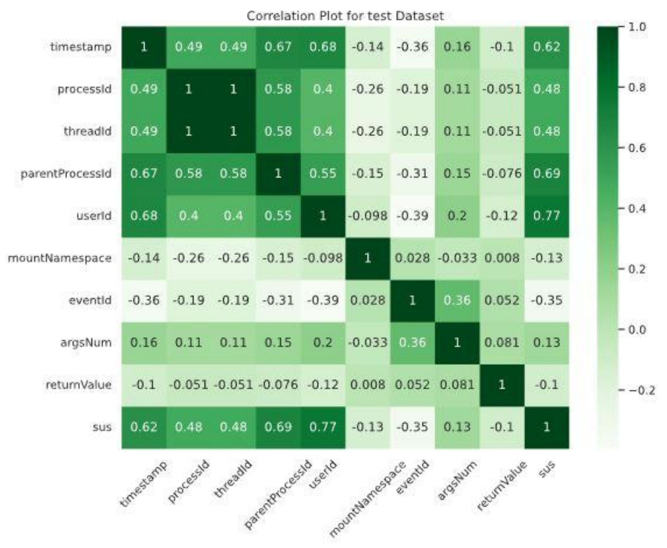
Initial exploratory data analysis yielded a few interesting findings. Notably, more IP addresses were seen in the training dataset, while only one was seen in the testing dataset. There also were more unique Process IDs, Thread IDs, and higher average timestamps seen in the training set than in the testing set. Additionally, we saw another user (user 1001) in the testing set that did not appear in the training set.

2.1 Correlations

In the training set, the User ID seems to be strongly correlated to the sus value, as seen by a correlation of 0.86 in Figure 2. ThreadID and ProcessID have a correlation of 1, so one of these columns should be removed when training the model. In the testing dataset, we saw relatively strong correlations of the timestamp, ParentProcessID and the UserID (0.62, 0.69, 0.77 respectively) to the sus value. This suggests that they may have some linear predictive relationship that could be exploited for detection purposes.

Figure 2. Correlation Plots





2.2 Processing:

The columns to be kept are ProcessID, ParentProcessID, UserID, MountNameSpace, EventID, ArgsNum, and ReturnValue. There are many ways to extract useful information from these features, but we took inspiration from the creators of the dataset and chose to map values to Operating System/Kernel events and non-OS events.

In Linux, ProcessIDs of 0, 1, and 2 as well as UserIDs of less than 1000 are used only for Kernel/OS processes. Newly added Users in the user space have IDs larger than 1000. This information can be useful because most malicious activity occurs in the user space. Additionally, MountNamespace can be mapped to a binary event of whether the value is 4026531840 or not. This value for the mount namespace indicates the /mnt directory which is where all manually mounted points are linked. It is also the mounting point for all OS users unless otherwise specified. Activity with a different mount namespace value could be indicative of activity in the user space.

The ReturnValue feature does not tell us whether or not the function came from an OS or Non-OS process. Instead, it tells us whether a function was completed successfully or not and can be mapped to success, success with a signal to the parent process, and an error. The ArgsNum value did not require any additional manipulation, so it will be left as is. This feature engineering should be sufficient enough to train a moderately successful anomaly detection model.

Table 2. Mappings for Processed Features

Processed Feature	Mappings
ProcessID	OS (0) / Non OS (1)
ParentProcessID	OS (0) / Non OS (1)
UserID	OS (0) / Non OS (1)
MountNameSpace	mnt directory (all non-OS users) (0) / other (1)
EventID	No Mapping
ArgsNum	No Mapping
ReturnValue	Success (0) / Success with value(1) / Error(2)

2.3 Analysis of Processed Feature

By creating a count plot for each feature grouped by the target variable, as seen in Figure 1 above, we can gain a few key insights. ProcessID, ParentprocessID, and mount namespace do not appear to have a clear relationship to the target class when comparing their distributions in the training and testing datasets. UserID shows a clear relationship with the anomaly class when the value is 1 (an OS user). ArgsNum is the number of arguments of the dropped Args feature. The value of 3 for ArgsNum appears to be a clear indicator of an anomaly. There does not appear to be an intuitive way to understand this relationship, though it is valuable information nonetheless. Additionally, a return value of 2 (which means an error was thrown after a function call) shows a clear relationship with the anomaly class.

3. Anomaly Detection Model Training

3.1 Introduction

This section discusses model training and selection. We will use three of the most common unsupervised learning algorithms for anomaly detection: Local Outlier Factor, One-Class SVM, and Isolation Forest. Even though we have access to training labels, we are choosing to use unsupervised learning algorithms because when in production in a live environment, no labels will be available for training. That being said, we will use the labels in the testing dataset to compare model performance.

3.2 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm is a technique used in unsupervised learning for anomaly detection. It works by calculating the deviation of a given data point's local density from its neighbors. Datapoints that have significantly lower densities are flagged as anomalies.

To start out, we will use the built-in parameters of the model with adjustments to only 2 arguments. We chose the value 5 for the number of neighbors parameter as that is often used in other anomaly detection problems, and the default value of 20 would lead to exorbitant computational time. Additionally, we set the contamination parameter to 0.05 (estimating 5% of the data being anomalous) as it is a fairly standard setting in anomaly detection as well as being the recommended value by the researchers who created the BETH dataset.

This model performed very poorly. The training for this base model took 41 minutes. If cross-validation was used to tune hyperparameters, computation cost and time would increase dramatically, making this algorithm unfavorable for our purposes. With an F1 score and Accuracy rating of less than 10%, it is unlikely that we will choose this model moving forward.

```
ROC: 0.4988
F1: 0.0850
Accuracy: 0.0927
```

3.3 One Class SVM

One-Class Support Vector Machine is an unsupervised learning method that is a variant of the original SVM model usually used for anomaly detection. This algorithm works by minimizing the hypersphere that encloses the data points belonging to the main class in the training data. All data points outside of this hypersphere are considered anomalies.

Similarly to the LOF model training, we will be mostly using the default parameters of the model as defined by the SKLEARN model, with the exception of the contamination parameter, Nu. We set this to 0.05 to maintain consistency in the model comparison phase.

This model performed much better than the LOF. An accuracy score of 0.935 and an F1 score of 0.816 show that this model is very promising. The one downside is that the model took 126 minutes to train. This is triple that of the LOF. At scale, this model would take a large amount of computational resources to bring training time down to a reasonable level. This performance shows promise but leaves something to be desired.

ROC: 0.8344

F1: 0.8168

Accuracy: 0.9351

3.4 Isolation Forest

The Isolation Forest is based on the well-known Decision Tree algorithm for classification. It works to identify outliers in the training dataset by randomly selecting a feature from the given set of features and then randomly selecting a split value between the range seen in that particular feature. This random partitioning results in shorter paths in trees containing the anomalous data points.

Similarly to the previous model training, we will use the default parameters provided by the sklearn module, with the exception of the random state value of 20 and the contamination value 0.05. The random state value controls the pseudo-randomness of the selection of the feature and split values. Choosing any integer for this parameter allows for reproducible results when calling this function multiple times.

This model performed very well. Similar to the One-Class SVM model, this model generated a high F1 score of 0.914 and an Accuracy score of 0.853. These are very promising results as typically, an F1 and accuracy score in the 0.60-0.80 range is considered good for anomaly detection. Additionally, this model only took 4 minutes to train, so its scalability is much higher than the other 2 algorithms.

ROC: 0.8479

F1: 0.9136

Accuracy: 0.8534

3.5 Model Selection and Hyper Parameter Tuning

Model Selection

Considering the performance and processing time of each algorithm, the clear choice is the Isolation Forest. While the performance was similar to that of the One-Class SVM model, this model took over 20 times longer to train than the Isolation Forest, which would pose problems for future scaling.

Cross Validation for Hyperparameter Tuning

In production, we would likely not have access to data labels as often each model is trained on the baseline activity of the endpoint or network they are protecting. That being said, this dataset is useful for creating an out-of-the-box model that could be retrained as more data is collected from the local environment where it is deployed.

During cross-validation, we will focus on tuning only high-impact parameters, as each additional parameter increases the complexity of the process, thus dramatically increasing computational cost. Using the Random Search Cross Validation function, we chose to tune the number of estimators parameter, contamination parameter, and max features parameter. The contamination parameter impacts the model performance most as it is used to estimate the underlying distribution of the anomalous class in the dataset. We will use the F1 score as the training metric because it is the harmonic mean of the recall and precision score. Theoretically, this should help minimize the false negative and false positive rates. Both of these metrics need to be as low as possible because a high false negative rate can lead to successful cyber attacks, and a high false positive rate can lead to the alert fatigue of analysts who are triaging anomalous activity, which could cause cyber attacks to slip through.

After hyperparameter tuning, the best model was found to have 280 estimators, 5 max features, and a contamination of 0.009. Training a best-fit model using these parameters increased the F1 score to 0.918 and the accuracy score to 0.863. These are marginal improvements, but they still show effectiveness for the underlying algorithm in producing effective models.

ROC:0.9147

F1: 0.9180

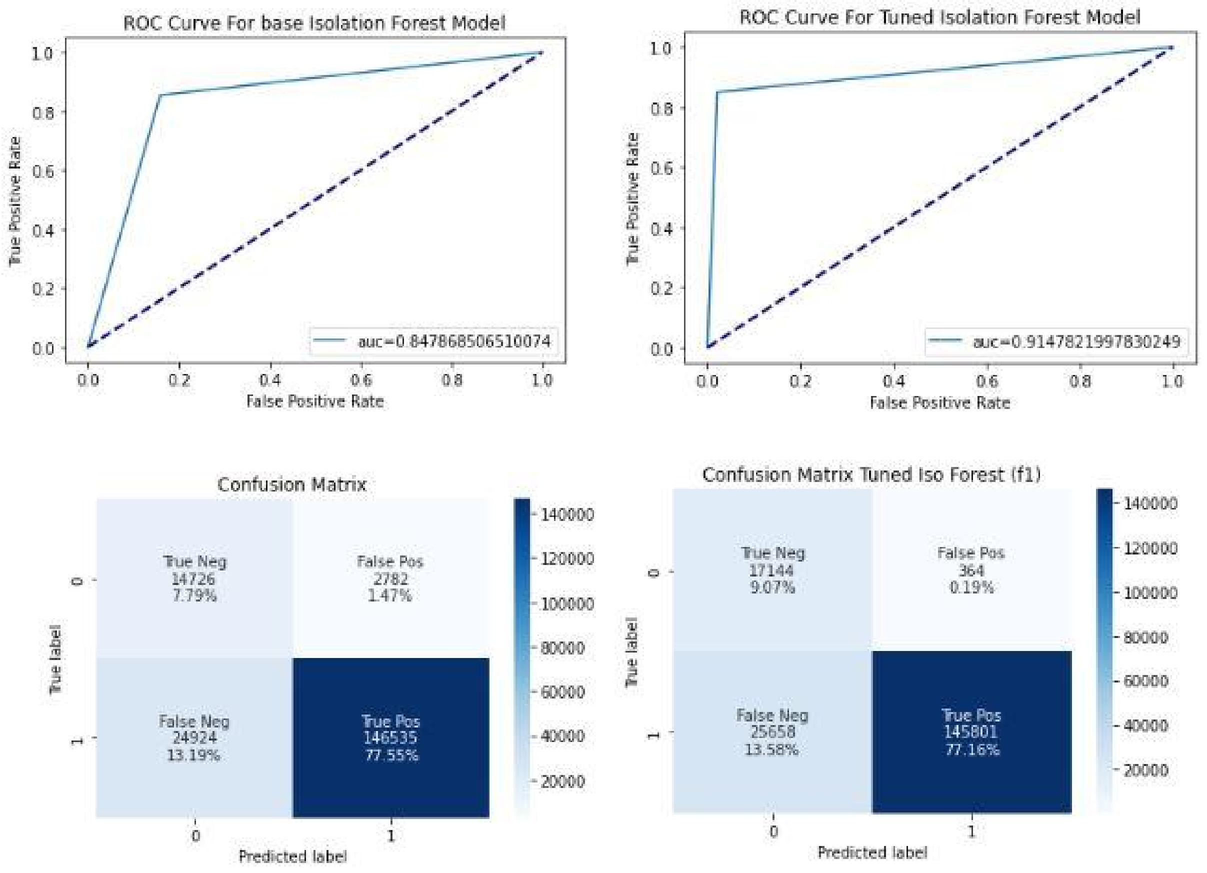
Accuracy: 0.8623

Recall: 0.8504

Precision: 0.9975

False Negative Rate: 13.58%

Figure 3. ROC plot and Confusion Matrix for Base and Tuned Isolation Forest Model



4. Conclusion

4.1 Results

Although this model performed well when measuring the F1 and Accuracy Score; other metrics paint a slightly different picture. Unfortunately, this model produces a False Negative rate of 13.58%. This suggests that 1 out of every 8 instances of suspicious activity would be misclassified as being in the base class. While in industry, layers of security are deployed to cover weaknesses of other products, this high False Negative rate would need to be optimized before the model could be viable in production.

4.2 Improvements

A few key improvements need to be implemented before this best-fit anomaly detection model can be put into production. The first improvement would be to tune the model based on the Recall metric versus the F1 score. This should optimize the False Negative rate and bring it down to a more acceptable level. In this same vein, we could tune all hyperparameters available in the algorithm. Although this is highly computationally expensive, it would provide the best model achievable. Moving forward, we could also explore more elaborate algorithms, such as deep learning neural networks, and techniques for increasing the performance of the baseline algorithm, such as bagging or boosting.

Once the best algorithm and hyperparameters are found, a data pipeline infrastructure could be built to transform raw log data into ingestible observations for iterative model training in a live environment. This would ensure that the most up-to-date model is always available to counteract the lightning-fast changes in the security landscape.

4.3 Final Thoughts

The findings in this report suggest some promise in using the Isolation Forest and One-Class SVM algorithms to fit models for anomaly detection in a Cyber Security Scope. That being said, deeper investigation is needed before a model can be deployed in a live environment. This report provided key insights into the problems and potential next steps for producing a production-capable Intrusion Detection System powered by machine learning.

References

[1] Highnam, K., Arulkumaran, K., Hanif, Z., & Jennings, N. R. (2021). *Beth Dataset: Real cybersecurity data for Anomaly Detection Research*. Retrieved October 15, 2022, from http://www.gatsby.ucl.ac.uk/~balaji/udl2021/accepted-papers/UDL2021-paper-033.pdf

[2] Kaspersky. (2021). *Machine Learning for Malware Detection*. Kaspersky. Retrieved September 2022, from https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf

[3] Krikorian, M. (2021, June 29). *Fraud detection applying unsupervised learning techniques*. Medium. Retrieved November 3, 2022, from https://medium.com/southworks/fraud-detection-applying-unsupervised-learning-techniques-4ae6f71b266f

[4] *Sklearn.ensemble.isolationforest*. scikit. (n.d.). Retrieved November 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[5] *Sklearn.neighbors.localoutlierfactor*. scikit. (n.d.). Retrieved November 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

[6] *Sklearn.svm.oneclasssvm*. scikit. (n.d.). Retrieved November 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html