Machine Learning Classification on Behavior-Based Security Alerts : A Comparative Study of Three Algorithms

LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg Department of Computer Science

Bachelor thesis:
Isak Walther
Junchao Zhang

© Copyright Isak Walther, Junchao Zhang
LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2024

Abstract

In the cybersecurity industry, security analysts are plagued by a high number of false positive alerts of various types. This takes up time and resources, and makes security analysts more prone to overlook true security threats. In collaboration with Orange Cyberdefense, this thesis investigates the ability of three machine learning algorithms, Decision Trees, Naive Bayes and Support Vector Machines (SVM), to classify behavioral security alerts. Using Scikit-learn, these three algorithms were trained and tested on synthetic data that consists of thousands of alerts. The results show that the Decision Tree algorithm has the highest performance in this alert classification task, closely followed by the SVM algorithm, with the Naive Bayes algorithm having the lowest performance. With the performance demonstrated by the algorithms, this thesis concludes that machine learning algorithms are able to assist security analysts prioritize true security threats.

Key words- Machine learning; Alert classification; False positive alerts; Decision tree; Naive bayes; Support Vector Machine; alert fatigue

Sammanfattning

Inom cybersäkerhetsbranschen plågas säkerhetsanalytiker av ett stort antal falska positiva varningar. Detta tar tid och resurser och gör säkerhetsanalytiker mer benägna att förbise verkliga säkerhetshot. I samarbete med Orange Cyberdefense undersöker detta examensarbete förmågan hos tre maskininlärningsalgoritmer, Decision Trees, Naive Bayes och Support Vector Machines (SVM), att klassificera beteendesäkerhetsvarningar. Med hjälp av Scikit-learn tränades dessa tre algoritmer och testades på syntetisk data som består av tusentals varningar. Resultaten visar att Decision Tree-algoritmen har den högsta prestandan i denna varningsklassificeringsuppgift, tätt följt av SVM-algoritmen, där Naive Bayes-algoritmen har den lägsta prestandan. Med den prestanda som algoritmerna visar, drar detta examensarbete slutsatsen att maskininlärningsalgoritmer kan hjälpa säkerhetsanalytiker att prioritera verkliga säkerhetshot.

Nyckelord- Maskininlärning; varningsklassificering; falska positiva varningar; Decision Tree; Naive Bayes; Support Vector Machine; larmtrötthet

Foreword

This bachelor thesis was made in cooperation with Orange Cyberdefense Sweden AB, which allowed us to gain experience in machine learning. Orange Cyberdefense provided key assistance to this thesis, as they provided access to key resources.

We would like to thank Herbert Urbanec, a security analyst at Orange Cyberdefense, who acted as a mentor at Orange Cyberdefense throughout the entire process, answering questions and giving insights that made the generated data used in this thesis more authentic. We would also like to thank Stefan Jönsson, who is the teamlead for the team at Orange Cyberdefense that gave us the opportunity to work with them. Lastly, we would like to thank Christian Gehrmann, our mentor at Lund University, for supporting us throughout the process, and Erik Larsson, who was the examiner for this thesis.

Thank you, for all the support and the experience.

Contents

Chapter 1

Introduction

1.1 Background

In today's digital landscape, the rise of endpoints, devices such as computers, has posed significant security challenges for organizations. To monitor these resources and respond to security threats, they may opt to use a platform such as Cortex XDR (Extended Detection and Response) by Palo Alto Networks. This is an endpoint security solution that detects threats and blocks malware by utilizing various tactics while also providing tools that give security analysts forensic information and the ability to investigate and respond to threats[1].

One of the key issues that Security Operations Centers (SOC) encounter is a high volume of false positive (FP) alerts[2] created by the monitor systems. Dealing with high amounts of false positives can cause alert fatigue in security analysts, which is when they become desensitized to security alerts, and can lead to lacking response and a higher chance of overlooking true security incidents. Working with Orange Cyberdefense, an organization that provides services and solutions in cybersecurity, this thesis will investigate the persistent problem of FPs in alert classification.

A common source of FP alerts that Orange Cyberdefense encounters in Cortex XDR is from Behavioral Indicators Of Compromise (BIOC). This method makes use of predefined sets of behavioral indicators, called BIOC rules[3]. Upon creation or activation, a BIOC rule starts to evaluate the historical data gathered in the Cortex XDR and keeps an eye out for any new matches in the incoming data stream in order to identify compromises. Sometimes these rules are too simple, which results in an excessive number of alerts being generated.

Machine learning algorithms[4], the technical focus of this thesis, have the ability to increase the accuracy of security alert classification and minimize the number of false positive alerts that security analysts have to investigate[5]. With high accuracy and minimal human involvement, the solution should be able to help classify alerts as True Positives (TP) or False Positives (FP) based on their content and context. Three algorithms, Decision Trees, Support Vector Machines (SVM) and Naive Bayes, will be utilized to create three baseline machine learning models with the goal of distinguishing between TPs and FPs.

The overall goal of this thesis is to offer recommendations and insights into the use of machine learning methods in security alert classification alongside platforms such as Cortex XDR. This solution can increase the security and reliability of security infrastructure for companies by increasing the accuracy and effectiveness of incident response.

1.2 Purpose

This thesis aims to evaluate the effectiveness of the Decision Tree, Naive Bayes and SVM algorithms in classifying false positive alerts generated from BIOC rules in Cortex XDR by implementing and comparing these algorithms using various metrics.

1.3 Goal Formulation

The goal of this thesis is to provide insights on using machine learning in order to minimize the workload of security analysts and contribute to the efficient and effective running of security monitoring and incident response procedures.

1.4 Problem Formulation

The following questions will be researched and answered in this thesis:

- Can machine learning algorithms be used to classify true positive and false positive alerts generated from BIOC rules in the Cortex XDR platform?
- Which of the three chosen machine learning algorithms (Decision Trees, SVMs, and Naive Bayes) performs best in terms of classifying false positive BIOC rule alerts from the Cortex XDR platform?
- Would the chosen algorithm(s) be able to improve the workflow at the company?

1.5 Motivation of Thesis

This thesis was chosen due to our interest in machine learning and its potential applications in the field of cybersecurity. The benefits for the company are significant, as implementing machine learning algorithms to minimize the amount of investigated false positive alerts could reduce alert fatigue for security analysts, save time, and improve overall alert detection. This would result in faster and better incident response for customers and increase trust in the company's protection of data. The benefits for society are also remarkable, as better security solutions would increase trust in digital systems overall.

1.6 Limitations

For this thesis, a set of limitations has been established:

- The only algorithms to be covered are Decision Trees, Naive Bayes and SVMs.
- Only alert classification pertaining to BIOC rules will be covered.
- Only alerts from the Cortex XDR platform will be covered.
- Deployment of the algorithms will not be covered.

1.7 Division of Labor

	Isak Walther	Junchao Zhang
Literature Review	30%	70%
Data Collection	60%	40%
Implementation	40%	60%
Testing	70%	30%
Report and Poster	50%	50%

Chapter 2

Terminology

- Behavioural Indicators Of Compromise(BIOCs): An indicator of malicious activity concerning files, processes, network activity and registry.
- BIOC Rule: A predefined set of behavioral indicators that Cortex XDR uses in order to detect specific behavior. Detected instances of this behavior are logged as alerts.
- Cortex XDR: A cybersecurity platform that provides endpoint security solutions.
- Cortex XDR Agent: A software component from Cortex XDR that monitors endpoint data and behavior and reports back to the Cortex XDR platform.
- Cross-validation: A method for validating model parameters and evaluate model performance in machine learning.
- Decision Tree: A machine learning algorithm that recursively splits data into subsets based on the most significant attribute, creating a tree-like structure to make predictions.
- Endpoint: A device that connects to a network.
- False Positive(FP) alert: An alert that is triggered even though there is an absence of malicious activity.
- Feature: The columns in a dataset, showing the input variables in an example, for example “Temperature” or “Age”.
- Hyperparameter: Input variables that dictate how a machine learning model learns and behaves during training.
- Hyperparameter Tuning: Finding optimal hyperparameters for machine learning algorithms not by manual but by certain algorithms.

- Label: The resulting value of an example, for example “spam” or “not spam”.
- Machine learning algorithm: A set of instructions that guides the learning process.
- Machine learning model: The practical outcome of applying an algorithm to data.
- Naive Bayes: An algorithm that assumes independence between features and calculates the probability of a label given the observed features.
- Scikit-learn: A Python library that provides functionality to implement machine learning algorithms.
- Support Vector Machine(SVM): An algorithm that categorizes data by finding a line between datapoints with the largest margin.
- True Positive(TP) alert: A legitimate alert that is triggered by malicious activity.

Chapter 3

Technical Background

This chapter provides a thorough overview of the technical foundations of this thesis, which is necessary for comprehending the methodology[Chapter 4][Chapter 6][Chapter 7] of this thesis.

3.1 Cortex XDR

Cortex XDR, a core resource for this thesis, is a security platform developed by Palo Alto Networks. To detect threats, it gathers data from endpoints, networks, and clouds, across multiple platforms[6].

In this thesis, endpoint detection and response features of Cortex XDR are used to help generate data. In order to monitor the endpoints in real time, a Cortex XDR agent will be deployed on the endpoint which in this case was a Windows virtual machine. The Cortex XDR agent serves as a companion software component that reports back to the Cortex XDR platform.

The real-time collection and transmission of telemetry data to the Cortex XDR platform is largely dependent on the Cortex XDR agent[7]. The Cortex XDR agent continually monitors endpoint activity and compares it to the security policies specified in Cortex XDR. Once a security event or anomaly is discovered, the Cortex XDR agent immediately forwards the pertinent information to Cortex XDR to conduct a further inspection and evaluation.[6]

3.2 Virtual Machines(VM) and VMWare

In order to utilize the Cortex XDR agent, it needed to be installed on an endpoint. In this thesis, a virtual machine (VM) was utilized, which is a software-based simulation of a physical computer system that enables simultaneous use of multiple operating systems on a single machine. This is achieved through specialized software called a hypervisor, which acts as a layer between the physical machine (the host machine) and the VM (often referred to as the "guest"). The hypervisor utilizes the resources of the host machine to allocate resources to the VM and provide it with a simulated environment that is isolated from the physical computer. Each virtual machine functions as an independent computer and can be equipped with their own simulated hardware, software, storage and network capability[8][9].

For this thesis, licenses were provided by Orange Cyberdefense for a hypervisor program, namely VMWare Workstation 17 Pro. This hypervisor program allowed simultaneous use of multiple VMs. Two VMs were used simultaneously in VMWare for this thesis, one based on Linux Ubuntu 18.04 x64, and one based on an evaluation version of Windows 11 x64 version 22H2. The Linux VM was provided by Orange Cyberdefense and was customized for security analysis, more specifically to intercept network traffic from other VMs and to route that traffic through a VPN. The Windows VM was used for generating alerts, and was a development environment provided by Microsoft, that permits developers to utilize an evaluation version of Windows up until a certain date[10]. A Cortex XDR agent was installed on the Windows VM in order to monitor all activity. The Windows VM was not modified any further, other than routing its internet traffic through the Linux VM. Figure 2.1 shows the described layout visually.

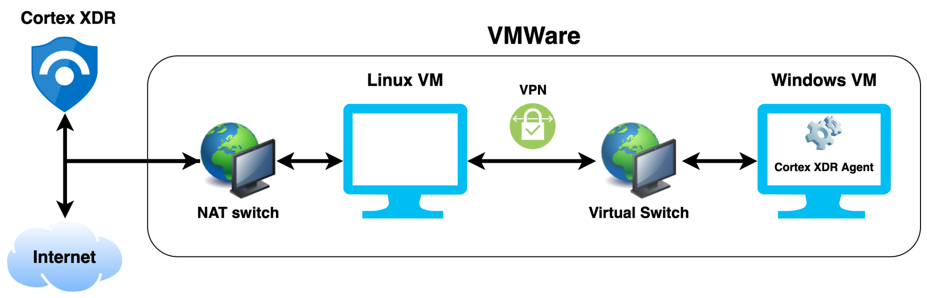


Figure 3.1: Layout of virtual machines and their networking.

3.3 Machine Learning

Machine Learning, the technical focus of this thesis, is a subfield of Artificial Intelligence, It is also a notion that enables computers to learn from examples and experiences and to mimic or emulate human decision making without being expressly programmed to do so[11].

Traditionally, manual analysis by humans would have produced superior choices. However, with the abundance of data generated by both computers and humans in today's world, humans are no longer able to process large-scale data in a short period of time. But with the introduction of machine learning by Arthur Samuel in 1959 and its evolution and innovation since then, machine learning has proved its superiority in handling large, complex datasets and automated decision-making, and has become a key part of artificial intelligence research[4][11].

Machine learning is divided into two types: supervised learning and unsupervised learning. Supervised learning, as the name implies, involves machine learning while being supervised and guided. Humans act as guides in this case, delivering data labeled with input and output features to machine learning models, and the models can forecast the relevant outputs following inputs and master the relationship between inputs and outputs by learning these data and altering the corresponding parameters. In contrast to supervised learning, unsupervised learning requires machine learning models to learn from unlabeled data. The training data used in unsupervised learning contains just input features with no associated output labels. Unsupervised learning differs from supervised learning in that the purpose and role of unsupervised learning is typically to identify structures and patterns in data[12][13][14].

In this thesis, three different supervised machine algorithms will be the focus of the thesis and will be presented in following subsections.

3.3.1 Support Vector Machines

Support Vector Machines is a technique that utilizes a decision boundary, called a hyperplane, in order to classify data[15][16]. It does so by finding a line between data points that correctly separates the data points to their respective groups. The data points that are closest to the hyperplane are called "support vectors", and the distance between the support vectors and the hyperplane is the margin. The goal of an SVM during training is to find a hyperplane that maximizes the margin, which is done in order to reduce misclassification of new data points.

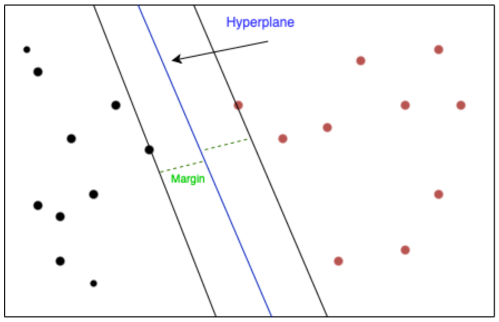


Figure 3.2: Example of the SVM in two dimensions

For any new data point, the following function is used in order to decide which side of the hyperplane the data point lands on.

$$g(x) = w^{\tau} x + b \tag{3.1}$$

- $w^{\tau} x$: The transpose (reorienting a vector from horizontal to vertical, or vice versa) of the vector w, which represents the hyperplane, multiplied by x, which represents the input vector consisting of the features of a data point.
- b: The bias term that defines the offset and position of the hyperplane from the origin.

While this type of decision boundary is one of many types of decision boundaries that are provided in Scikit-learn, this thesis focuses on linear decision boundaries.

3.3.2 Naive Bayes

Naive Bayes is a collection of algorithms that are based on using Bayes' Theorem[17][18], with the assumption that each feature in a dataset is conditionally independent (in other words, unrelated to each other), hence the "Naive" part of the name. Bayes' Theorem relates the probability of an event 'Y' given another event 'X' to the probabilities of 'Y' and 'X' individually:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{3.2}$$

- $P(Y|X)$: The probability of event Y given that event X has occurred.
- $P(X|Y)$: The probability of event X given that event Y has occurred.
- $P(Y)$: The probability of A occurring on its own.
- $P(X)$: The probability of B occurring on its own.

Introducing 'n' number of conditionally independent variables (X_n) gives the following formula:

$$P(Y|X_1...X_n) = \frac{P(X_1...X_n|Y)P(Y)}{P(X_1) * P(X_2)... * P(X_n)} \tag{3.3}$$

In tasks involving classification, a Naive Bayes algorithm uses the theorem to choose the outcome with the highest probability.

3.3.3 Decision Trees

As the name implies, Decision Trees are tree-structured supervised machine algorithms that are frequently applied to address regression and classification issues[19][20]. The main parts of a Decision Tree are the root node, internal nodes, and leaf nodes. The root node symbolizes the entire set of sample data, the internal nodes represent the tests that correspond to a feature attribute, and the leaf nodes represent the decisions outcomes.

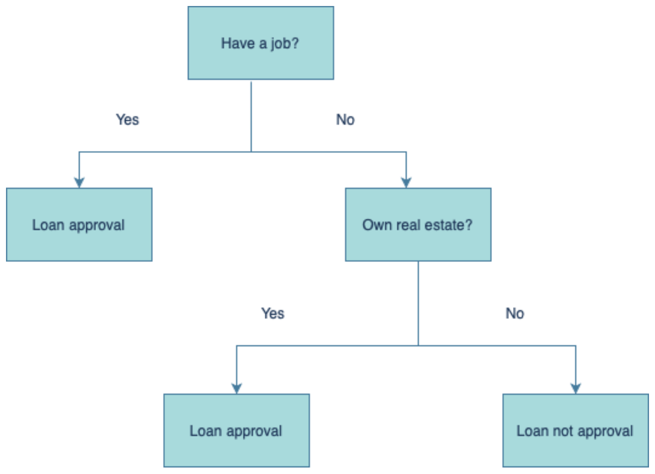


Figure 3.3: Example of the Decision Trees

There are three generally used algorithms for Decision Trees, ID3, C4.5, and CART. Machine learning researcher J. Ross Quinlan devised and published the first two in 1986 and 1993, while statisticians Breiman and Stone et al. suggested the CART algorithm in 1984.

ID3, on the other hand, is the most commonly used of these three algorithms.The core concept of the ID3 algorithm is to partition the data into new nodes by measuring the selection of features in terms of information gain, always selecting the feature with the highest information gain.

While the information gain of a feature has a larger number, it means that it is better at reducing the uncertainty. When the uncertainty is reduced, the Decision Tree is able to make better predictions[21]. In order to identify the information gain, the information entropy and the conditional entropy are the two values that must be known[19].

The information entropy indicates the level of uncertainty in the data:

$$H(D) = -\sum_{i=1}^{n} P_i \log_2 P_i \tag{3.4}$$

The conditional entropy indicates the uncertainty in the data after the splitting using a particular attribute:

$$H(D|A) = \sum_{i=1}^{n} P_i H(D|A = a_i) \tag{3.5}$$

The difference between the two values represents the amount of information gained and it can be conveyed by the following equation:

$$I(D, A) = H(D) - H(D|A) \tag{3.6}$$

In this thesis, the CART algorithm will be emphasized and applied. The core concept of the CART algorithm is similar to the ID3 algorithm, instead of using information gain in the ID3 algorithm, CART uses the Gini's impurity index to partition the data.

$$Gini(D) = 1 - \sum_{i=1}^{n} p_i^2 \tag{3.7}$$

As in the case of information gain in ID3, CART has a certain feature applied to divide the data D into D_1 and D_2 and have the expression:

$$Gini(D, A) = Gini(D_1)\frac{|D_1|}{D} + Gini(D_2)\frac{|D_2|}{D} \tag{3.8}$$

$Gini(D, A)$ represents the level of impurity of the data after selecting a certain feature.When the $Gini(D, A)$ is relatively small, it indicates a lower level of impurity, which means that the feature is preferable to partition the data.

3.4 Python

Python was used in this thesis, as it is an ideal choice for scientific computing and data analysis because of its simplicity and diverse libraries, and is renowned for clarity and readability [22]. It is extensively utilized in many different industries and has a wide range of applications including web development,data analysis,machine learning and artificial intelligence.

3.5 Scikit-learn

While Python was chosen as the programming language used in this thesis, it needed a library in order to implement machine learning algorithms. Scikit-learn, also known as "sklearn" is a robust and user-friendly machine learning library that offers an extensive choice of algorithms and tools for different tasks,such as classification and regression[23].

Other well-known python libraries like Numpy and Matplotlib are easily integrated with Scikit-learn. Given this connection, users may take advantage of the machine learning capabilities of Scikit-learn as well as the data processing, analysis of these libraries. It enables a fluid and effective approach.

3.5.1 Preprocessing Tools

Raw data from the real-world may not always be suitable for a machine learning algorithm right away. If this is the case, the data needs to go through preprocessing, which is the process of manipulating data before passing it onto the machine learning algorithm that uses it, in order for the algorithm to easily understand the data and perform its task effectively[24]. When working with categorical or textual data, this would be encoding that data into numerical values, so that the machine learning algorithms can properly do their mathematical calculations. Scikit-learn provides these kinds of tools natively[25].

LabelEncoder and OrdinalEncoder

For categorical features in data, Scikit-learn provides, among others, LabelEncoder and OrdinalEncoder. Both encoders transform each unique value that they find in data into a number, with a defining difference that only OrdinalEncoder can preserve the order of values[26][27]. This means that encoding categories such as "Low", "Medium" and "High" can be encoded to "0", "1" and "2" with OrdinalEncoder, but that same order is not guaranteed with LabelEncoder.

TfidfVectorizer

When encoding textual data, some cases demand the preservation of the importance of each word in a collection of texts. This can be done with TfIdfVectorizer. It is able to transform texts into a matrix of Term-Frequency Inverse Document-Frequency (TF-IDF) features[28].

Term-frequency (TF) purely refers to how many times a word appears in a single given text, while inverse document-frequency (IDF) is the component that describes a word's rarity across all texts in the data. The TF and IDF components are multiplied together in order to calculate how much meaning, or "weight", a word has in any of the given texts, shown as a number[29]. This number is sometimes called TF-IDF score. If a word rarely appears in an entire collection of texts, then it is given a higher TF-IDF score than a word that appears frequently, meaning that the word that rarely appears is more interesting or important.

The matrix that TfidfVectorizer creates has rows that represent texts and columns that represent words from the collection of texts, where the value in each cell shows the TF-IDF score of a word in a particular text[29].

StandardScaler

Values found in one feature of a dataset can sometimes have a different range than the values of another feature, which can impact performance of algorithms when comparing features[25]. Standardization of data is when the ranges of values in different features are scaled to a similar scale, so that computation in distance-based algorithms, such as SVMs, are influenced by different features the same amount[30]. A solution that Scikit-learn provides for this process is StandardScaler. StandardScaler works by using the following formula to standardize the values of a feature:

$$z = \frac{x - u}{s} \tag{3.9}$$

In this formula, 'x' is the value being standardized, 'u' is the mean of the values in the feature and 's' is the standard deviation of the values in the feature[31].

3.5.2 Metrics

In many cases, it is often necessary to evaluate the performance of a machine learning model. The metric functions can be easily called in Scikit-learn to compute evaluation metrics. In this thesis, 4 different evaluation metrics were used.

Confusion matrix

Comprehending and computing evaluation metrics requires an understanding of the confusion matrix. The confusion matrix is a two dimensional table that illustrates the correspondence between the actual values and the predicted values of a machine learning model [32]. Based on the combination of actual labels and predicted labels, the samples are categorized in the table as True Positive(TP),True Negative(TN),False Positive(FP) and False Negative(FN). TP and TN represent samples that are correctly predicted while FP and FN are the opposite.

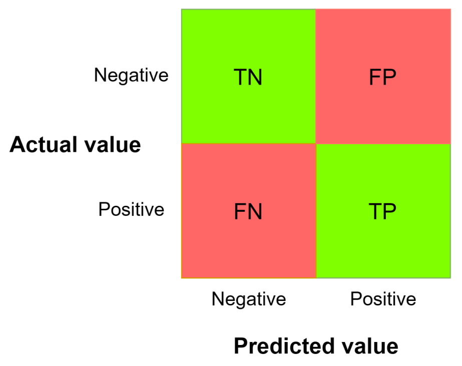


Figure 3.4: Example of the confusion matrix

Accuracy Score

A machine learning model's accuracy score can be derived by calculating the accuracy rate, which refers to the proportion of correctly predicted samples of the total sample size. The accuracy formula can be written as follows

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.10}$$

The accuracy rate has an obvious defect: when the data distribution is not balanced, with some categories having too many samples and others having too few samples, the category with the highest proportion is bound to be the most important factor influencing the accuracy rate[33][32].

Recall Score

The recall score can be represented by following formula:

$$Recall = \frac{TP}{TP + FN} \tag{3.11}$$

which shows an algorithm's ability to find all TP samples in a dataset[34].

Precision Score

The precision score of a machine learning model refers to the ratio of correctly predicted positive samples to all correctly predicted samples. The following formula can be used to represent the precision[35]:

$$Precision = \frac{TP}{TP + FP} \tag{3.12}$$

F1-score

The machine learning model's ability to identify negative samples improves with increased precision score, but it will be less effective at recognizing positive samples, which will have an impact on recall score. Recall and precision are mutually exclusive.The F1 score is a combination of the precision score and the recall score, a balanced performance is often indicated by a high F1 score.[32] The following formula can be used to represent the F1 score[36]:

$$F1 = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.13}$$

3.6 Pandas

A comprehensive Python library for data analysis and manipulation is called Pandas. Intuitive data structures like DataFrame are available for processing structured data. Almost any data manipulating task, including data cleansing, filtering and transformation can be done with it. Pandas also supports a variety of input/output types and can smoothly combine with other libraries[37].

Chapter 4

Methodology

This chapter provides an overview of the approach used to investigate the objectives of this thesis, as well as outlining the process for each phase of the work.

4.1 Phase 1: Literature Review

In order to gain a comprehensive idea of the existing work relevant to the topic of this thesis, a preliminary literature review was conducted. Through reading a variety of conference papers and journal articles, this literature review aims to identify key themes, methodologies and gaps in current work. The insights from this literature review resulted in the "Related Work" chapter, found later in this thesis[Chapter 5].

4.2 Phase 2: Data Collection

The machine learning models in this thesis needed to be tailor-made for alert classification, thus requiring a suitable dataset that allows the models to be trained and tested. In this case it was in the form of alerts generated by BIOC rules. For this thesis, access was provided by Orange Cyberdefense to their demo environment, which does not contain any confidential customer data, but contains BIOC rules that may be used in customer environments. This was the chosen approach since Orange Cyberdefense giving access to real-world data could have exposed confidential customer data.

After gaining access to the demo environment, three BIOC rules of differing severity and category were chosen in order to create a diverse dataset. To generate alerts for these BIOC rules, certain commands had to be entered into PowerShell in the Windows VM. Using techniques described later in this thesis[Chapter 6], 27487 alerts were generated in a way that reflected real-world usage of commands that trigger the selected BIOC rules and provided realistic data to train the machine learning algorithms on.

This is a synopsis of phase 2. A detailed description of methods and techniques used in this phase can be found in the ”Data Collection” chapter[Chapter 6].

4.3 Phase 3: Data Preprocessing and Initial Implementation of Machine Learning Models

Although a modest amount of data had been generated and retrieved, several features and values in these features needed to be preprocessed in order for the three machine learning algorithms to process the data and learn from it.

The generated data had significantly numerous features to handle alongside. Therefore, with the assistance of a security analyst at Orange Cyberdefense, the number of features was reduced. Subsequently, a new column named “Label” was added to the dataset, categorizing each alert as either a False Positive (FP) or a True Positive (TP).

The data was then preprocessed in preparation for the machine learning algorithms to learn from it. By applying techniques detailed later in this thesis[Chapter 7], the data was successfully preprocessed, enabling the machine learning algorithms to use it to learn. Initial testing revealed that the SVM algorithm performed significantly worse than the Decision Tree and Naive Bayes algorithms, which was discovered to be because of the data being passed to the SVM algorithm not being scaled. Scaling the data that was passed to the SVM algorithm enhanced its performance, making it comparable to the others.

This is a synopsis of phase 3. A detailed description of methods and techniques used in this phase can be found in the ”Data Preprocessing and Initial Implementation of Machine Learning Models” chapter[Chapter 7].

4.4 Phase 4: Testing and Refinement of Machine Learning Models

In order to further test and evaluate the machine learning models, four classification evaluation metrics were used in this thesis, accuracy score, precision score, recall score and f1 score. These four metrics are commonly used for evaluating classification models and can give a quantitative assessment of models performance. These evaluation metrics can be conveniently calculated using Scikit-learn’s built in functions and the result of tests can be found in the results chapter [Chapter 8] later in this thesis[38].

To achieve the best results for the models, the external variables (hyperparameters) that are used when training the machine learning models can be adjusted. In this thesis, the hyperparameter tuning method known as GridSearch is employed. Given a certain number of suitable hyperparameters, the best-performing combination of hyperparameters was found by traversing through each combination of given hyperparameters.[39] In Scikit learn, such hyperparameter tuning can be implemented easily by calling the GridSearchCV() function, which is a combination of GridSearch and k-fold cross-validation. K-fold cross-validation ensures that the model is not overfitting or underfitting by randomly splitting the training subset into k smaller subsets, where k-1 subsets will be used for training and the remaining set will be used to validate the model's performance. Such a combination between GridSearch and Cross-validation on every possible combination of hyperparameters ensures the accuracy and reliability of the evaluation metrics in order to find the best possible mixture of hyperparameters.[39][40][41]

After implementing the optimal hyperparameters identified by GridSearchCV, the performance of machine learning models marginally improved[Chapter 8]. The results demonstrate an adjustment of hyperparameters can enhance model performance and improve estimated outcomes, which is clarified in the results chapter[Chapter 8].

Chapter 5

Related work

The usage of computer resources and network infrastructures by individuals and organizations in the field of information security has significantly increased since 2000. The need to take precautions against digital assets from different assaults has become increasingly apparent as our dependence on networked systems grows. Consequently, the information security profession has seen significant transformation, leading to heightened research endeavors aimed at fortifying defenses against malevolent assaults[42].

At the core of detecting and preventing cyberattacks are Security Incident and Event Management Systems (SIEM) and Intrusion Detection Systems (IDS). These systems are now sufficient protection for digital assets, keeping a close eye on system activity and network traffic in order to spot and prevent any breaches. The desire to fully realize the revolutionary potential of machine learning algorithms has attracted significant scholarly attention, coinciding with the development of these protective mechanisms[43][44].

With the introduction of machine learning techniques, there is optimism that they can be applied to improve the effectiveness of data network security. Research has demonstrated how machine learning algorithms are becoming increasingly effective at improving threat detection accuracy and reducing the number of false alarms in SIEM and IDS systems[45].

Pietraszek[46] made groundbreaking contributions that embodied a fundamental change in the field of alert classification systems, leading to the creation of the Alert Classification System (ALAC), an adaptive classifier. Pietraszek's pioneering research established an advanced system for differentiating between real threats and benign anomalies, which relieved security analysts of the effort of verifying the truthfulness of alerts.

Using the K-nearest neighbors (KNN) method, Kwok and Law[47] advanced the field of alert categorization in further research. Their inventive approach made use of similarities amongst incoming signals to create a model of normalcy from which deviations suggestive of possible security breaches could be dynamically identified. They improved alert classification techniques by including the KNN algorithm, enabling intrusion detection systems to distinguish between legitimate threats and innocuous activity. The study conducted by Lam and Kwok illustrated how KNN(machine learning)may be used to improve the adaptability of intrusion detection systems, which helps to further the continuous development of cybersecurity procedures.

An important turning point in the development of alert recognition and classification techniques was reached by Meng and Kwok's[48] innovative contributions.Their thorough examination of several machine learning algorithms demonstrates the revolutionary potential of adaptive false alarm filters, which have the capacity to significantly reduce the flood of false positives that overwhelm security analysts and present a viable path to improve the effectiveness of intrusion detection systems (IDSs). Meng and Kwok discover the adaptive nature of false alarm filters by exploring the complexities of machine learning, which helps them to continuously improve their decision-making procedures. This flexibility helps the filters differentiate between legitimate threats and benign activity, improving the efficacy of intrusion detection systems (IDSs) in identifying and stopping malicious activity.

Additionally, the research conducted by Subbulakshmi, George and Shalinie [49] makes a significant addition to the discussion on false positive mitigation inside IDS. The research presents a novel two-phase automatic alert categorization system designed to help analysts distinguish false positives. In order to create meta-alerts that aid in contextual analysis and pattern recognition, incoming alerts are first carefully normalized and clustered. By utilizing machine learning techniques, the system effectively reduces the number of warnings by identifying and eliminating false positives. In order to precisely and effectively automate the classification process, the second step makes utilization of advanced machine learning techniques. The authors validate the joint effort to improve network security architecture by demonstrating the system's effectiveness in lowering the volume of alerts and limiting false positives through extensive real-world testing.

Likewise, a thorough examination of alert management in IDSs is provided by Alsubhi, Al-Shaer, and Boutaba[50]. Alsubhi et al. tackles the problem of security analysts in IDSs being inundated with notifications. The authors provide an alert rescoring method that introduces a fuzzy logic-based approach to prioritize alerts and dynamically alter scores. This method simplifies the handling of alerts, enabling administrators to better allocate resources and strengthen their reaction to threats. The study provides important insights into IDS alert management and network security operations by validating the suggested technique in real-world settings. To summarize, the thesis offers an intricate structure for prioritizing alerts in intrusion detection systems (IDSs), hence bolstering cybersecurity resistance to constantly changing threats.

The DARPA datasets, which are synthetic datasets gathered from simulated medium-sized computer networks and includes information about network traffic and audit logs[51][52][53], and Snort, an open-source network-based intrusion detection system[54], were utilized by all the aforementioned studies to generate alerts data for machine learning model validation or training by running Snort on the dataset.

Although the significant research mentioned above has offered valuable insights into the effectiveness of machine learning algorithms in the context of network-based security, there is still a glaring gap in the application of these approaches to the classification of host-based security alerts. The necessity of filling this void emphasizes the need for more research, and the ramifications go beyond the confines of academia to include the practical needs of strengthening cyber defenses in an increasingly dangerous digital environment.

In this thesis project, validating machine learning with host-based security will be the focus of this research. The conceptual underpinnings of the pioneering studies previously described inform the methods utilized to generate alerts data that was used to train the machine learning models in this work. However, this study differs from earlier research, where the DARPA dataset used to trigger alerts are not directly relevant to this study. Despite considerable efforts, no suitable dataset for this work was uncovered.

Given this, the method used in this study was to manually generate alerts that imitate real-world events. This varies from previous data generating methods in that it requires more manual work. However, the considerable manual authoring of the data increases its relevance to the purpose of this thesis.

Furthermore, in this study, Cortex XDR was introduced as the mainstay of this thesis' data generation methodology, in contrast to the previously mentioned research in the network-based field that employed Snort as the tool for detecting and generating alerts. This decision was influenced by a security analyst at Orange Cyberdefense and Cortex XDR's capabilities, as discussed in the technical background chapter. To better achieve the objectives of this study, Cortex XDR takes the place of Snort by offering a more appropriate platform.

Chapter 6

Data Collection

This chapter provides and in-depth description of the methodology for phase 2 of this thesis, describing the tools and techniques that were used as well as choices that were made.

6.0.1 Selection of Used Data

The dataset used in this thesis consists of alerts from three BIOC rules with the following names:

- **Possible LSASS memory dump**
  This rule detects if the command line was used to execute a command containing "-ma lsass.exe" anywhere in the command. This could be used by attackers in order to retrieve credentials from a lsass.exe (Local Security Authority Subsystem Service) memory dump. A memory dump is a file that contains the data of a system's memory, sometimes including information from processes that were active at the time the memory dump was created [55]. Alerts from this rule are of high severity and are of the type "Credential Access".

- **New local user created via Powershell command line**
  This rule detects if the command line was used to execute a Powershell command containing "New-LocalUser" anywhere in the command. This could be used by attackers in order to maintain access to machines. Alerts from this rule are of medium severity and are of the type "Persistence".

- **Collecting audio via Powershell Command**
  This rule detects if the command line was used to execute any of several Powershell commands that can manipulate audio devices. This could be used by attackers in order to collect audio from a microphone. Alerts from this rule are of low severity and are of the type "Collection".

These are some of many pre-configured rules that Palo Alto Networks provides. These three were specifically selected in order to provide data on alerts of varying severities and categories.

6.0.2 Generation of Alerts

In order to generate a single alert from one of the chosen BIOC rules, an instance of Powershell had to execute a command that would trigger one of the rules. In order to gain a moderate dataset of both FPs and TPs, this would have to be done thousands of times, so for the purpose of automating this process, batch files were utilized.

Batch files (also known as batch scripts) can, according to Microsoft, be used to simplify routine or repetitive tasks [56], and they describe a batch file as a text file that contains commands. In other words, they are executable files that can execute code, and it was through these that alerts were generated for this thesis.

Three batch files were created, one for each BIOC rule. As previously stated, the purpose of these batch files were to automate the process of generating alerts. They did this by entering commands into Powershell that fit the described behavior of a BIOC rule, which in turn caused the Cortex XDR agent to detect the behavior and report it in the form of an alert.

Commands that fit the behavior of one of the BIOC rules were not always the only commands entered into Powershell during each execution of a batch file, but could instead be one of many commands that were entered at the same time. The other commands present were benign Powershell commands that had nothing to do with the behavior of the BIOC rules, such as Get-Date or Get-Random, which return the current date and time or a random number, respectively[57][58].All three batch files shared 22 of these benign commands that were used in order to introduce noise to the alerts. Noise is random fluctuations in data that are normally unwanted as it can lead to difficulty in identifying the patterns of the data[59]. Despite this, noise was included in many of the generated alerts by including benign commands along with the command that triggered the alert. This was purposefully done in order to make the behavior that triggers the BIOC rules more realistic, since commands that fit the behavior of BIOC rules are often buried amongst irrelevant commands (H. Urbanec, personal communications).

Alongside the benign commands, each batch file had two lists of 4-5 commands each that were relevant to their BIOC rule. One list contained commands that would trigger a FP alert, and the other contained commands that would trigger a TP alert. All commands used were approved by a security analyst at Orange Cyberdefense (H. Urbanec, personal communications).

Each batch file created an output string of varying length containing the triggering command. If noise commands were present, the triggering command was placed in a random position among the noise commands. The output string was then used in the command line, which executed the commands in Powershell. The basic structure for each batch file was as follows:

1. Randomly generate alphanumeric strings and numbers that can be used in both the noise commands and triggering commands. These strings and numbers were used as command parameters, usernames and/or passwords.
2. Decide if the generated alert will be a FP or TP alert.
3. Decide on the total number of commands to be used (1-10 commands, including the command that triggers the alert).
4. Generate a random index for where in the output the trigger command will be, based on the number chosen in the previous step.
5. Randomly choose a random command that triggers the alert. This is chosen from one of the batch file's lists containing FP and TP commands. The list which the program chose a command from was based on the decision made in step 2.
6. Generate the output. This was done by appending strings to an variable named "output", which would be used in the command line later.
   (a) If the index chosen for the triggering command is 0 (at the beginning), the triggering command is appended. Otherwise, one of the noise commands is appended.
   (b) Run a for-loop that runs for the number decided in step 3.
      i. If the number of the current loop matches the index of where the triggering command is to be inserted, append the triggering command. Otherwise, append one of the noise commands.
7. Use the command "start cmd /c 'powershell.exe %output%'". This command initiated a new instance of the command line and executed the commands in "output" in Powershell, after which the instance terminated.

This structure ensured that an alert could reliably and automatically generate output that, according to a security analyst at Orange Cyberdefense, was suitable for the data needed in this thesis (H. Urbanec, personal communications). Each time one of the batch scripts executed, the Cortex XDR agent in the Windows VM would detect the behavior described in the BIOC rules and generate an alert.

Executing a batch file can be done by double-clicking the batch file. AlphaClicker, an open-source autoclicker software[60], was used for repeatedly double-clicking on the batch files in order to execute them a set number of times.

After generating 27487 alerts, they could be retrieved from the Cortex XDR BIOC dashboard by using filters to only view the generated alerts and downloading them to a .tsv (tab separated values) file.

Chapter 7

Data Preprocessing and Initial Implementation of Machine Learning Models

This chapter provides and in-depth description of the methodology for phase 3 of this thesis, describing the tools and techniques that were used as well as choices that were made.

7.0.1 Importing and Preparing Data for Preprocessing

The generated data was utilized in code by first creating a Pandas dataframe and using its 'read_csv()' function in order to import the data from the TSV file[61] retrieved from Cortex XDR.

The dataset initially had numerous features. While this might sound positive, having more features can lead to complex datasets that could require more computational resources and datapoint in order to find relations between features, this is called the "Curse of Dimensionality"[62][63]. To avoid this, some features in the dataset were removed. The 19 features that remained were verified by a security analyst at Orange Cyberdefense as essential features that would be used when analyzing an alert (H. Urbanec, personal communications), and were considered numerous enough to give the machine learning algorithms used in this thesis enough information to work with. Following this, a new column was added to the dataset named "Label", which labeled each alert was a FP or TP alert.

7.0.2 Preprocessing of Data

As mentioned in the technical background [Chapter 3, section 3.5.1] textual and categorical data that is not numerical need to be preprocessed in order for machine learning algorithms to effectively learn from the data. In the dataset used in this thesis, some features were numerical and did not need any preprocessing, while several features were categorical, textual or both.

The first step taken on preprocessing the data for this thesis was imputation, in other words, handling missing values in the dataset. Pandas provides a suitable solution to this, "fillna()". This function fills missing values using the methodology that it is provided[64], giving it the ability to fill different types of values in various ways. In this case, the function was used in order to fill in missing values in two features, "Process execution signer" and "Initiator TID".

In Cortex XDR, "Process execution signer" shows the name of the signer of the certificate of the process that triggered the alert[7]. The certificate certifies that the program has not been manipulated since the signature was written [65]. "Initiator TID" shows an identification number of the thread in the running process that triggered the alert[7]. The missing values in "Process execution signer" were filled with "Unknown" and with "0" in "Initiator TID". The value "0" was used for "Initiator TID" because a TID in Windows can never be "0"[66], making it a suitable value for a missing TID.

Timestamps also required preprocessing because of the timestamps in the dataset initially being strings with a format such as "Feb 1st 2024 00:00:00". This was not data that the machine learning algorithmss could learn from, therefore the timestamps were converted to numerical values. This was done by first stripping the timestamps of suffixes to dates such as "st", "nd", "rd" and "th", and then using built-in functions in Python to convert the timestamps to Portable Operating System Interface (POSIX)[67] timestamps. This type of timestamp represents the number of seconds that have elapsed since "the epoch", which refers to midnight of January 1st, 1970, as a single floating point number[68][69][70]. For example, this conversion would show "Feb 1st 2024 00:00:00" as "1706742000" instead, providing a suitable numerical conversion.

OrdinalEncoder was used for the "Severity" feature, which had the categories "Low", "Medium" and "High", and was used in order to preserve that order. LabelEncoder was used for 11 categorical features, both numerical and textual, that did not inherently have any order.

Three of the features in the dataset contained strings that showed the commands that were entered in the command line and triggered an alert. To reiterate, only one command in each alert was relevant in making an alert a FP or TP alert. This means that these features had to be encoded in such a way that showed how much meaning each word had in each alert. TfidfVectorizer was the chosen encoder for this task. After encoding, analyzing TfidfVectorizers' ranking of words in the dataset showed that it correctly ranked words such as "Get", which appeared in several of the noise commands, lower than words that truly caused an alert to be FP or TP. TfidfVectorizer was also given a vocabulary of the words from commands that triggered the alerts so as to avoid TfidfVectorizer giving higher TF-IDF scores to randomly generated words, which only appear in one alert each, compared to words in the dataset that were truly important.

The dataset was then split into two subsets, a training subset and a testing subset. Doing this prevents a machine learning model from training and testing on the same data, which would give it a perfect prediction accuracy, only for it to fail to perform as well when given new data, which is called "overfitting"[41]. Scikit-learn's "train_test_split()" function provides functionality to split a dataset into training and testing subsets in various ways[71]. For this thesis, 70% of the dataset (19240 alerts) was reserved for training, while the remaining 30% (8247 alerts) was reserved for testing. This split ratio is commonly used on datasets in machine learning, along with a ratio that reserves 80% for training and 20% for testing [72]. However, the 70/30 ratio was chosen in order to have more data to evaluate the performance of the machine learning models on. The entire preprocessing process was implemented as a function that could be called by other classes, in order to have the splitting of the dataset be different every time an algorithm wanted to learn from the dataset.

7.0.3 Initial Implementation of Machine Learning Models and Initial Testing

In order to test and verify that the data was correctly preprocessed for the three algorithms used in this thesis, minimally viable code was written for each algorithm in order to check if they would indeed process the data and give an accuracy score. For each of the three algorithms, a Python class was created. In each, the training and testing subsets of the dataset were loaded, and then used by the algorithms to learn the patterns of the dataset.

For ìmplementing Decision Trees used for classification tasks, Scikit-learn provides DecisionTreeClassifier[73]. SVMs are provided in various ways, and for this thesis, svm.SVC (C-Support Vector Classification)[74] was used. This implementation provides various kernel functions, which dictate the type of decision boundary used in classification, however this thesis pertains to the linear kernel, due to it utilizing linear decision boundaries.

As mentioned in the technical background [Chapter 3, section 3.3.1], Naive Bayes is a collection of algorithms. Each implementation is better suited for one type of feature[18], therefore a combination of some of the Naive Bayes algorithms would be desirable for the dataset used in this thesis. However, this thesis pertained to baseline models, which meant only choosing one implementation of Naive Bayes. This approach provided a baseline model that avoided potential overfitting from more complex models, while still respecting the independence assumption inherent in Naive Bayes. With this in mind, all implementations of Naive Bayes were tested, and after comparing accuracy scores of each implementation, the Bernoulli implementation, BernoulliNB, was chosen. This implementation assumes that each feature is a binary-valued[75], and although many features of the dataset were not binary, this implementation performed with substantially higher accuracy than other implementations such as GaussianNB, MultinomialNB and ComplementNB. Scikit-learn provides one more implementation, CategoricalNB, however it was not possible to get an accuracy score from this implementation because it required more computer memory than what was available in order to run.

After numerous attempts of preprocessing, the dataset was preprocessed properly, which was evident given that each algorithm processed the data and gave an accuracy score without error. However, the SVM implementation performed with a significantly worse score than the Decision Tree or Naive Bayes implementation, which had similar scores. After investigation, this was due to the data being used by the SVM algorithm not being standardized. To rectify this, StandardScaler was employed to standardize the data. This method has been shown to improve performance on SVMs[76], and after utilizing it to standardize the dataset, the accuracy score of the SVM did indeed increase significantly to an accuracy score similar to the other algorithms. StandardScaler was not used for the data used with the Decision Tree or Naive Bayes algorithms, as testing showed that data standardization did not impact the accuracy scores of these two algorithms whatsoever.

Chapter 8

Results

This chapter provides an overview of the performance of the three algorithms, decision trees, Naive Bayes and SVM, in classifying alerts from Cortex XDR pertaining to BIOC rules. The results are provided in the form of tables to show concise performance metrics, and confusion matrices to visualize the performance of the algorithms.

8.0.1 Performance Metrics

Tables

Table 8.1: Machine Learning Model Metrics: Before Hyperparameter Tuning

Model Name	Accuracy	Precision	Recall	F1-Score
Decision Tree	99.67%	99.65%	99.76%	99.70%
Naive Bayes (Bernoulli)	94.25%	90.86%	99.51%	94.99%
SVM(Linear)	99.49%	99.34%	99.73%	99.54%

Table 8.1 shows the baseline performance of the three machine learning algorithms, before hyperparameter tuning, with the four performance metrics used in this thesis, accuracy, precision, recall and F1-score. The Decision Tree algorithm showed the highest accuracy at 99.67%, closely followed by the SVM algorithm by a 0.18% lower accuracy. The Naive Bayes algorithm showed the lowest accuracy of 94.25%, showing that the Decision Tree and SVM algorithms were more successful in correctly predicting the label of the alerts. The same order is shown for precision, with both the Decision Tree and SVM algorithms showing over 99% precision and the Naive Bayes algorithm showing just over 90%. This showed that the Naive Bayes algorithm does not correctly identify TPs as well as the other algorithms.The same order is yet again shown in recall, although all algorithms showed a recall score of over 99% each, meaning that all three algorithms could correctly identify over 99% of TP alerts. Lastly, both the Decision Tree and SVM algorithm showed F1-scores at over 99% while the

Naive Bayes algorithm showed 94.99%. This showed that the Decision Tree and SVM algorithms had better overall performance in minimizing FP and FN classification than the Naive Bayes algorithm.

Table 8.2: Machine Learning Model Metrics: After Hyperparameter Tuning

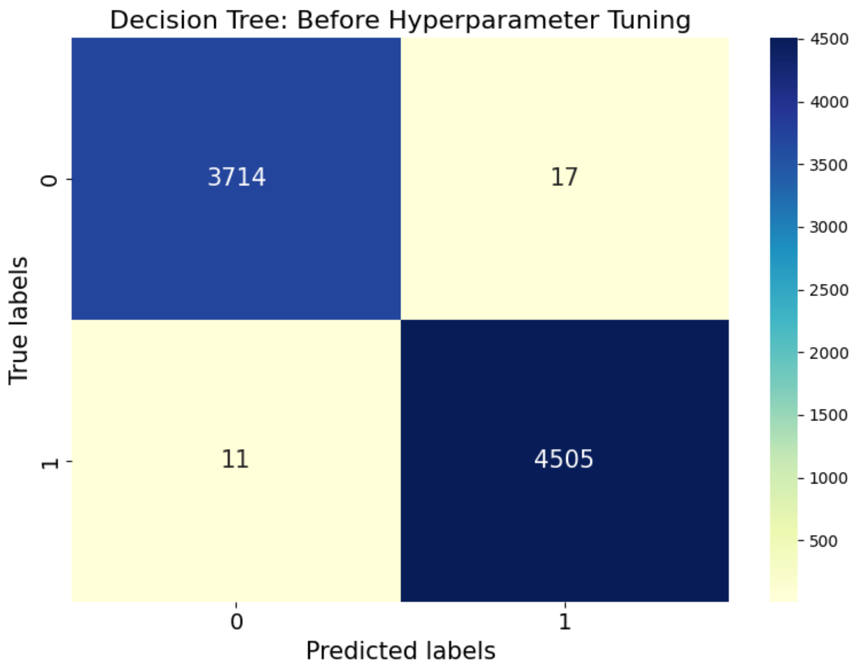
Model Name	Accuracy	Precision	Recall	F1-Score
Decision Tree	99.77%	99.71%	99.87%	99.79%
Naive Bayes (Bernoulli)	94.69%	91.56%	99.47%	95.35%
SVM(Linear)	99.53%	99.27%	99.87%	99.57%

Table 8.2 shows the performance of the three algorithms after tuning the hyperparameters of each algorithm. The order of highest to lowest performant of the algorithms in each metric stayed the same, although each algorithm showed a small improvement in overall performance. Scores in each metric were improved in each algorithm, with the exception of the recall score of the Naive Bayes algorithm lowering by 0.04% and the precision score of the SVM algorithm lowering by 0.07%. Even though these two scores were worsened, the overall average score of each algorithm improved, with the Decision Tree scores improving by an average of 0.09%, Naive Bayes scores improving by an average of 0.365% and SVM scores improving by an average of 0.035%.

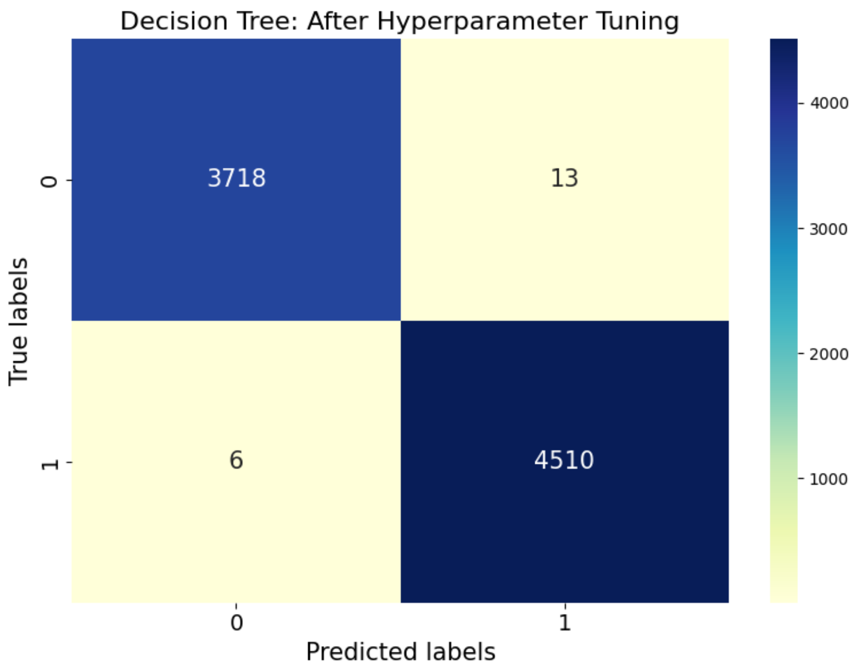
While the machine learning models initially demonstrated promising baseline performance, the marginal improvements to their overall performance show hyperparameter tuning as a noteworthy step that ultimately improved the machine learning models.

8.0.2 Confusion Matrices

In the following confusion matrices, TPs represent correct classifications of TP alerts, TNs represent correct classifications of FP alerts, FPs represent incorrect classifications of FP alerts and FNs represent incorrect classifications of TP alerts.



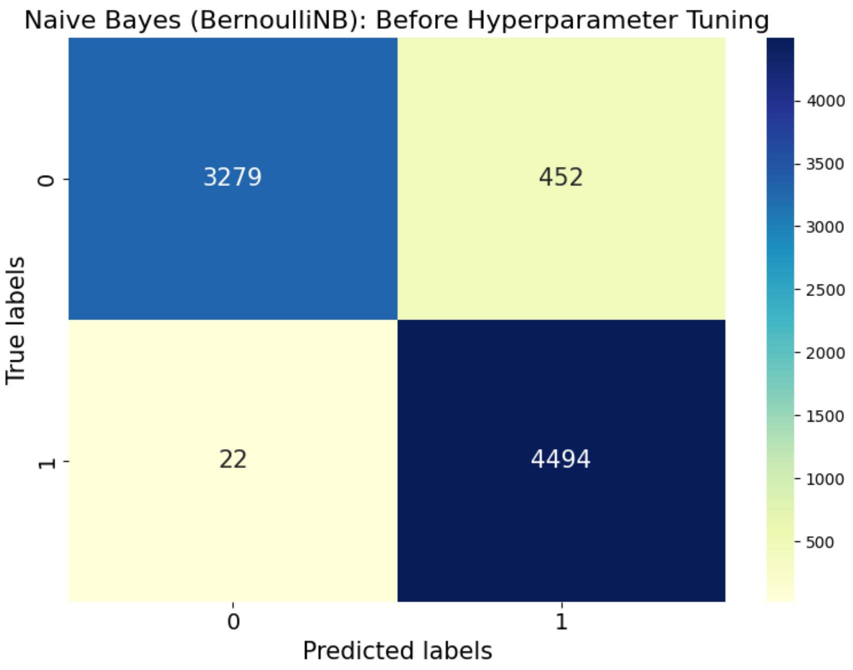
(a) Confusion matrix before hyperparameter tuning



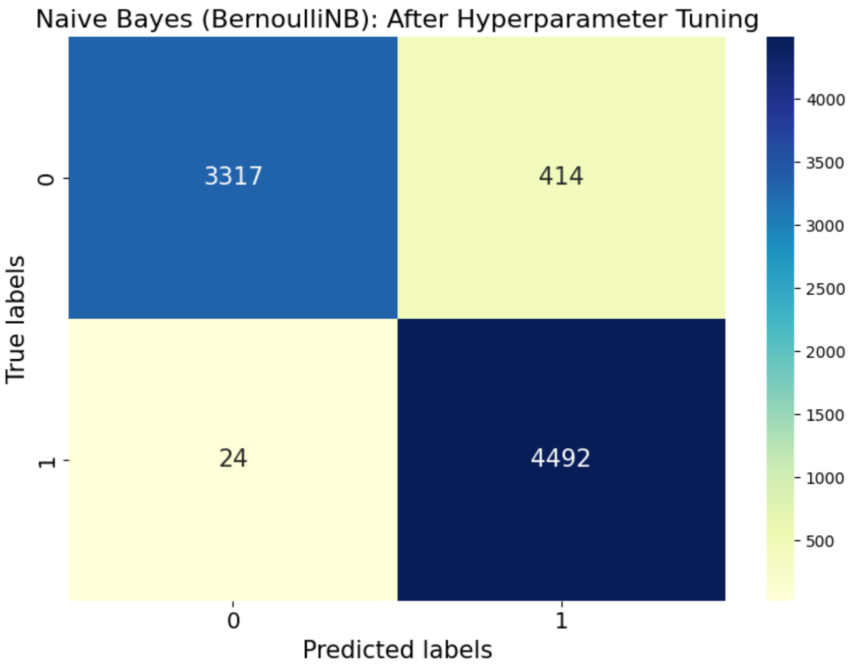
(b) Confusion matrix after hyperparameter tuning

Figure 8.1: Confusion matrices for the Decision Tree algorithm

As shown in figure 8.1(a) and figure 8.1(b), hyperparameter tuning on the Decision Tree algorithm increased the number of TP and TN classifications, while decreasing the number of FP and FN classifications. This shows that hyperparameter tuning was overall positive for this algorithm.



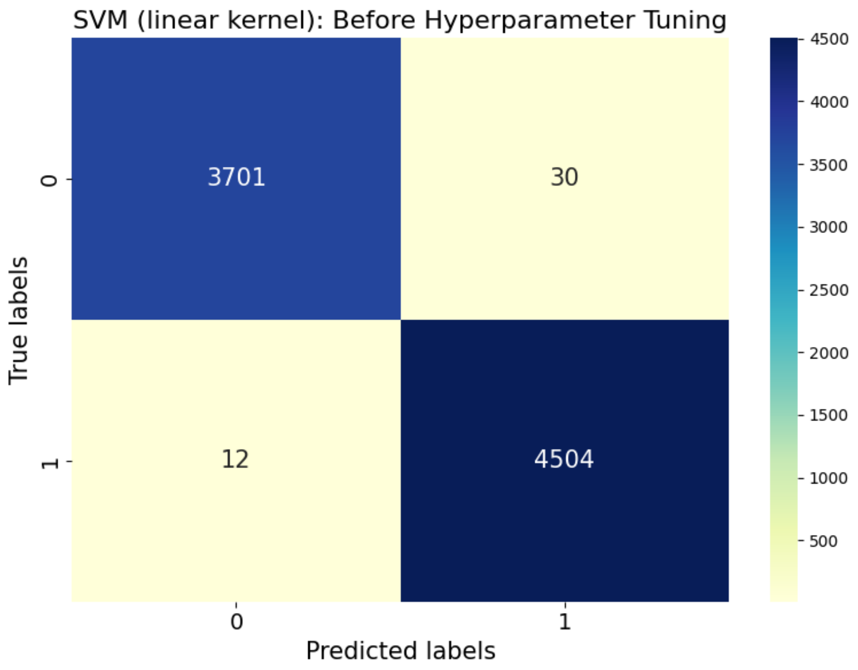
(a) Confusion matrix before hyperparameter tuning



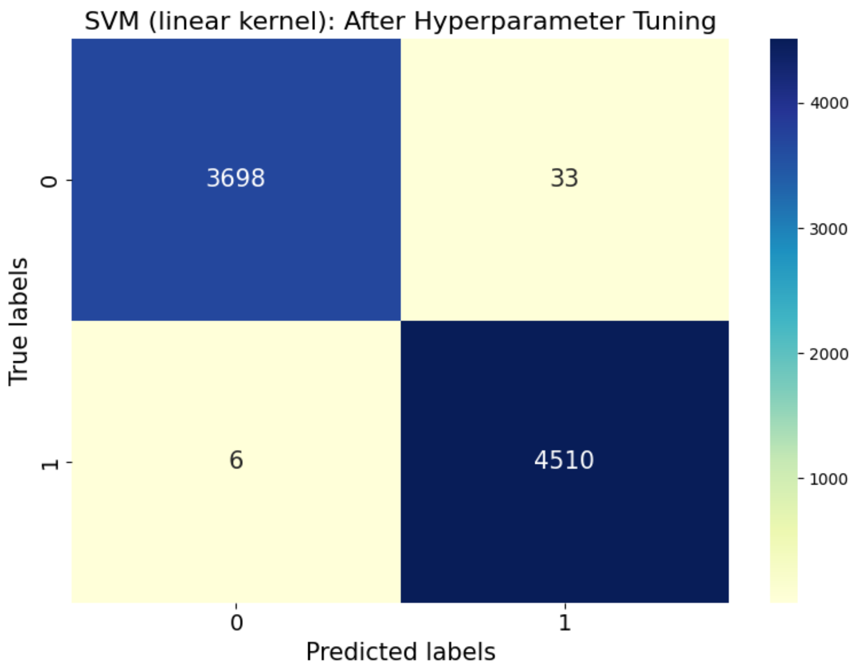
(b) Confusion matrix after hyperparameter tuning

Figure 8.2: Confusion matrices for the Naive Bayes algorithm

For the Naive Bayes algorithm, figure 8.2(a) and figure 8.2(b) show that hyperparameter tuning increased the number of TN classification, while decreasing FN classifications. Tuning did, however, increase FP classifications by 9% (2 classifications) and decrease TP classifications, although only by 0.000445%(2 classifications). Although the decrease in TP classifications is rather insignificant, the amount of decrease in TP classifications may scale with the size of the dataset.



(a) Confusion matrix before hyperparameter tuning



(b) Confusion matrix after hyperparameter tuning

Figure 8.3: Confusion matrices for the SVM algorithm

A similar development occurs for the SVM algorithm, as shown in figure 8.3(a) and figure 8.3(b). Hyperparameter tuning increased the number of TP classifications and decreased the number of FN classifications, although the number of FP classifications increased and TN classifications decreased.

Chapter 9

Conclusion

In this thesis, we propose and demonstrate the concept of applying machine learning to alert classification. We reviewed the background knowledge related to machine learning and the technical knowledge associated with the three machine learning algorithms that were applied in this thesis. We also investigated and reviewed earlier research works on applying machine learning to alert classification. These earlier research works are focused on network-based alert classification and cannot directly contribute to the current study of endpoint-based alert classification, but we still receive a certain amount of inspiration from them.

After conducting a series of data collection, preprocessing and model training, our results showed that machine learning is significantly effective in classifying true positive and false positive behavioral alerts generated by Cortex XDR which was used in this thesis. The results of this thesis also showed that among the three machine algorithms applied in this thesis, the Decision Tree algorithm ranked highest in this alert classification study by comparing all the evaluation metrics. It has shown a great effectiveness in classifying true and false positive alerts. In addition, we noticed that the results obtained by Support Vector Machine after tuning its hyperparameter are quite comparable to the Decision Tree, which shows that Support Vector Machine is also very efficient in the performance of classification alerts.

As a result of this thesis, we found that all three machine learning algorithms have more than 90 percent accuracy for alert classification, which shows their applicability in the real-world. This thesis confirms that machine learning has the potential to assist security analysts of Orange Cyberdefense as a way to optimize their workflow and reduce alert fatigue as well as its negative impact.

The potential social benefits of this thesis were also significant. For the cybersecurity industry, the benefits of machine learning technology are immeasurable. The most obvious benefit being that for security analysts machine learning can reduce alert fatigue as previously mentioned, with machine learning, security analysts are able to prioritize and process potential security events more efficiently and faster for those security activities that require immediate intervention.

9.0.1 Future work

There are many aspects in the work done in this thesis that can be improved. Unfortunately, in this thesis, due to privacy concerns, we were unable to use data from the real-world to build and train machine learning models. While the models showed promising performance on synthetic data, it is important to acknowledge that their performance on real-world data may not directly correlate. Although the synthetic data was designed to simulate real-world behavior, it can not fully capture the complexity and and variability inherent in real-world data. In the future we would like to see real-world data being used when training models to truly validate the usability of machine learning in the real world.

Moreover, in the future we would also like to collect more data from more different categories of alerts and preprocess the data in a more sophisticated and detailed way. In the machine learning part, we would like to include other machine learning algorithms and even more advanced deep learning algorithms such as artificial neural networks in the study. For this, we would need to conduct in-depth research and study on machine learning and artificial intelligence in the future.

References

[1] Palo Alto Networks, *Cortex XDR - Extended Detection and Response.* [Online]. Available: https://www.paloaltonetworks.com/cortex/cortex-xdr.

[2] B. A. Alahmadi, L. Axon, and I. Martinovic, “99% False Positives: A Qualitative Study of {SOC} Analysts’ Perspectives on Security Alarms,” en, 2022, pp. 2783–2800, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/alahmadi.

[3] Palo Alto Networks, *Working with BIOCs - Palo Alto Networks documentation portal.* [Online]. Available: https://docs-cortex.paloaltonetworks.com/r/Cortex-XDR/Cortex-XDR-Pro-Administrator-Guide/Working-with-BIOCs.

[4] S. Brown, *Machine learning, explained*, MIT Sloan, Apr. 2021. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

[5] G. Apruzzese, P. Laskov, E. Montes de Oca, *et al.*, “The Role of Machine Learning in Cybersecurity,” *Digital Threats: Research and Practice*, vol. 4, no. 1, 8:1–8:38, Mar. 2023. DOI: 10.1145/3545574. [Online]. Available: https://dl.acm.org/doi/10.1145/3545574.

[6] A. 23 and 2. a. 0. AM, *Cortex xdr datasheet*, Palo Alto Networks. [Online]. Available: https://www.paloaltonetworks.com/resources/datasheets/cortex-xdr.

[7] *Palo alto networks documentation portal*, docs-cortex.paloaltonetworks.com. [Online]. Available: https://docs-cortex.paloaltonetworks.com/r/Cortex-XDR/Cortex-XDR-Pro-Administrator-Guide/Alerts.

[8] *What is a Virtual Machine? — VMware Glossary*, en-US. [Online]. Available: https://www.vmware.com/topics/glossary/content/virtual-machine.html.

[9] *What are virtual machines? — IBM*, en-us. [Online]. Available: https://www.ibm.com/topics/virtual-machines.

[10] jdanyow, *Download a Windows virtual machine - Windows app development*, en-us. [Online]. Available: `https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/`.

[11] IBM, *What is machine learning?* IBM, 2023. [Online]. Available: `https://www.ibm.com/topics/machine-learning`.

[12] IBM, *What is supervised learning? — ibm*, IBM, 2023. [Online]. Available: `https://www.ibm.com/topics/supervised-learning`.

[13] *What is unsupervised learning?* Google Cloud. [Online]. Available: `https://cloud.google.com/discover/what-is-unsupervised-learning`.

[14] *Supervised vs unsupervised learning - difference between machine learning algorithms - aws*, Amazon Web Services, Inc. [Online]. Available: `https://aws.amazon.com/compare/the-difference-between-machine-learning-supervised-and-unsupervised/`.

[15] M. Awad and R. Khanna, "Support vector machines for classification," *Efficient Learning Machines*, pp. 39–66, 2015. DOI: `10.1007/978-1-4302-5990-9_3`.

[16] *An idiot's guide to support vector machines (svms) r. berwick, village idiot svms: A new generation of learning algorithms*. [Online]. Available: `https://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf`.

[17] *What Are Naïve Bayes Classifiers? — IBM*, en-us. [Online]. Available: `https://www.ibm.com/topics/naive-bayes`.

[18] *1.9. Naive Bayes*, en. [Online]. Available: `https://scikit-learn/stable/modules/naive_bayes.html`.

[19] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, Mar. 1986. DOI: `10.1007/bf00116251`. [Online]. Available: `http://www.hunch.net/~coms-4771/quinlan.pdf`.

[20] K. Chumachenko, "MACHINE LEARNING METHODS FOR MALWARE DETECTION AND CLASSIFICATION," en,

[21] S. Tangirala, "Evaluating the Impact of GINI Index and Information Gain on Classification using Decision Tree Classifier Algorithm*," en, *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 2, 2020, ISSN: 21565570, 2158107X. DOI: `10.14569/IJACSA.2020.0110277`. [Online]. Available: `http://thesai.org/Publications/ViewPaper?Volume=11&Issue=2&Code=IJACSA&SerialNo=77`.

[22] P. S. Foundation, *3.12.0 documentation*, Nov. 2023. [Online]. Available: `http://docs.python.org`.

[23] Scikit-Learn, *User guide: Contents — scikit-learn 0.22.1 documentation*, 2019. [Online]. Available: `https://scikit-learn.org/stable/user_guide.html`.

[24] I. Idan Novogroder, *Data preprocessing in machine learning: Steps best practices*, Git for Data - lakeFS, Mar. 2024. [Online]. Available: `https://lakefs.io/blog/data-preprocessing-in-machine-learning/`.

[25] *6.3. preprocessing data — scikit-learn 0.22.2 documentation*, scikit-learn.org. [Online]. Available: `https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features`.

[26] scikit-learn developers, *Sklearn.preprocessing.labelencoder — scikit-learn 0.22.1 documentation*, Scikit-learn.org, 2019. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html`.

[27] *Sklearn.preprocessing.ordinalencoder*, scikit-learn. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html`.

[28] S. learn, *Sklearn.feature_extraction.text.tfidfvectorizer — scikit-learn 0.20.3 documentation*, Scikit-learn.org, 2018. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`.

[29] *6.2. feature extraction*, scikit-learn. [Online]. Available: `https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting`.

[30] Z. Jaadi, *When and why to standardize your data?* Built In, Sep. 2019. [Online]. Available: `https://builtin.com/data-science/when-and-why-standardize-your-data`.

[31] Scikit-Learn, *Sklearn.preprocessing.standardscaler — scikit-learn 0.21.2 documentation*, Scikit-learn.org, 2019. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`.

[32] J. Murel and E. Kavlakoglu, *The confuse matrix*, IBM.com, Jan. 2024. [Online]. Available: `https://www.ibm.com/content/dam/connectedassets-adobe-cms/worldwide-content/creative-assets/s-migr/ul/g/c8/a7/binary-matrix.component.complex-narrative-xl-retina.ts=1712087356966.png/content/adobe-cms/us/en/topics/confusion-matrix/jcr:content/root/table_of_contents/body/content_section_styled/content-section-body/complex_narrative_390941229/items/content_group/image`.

[33] Google, *Classification: Accuracy—machine learning crash course*, Google Developers, 2019. [Online]. Available: `https://developers.google.com/machine-learning/crash-course/classification/accuracy`.

[34] *Sklearn.metrics.recall$_s$core*, scikit-learn. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score`.

[35] $Sklearn.metrics.precision_score|scikit-learn0.24.1documentation$, scikit-learn.org. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score`.

[36] *Sklearn.metrics.f1$_s$core*, scikit-learn. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score`.

[37] Pandas, *User guide — pandas 1.0.1 documentation*, 2014. [Online]. Available: `https://pandas.pydata.org/docs/user_guide/index.html`.

[38] *3.3. metrics and scoring: Quantifying the quality of predictions — scikit-learn 0.22.1 documentation*, scikit-learn.org. [Online]. Available: `https://scikit-learn.org/stable/modules/model_evaluation.html`.

[39] $Sklearn.model_selection.gridsearchcv|scikit-learn0.22documentation$, Scikit-learn.org, 2019. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV`.

[40] AWS, *What is overfitting? - overfitting - aws*, Amazon Web Services, Inc. [Online]. Available: `https://aws.amazon.com/what-is/overfitting/`.

[41] SciKit-Learn, *3.1. cross-validation: Evaluating estimator performance — scikit-learn 0.21.3 documentation*, Scikit-learn.org, 2009. [Online]. Available: `https://scikit-learn.org/stable/modules/cross_validation.html`.

[42] R. E. Crossler, A. C. Johnston, P. B. Lowry, Q. Hu, M. Warkentin, and R. Baskerville, "Future directions for behavioral information security research," *Computers Security*, vol. 32, pp. 90–101, Feb. 2013. DOI: `10.1016/j.cose.2012.09.010`.

[43] S. M. M. Hossain, R. Couturier, J. Rusk, and K. B. Kent, "Automatic event categorizer for SIEM," in *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '21, USA: IBM Corp., Nov. 2021, pp. 104–112.

[44] T. Ban, N. Samuel, T. Takahashi, and D. Inoue, "Combat Security Alert Fatigue with AI-Assisted Techniques," en, in *Cyber Security Experimentation and Test Workshop*, Virtual CA USA: ACM, Aug. 2021, pp. 9–16, ISBN: 978-1-4503-9065-1. DOI: `10.1145/3474718.3474723`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3474718.3474723`.

[45] T. Ban, T. Takahashi, S. Ndichu, and D. Inoue, "Breaking Alert Fatigue: AI-Assisted SIEM Framework for Effective Incident Response," en, *Applied Sciences*, vol. 13, no. 11, p. 6610, Jan. 2023, Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417. DOI: `10.3390/app13116610`. [Online]. Available: `https://www.mdpi.com/2076-3417/13/11/6610`.

[46] T. Pietraszek, "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection," en, in *Recent Advances in Intrusion Detection*, E. Jonsson, A. Valdes, and M. Almgren, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2004, pp. 102–124, ISBN: 978-3-540-30143-1. DOI: `10.1007/978-3-540-30143-1_6`.

[47] L. F. Kwok and K. H. Law, "IDS false alarm filtering using KNN classifier," vol. 3325, Aug. 2004, pp. 114–121, ISBN: 978-3-540-24015-0. DOI: `10.1007/978-3-540-31815-6_10`.

[48] Y. Meng and L. F. Kwok, "Adaptive False Alarm Filter Using Machine Learning in Intrusion Detection," en, in *Practical Applications of Intelligent Systems*, Y. Wang and T. Li, Eds., ser. Advances in Intelligent and Soft Computing, Berlin, Heidelberg: Springer, 2012, pp. 573–584, ISBN: 978-3-642-25658-5. DOI: `10.1007/978-3-642-25658-5_68`.

[49] T. Subbulakshmi, M. George, and S. M. Shalinie, "Real Time Classification and Clustering Of IDS Alerts Using Machine Learning Algorithms," *International Journal of Artificial Intelligence & Applications*, vol. 1, Jan. 2010. [Online]. Available: `https://www.researchgate.net/publication/45706056_Real_Time_Classification_and_Clustering_Of_IDS_Alerts_Using_Machine_Learning_Algorithms`.

[50] K. Alsubhi, E. Al-Shaer, and R. Boutaba, "Alert prioritization in Intrusion Detection Systems," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, ISSN: 2374-9709, Apr. 2008, pp. 33–40. DOI: `10.1109/NOMS.2008.4575114`. [Online]. Available: `https://ieeexplore.ieee.org/document/4575114`.

[51] *1998 DARPA Intrusion Detection Evaluation Dataset — MIT Lincoln Laboratory*. [Online]. Available: `https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset`.

[52] *1999 DARPA Intrusion Detection Evaluation Dataset — MIT Lincoln Laboratory*. [Online]. Available: `https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset`.

[53] *2000 DARPA Intrusion Detection Scenario Specific Datasets — MIT Lincoln Laboratory*. [Online]. Available: `https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-datasets`.

[54] *Snort - Network Intrusion Detection & Prevention System*. [Online]. Available: `https://www.snort.org/`.

[55] Deland-Han, *Memory dump file options - Windows Server*, en-us, Dec. 2023. [Online]. Available: `https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/memory-dump-file-options`.

[56] *Using batch files*, en-us, Sep. 2009. [Online]. Available: `https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb490869(v=technet.10)`.

[57] sdwheeler, *Get-Date (Microsoft.PowerShell.Utility) - PowerShell*, en-us. [Online]. Available: `https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-date?view=powershell-7.4`.

[58] sdwheeler, *Get-Random (Microsoft.PowerShell.Utility) - PowerShell*, en-us. [Online]. Available: `https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-random?view=powershell-7.4`.

[59] *What is Noise in Machine Learning*, en. [Online]. Available: `https://dataheroes.ai/glossary/noise-in-machine-learning/`.

[60] Elliot, *Robiot/AlphaClicker*, original-date: 2021-07-30T16:45:24Z, Feb. 2024. [Online]. Available: `https://github.com/robiot/AlphaClicker`.

[61] $Pandas.read_csv|pandas2.2.2documentation$, pandas.pydata.org. [Online]. Available: `https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html#pandas.read_csv`.

[62] N. Venkat, *The Curse of Dimensionality: Inside Out.* Sep. 2018. DOI: `10.13140/RG.2.2.29631.36006`.

[63] L. Chen, "Curse of Dimensionality," en, in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds., Boston, MA: Springer US, 2009, pp. 545–546, ISBN: 978-0-387-39940-9. DOI: `10.1007/978-0-387-39940-9_133`. [Online]. Available: `https://doi.org/10.1007/978-0-387-39940-9_133`.

[64] *Pandas.dataframe.fillna — pandas 1.0.5 documentation*, pandas.pydata.org. [Online]. Available: `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html`.

[65] *What is code signing? — digicert faq*, www.digicert.com. [Online]. Available: `https://www.digicert.com/faq/code-signing-trust/what-is-code-signing`.

[66] Karl-Bridge-Microsoft, *Thread handles and identifiers - win32 apps*, Microsoft, Jan. 2021. [Online]. Available: `https://learn.microsoft.com/en-us/windows/win32/procthread/thread-handles-and-identifiers`.

[67] *Ieee standards association*, IEEE Standards Association. [Online]. Available: `https://standards.ieee.org/ieee/1003.1/7101/`.

[68] *Time — time access and conversions*, Python documentation. [Online]. Available: `https://docs.python.org/3/library/time.html#epoch`.

[69] *Time — time access and conversions — python 3.10.5 documentation*, docs.python.org. [Online]. Available: `https://docs.python.org/3/library/time.html#time.time`.

[70] *Datetime — basic date and time types*, Python documentation. [Online]. Available: `https://docs.python.org/3/library/datetime.html#datetime.datetime.timestamp`.

[71] , $Sklearn.model_selection.train_test_split|scikit-learn0.20.3documentation$, Scikit-learn.org, 2018. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`.

[72] J. Roller, *A Practical Guide to Working with Testing and Training Data in ML Projects*, en-US, Jun. 2023. [Online]. Available: `https://www.computer.org/publications/tech-news/trends/machine-learning-projects-training-testing/`.

[73] scikit learn, *Sklearn.tree.decisiontreeclassifier — scikit-learn 0.22.1 documentation*, Scikit-learn.org, 2019. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`.

[74] *Sklearn.svm.svc — scikit-learn 0.22 documentation*, Scikit-learn.org, 2019. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`.

[75] *Sklearn.naive$_b$ayes.bernoullinb*, scikit-learn. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB`.

[76] M. M. Ahsan, M. A. P. Mahmud, P. K. Saha, K. D. Gupta, and Z. Siddique, "Effect of data scaling methods on machine learning algorithms and model performance," *Technologies*, vol. 9, p. 52, Jul. 2021. DOI: `10.3390/technologies9030052`.