FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Asad Hasan

APPLICATION OF MACHINE LEARNING ON CYBER THREAT INTELLIGENCE DATA

Master's Thesis
Degree Programme in Computer Science and Engineering
June 2024

Hasan A. (2024) Application of Machine Learning on Cyber Threat Intelligence Data. University of Oulu, Degree Programme in Computer Science and Engineering, 66 p.

ABSTRACT

Cybersecurity stands as a paramount concern in today's digital landscape, with the proliferation of cyber threats posing significant risks to individuals, organizations, and governments worldwide. In response to this escalating challenge, the fusion of machine learning (ML) techniques with cyber threat intelligence (CTI) data emerges as a promising approach which can help develop defense mechanisms against malicious activities. This thesis investigates the application of ML algorithms to CTI datasets for predicting the categories of cyber attacks, aiming to model CTI datasets for ML and develop a predictive model capable of discerning attack categories amidst the complex cyber threat landscape. Methodologically, the research employs a systematic approach to dataset processing and ML classifier training, encompassing feature identification, engineering, and selection, as well as iterative model refinement and validation. The envisioned outcomes include the ability to model CTI datasets to effectively predict attack categories with confidence. Through extensive data processing, exploratory data analysis, and feature engineering, this research contributes to the advancement of cybersecurity by providing a robust framework for modelling CTI data and accurately predicting cyber attack categories.

Keywords: Cybersecurity, Cyber threats, Machine learning (ML), Cyber threat intelligence (CTI) data, Defense mechanisms, Predicting the categories of cyber attacks, Dataset processing, ML classifier training, Feature engineering, Exploratory data analysis, Predict attack categories, Cyber threat landscape

TABLE OF CONTENTS

FOREWORD

This thesis was completed as part of the NEWSROOM project, funded by the European Union under the European Defence Fund (GA no. 101121403 - NEWSROOM). Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

I want to thank the Oulu University Secure Programming Group (OUSPG) and my supervisor, Kimmo Halunen, for making this work possible and helping me deliver it within the University's specified timeline. Special thanks go to my second supervisor, Jaakko Suutala, who guided me in technical situations and corrected my direction. Lastly, I want to thank Arctic Security for graciously providing datasets for this research.

Oulu, June 18th, 2024

Asad Hasan

LIST OF ABBREVIATIONS AND SYMBOLS

AUC	Area Under the Curve
AI	Artificial Intelligence
BGP	Border Gateway Protocol
BRNN-LSTM	Bidirectional Recurrent Neural Network - Long Short-Term Memory
CTI	Cyber Threat Intelligence
CVEs	Common Vulnerabilities and Exposures
DL	Deep Learning
DNS	Domain Name System
EDA	Exploratory Data Analysis
EWS	Early Warning System
GBDT	Gradient Boosted Decision Tree
GPUs	Graphics Processing Units
HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrusion Detection Systems
IOCs	Indicators of Compromise
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
ML	Machine Learning
MLlib	Machine Learning library
MITRE-ATT&CK	MITRE Adversarial Tactics, Techniques, and Common Knowledge Framework
NCC	Normalized Cross-Correlation
NLP	Natural Language Processing
NLPT	Natural Language Processing Techniques
NLTK	Natural Language Toolkit
OSINT	Open-Source Intelligence
OOB	Out-of-Bag
ROC	Receiver Operating Characteristic
RF	Random Forest
SIEM	Security Incident and Event Management
SVM	Support Vector Machine
TPUs	Tensor Processing Units
TTPs	Tactics, Techniques, and Procedures
UML	Unified Modeling Language

1. INTRODUCTION

In an era marked by the omnipresence of digital infrastructure, cybersecurity has emerged as a critical concern for individuals, organizations, and governments alike [1, 2]. The exponential growth in cyber threats poses substantial risks to data integrity, operational continuity, and financial stability [3]. Amidst this backdrop, the application of machine learning (ML) techniques on cyber threat intelligence (CTI) data could prove vital in fortifying defense mechanisms against malicious activities and actors.

This thesis explores the fusion of machine learning algorithms with CTI datasets to predict the categories of cyber attacks. The dataset under scrutiny encompasses a plethora of information pertaining to observations of companies and software vulnerabilities. By harnessing the power of ML, the aim is to develop a predictive model capable of discerning attack categories.

The rationale behind this research stems from the pressing need to address the escalating challenges posed by cyber threats [3]. Incidents of data breaches, ransomware attacks, and other forms of cyber assaults continue to exact a heavy toll on organizations, both in terms of financial losses and reputational damage [1, 3]. Despite the burgeoning interest in cybersecurity, there exists a noticeable dearth of significant research endeavors exploring the application of ML techniques on CTI data in light of relevant work [4, 5, 6]. Consequently, this research aims to lay the groundwork for designing robust systems capable of predicting the category of cyber attacks.

Methodologically, the research methodology entails a systematic approach to processing the acquired dataset and training the ML classifier. Key features relevant to cyber threat intelligence will be identified and the rest will be engineered. Afterwards, appropriate features will be selected to facilitate the training process. The dataset will be partitioned into training and testing sets, with various sampling techniques such as random sampling being employed to ensure robustness and generalizability of the predictive model. Through iterative refinement and validation, the classifier will be fine-tuned to accurately predict the category of cyber attacks.

The envisioned outcomes of this research encompass the ability to predict attack categories with a certain degree of confidence. Specifically, the predictive model aims to categorize cyber threats into one of four classes: suspected compromise, known vulnerabilities, public exposure, and potential threats. **Suspected compromise** involves identifying specific recipient assets that have been observed by a third party to be compromised [7], prompting immediate incident response to minimize damage. **Known vulnerabilities** focus on identifying and managing technical vulnerabilities, often enumerated through common vulnerabilities and exposures (CVEs), which require fixes to prevent exploitation [7]. **Public exposure** pertains to identifying services or ports that are publicly exposed to the Internet [7], necessitating proper configuration management to secure them. **Potential threats** involve enumerating observations that can cause harm to the organization, such as a service being blocked by third parties, but require further analysis for specific attribution and mitigation [7]. By outlining the nature and severity of cyber threats, organizations can proactively devise countermeasures and allocate resources judiciously to mitigate risks and fortify their cybersecurity posture [2].

In summary, this thesis contributes to the field of cybersecurity by harnessing the power of machine learning to predict cyber attack categories. By leveraging

CTI data, the research emphasizes extensive data processing, including cleaning and handling null values, to ensure data quality. Through comprehensive exploratory data analysis (EDA), patterns and relationships within the data are uncovered, providing crucial insights. Additionally, feature engineering is employed to transform raw data into meaningful features that enhance the predictive power of robust ML algorithms. Together, these steps aim to build a sophisticated framework for accurately predicting categories of cyber attacks.

1.1. Research Objectives

With cyber threat intelligence (CTI) data serving as a crucial repository of insights into the tactics, techniques, and procedures (TTPs) employed by threat actors, this study investigates the feasibility of leveraging machine learning techniques to predict the category of cyber attacks based on CTI data. The thesis aim is to asnwer one main research question:

- Can machine learning be effectively applied to cyber threat intelligence data to predict the category of cyber attacks?

Two sub-research questions are created to answer the main research question:

- How CTI data can be modeled to achieve successful attack category prediction?
- What are the primary input features crucial for accurate prediction, and why does the system consider them significant?

In pursuit of these objectives, the study will delve into the intricacies of CTI data analysis and machine learning methodologies, aiming to establish a robust framework that acts as ground-basis for predictive analysis in cybersecurity. By effectively modeling CTI data for machine learning, exploring the significance of key input features and their role in enhancing category prediction accuracy, this research contributes valuable insights into the potential of utilizing CTI data with machine learning in bolstering cyber threat prediction capabilities.

1.2. Structure of Thesis

The structure of this thesis is organized into several chapters. The introductory chapter sets the stage by acquainting the reader with the overarching topic and the research problem at hand. It delves deeper into the research objectives, explicating the research goals guiding this investigation. Chapter two, unearths the theoretical underpinnings, literature review and relevant work related to this research. Chapter three describes the research design used to answer the research objectives. Fourth Chapter briefly explains the implementation of the research design. In the fifth Chapter, results gathered from the experiment are presented followed by a subsequent Section dedicated to discuss the results. The thesis concludes with a summary in the end.

2. LITERATURE REVIEW

Machine learning is a branch of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to progressively improve their performance on a specific task through experience or data [8]. Rather than being explicitly programmed to accomplish a task, machine learning algorithms use patterns and inference to learn from data and make predictions or decisions. The essence of machine learning lies in its ability to automatically identify patterns and make intelligent decisions without explicit human intervention. This field encompasses a wide range of techniques and methodologies, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [9]. **Supervised learning** is a type of machine learning where the model is trained on labeled data. In this approach, each training example is paired with an output label [10]. In **unsupervised learning**, a model is trained on data without labeled response. The model tries to learn the underlying structure of the data by identifying patterns, relationships, or clusters within the input data [11]. **Semi-supervised learning** is a hybrid approach that uses both labeled and unlabeled data for training [9]. Lastly, in **reinforcement learning** an agent learns to make decisions by performing actions in an environment to maximize some notion of cumulative reward. [9]

Cyber threat intelligence data refers to information collected, analyzed, and disseminated to understand cyber threats, including their actors, tactics, techniques, and procedures, as well as indicators of compromise (IOCs) [12]. It enables organizations to make informed decisions at strategic, operational, and tactical levels by collecting and analyzing information about threats, including actors, tools, techniques, and procedures. While definitions may vary [12], CTI involves the interpretation of raw data into meaningful information that addresses the intent, opportunity, and capability of adversaries to cause harm. This data is gathered from various sources, such as open-source intelligence (OSINT), dark web monitoring, malware analysis, intrusion detection systems (IDS), and security incident and event management (SIEM) tools. These terms are elaborated in more detail in Table 1.

Table 1. Definitions of few key terms associated with cybersecurity

Term	**Definition**
Open-Source Intelligence (OSINT)	Collection and analysis of publicly available information to generate actionable intelligence for security assessments, competitive intelligence, and threat analysis [13].
Dark Web Monitoring	Tracking and analyzing activities on the dark web to identify potential threats such as data breaches, cybercrime activities, or illicit trade. Dark Web refers to a network of cryptographically hidden sites [14].
Malware Analysis	Examining malicious software to understand its behavior, origins, and impact. Involves static and dynamic analysis to develop detection and mitigation strategies. [15]

Term	Definition
Intrusion Detection Systems (IDS)	Security tools designed to detect and monitor unauthorized access or breaches in a network or system [16].
Security Incident and Event Management (SIEM) Tools	Systems that provide real-time analysis of security alerts generated by applications and network hardware, collecting and analyzing log and event data [17].

In recent years, the application of artificial intelligence and machine learning in cybersecurity has garnered significant attention due to its potential to revolutionize threat detection, prediction, and mitigation strategies [18]. A critical examination of existing literature reveals a plethora of research endeavors exploring various aspects of this intersection [4, 5, 6]. Within this domain, several studies have focused on enhancing anomaly detection, network intrusion prevention, and predictive analysis through the utilization of advanced algorithms and pattern recognition capabilities offered by AI and ML [19, 20, 5]. Specifically, methodologies such as neural networks, genetic algorithms, and natural language processing have been explored, highlighting the interdisciplinary nature of AI and ML applications in cybersecurity [21, 22]. Moreover, within the specific domain of cyber threat intelligence (CTI), AI and ML methodologies hold immense potential for automating data processing, extracting actionable insights from disparate data sources, and facilitating predictive analysis [4, 5]. Notably, research in this area has emphasized the need for a nuanced understanding of CTI's capabilities and its implications for organizational cybersecurity strategies, underscoring the pivotal role of AI and ML technologies in augmenting traditional cybersecurity measures [6].

Despite the growing recognition of AI and ML's potential in cybersecurity, organizations continue to face challenges in fully harnessing CTI and implementing AI-driven solutions effectively. One of the primary motivations behind research in this area is the escalating volume and impact of security incidents globally [23], which necessitates innovative approaches to fortify organizational cyber defenses. Cyber attacks not only pose significant financial and reputational risks to organizations but also undermine the integrity of critical infrastructure and national security [2]. Consequently, there is a pressing need for proactive measures to detect, predict, and mitigate cyber threats in real-time, which has spurred interest in leveraging AI and ML technologies for CTI.

In addressing the aforementioned challenges, researchers have proposed various methodologies and frameworks aimed at harnessing AI and ML for CTI [21, 24, 25, 6]. One approach involves the development of predictive models capable of forecasting cyber attacks based on historical data and key indicators. For instance, in [21], the authors introduce a novel approach utilizing a deep learning framework called BRNN-LSTM for forecasting cyber threats. Through empirical studies, the paper demonstrates the superiority of BRNN-LSTM over traditional statistical methods in predicting cyber attack rates, highlighting its potential for improving accuracy in cyber threat forecasting. Similarly, in [4], the authors investigate the feasibility of predicting cyber attacks solely based on publicly available data. By training a machine

learning algorithm on attack mentions from public repositories, the study illustrates the potential of such an approach, albeit with limitations in scope and accuracy.

Another approach involves the utilization of AI and ML algorithms for automated data processing and extraction of actionable insights from CTI datasets. The authors in [26] propose a framework for detecting cybersecurity events from online social media platforms. By employing dynamic query expansion methods and novel strategies, the mentioned study [26] identifies and characterizes cyber attacks dynamically. Extensive empirical evaluations underscore the effectiveness of the proposed framework in detecting various cyber threats early. Additionally, in [25], the author advocates for leveraging the MITRE ATT&CK framework to enhance organizations' understanding of their security solutions and threat detection capabilities. Through a meticulous analysis of real-world threat intelligence reports, this study identifies detection gaps and provides insights for strengthening organizational defense mechanisms.

One notable study [5] investigates the feasibility of forecasting cyber security incidents for organizations, aiming to understand if such incidents can be predicted before they occur. According to [5], its application in prediction, except for some instances, is relatively less explored.

In implementing these methodologies, researchers have emphasized the importance of integrating AI, ML and deep learning driven solutions into existing cybersecurity frameworks and workflows [27, 6, 20]. This necessitates collaboration between cybersecurity experts, data scientists, and IT professionals to develop tailored solutions that address specific organizational needs and challenges [27]. Furthermore, researchers have highlighted the significance of continuous evaluation and refinement of AI and ML models to adapt to evolving cyber threats and ensure optimal performance in real-world scenarios [27]. Ultimately, by synthesizing insights from existing literature and leveraging AI and ML technologies effectively, organizations can fortify their cybersecurity posture and mitigate the risks posed by cyber threats in an increasingly interconnected digital landscape.

2.1. Methodological Insights from 'Cloudy with a Chance of Breach: Forecasting Cyber Security Incidents'

This Section describes the detailed methodology used to predict cybersecurity incidents in [5] because it highly correlates with the research work of this study. In this study, features are engineered and predictions are made, with consideration for short-term and long-term forecasting scenarios based on the time period leading up to the incident month. Contrary to the objectives of the main research, this approach emphasizes on the utilization of temporal features for forecasting security incidents.

In [5] security posture is evaluated through measurements of misconfigurations and malicious activities originating from networks. Mismanagement symptoms, such as open recursive resolvers, DNS source port randomization issues, BGP misconfigurations, untrusted HTTPS certificates, and open SMTP mail relays, are examined using data snapshots collected in early 2013. Paper reports over 200,000 misconfigured DNS resolvers and about 10.3 million untrusted HTTPS certificates giving an indicator about the size of dataset. Malicious activity data spanning from May 2013 to December 2014, sourced from reputation blacklists, categorize spam,

phishing, malware, and scanning activities. Security incident data, collected from August 2013 to December 2014, includes publicly available reports from the VERIS Community Database, Hackmageddon, and the Web Hacking Incidents Database. Only incidents directly related to cybersecurity issues are considered, excluding those from physical attacks or internal misoperation.

The feature set employed in the study [5] comprises primary and secondary features, totaling 258 attributes. The primary features encompass five mismanagement symptoms and time series data for malicious activities, along with organization size. A single snapshot of each mismanagement symptom is reported to be used mainly due to systems not being re-configured on a daily or weekly basis. Mismanagement symptoms are quantified as ratios of misconfigurations to total systems, while malicious activity time series represent counts of blacklisted IPs for spam, phishing, and scanning activities over time. Secondary features, derived from time series data, aim to capture behavioral patterns in malicious activities, including persistency and change. These features are quantified through statistics such as the magnitude and duration of "good" and "bad" periods, as well as the frequency of transitions between these states. Both primary and secondary features are examined for their predictive power, with distinct distributions observed between victim and non-victim organizations, suggesting their relevance in predicting security incidents.

The training and testing procedure for constructing the predictor involves two key steps in the referenced article [5]. Firstly, in the training step, a subset of incident or victim organizations (referred to as Group(1)) is selected based on the time stamps of reported incidents, along with a comparable-sized randomly selected set of non-victim organizations (referred to as Group(0)). This random sub-sampling of non-victim organizations is necessary due to the vast disparity in the dataset, with close to three million records for non-victim organizations compared to less than a thousand records for victim organizations. Each victim organization's feature set includes mismanagement symptoms and various time series collected over specific periods prior to the incident occurrence, while for non-victim organizations, features are collected over a period leading up to the first incident in Group(1). Secondly, in the testing step, victim organizations not included in Group(1) and a larger set of randomly selected non-victim organizations are used. Feature sets for testing are obtained similarly for the training process, with consideration for short-term and long-term forecasting scenarios based on the time period leading up to the incident month. The classifier's output is a risk probability, which is converted into a binary label using a threshold. By moving this threshold, the author obtained different performances constituting a receiver operating characteristic (ROC) curve.

The training and testing procedure was repeated multiple times resulting in multiple classifiers. Therefore, reported testing results are averages over all classifier versions in this referenced study [5].

2.2. Methodological Insights from 'A Framework for Fast and Efficient Cyber Security Network Intrusion Detection Using Apache Spark'

This Section describes methodology used in [20] to develop a cyber security intrusion detection system, which is relevant to the work of this study. The paper [20] discusses

feature selection schemes, describes tests for evaluating feature relevance and explores various classification-based intrusion detection schemes which are relevant to the main work.

The paper [20] begins by discussing feature selection schemes, including correlation-based and chi-squared algorithms, which evaluate the relevance and inter-correlation of features to improve system performance. The correlation-based feature selection algorithm evaluates the usefulness of individual features and their inter-correlation. The chi-square metric measures the differences in distributions of categorical variables. Author describes Apache Spark's MLlib library's Pearson's chi-squared tests for evaluation of goodness of fit and independence. They are used to determine whether distributions of values of a feature for two different classes are distinct or not by using a t-test. According to the paper, a feature can be included if attributes are found to be distinct within a certain confidence interval.

Subsequently, classification-based intrusion detection schemes are explored, such as logistic regression, support vector machine (SVM), Naïve Bayes, random forest, and gradient boosted decision tree (GBDT). The author states that these algorithms are chosen for their effectiveness in modeling posterior probabilities, handling high-dimensional datasets, and minimizing loss functions iteratively. The paper emphasizes the importance of each algorithm's strengths, such as logistic regression's simplicity and SVM's ability to create maximum-margin hyperplanes. Additionally, it highlights the benefits of ensemble methods like random forest and GBDT in reducing over-fitting and improving prediction power. Throughout, the methodology is illustrated with a visual framework, depicting the interaction between feature selection, machine learning algorithms, and intrusion detection models. Overall, the approach outlined in the paper offers a systematic and comprehensive strategy for building a robust intrusion detection system capable of identifying cyber threats.

3. RESEARCH DESIGN

In this chapter, the research design structured to carry out the research is highlighted along with details. The dataset for this research is acquired from Arctic Security's [28] early warning service [29]. The design for this research consists of multiple stages which are summarized in Table 2.

Table 2. Summary of methodology in this research

Section #	Title	Description
3.1	Tools and Technologies	Information about tools and technologies to carry out the system design.
3.2	Datasets	Information about datasets.
3.3	Data pre-processing	Combining datasets, examining the data for inconsistencies, and addressing missing values resulting in a clean dataset for further analysis.
3.4	Feature Engineering	Scaling numerical features to ensure uniformity in their magnitude and encoding text-based variables into numerical representations. Creating new features based on domain knowledge if required.
3.5	Data Visualization	Visualizing data distributions, relationships, and patterns to gain insights for feature selection and model building processes.
3.6	Machine Learning	Selecting relevant features for model training. Building predictive models using selected features and evaluating performance.

Supervised learning is a fundamental approach in machine learning where algorithms are trained on labeled data to predict or classify new data points [30]. In supervised learning, the algorithm learns from a dataset containing input-output pairs, also known as labeled examples [30]. The goal is to learn a mapping from input features to output labels, allowing the algorithm to make predictions or decisions on unseen data [30].

Before going into the specifics of research design, it is of paramount importance to understand the structure of this research. This is a ***supervised machine learning problem*** where a set of independent features will be used to predict the dependent feature. The dependent feature is ***'category'***. A set of independent features will be engineered using the acquired datasets to predict the 'category' i.e. dependent feature. Figure 1 shows popular types of machine learning and highlights this research's problem statement as ***supervised learning classification***.

3.1. Tools and Technologies

To carry out the research design, a set of tools and technologies is utilized. Moreover, a set of key machine learning terms are utilized throughout the design phase. The information regarding tools and technologies is summarized in Table 3.

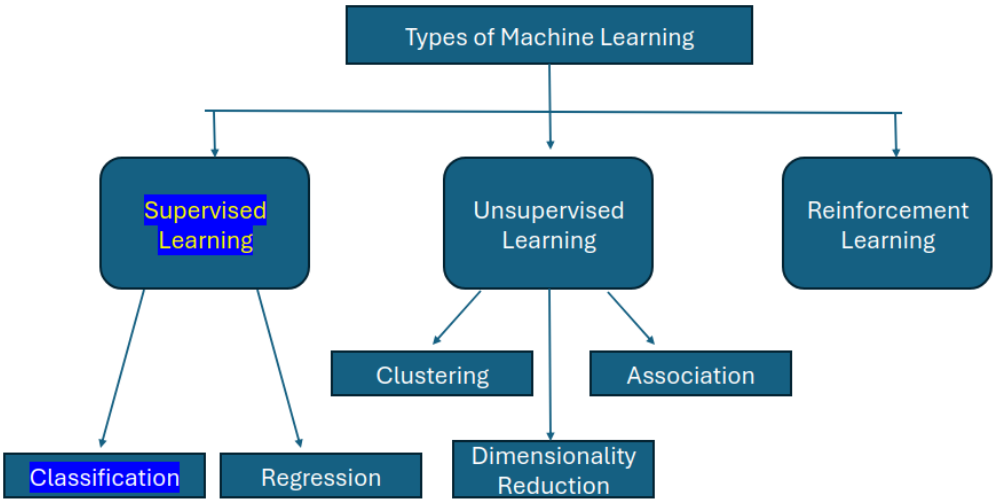


Figure 1. Popular types of machine learning.

Table 3. Tools and technologies used in this research

Tool/Technology	**Description**
Google Colab[1]	A cloud-based platform for running and managing Jupyter notebooks, providing access to computing resources such as GPUs and TPUs.
Google Drive[2]	A file storage and synchronization service that allows users to store, share, and collaborate on files and folders, including Jupyter notebooks and datasets used in the project.
scikit-learn[3]	A popular open-source machine learning library for Python, providing simple and efficient tools for data analysis and machine learning tasks.
Pandas[4]	A Python library for data manipulation and analysis, providing data structures and functions for working with structured data, such as tabular data and time series.
NumPy[5]	A fundamental package for scientific computing in Python, providing support for multidimensional arrays, mathematical functions, and linear algebra operations.

Continued on next page

[1] https://colab.google/
[2] https://www.google.com/drive/
[3] https://scikit-learn.org/stable/
[4] https://pandas.pydata.org/
[5] https://numpy.org/

Table 3 – continued from previous page

Tool/Technology	**Description**
Matplotlib[6]	A Python library for creating static, interactive, and animated visualizations, including plots, charts, and histograms, to analyze and communicate data insights.
Seaborn[7]	A Python data visualization library based on Matplotlib, providing a high-level interface for creating attractive and informative statistical graphics.
NLTK[8]	A Python library for natural language processing (NLP), providing tools and resources for text analysis, tokenization, stemming, tagging, parsing, and semantic reasoning.
CountVectorizer[9]	A scikit-learn feature extraction method for converting text data into numerical feature vectors, representing the frequency of each word in the document corpus.
Bag of Words[10]	A text representation model used in NLP, where each document is represented as a bag of words, ignoring grammar and word order but preserving word frequency information.
One-Hot Encoding[11]	A categorical variable encoding technique used in machine learning to convert categorical data into numerical format by representing each category as a binary vector.
Encoder[12]	A machine learning model component or algorithm used to transform categorical variables into numerical representations, such as ordinal encoder.
Random Forest Classifier[13]	An ensemble learning method for classification and regression tasks, where multiple decision trees are trained on random subsets of the training data and their outputs are aggregated to make predictions.

[6] https://matplotlib.org/

[7] https://seaborn.pydata.org/

[8] https://www.nltk.org/

[9] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[10] https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/

[11] https://www.geeksforgeeks.org/ml-one-hot-encoding/

[12] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html

[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

For reader's reference, key machine learning terminologies that are utilized and relevant to this research statement are presented below:

1. **Independent Features (Predictors)**: Variables used as input for the model to predict the dependent variable. They are also known as predictors or input features.

2. **Dependent Feature (Target)**: The variable that the model aims to predict. It is also known as the response variable or output.

3. **Feature**: An individual measurable property or characteristic of a phenomenon being observed. In datasets, features are represented as columns.

4. **Label**: The true outcome for each observation in the dataset. In classification, labels represent the category to which each observation belongs.

5. **Training Set**: A subset of the dataset used to train the model. It includes both independent features and the dependent feature.

6. **Test Set**: A subset of the dataset used to evaluate the performance of the trained model. It includes independent features and the true values of the dependent feature.

7. **Overfitting**: A modeling error that occurs when the model learns not only the underlying pattern but also the noise in the training data, resulting in poor performance on new, unseen data.

8. **Underfitting**: A modeling error that occurs when the model is too simple to capture the underlying pattern of the data, resulting in poor performance on both the training data and new, unseen data.

9. **Cross-Validation**: A technique used to evaluate the model's performance by splitting the data into multiple subsets, training the model on some subsets, and validating it on others to ensure it generalizes well.

10. **Confusion Matrix**: A table used to evaluate the performance of a classification model by showing the true positives, true negatives, false positives, and false negatives.

3.2. Datasets

The dataset used in this research comprises event data for four organizations, each identified by a unique configuration file containing their respective network ranges and organization names. The dataset spans the year 2023 and is stored in JSON format. The events captured vary in volume across organizations, ranging from a few hundred to 74,000 observations. These events are filtered based on the network ranges associated with each organization, ensuring that only relevant data is included for analysis. The network ranges are derived from publicly available records for the organizations. It's important to note that this dataset is a subset of the larger dataset utilized in the Arctic

EWS, focusing specifically on events relevant to the selected organizations during the specified time period, i.e., the year 2023.

For the sake of simplicity, these four datasets are referred to as dataset 1, dataset 2, dataset 3 and dataset 4. Figure 2 highlights the row versus column count in each of these datasets.

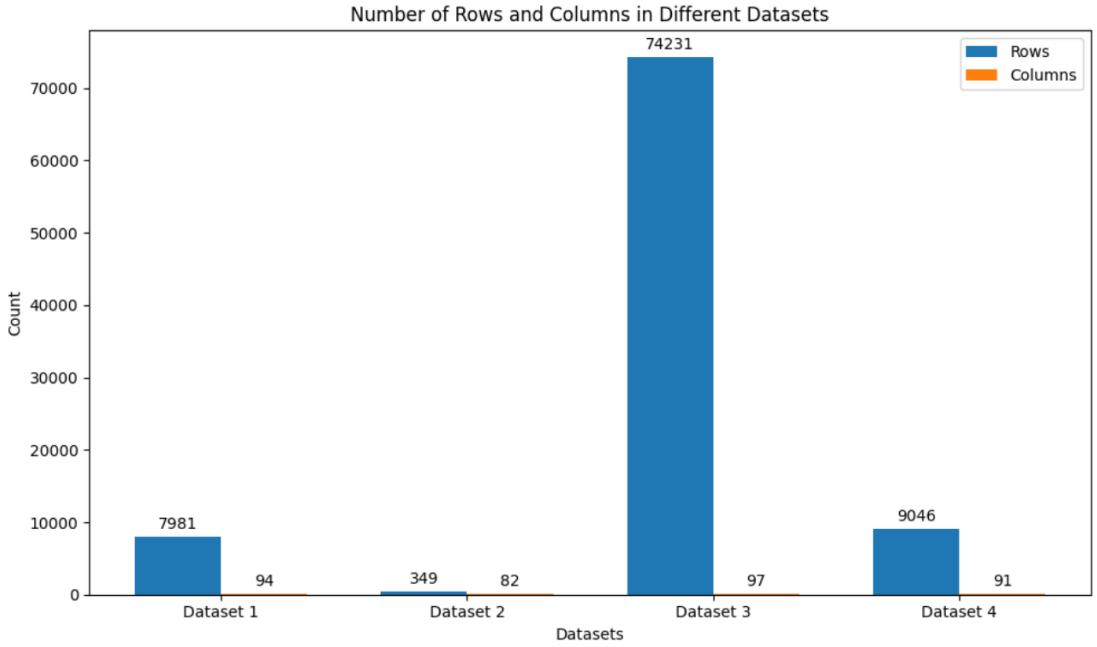


Figure 2. Row versus column count in each dataset.

For the initial step, it is beneficial to categorize all features in the dataset based on specific attributes. In this research, the 88 initial features or columns which were common across the four datasets were grouped into 5 broad categories, as shown in Table 4.

Table 4. Distinct broad categories of features

Category	**Description**
Temporal features	Features related to time or temporal aspects
Geospatial features	Features related to location or geographical aspects
Confidential features	Features that require confidentiality or privacy
Irrelevant features	Features that are not relevant to the analysis of this research
Relevant features	Features that are relevant to the analysis of this research

3.3. Data Pre-Processing

Data pre-processing is a crucial step in data analysis and machine learning tasks, involving the transformation and preparation of raw data to make it suitable for further analysis and machine learning tasks. It aims to improve data quality, remove inconsistencies, caters null values and prepares it for subsequent modeling processes. Following three steps are carried out to accomplish this:

1. Exploratory analysis of each dataset
2. Combining/concatenating datasets to have one uniform dataset
3. Handling null values in this uniform and consistent dataset

3.3.1. Handling Null Values

According to Figure 2, dataset 2 contained only 349 records which were insufficient for this study and this dataset was dropped. Further analysis was carried out on dataset 1, dataset 3 and dataset 4 to find consistent columns. 88 columns were found to be common. Therefore, these three datasets were concatenated on those 88 columns resulting in one uniform dataset of 91,258 rows by 88 columns. Some of the records had null value for 'category' column. All such records were dropped leading to a final uniform and consistent dataset of 90,686 rows by 88 columns.

Using python function presented in Listing 3.1, percentages of null count in each of the 88 columns are studied. Accordingly, a strategy is devised to cater the null counts.

Listing 3.1. Python function to count null values percentage and list by column name

```
def count_null_values(df):
    null_counts = df.isnull().sum()
    null_counts = null_counts[null_counts > 0]
    total_rows = len(df)
    null_percentages = null_counts / total_rows * 100
    null_percentages = null_percentages.round(2)
    null_counts_df = pd.DataFrame({
        'Column Name': null_counts.index,
        'Null Count': null_counts.values,
        'Null Percentage': null_percentages.values
    })
    return null_counts_df
```

Out of the 88 columns in the dataset, 71 columns contain null values. Table 5 illustrates the null count in each of these 71 columns. For example, at index 0, the column 'query' has 8740 null values out of a total of 90,686 values.

To ensure a sufficient dataset size, it is decided that at least 80,000 rows should be present. While this threshold can be adjusted based on requirements, for this study, a threshold of 80,000 is chosen. Therefore, any column with more than 10,686 null counts or **11.78%** in Table 5 is dropped. This results in a list of 45 columns being dropped from the dataset.

Table 5. Summary of null count in the dataset

Index	Column Name	Null Count	Null Percentage
0	query	8740	9.64
1	reported asn	135	0.15
2	reported city	8762	9.66
3	reported cc	134	0.15
4	reported country	8762	9.66
5	domain	22206	24.49

Table 5 (Continued)

Index	Column Name	Null Count	Null Percentage
6	reverse dns	43194	47.63
7	isp	8762	9.66
8	reported latitude	8762	9.66
9	reported longitude	8762	9.66
10	reported organization	8762	9.66
11	port	7	0.01
12	product	38426	42.37
13	source time	17	0.02
14	transport protocol	19	0.02
15	product version	55201	60.87
16	x509 expired	43958	48.47
17	x509 not after	43958	48.47
18	x509 not before	43958	48.47
19	x509 subject cn	43960	48.47
20	ssl cipher version	43958	48.47
21	ssl cipher name	43958	48.47
22	ssl cipher bits	43958	48.47
23	http host	31475	34.71
24	http title	49435	54.51
25	http favicon location	70746	78.01
26	http location	31475	34.71
27	shodan module	8740	9.64
28	region code	8762	9.66
29	additional information	38651	42.62
30	vulnerability	79381	87.53
31	protocol	10019	11.05
32	service	10582	11.67
33	description url	104	0.11
34	feed url	8	0.01
35	geoip cc	155	0.17
36	bgp prefix	15	0.02
37	asn	15	0.02
38	cymru cc	15	0.02
39	registry	15	0.02
40	bgp prefix allocated	15	0.02
41	as allocated	15	0.02
42	as name	15	0.02
43	ctl domain name count	45191	49.83
44	cpe23	49454	54.53
45	cpe	49380	54.45
46	shodan tags	80894	89.2
47	os name	80277	88.52
48	php version	82884	91.4
49	vulnerability verified	82475	90.95

Table 5 (Continued)

Index	Column Name	Null Count	Null Percentage
50	wordpress version	88967	98.1
51	weakness	51878	57.21
52	reputation score	82058	90.49
53	reputation key	82058	90.49
54	days seen	82058	90.49
55	active detections	82058	90.49
56	passive detections	82058	90.49
57	detection method	82058	90.49
58	destination port	81991	90.41
59	malware family	90674	99.99
60	shared resource	73204	80.72
61	ip version	46604	51.39
62	nginx version	88504	97.59
63	related domain name	69698	76.86
64	shodan tag	82448	90.92
65	x509 subject organization	80721	89.01
66	source	74567	82.23
67	reported product	79647	87.83
68	reported product version	82717	91.21
69	component	90059	99.31
70	reported category	90303	99.58

After applying the null value cut-off threshold, the dataset now consists of 90,686 rows and 43 columns, still containing some null values. Only columns with the most null values were dropped. To handle the remaining null values, the dropna() method from the pandas module was used, resulting in a dataset of 79,017 rows and 43 columns. This dataset is now free of any null values and can be further processed in the next Section to engineer independent features for the machine learning model.

3.4. Feature Engineering

Feature engineering is another crucial step in the machine learning pipeline that involves the process of using domain knowledge to extract, transform, and select the most relevant features from the dataset. CTI data usually contains information in the form of textual representation such as urls, IP addresses, protocols used etc. However, machine learning models operate on numerical inputs only. Therefore, feature engineering is a necessary step to transform CTI data's textual information into numerical features while also maintaining their efficacy. The pie-chart in Figure 3 depicts the numerical vs non-numerical features percentage on studied datasets.

Before diving into the intricacies of feature engineering for this research, the following key points are presented:

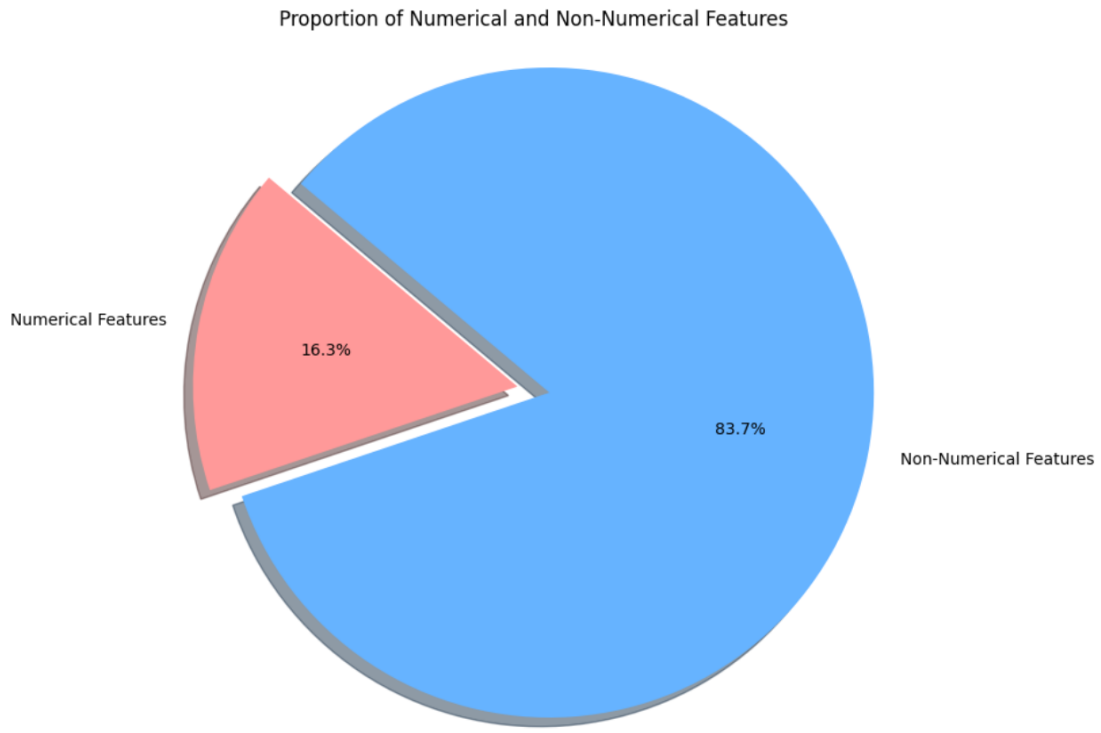


Figure 3. Proportion of numerical and non-numerical features in processed dataset.

1. The objective of this research is to find out the category of cyber attack. It is not the prediction of cyber attacks. Therefore, temporal features can be dropped.

2. The objectives of this research does not cover identifying sources of attacks or victims. Therefore, geospatial features are dropped.

3. Confidential information features are dropped to maintain anonymity.

4. CTI datasets can often contain repetitive and extra information. Thus, irrelevant features are also dropped.

Application of points above stems into a dataset of 79,017 rows and 17 columns. Python list comprehension is utilized to achieve this. Such exhaustive data pre-processing lead to a loss of certain target class labelled 'suspected compromise'.

3.4.1. Concatenating Data for Missing Target Class

To address the issue of missing target class of 'suspected compromise' on pre-processed data due to handling null values, the focus was on extracting relevant records from the originally concatenated dataset. The work began with pre-processed dataset from Section 3.3.1, which contains 90,686 rows and 88 columns. For the analysis, 17 columns matching the dataset from Section 3.4 were selected.

After selecting these columns, all records where the 'category' column was labeled as 'suspected compromise' were extracted, resulting in a dataframe of 8740 rows and 17 columns. The pandas groupby() method was then applied. An analysis of this

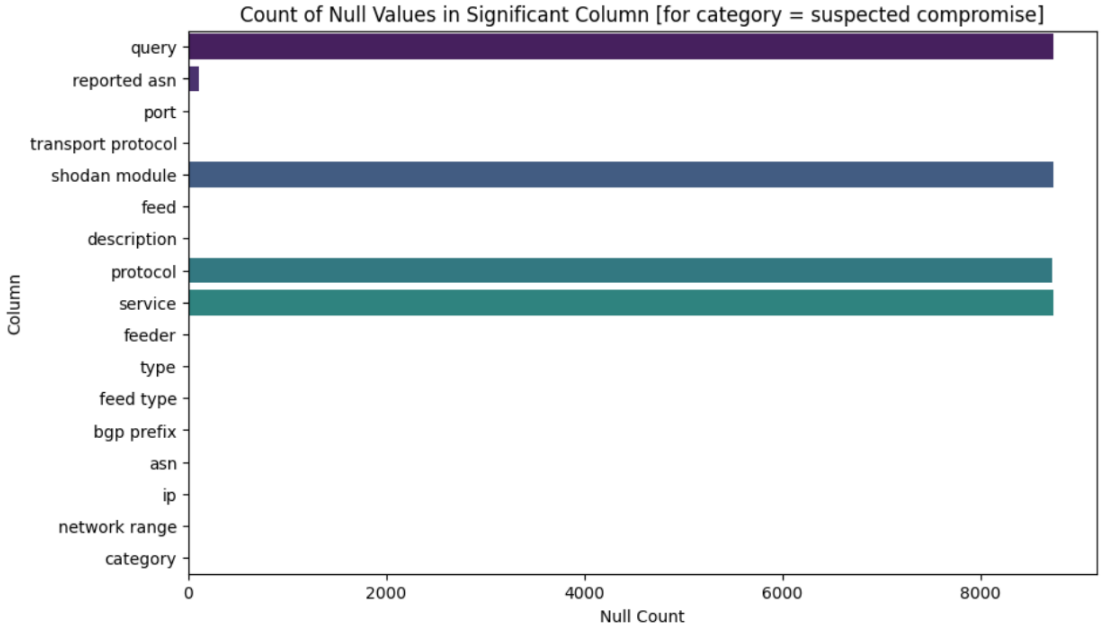


Figure 4. Null count for relevant features or columns where category is 'suspected compromise'.

specific subset of data revealed that the following columns contained almost no data for the 'suspected compromise' category: 'query', 'shodan module', 'protocol', and 'service'. This information is summarized in the horizontal bar chart shown in Figure 4.

To ensure a uniform dataset capable of predicting this category as well, these columns had to be dropped. Otherwise, the prediction of this category would have to be eliminated due to the nature of the dataset. The result was a dataframe of 8604 rows and 13 columns. This dataframe was joined with the original pre-processed dataset from Section 3.4 using the pandas concat() function, leading to a complete, uniform dataset of 87,621 rows and 13 columns ready for the feature engineering study.

3.4.2. Feature Engineering: Textual to Numeric

Feature engineering is a critical process in preparing textual data for machine learning models, involving the transformation of text into numerical representations that algorithms can utilize. Two prominent methods for this transformation are one-hot encoding and the bag-of-words approach. One-hot encoding converts categorical text data into binary vectors, where each unique text category is represented as a distinct vector with a single high (1) value and the rest low (0), enabling the model to process nominal variables effectively [31]. The bag-of-words model, on the other hand, converts text into fixed-length vectors based on word frequency, capturing the presence or absence of words in a document without considering word order [32].

The machine learning-ready dataset is taken and all textual columns are transformed into numeric during this stage of research design. Textual data can be classified into two broad categories based on the relationship between categories:

1. Nominal Variable (Categorical): This type of variable comprises a finite set of discrete values with no inherent relationship between the values.

2. Ordinal Variable: This type of variable comprises a finite set of discrete values with a ranked ordering between the values.

In this dataset, there are no ordinal variables. For categorical variables where no ordinal relationship exists, integer encoding is insufficient. Therefore, one-hot encoding is used for these categorical variables to transform them into numeric format.

3.4.3. Handling IP Addresses

IP addresses are a complex set of information that cannot be processed by machine learning algorithms directly because of their unique multi-dotted notation. CTI data contains IP addresses in majority of the cases and it's vital to devise a strategy to cater this piece of information while maintaining the original relationship between these IP addresses. In [33] author describes three different pathways to handle IP addresses. One-hot encoding is a common method [34] utilized in machine learning. However, in-case of IP addresses it does not preserve the relation between IP addresses [33]. Another approach described in [33] entails converting the IP address to a binary number, and then convert the binary number to an integer.

For the scope of this research, a different path in which we strip the dots from the dotted-decimal and simply add zeros to the left to complete each octet to exactly three digits [33] is followed. This method maintains the relationship between IP addresses and helps the recognition of adjacent nodes and networks.

3.4.4. Handling Textual Columns with High Categorical Count

Table 6 highlights columns that require more processing based on the different categories that appeared in them after grouping by.

Table 6. Summary of unique category counts

Column	**Unique Category Count**
Feed	86
Description	91
Type	10

To process these, a combination of bag of words and one-hot encoding is used. The "bag" of words model disregards the order and structure of words in a document, focusing solely on whether certain words are present regardless of their position [32] and was applied to 'description' feature. 'Feed' and 'type' on the other hand were handled with one-hot encoding.

3.4.5. Engineering Features Using Bag of Words

Bag of words is applied to description column. However, for it to be effective natural language processing needs to be applied to remove unnecessary words.

1. **Applying NLP on description column:** Each description record contains up to 33 words. Tokenization, stop word removal, and lemmatization are applied to the description column. Notably, there are 91 unique descriptions for 87,621 records. This process eliminates unnecessary words from each description, streamlining the textual data for subsequent analysis.

2. **Applying bag of words on description column:** The function in Listing 3.2 was defined to apply bag of words. The final result is a set of 239 new features emerging from the description column.

Listing 3.2. Python function to apply bag of words on description column

```
def apply_bag_of_words(df, column_name):
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(df[column_name])

    feature_names = vectorizer.get_feature_names_out()
    bag_of_words_df=pd.DataFrame(X.toarray(), columns=feature_names)

    df.drop(columns=[column_name], inplace=True)
    df = pd.concat([df, bag_of_words_df], axis=1)

    return df
```

3.4.6. Engineering Features Using One-Hot Encoding

One-hot encoding is one way of converting text to numeric representation. It is performed for following columns:

1. Feed
2. Type
3. Transport protocol
4. Feeder

Additionally, the column 'feed type' is dropped due to its univariate nature, which does not contribute significantly to the analysis. The process of one-hot encoding involves converting categorical variables into binary vectors, thereby expanding the feature space. This transformation is executed using the pd.get_dummies() function from the pandas library, which efficiently performs the one-hot encoding operation. Alternatively, the scikit-learn library provides another option for accomplishing this task. The dataset used for this transformation already incorporates text processed features of description column.

3.4.7. Summary of Features Engineered

The dataset underwent several feature engineering processes to prepare it for machine learning tasks. The key transformations are summarized as follows:

- **Description Column:** Utilizing the bag-of-words technique, the 'Description' column was processed, resulting in an expansion from 1 to 239 features.
- **Feed Column:** One-hot encoding was applied to the 'Feed' column, expanding the feature set from 1 to 86.
- **Type Column:** Similarly, the 'Type' column underwent one-hot encoding, increasing the feature count from 1 to 10.
- **Transport Protocol Column:** Employing one-hot encoding, the 'Transport Protocol' column was transformed, resulting in an augmentation from 1 to 2 features.
- **Feeder Column:** One-hot encoding was also utilized for the 'Feeder' column, expanding the feature set from 1 to 3.
- **IP, BGP prefix, and Network Range Columns:** Custom encoding was utilized to encode columns containing IP addresses. The three mentioned columns were replaced with 3 encoded columns.

Dataset now has 347 unique features.

3.5. Data Visualization

Data visualization and exploratory data analysis (EDA) are crucial processes in the early stages of data analysis. Data visualization involves the graphical representation of data through charts, graphs, and other visual tools, facilitating the intuitive understanding of complex datasets. EDA employs a variety of techniques to summarize the main characteristics of data, often using visual methods. These combined approaches help to identify patterns, anomalies, and relationships within the data.

Because this research problem required extensive processing on the dataset at multiple stages of the research design phase, a brief overview is provided in this Section about what was achieved from the initial dataset. The Figure 5 presents the timeline of data-processing.

In machine learning classification problems, ensuring a balanced dataset is essential for building effective models. However, this dataset presents a significant imbalance in its category distribution as depicted in Figure 6. Specifically, the prediction task involves three categories, but one category contains a disproportionately high number of records compared to the other two. This imbalance leads to a biased dataset, where the model is likely to favor the majority class during training. Such bias can result in poor predictive performance and generalization, particularly for the underrepresented categories. However, there are techniques present which will be presented in the next

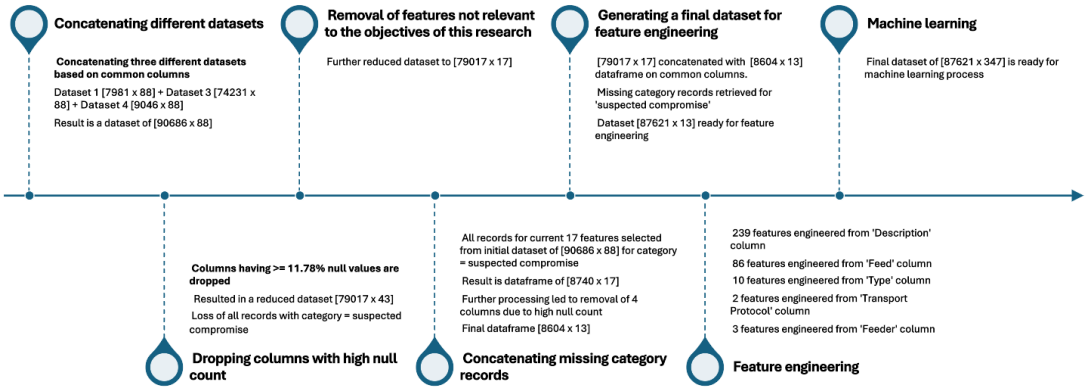


Figure 5. Timeline of data processing.

chapter to handle such scenarios. However, techniques to address this issue will be presented in the next chapter.

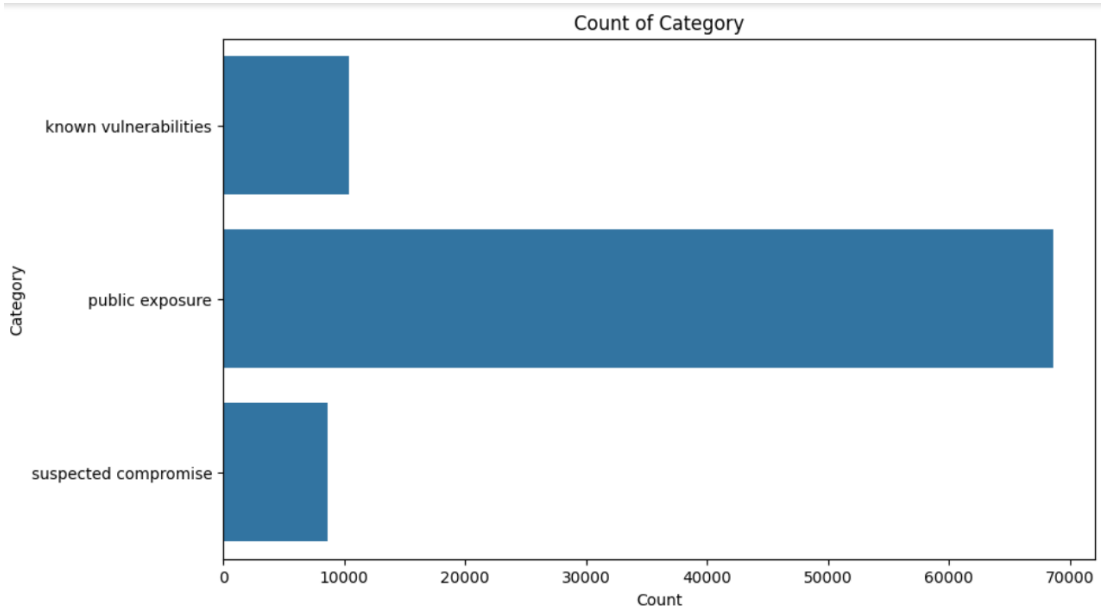


Figure 6. Imbalance in the dataset for dependent feature, i.e., category.

3.6. Machine Learning

Random forests, also known as random decision forests, are ensemble learning methods utilized for tasks such as classification, regression, and others. These methods involve constructing numerous decision trees during training. In classification tasks, the random forest's output is determined by the class chosen by the majority of trees. [35]

The machine learning method for the category prediction chosen for the scope of this work is random forest classifier. According to the article [36] random forest are powerful machine learning algorithms known for their accuracy and versatility. They

work by combining multiple decision trees, creating a more robust model than any single tree. The pseudo code below adopted from [37] describes the working of a typical random forest classifier:

Listing 3.3. Random forest algorithm pseudo code adopted from [37]

```
Input: N - Quantitative amount of bootstrap samples
       M - Total number of features
       m - Sample size
       k - Next node
Output: A Random Forest (RF)
Steps:
1. Creates N bootstrap samples from the dataset.
2. Every node (sample) taking a feature randomly of size
   m where m<M.
3. Builds a split for the m features selected in Step 2
   and detects the k node by using the best split point.
4. Split the tree iteratively until one leaf node is
   attained and the tree remains completed.
5. The algorithm is trained on each bootstrapped sample
   independently.
6. Using trees classification voting predicted data is
   collected from the trained trees (n).
7. The final RF model is built using the peak voted
   features.
8. return RF
```

.

A common problem with random forest classifier is over-fitting and under-fitting. In-order to avoid this understanding the impact of depth and number of trees in random forests is important.

4. IMPLEMENTATION

In this Chapter, a random forest classifier is applied to the processed CTI dataset. The methodology encompasses data pre-processing, including cleaning, feature engineering and selection which was done in previous Section. In particular, this Chapter talks about the construction and training of the random forest model. The implementation involves configuring the classifier with appropriate parameters, training it on the prepared dataset, and rigorously evaluating its performance using standard metrics.

According to [35] a random forest is a meta-estimator that constructs multiple decision tree classifiers on various sub-samples of the dataset, utilizing averaging to enhance predictive accuracy and mitigate over-fitting. It's an ensemble method of learning where each tree in the forest employs the best split strategy. [35]

Numerous metrics are often used to evaluate the performance of a classifier. One of the most straightforward metrics is classification accuracy, which calculates the proportion of correctly classified instances out of the total instances. However, accuracy alone may not provide a complete picture, especially when dealing with imbalanced datasets. Logarithmic loss, also known as log loss, measures the performance of a classifier where the predicted output is a probability value between 0 and 1. A lower log loss indicates better performance. The area under the curve metric is commonly used for binary classification models, with a higher AUC indicating better performance. The F1 score is a metric that considers both precision and recall, providing a balance between the two. Precision is the ratio of correctly predicted positive observations to the total predicted positives, while recall is the ratio of correctly predicted positive observations to all actual positives. The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both. Lastly, the confusion matrix is a table that summarizes the performance of a classification algorithm. It shows the number of true positives, false positives, true negatives, and false negatives, providing insight into the model's performance across different classes. [38]

A typical implementation life cycle for the scope of this research is present as a flowchart Figure 7.

4.1. Random Forest Classifier

In a random forest classifier, multiple decision trees are constructed using different random subsets of the data and features. Each decision tree acts as an expert, providing its classification opinion based on its unique exposure to a portion of the dataset. By creating n decision trees, each with its own perspective, the RF classifier leverages the diversity of opinions to enhance predictive accuracy and robustness. For classification tasks, the predictions from all decision trees are collected, and the most popular prediction is selected as the final output. [39]

Two key parameters in a Random Forest classifier are `n_estimators` and `max_depth`. The `n_estimators` parameter represents the number of decision trees in the forest. Increasing this hyperparameter generally improves the performance of the model but also increases the computational cost of training and predicting. The `max_depth` parameter specifies the maximum depth of each decision tree in the

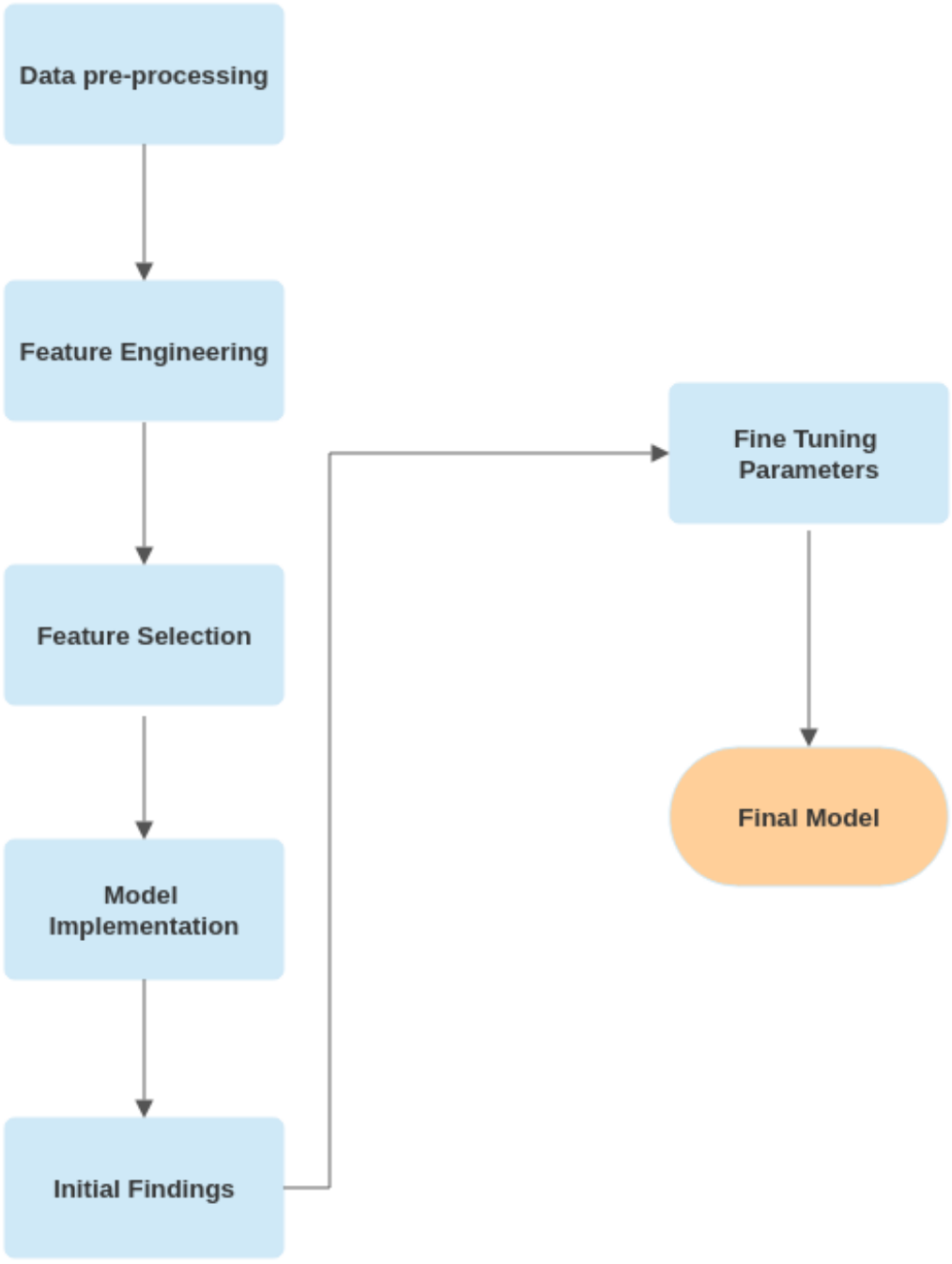


Figure 7. Typical implementation life cycle for the scope of this research.

forest. Setting a higher value for `max_depth` can lead to overfitting, while setting it too low can lead to underfitting. [39]

4.2. Performance Metrics

The following metrics will be used to evaluate the performance of the model:

- **Accuracy**: The proportion of correctly classified instances out of the total instances.
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
- **Precision**: The ratio of correctly predicted positive observations to the total predicted positives.
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Recall**: The ratio of correctly predicted positive observations to all actual positives.
$$\text{Recall} = \frac{TP}{TP + FN}$$
- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two.
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
- **Confusion Matrix**: A table that summarizes the performance of a classification algorithm, showing the number of true positives, false positives, true negatives, and false negatives.

Confusion Matrix for 3-Class Classification Problem

$$\begin{bmatrix} \text{TP}_{00} & \text{FP}_{01} & \text{FP}_{02} \\ \text{FN}_{10} & \text{TP}_{11} & \text{FP}_{12} \\ \text{FN}_{20} & \text{FN}_{21} & \text{TP}_{22} \end{bmatrix}$$

Where

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives
- TP_{ii} are the true positives for class i.

- FP_{ij} are the false positives for class i when the actual class is j.
- FN_{ji} are the false negatives for class j when the predicted class is i.

Furthermore, k-fold cross validation will be employed as a crucial step to evaluate the performance of the model. K-fold cross validation involves splitting the dataset into k subsets, known as folds. The model is trained on k-1 subsets while the remaining subset is used for evaluation. This process is repeated k times, with each subset serving as the evaluation set once. This method ensures that every data point is used for both training and testing, providing a comprehensive assessment of the model's performance. By averaging the results from each iteration, k-fold cross validation helps to mitigate the variance associated with a single train-test split, leading to a more robust and reliable evaluation of the model. [40]

4.3. Other Evaluation Techniques to Consider

Before presenting the construction and training of the actual model, there are other important evaluation techniques to consider. Their definitions are presented below:

- **Stratified Sampling**: Stratified sampling is a method of sampling that involves dividing a population into smaller groups, known as strata, that share similar characteristics. The sample is then drawn from each stratum proportionally to ensure that each subgroup is adequately represented in the overall sample. This technique is particularly useful when dealing with imbalanced datasets to ensure that minority classes are represented in the sample. [41]
- **Bootstrapping**: Bootstrapping is a statistical method that involves repeatedly sampling with replacement from a dataset to create multiple simulated samples. This technique allows for the estimation of the sampling distribution of a statistic and is used to assess the accuracy and stability of statistical estimates. [42]
- **Out-of-Bag (OOB) Dataset**: In the context of ensemble methods like Random Forests, the out-of-bag (OOB) dataset consists of the instances from the original dataset that are not included in a particular bootstrap sample. The OOB dataset is used to estimate the performance of the model on data not seen during training, providing an unbiased validation method. [35]

4.4. Constructing the Model

The dataset used in this study consists of 87,621 rows and 347 columns, encompassing 347 features. The following key steps were undertaken to prepare the data for model training and evaluation:

1. **Stratification by Category:**
    - The dataset was divided into three distinct categories.
2. **Randomization within Categories:**

- Each category was randomized to eliminate any inherent ordering.

3. **Extraction of Independent Test Set:**
    - 25% of the data from each randomized category was extracted and set aside.
4. **Combination and Re-randomization:**
    - The remaining 75% of data from each category was combined.
    - The combined dataset was randomized to ensure thorough mixing.
5. **Final Dataset Segregation:**
    - 25% of the re-randomized combined dataset was separated to form an independent test set.
    - The remaining 75% of the data was used for model training and validation.
6. **Model Training:**
    - The model was trained on the remaining 75% of the data with an 85:15 train-test split. An 85:15 train-test split means that 85% of the data was used for training and 15% for testing.

This approach ensured a balanced and representative sample for training while maintaining an unbiased test set for the final evaluation of the model's performance.
Ordinal Encoder is a data preprocessing technique used in machine learning to encode categorical features into numerical representations. It assigns a unique integer to each category, preserving the ordinal relationship between the categories. This encoding is suitable for categorical features where the order of the labels is meaningful but it can also be used with unordered categories. [43]
The dependent feature, which corresponds to the category of the data was encoded using 'OrdinalEncoder' as follows:

- Number 0 corresponds to category: known vulnerabilities
- Number 1 corresponds to category: public exposure
- Number 2 corresponds to category: suspected compromise

This encoding ensures that the categories are represented numerically for use in the machine learning model.

4.5. Evaluating Performance

The processed CTI dataset used in this implementation comprises 87,621 rows and 347 columns, encompassing 347 features. The data preparation process involved several steps to ensure a balanced and comprehensive evaluation of the model. Initially, the dataset was stratified by category, segregating it into three distinct categories. Each

category was then subjected to randomization to eliminate any inherent ordering. From each randomized category, 25% of the data was extracted and set aside. The remaining 75% of the data from each category was combined, resulting in a new dataset that was once again randomized to ensure thorough mixing of instances from all categories. Random forest classifier model was trained on this dataset with 85:15 train-test split. Hyperparameter tuning was applied to find the best model parameters. Table 7 shows the fine-tuned parameters for the best performing model.
However, 25% of this combined dataset was separated to form an independent test set. This independent test set, comprising a stratified and balanced sample, was reserved for the final evaluation of the best performing model. The remaining 75% of the data was used for training and validating the model. This approach ensured that the model was trained on a representative and sufficiently large portion of the dataset while maintaining a separate and unbiased test set for final performance assessment. As highlighted in Section 4.2 and Section 4.3, accuracy, precision, recall, F1 score, confusion matrix and k-fold cross validation were utilized to evaluate model's performance. These results are presented in the next chapter.

Table 7. Best performing random forest classifier parameters

Parameter	**Value**
max_depth	3
n_estimators	5
random_state	42
Train-Test Split	85:15

5. RESULTS

In this Chapter, the results of the Section 4 are shared. A random forest classifier model was utilized to evaluate the performance of the classification task. The dataset was split into training and testing sets with an 85:15 ratio. Additionally, 25% of the original sample data was reserved for a stratified sampling test with maintained class distributions.

5.1. Hyperparameter Tuning of Random Forest

At first, the performance metrics of the random forest models with varying max_depth and n_estimators were studied to demonstrate the impact of hyperparameter tuning on model accuracy, precision, recall, and F1 score. As shown in Table 8, different combinations of these hyper-parameters lead to significant variations in performance. For instance, a shallow tree with max_depth of 1 and n_estimators of 1 or 10 shows moderate accuracy and F1 scores (0.78 and 0.68-0.69 respectively), indicating underfitting. In contrast, increasing the depth to 3 improves the performance, with accuracy reaching up to 0.92 and F1 scores improving correspondingly. Further increasing max_depth to 5 and 7, along with low number of trees (n_estimators ranging from 1 to 5), results in near-perfect performance metrics, indicating a fitting model.
However, excessively increasing the hyperparameters can lead to overfitting. For example, a model with max_depth of 5 and n_estimators of 10 achieves accuracy and F1 score of 0.99. This overfitting scenario highlights the importance of balancing model complexity with generalizability through careful hyperparameter tuning and validation techniques.

Table 8. Performance metrics of random forest models with varying max_depth and n_estimators

max_depth	n_estimators	Accuracy	Precision	Recall	F1 score
1	1	0.78	0.61	0.78	0.68
1	10	0.78	0.73	0.78	0.69
3	1	0.91	0.91	0.91	0.89
3	10	0.92	0.92	0.92	0.91
5	1	0.89	0.90	0.89	0.85
5	10	0.99	0.99	0.99	0.99
7	5	0.96	0.96	0.96	0.96
7	10	0.99	0.99	0.99	0.99

Additionally, the results of hyperparameter tuning plays a pivotal role in determining feature importance. As evidenced by the Table 9, variations in hyperparameters such as max_depth and n_estimators can lead to substantial shifts in the identification of contributing features versus non-contributing ones. This underscores the sensitivity of feature importance to the efficacy of model training and its coverage of the data. These findings emphasize the criticality of fine-tuning model configurations to achieve optimal feature discernment.

Table 9. Feature contribution of random forest models with varying parameters

max_depth	n_estimators	Contributing Features	Non-Contributing Features
1	1	1	345
1	10	9	337
3	1	5	341
3	10	33	313
5	1	11	335
5	10	58	288

5.2. Best Performing Model

Findings from the hyperparameter tuning process culminated in the construction of the best-performing model. Details regarding these optimized parameters are provided in Section 4.5. The subsequent analysis yielded the following results:

- **Accuracy:** 0.9653
  This metric indicates that 96.53% of the total instances were correctly classified by the model.
- **Precision:** 0.9668
  Precision reflects the proportion of true positive predictions among all positive predictions, showing the model's ability to correctly identify positive instances.
- **Recall:** 0.9653
  Recall measures the proportion of true positive instances that were correctly identified by the model, indicating the model's effectiveness in capturing all relevant instances.
- **F1 Score:** 0.9633
  The F1 score is the harmonic mean of precision and recall, providing a balance between the two metrics and offering a single measure of the model's performance.

To further illustrate the performance of the model, the confusion matrix for the 3-class classification problem is presented in the Figure 8.

5.2.1. Utilizing K-Fold Cross Validation to Test Generalization Ability

The performance of the best performing model was also assessed using k-fold cross-validation with n=5 and n=10 to evaluate its generalization ability. Table 10 presents the key metrics, including accuracy, precision, recall, and F1 score, for both scenarios. Overall, the results indicate that the RF classifier performs consistently well across both n values, with slight variations observed in evaluated metrics. These findings suggest that the model is robust and capable of maintaining performance across different fold sizes. 5-fold and 10-fold cross validation confusion matrix are presented in Figure 9 and 10 respectively.

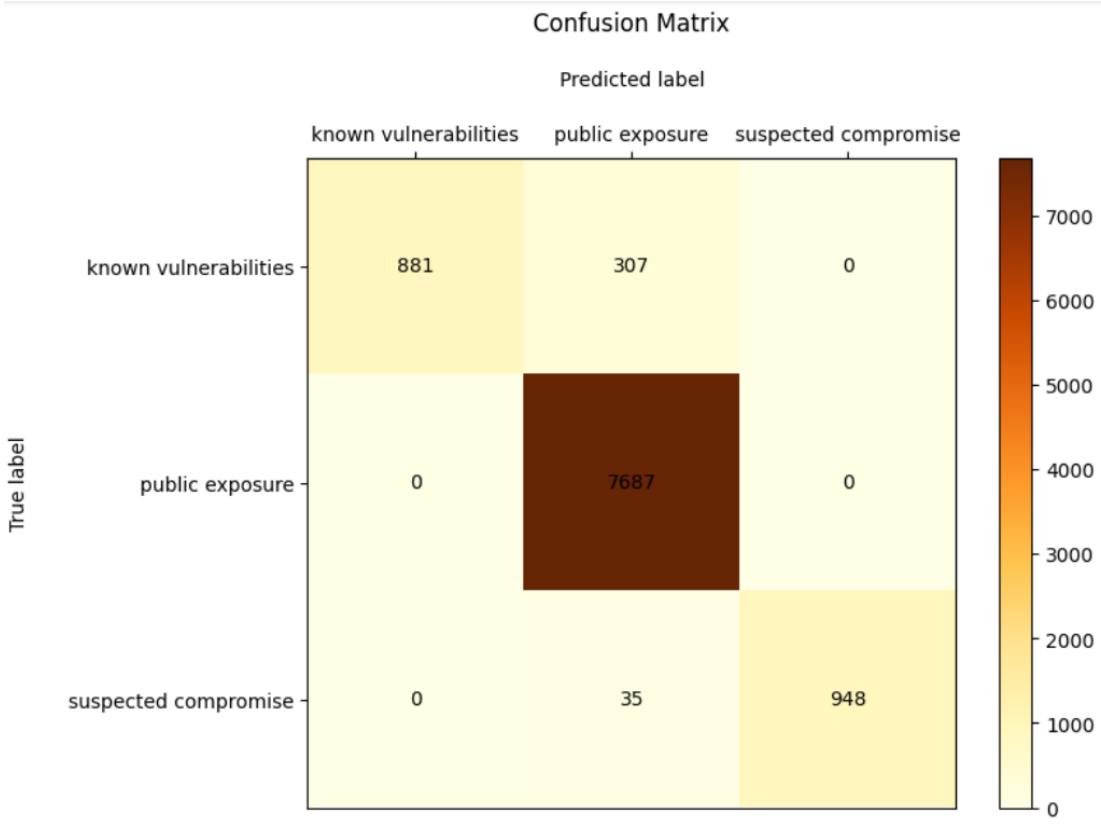


Figure 8. Confusion matrix for train-test split best performing model.

Table 10. Comparison of k-fold cross-validation results for $n = 5$ and $n = 10$

Metric	$n = 5$	$n = 10$
Accuracy	0.9520	0.9445
Precision	0.9557	0.9486
Recall	0.9520	0.9445
F1 Score	0.9447	0.9376

5.2.2. Utilizing Stratified Sample to Evaluate the Best Performing Model

The random forest model was initially trained on 75% of the original dataset, with an 85:15 train-test split to optimize the training process. The trained RF model was saved using joblib [44] for final testing with stratified sample. An unseen stratified sample, comprising 25% of the original dataset as mentioned in Section 4.4, was loaded to evaluate the model's final performance. The saved model was then applied to the stratified test dataset. The performance metrics gathered from this testing phase demonstrating the model's efficacy are presented in terms of accuracy, precision, recall, F1 score in Table 11 and the corresponding confusion matrix in Figure 11.

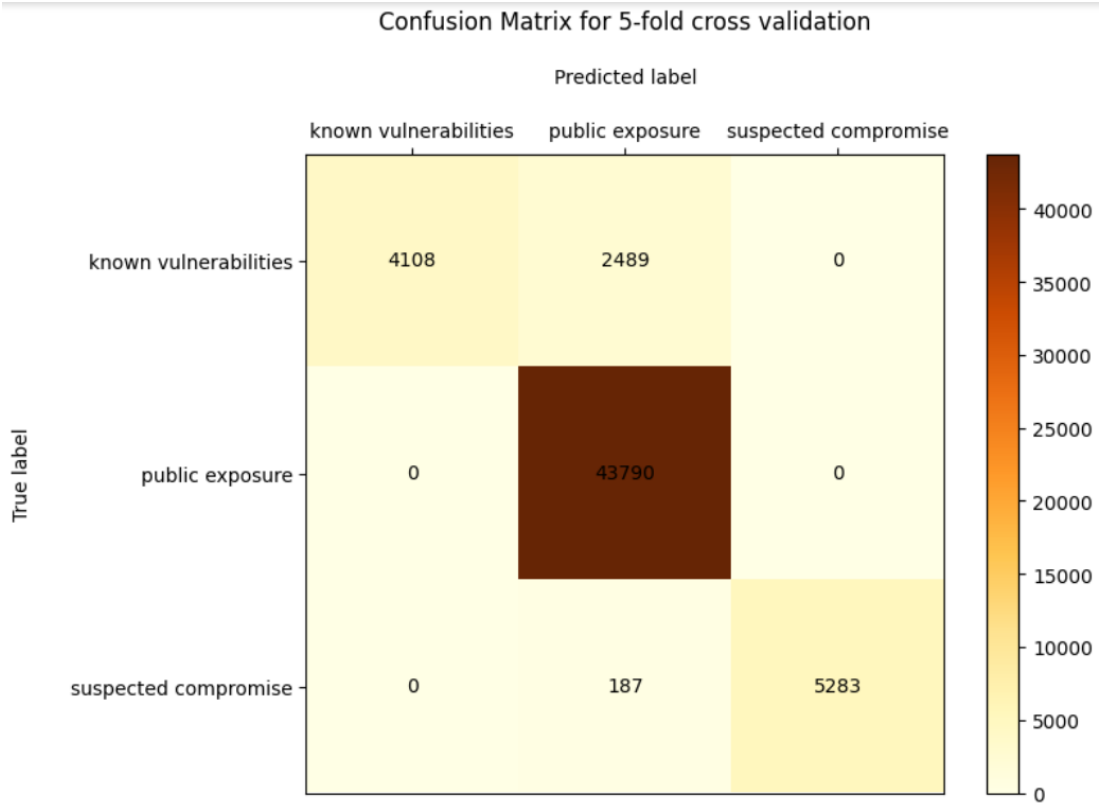


Figure 9. 5-fold cross validation confusion matrix for the best performing model.

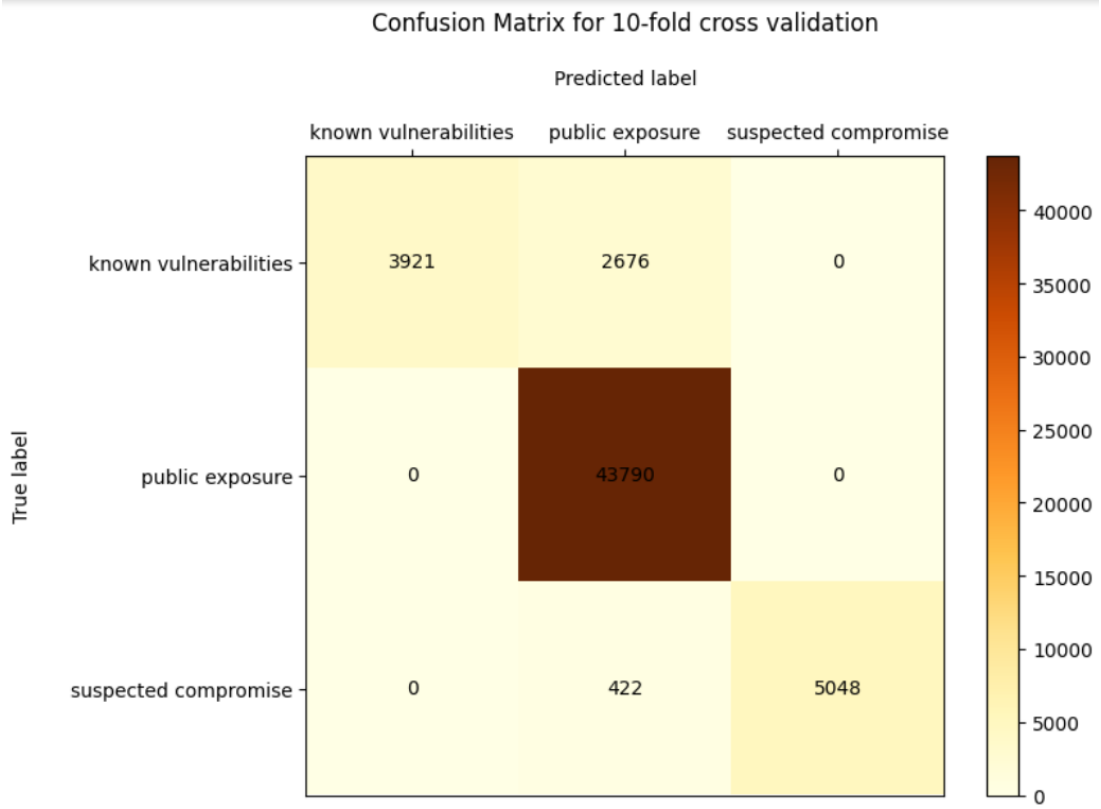


Figure 10. 10-fold cross validation confusion matrix for the best performing model.

Table 11. Performance metrics with stratified sampling

Metric	Value
Accuracy	0.9217
Precision	0.9288
Recall	0.9217
F1 Score	0.9044

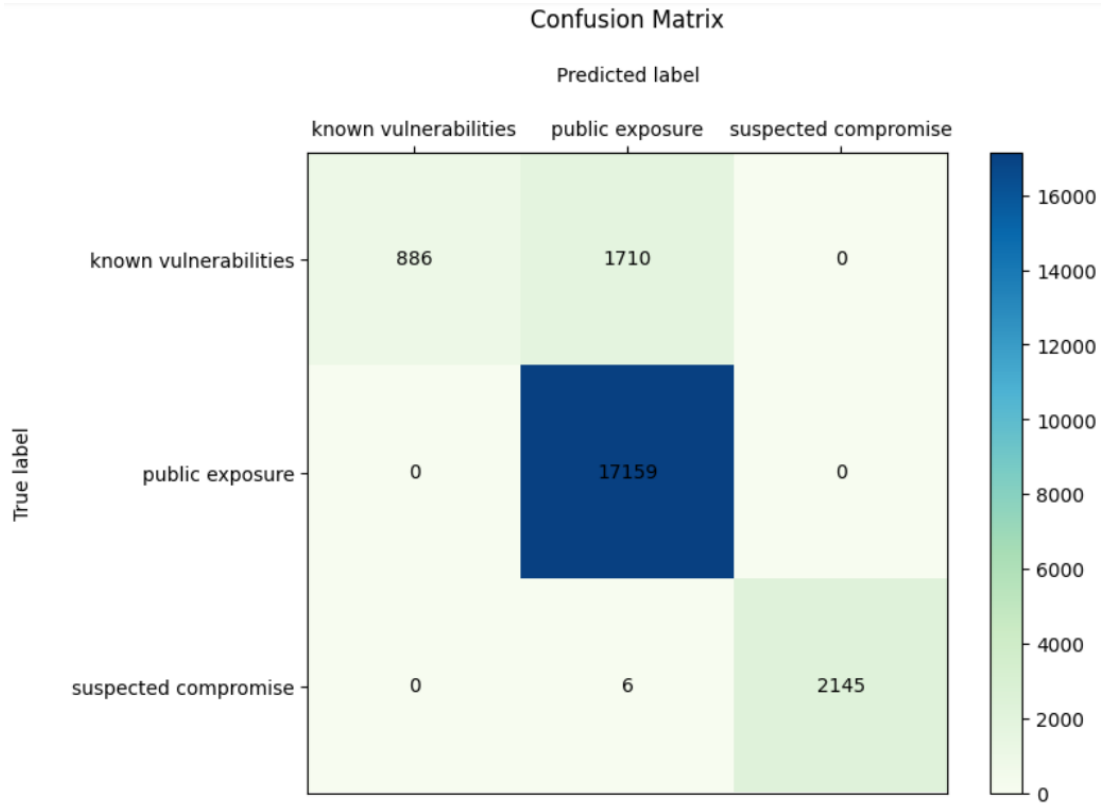


Figure 11. Confusion matrix for the stratified sampling test with the best performing model.

5.3. Results for Important Input Features

In random forest classifiers, feature importance scores are a crucial metric for understanding the contribution of each input variable to the model's predictive performance. These scores are calculated based on the decrease in impurity (e.g., Gini impurity) that each feature causes when used in the construction of the decision trees within the RF ensemble. It is important to note that the sum of all feature importance scores in a Random Forest model is always equal to 1. Features that lead to greater impurity reduction are deemed more important, as they are more effective in splitting the data and making accurate predictions. [35]
To identify the most important input features, the best performing model from Section 5.2 and two different RF classifiers were trained. The *max_depth* parameter was kept constant to a value of 3 while values of *n_estimators*, denoted by n varied. Specifically, the classifiers were trained with $n = 5$, $n = 10$, and $n = 100$. The feature importance scores were then extracted and compared across these models. The variable n in this section refers to the values of *n_estimators* used in the RF classifiers.
In the investigation, it was observed that altering the `n_estimators` parameter of the random forest classifier had a notable impact on the number of contributing features and their values. Specifically, when the number of estimators (n) was set to 5, fewer features were identified as significant contributors to the model's predictions. This is likely due to the smaller ensemble size leading to a more limited exploration of the feature space. However, as the n value increased, a broader range of features began to contribute to the model. This can be attributed to the larger number of trees in the forest, which enhances the model's ability to explore and utilize more features for making accurate predictions. Consequently, a higher `n_estimators` value tends to incorporate wider set of features as highlighted in bar plot in Figure 12. The difference in feature score values is presented as a heat-map in Figure 13.
The result of the 20 most important features with the original random forest classifier as constructed in Section 4.4 with $max_{\text{depth}} = 3$ and $n_{\text{estimators}} = 5$ is shown in Figure 14.

5.4. Research Objectives Results

The thesis aimed to answer one main research question:

- Can machine learning be effectively applied to cyber threat intelligence data to predict the category of cyber attacks?

Two sub-research questions were created to answer the main research question:

- How CTI data can be modeled to achieve successful attack category prediction?
- What are the primary input features crucial for accurate prediction, and why does the system consider them significant?

To address the **first sub-question**, the thesis employed a range of data pre-processing techniques on the CTI dataset and engineered new features. These included handling null values, applying bag-of-words to textual columns, performing one-hot encoding

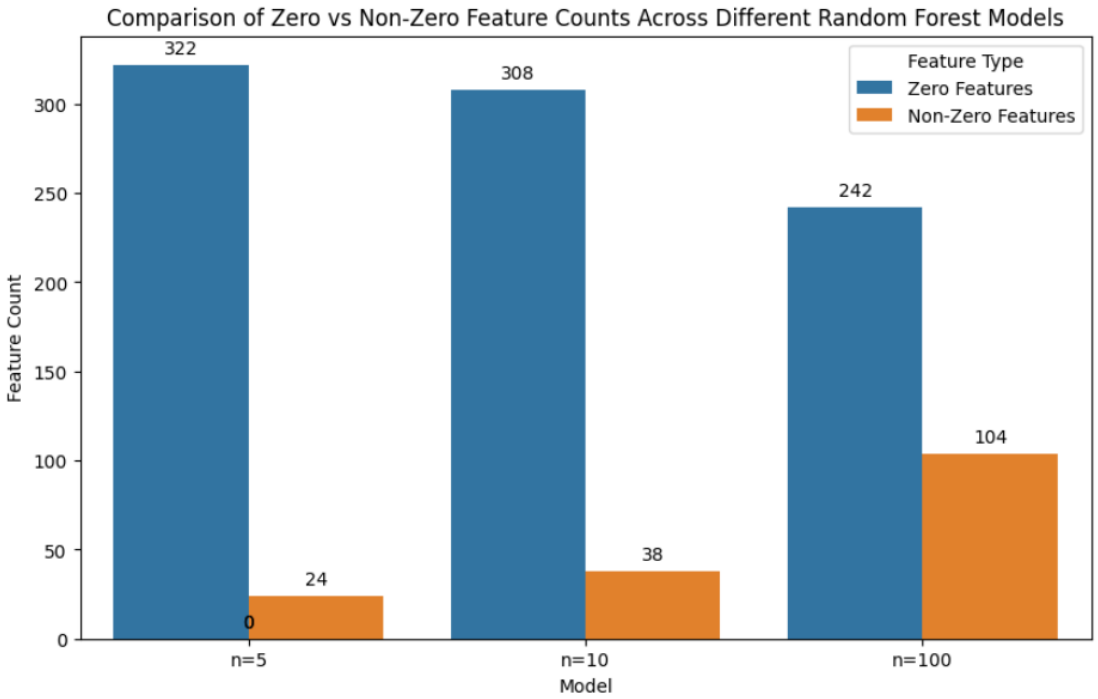


Figure 12. Comparison of zero vs. non-zero feature counts across random forest models with different numbers of estimators (n=5, n=10, n=100).

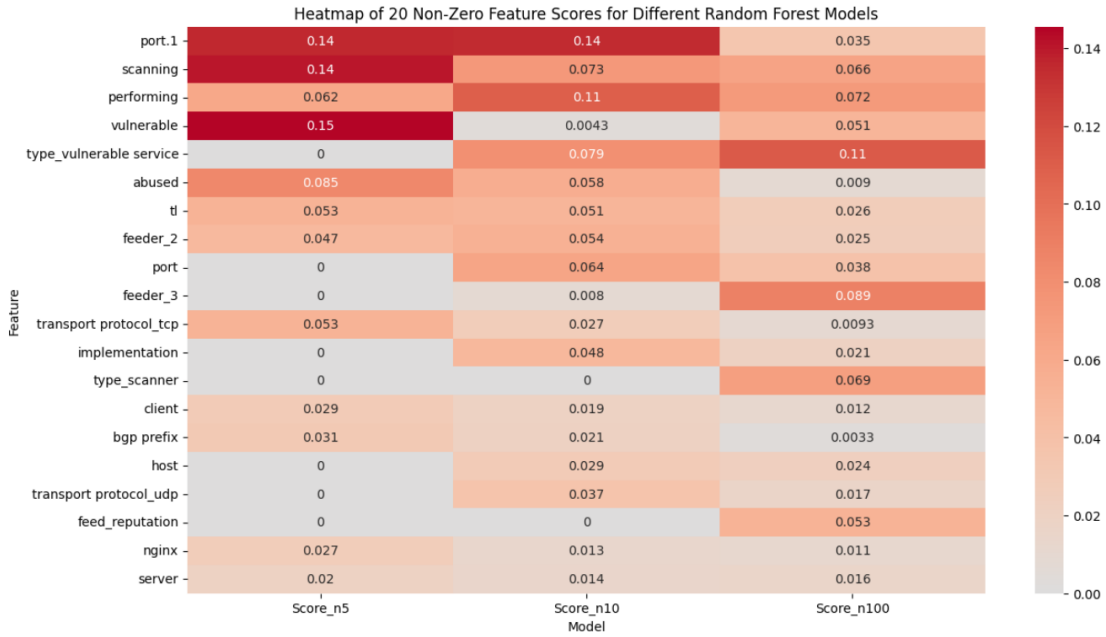


Figure 13. Heatmap of 20 non-zero contributing features common across three different Random Forest models.

for categorical variables, and utilizing NLP to reduce textual descriptions of CTI data. Majority of columns in a CTI data contain text in one form or another. Therefore, feature engineering is a crucial step to convert textual representations into numerical while also retaining key information. Furthermore, a custom method was devised to handle IP address values within the dataset. This custom approach ensured that

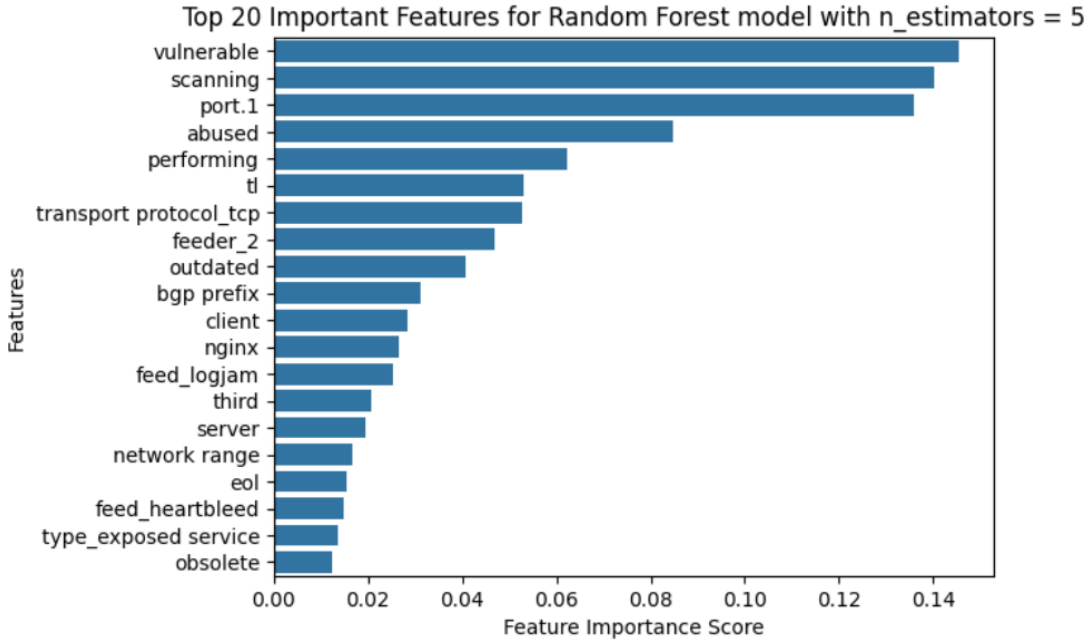


Figure 14. Top 20 most contributing features for the best performing random forest classifier with $max_{\text{depth}} = 3$ and $n_{\text{estimators}} = 5$ and their feature scores.

IP addresses were appropriately transformed into a numerical format while also maintaining their relationships.

For the **second sub-question**, the thesis investigated the importance of input features using random forest classifiers with different values of n (number of estimators). It was found that the choice of n significantly influenced the number of contributing features and their importance scores. A lower n value (e.g., 5) resulted in fewer features being identified as significant contributors, indicating a less focused exploration of the feature space. However, as n increased, a wider range of features began to contribute, suggesting a more comprehensive exploration of the feature space and a greater utilization of features for accurate predictions. The most important primary input features for three different models having feature score >= 0.05 can be seen in the grouped bar chart in Figure 15.

In answer to the **main question**, the research demonstrated that machine learning can be effectively applied to CTI data to predict the category of cyber attacks with greater than 90% accuracy. By pre-processing the data and utilizing ML models such as RF classifiers, the thesis showcased how CTI data can be transformed into machine learning features to predict category of cyber attack. Furthermore, this research lays the framework for processing CTI data to enable application of machine learning on it.

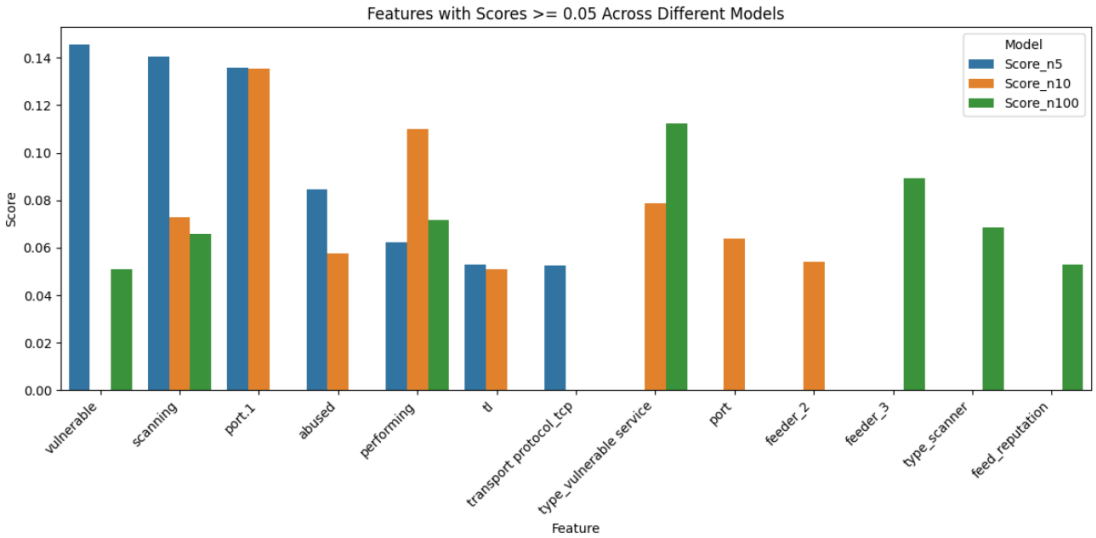


Figure 15. Features with scores >= 0.05 across different models with constant max_depth and varying n_estimator values.

6. DISCUSSION

In the literature review in Chapter 2, two notably correlating studies were presented. Specifically, the research design phase in [5] followed an approach similar to that of this study. The author of [5] categorized the feature space into relevant groups, such as separating temporal features, akin to the method used in this research. However, this study diverges in that temporal features were not utilized, as it is not a time-based prediction problem. Additionally, while [5] presented 258 features, this study incorporates 346 input features.
In contrast to [20], where multiple classifiers were employed, this research exclusively utilized a random forest classifier to generate results and gather feature importance. The subsequent sections of this discussion chapter present the results of applying machine learning techniques to a cyber threat intelligence dataset for predicting the category of cyber attacks. The primary research question focused on the effectiveness of machine learning in this context, with two sub-questions addressing how CTI data can be effectively modeled for attack category prediction and identifying the key input features crucial for accurate predictions.

6.1. Modelling CTI Datasets for Machine Learning

Notably, any CTI dataset contains lots of textual information in the form of urls, protocol type, IP addresses etc. Machine learning cannot process textual information directly and requires techniques such as NLP [45]. Therefore, to apply ML and extract results, a crucial step is to model the data correctly for ML. The original data started with 88 features out of which only a sub-group of 13 features was considered relevant to the prediction of cyber attack category. Numerous columns containing useful information were dropped due to huge percentage of null values. These can be referenced from Table 5. Out of these 13 features, one was dependent feature and rest of the 12 features were utilized to engineer 346 new features, all numerical.
The CTI dataset provided by Arctic Security [28] contained many useful subsets of features. Due to limitations of the dataset acquired, it contained 0.002% records with potential threats category. Unfortunately, research could not explore the prediction of this category type and it was dropped because of low record count. The results then talk about rest of the three categories as presented in harmonization document [7] provided by the Arctic Security [28]. As a first step, it's good practice to group all the features present in the provided dataset based on certain attributes. For the scope of this research, the 88 starting features or columns in the dataset were divided into 5 distinct broad categories Table 4. It is important to note that not all data is relevant to every machine learning problem. Each ML problem comes with it's own set of requirements regarding what data could be considered relevant. Therefore, in the initial study it is important to divide the dataset into relevant broad group of features. Then, based on the research problem relevant group of features can be selected and others can be dropped. With manual study, only 17 features out of 88 were found relevant to the objectives of this research. This reflects how individualistic problem definitions can have varying data requirements.

6.2. Random Forest Classifier Implemntation

Understanding the role of numerous parameters while training a model plays a crucial role in any ML problem. In the context of this research, a random forest classifier was utilized and two key parameters were explored namely `max_depth` and `n_estimators`. Setting value of `max_depth` had huge impact on the model's fitting and values of `max_depth` > 3 resulted in significant overfitting leading to even 99% accuracy as highlighted in Table 8. On the other hand, `n_estimators` did not play a crucial role in improving the accuracy and F1 scores of the model. However, it was found that varying `n_estimators` values had significant effect on the importance of contributing input features. Different models with different `n_estimators` values such as n=5, n=10 and n=100 were explored to see the impact. In any ML problem it is vital to experiment with parameters to see how they affect the overall performance and relevant research objectives.

6.2.1. Analyzing Confusion Matrix

In analyzing the confusion matrix, several insights were gained regarding the model's performance. Notably, the model achieved a perfect score in predicting instances of public exposure. However, it faced challenges in distinguishing between known vulnerabilities and suspected compromises, with instances of misclassification observed in these categories. Specifically, known vulnerabilities were often classified as public exposure, while suspected compromises were sometimes also misclassified as public exposure.

The model's performance over seen data through train:test split did a better job at classifying known vulnerabilities. 881 records were correctly classified and 307 were misclassified as public exposure as shown in Figure 8. However, when tested the same model on unseen data there was drastic reduction in the correct classification of known vulnerabilites category. The model predicted 886 records correctly yet 1710 of them were misclassified as public exposure as in Figure 11. This suggests that the model may not have been adequately trained to recognize the class/category of known vulnerabilities. One potential reason for this could be issues during the CTI data modeling for machine learning stage, where columns relevant to this category might have been dropped due to a high null count, mislabeling, or the loss of some information during feature engineering. Additionally, the class imbalance in the original dataset, as depicted in the pie chart shown in Figure 16, could also have contributed to this issue.

Overall, the research sheds light on the potential of machine learning in enhancing cyber threat intelligence analysis, but also highlights the need for further refinement to improve the model's ability to differentiate between different categories of cyber threats.

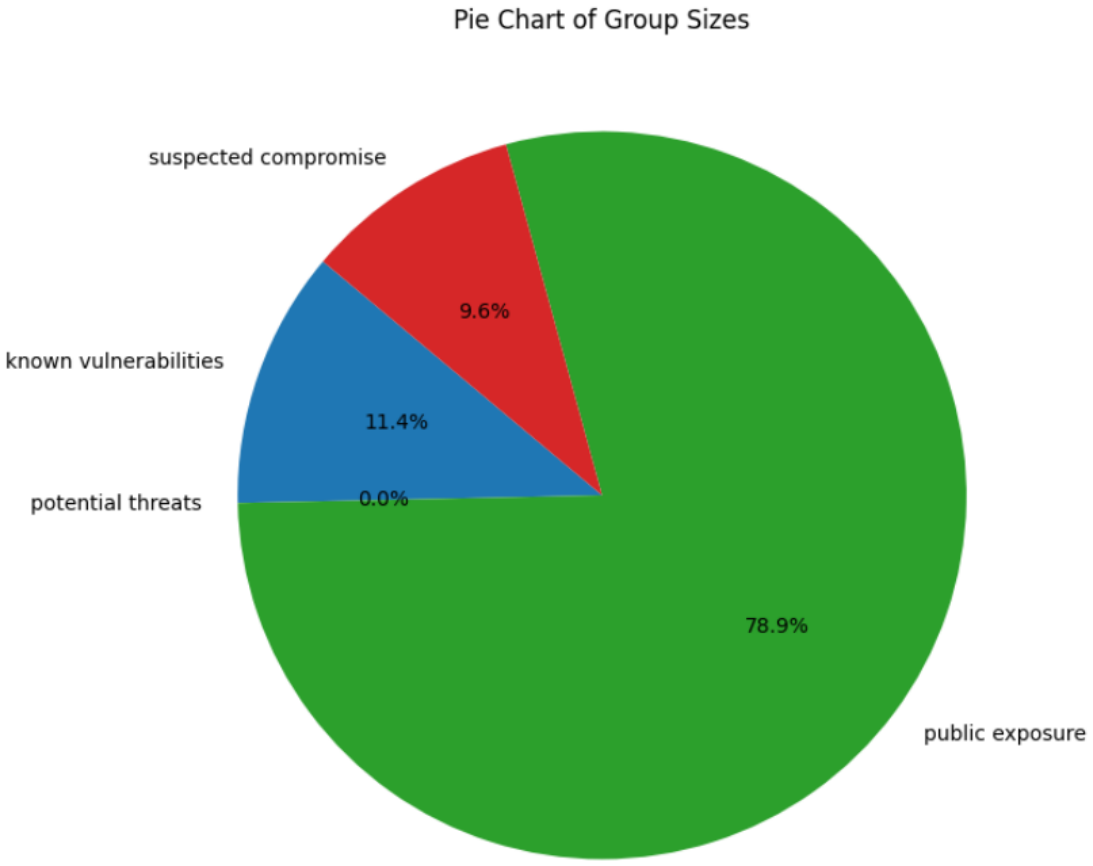


Figure 16. Pie-chart presenting category class imbalance in original dataset.

7. SUMMARY

The research aimed to investigate the effectiveness of applying machine learning techniques to cyber threat intelligence data for predicting the category of cyber attacks. Through a series of experiments, the study addressed two sub-research questions focused on optimizing CTI data modeling and identifying primary input features crucial for accurate predictions. To address the first sub-question, various data preprocessing techniques, including feature engineering methods such as bag-of-words and natural language processing, were employed to convert textual representations into numerical features while retaining key information. Additionally, a custom approach was developed to handle IP address values within the dataset. For the second sub-question, the research explored the importance of input features using random forest classifiers with different values of $n_{\text{estimators}}$. The results indicated that the choice of $n_{\text{estimators}}$ significantly influenced the number of contributing features and their importance scores, with a greater utilization of features leading to more accurate predictions. Ultimately, the study demonstrated that machine learning can be effectively applied to CTI data, achieving over 90% accuracy in predicting cyber attack categories. This research contributes to the understanding of processing CTI data for machine learning applications and lays the groundwork for future advancements in this field.

8. REFERENCES

[1] Katzan H. (2016) Contemporary issues in cybersecurity. Journal of Cybersecurity Research (JCR) 1, pp. 1–6. URL: https://doi.org/10.19030/jcr.v1i1.9745.

[2] Dawson M., Bacius R., Gouveia L.B. & Vassilakos A. (2021) Understanding the challenge of cybersecurity in critical infrastructure sectors. Land Forces Academy Review 26, pp. 69–75. URL: https://doi.org/10.2478/raft-2021-0011.

[3] Hussain A., Mohamed A. & Razali S. (2020) A review on cybersecurity: Challenges & emerging threats. In: Proceedings of the 3rd International Conference on Networking, Information Systems & Security, NISS '20, Association for Computing Machinery, New York, NY, USA. URL: https://doi.org/10.1145/3386723.3387847.

[4] Onoh G. (2018) Predicting cyber-attacks using publicly available data 6, pp. 26–44.

[5] Liu Y., Sarabi A., Zhang J., Naghizadeh P., Karir M., Bailey M. & Liu M. (2015) Cloudy with a chance of breach: Forecasting cyber security incidents. In: 24th USENIX Security Symposium (USENIX Security 15), USENIX Association, Washington, D.C., pp. 1009–1024. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/liu.

[6] Montasari R., Carroll F., Macdonald S., Jahankhani H., Hosseinian-Far A. & Daneshkhah A. (2020) Application of Artificial Intelligence and Machine Learning in Producing Actionable Cyber Threat Intelligence. pp. 47–64.

[7] Security A., Harmonization. https://github.com/arcticsecurity/public/blob/master/docs/Harmonization.md. Accessed: 2024-05-28.

[8] Mendonça M.O., Netto S.L., Diniz P.S. & Theodoridis S. (2024) Chapter 13 - machine learning: Review and trends , pp. 869–959URL: https://www.sciencedirect.com/science/article/pii/B9780323917728000193.

[9] Team E. (2022), What is machine learning? a definition. https://www.expert.ai/blog/machine-learning-definition/.

[10] Goodfellow I., Bengio Y. & Courville A. (2016) Deep Learning. MIT Press. http://www.deeplearningbook.org.

[11] Murphy K.P. (2012) Machine Learning: A Probabilistic Perspective. MIT press.

[12] Abu M.S., Selamat S.R., Ariffin A. & Yusof R. (2018) Cyber threat intelligence–issue and challenges. Indonesian Journal of Electrical Engineering and Computer Science 10, pp. 371–379.

[13] Lowenthal M.M. (2016) Intelligence: From Secrets to Policy. CQ Press, 7th ed.

[14] Moore D. & Rid T. (2016) Cryptopolitik and the darknet. Survival 58, pp. 7–38. URL: https://doi.org/10.1080/00396338.2016.1142085.

[15] CyberNX (2024), What is malware analysis and how it unveils the anatomy of malicious software. URL: https://www.cybernx.com/a-what-is-malware-analysis-and-how-it-unveils-the-anatomy-of-malicious-software.

[16] Wikipedia contributors (2024), Intrusion detection system. https://en.wikipedia.org/wiki/Intrusion_detection_system. [Online; accessed 7-June-2024].

[17] Wikipedia contributors (2024), Security information and event management. https://en.wikipedia.org/wiki/Security_information_and_event_management. [Online; accessed 7-June-2024].

[18] .

[19] Evangelou M. & Adams N.M. (2020) An anomaly detection framework for cyber-security data. Computers Security 97, p. 101941. URL: https://www.sciencedirect.com/science/article/pii/S0167404820302170.

[20] Gupta G.P. & Kulariya M. (2016) A framework for fast and efficient cyber security network intrusion detection using apache spark. Procedia Computer Science 93, pp. 824–831. URL: https://www.sciencedirect.com/science/article/pii/S1877050916314806, proceedings of the 6th International Conference on Advances in Computing and Communications.

[21] Fang X., Xu M., Xu S. & Zhao P. (2019) A deep learning framework for predicting cyber attacks rates. EURASIP Journal on Information Security 2019.

[22] Georgescu T.M. (2020) Natural language processing model for automatic analysis of cybersecurity-related documents. Symmetry 12. URL: https://www.mdpi.com/2073-8994/12/3/354.

[23] Cristea L.M. (2020) Current security threats in the national and international context. Journal of accounting and management information systems 19, pp. 351–378.

[24] Bebeshko B., Khorolska K., Kotenko N., Kharchenko O. & Zhyrova T. (2021) Use of neural networks for predicting cyberattacks. Cybersecurity Providing in Information and Telecommunication Systems .

[25] Kinnunen J. (2022) Threat Detection Gap Analysis Using MITRE ATT&CK Framework. Master's thesis, Master's Degree Programme in Information Technology, Cyber Security, Information and Communications Technology, Enfo Oyj. Permission for web publication: yes.

[26] Khandpur R.P., Ji T., Jan S., Wang G., Lu C.T. & Ramakrishnan N. (2017) Crowdsourcing cybersecurity: Cyber attack detection using social media .

[27] Al-Mansoori S. & Salem M.B. (2023) The role of artificial intelligence and machine learning in shaping the future of cybersecurity: Trends, applications, and ethical considerations. International Journal of Social Analytics 8, p. 1–16. URL: https://norislab.com/index.php/ijsa/article/view/36.

[28] Arctic Security. https://www.arcticsecurity.com/about-us. Accessed: June 2024.

[29] Arctic Security, Arctic security early warning service. https://www.arcticsecurity.com/products/early-warning-service. Accessed: June 7, 2024.

[30] Hastie T., Tibshirani R. & Friedman J. (2009) The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.

[31] GeeksforGeeks (2024), One hot encoding in machine learning. URL: https://www.geeksforgeeks.org/ml-one-hot-encoding/, last accessed on May 15, 2024.

[32] Brownlee J. (2019), A gentle introduction to the bag-of-words model. URL: https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[33] Alani M. (2021), Handling ip addresses in machine learning datasets. URL: https://www.mohammedalani.com/tutorials/handling-ip-addresses-in-machine-learning-datasets/.

[34] Samuels J. (2024) One-hot encoding and two-hot encoding: An introduction .

[35] Breiman L. (2001) Random forests. Machine learning 45, pp. 5–32.

[36] GeeksforGeeks (Year), The effects of the depth and number of trees in a random forest. https://www.geeksforgeeks.org/the-effects-of-the-depth-and-number-of-trees-in-a-random-forest/. Last accessed: 03 Apr, 2024.

[37] Hambali M., Oladele T., S. A., Kumar A. & Gao W. (2022) Feature selection and computational optimization in high-dimensional microarray cancer datasets via infogain-modified bat algorithm. Multimedia Tools and Applications 81.

[38] GeeksforGeeks (2024), Metrics for Machine Learning Model. https://www.geeksforgeeks.org/metrics-for-machine-learning-model/. [Online; accessed 3-June-2024].

[39] DataCamp Team (2023), Random Forest Classification with Scikit-Learn. URL: https://www.datacamp.com/tutorial/random-forests-classifier-python, [Online; accessed 3-June-2024].

[40] GeeksforGeeks (2023), Cross Validation in Machine Learning. https://www.geeksforgeeks.org/cross-validation-machine-learning/. [Online; accessed 3-June-2024].

[41] Scheaffer R.L., Mendenhall W. & Ott L. (1996) Elementary Survey Sampling. Duxbury Press.

[42] Efron B. & Tibshirani R.J. (1993) An Introduction to the Bootstrap. CRC press.

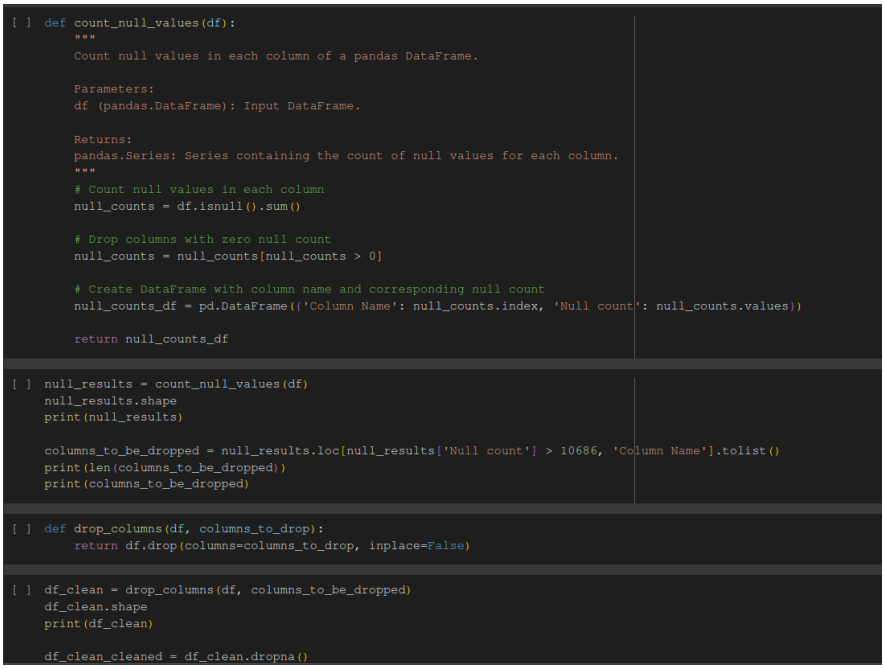
[43] Shin H.C. (2018) Ordinal encoder. In: Encyclopedia of Database Systems, Springer, 2nd ed., pp. 2421–2421.

[44] Varoquaux G., Gouillart E., Michel V., Vanderplas J. & Gramfort A., Joblib: Running python functions as pipeline jobs. https://joblib.readthedocs.io/en/stable/.

[45] Jurafsky D. & Martin J.H. (2020) Speech and Language Processing. Pearson Education, Inc.

9. APPENDICES

Appendix 1 Sample code snippets from Google Colab notebook
Appendix 2 Data processing pipeline for feature engineering



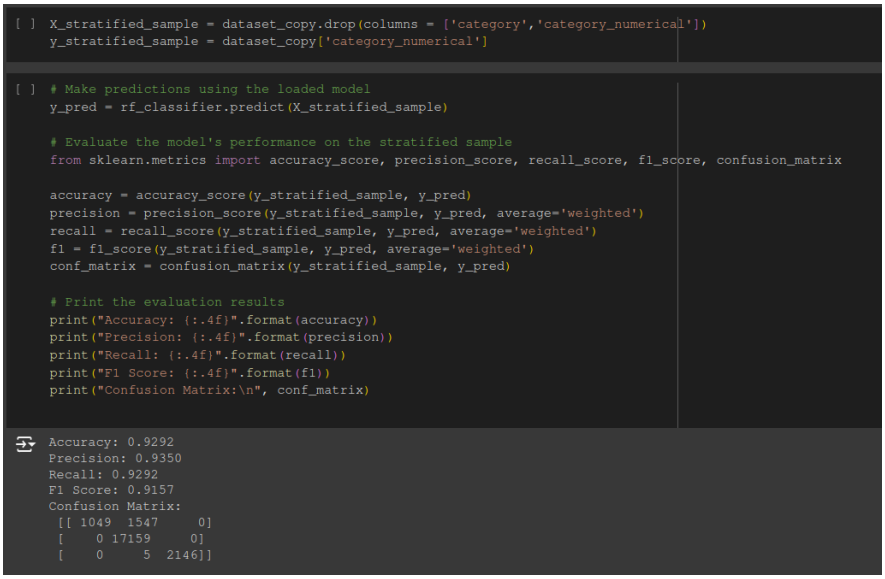
```
[ ] def count_null_values(df):
        """
        Count null values in each column of a pandas DataFrame.

        Parameters:
        df (pandas.DataFrame): Input DataFrame.

        Returns:
        pandas.Series: Series containing the count of null values for each column.
        """
        # Count null values in each column
        null_counts = df.isnull().sum()

        # Drop columns with zero null count
        null_counts = null_counts[null_counts > 0]

        # Create DataFrame with column name and corresponding null count
        null_counts_df = pd.DataFrame({'Column Name': null_counts.index, 'Null count': null_counts.values})

        return null_counts_df
```

```
[ ] null_results = count_null_values(df)
    null_results.shape
    print(null_results)

    columns_to_be_dropped = null_results.loc[null_results['Null count'] > 10686, 'Column Name'].tolist()
    print(len(columns_to_be_dropped))
    print(columns_to_be_dropped)
```

```
[ ] def drop_columns(df, columns_to_drop):
        return df.drop(columns=columns_to_drop, inplace=False)
```

```
[ ] df_clean = drop_columns(df, columns_to_be_dropped)
    df_clean.shape
    print(df_clean)

    df_clean_cleaned = df_clean.dropna()
```

Figure 1. A sample code snippet to count null values in each column and drop columns having null count > 11.78%.



```
[ ] X_stratified_sample = dataset_copy.drop(columns = ['category','category_numerical'])
    y_stratified_sample = dataset_copy['category_numerical']
```

```
[ ] # Make predictions using the loaded model
    y_pred = rf_classifier.predict(X_stratified_sample)

    # Evaluate the model's performance on the stratified sample
    from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

    accuracy = accuracy_score(y_stratified_sample, y_pred)
    precision = precision_score(y_stratified_sample, y_pred, average='weighted')
    recall = recall_score(y_stratified_sample, y_pred, average='weighted')
    f1 = f1_score(y_stratified_sample, y_pred, average='weighted')
    conf_matrix = confusion_matrix(y_stratified_sample, y_pred)

    # Print the evaluation results
    print("Accuracy: {:.4f}".format(accuracy))
    print("Precision: {:.4f}".format(precision))
    print("Recall: {:.4f}".format(recall))
    print("F1 Score: {:.4f}".format(f1))
    print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 0.9292
Precision: 0.9350
Recall: 0.9292
F1 Score: 0.9157
Confusion Matrix:
 [[ 1049  1547     0]
 [    0 17159     0]
 [    0     5  2146]]
```

Figure 2. Code with output showing performance metrics for Random Forest classifier with max_depth=3, n_estimators=10, and random_state=42.

0.1. Google Colab Code

Here is the code from Google Colab notebook for the feature engineering pipeline. This pipeline consists of two parts. The input of the first part 1 is saved as '.csv' file and fed to the second part 2 of the pipeline for further processing.

```
# -*- coding: utf-8 -*-
"""Master_Thesis_Feature_Engineering.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
       k8tTuYvRPJEO3A-HB1fSMiWThTPzPXn4
"""

import pandas as pd
## Display all the columns of the dataframe

pd.pandas.set_option('display.max_columns',None)

from google.colab import drive
drive.mount('/content/gdrive')

dataset = pd.read_csv('/content/gdrive/MyDrive/
   Masters_thesis/final_analysis_dataset.csv', low_memory
    = False)
print(dataset.head(5))
print(dataset.shape)
dataset.columns

"""# **Divide the features into numeric and textual**"""

# list of numerical variables
numerical_features = [feature for feature in dataset.
   columns if dataset[feature].dtypes != 'O']

print('Number of numerical variables: ', len(
   numerical_features))

# visualise the numerical variables
dataset[numerical_features].head()

import pandas as pd
import matplotlib.pyplot as plt

# Assuming `dataset` is your DataFrame
```

```
37
38 # Identify numerical and non-numerical features
39 numerical_features = [feature for feature in dataset.
      columns if dataset[feature].dtypes != 'O']
40 non_numerical_features = [feature for feature in dataset.
      columns if dataset[feature].dtypes == 'O']
41
42 # Count the number of numerical and non-numerical
      features
43 num_numerical = len(numerical_features)
44 num_non_numerical = len(non_numerical_features)
45
46 # Data for the pie chart
47 labels = ['Numerical Features', 'Non-Numerical Features']
48 sizes = [num_numerical, num_non_numerical]
49 colors = ['#ff9999','#66b3ff']
50 explode = (0.1, 0)  # explode 1st slice (Numerical
      Features)
51
52 # Plotting the pie chart
53 plt.figure(figsize=(8, 8))
54 plt.pie(sizes, explode=explode, labels=labels, colors=
      colors, autopct='%1.1f%%',
55         shadow=True, startangle=140)
56 plt.title('Proportion of Numerical and Non-Numerical
      Features')
57 plt.axis('equal')  # Equal aspect ratio ensures that pie
      is drawn as a circle.
58 plt.show()
59
60 """# Temporal Features (E.g time, year e.t.c)
61
62 """
63
64 #list of temporal features
65 temporal_features = ['source time', 'observation time']
66
67 """# Geospatial Features (E.g place, network owner, isp e
      .t.c)
68
69 """
70
71 geospatial_features = ['reported city','reported cc','
      reported country','isp','reported organization','
      region code','geoip cc','cymru cc','bgp prefix
      allocated','as allocated','as name','cc','network
      owner',
```

```
                        'network cc','network name']

"""# Other Irrelevant Features"""

irrelevant_features = ['description url','feed url','
    registry','hub connector']

"""# Confidential Features"""

confidential_features = ['reported latitude','reported
    longitude','latitude','longitude','uuid']

"""# Drop columns not required"""

list_of_columns_to_drop = temporal_features +
    geospatial_features + irrelevant_features +
    confidential_features

total_columns_to_drop = len(temporal_features) + len(
    geospatial_features) + len(irrelevant_features) + len(
    confidential_features)
print(total_columns_to_drop)

dataset_new = dataset.drop(columns =
    list_of_columns_to_drop)

print(dataset_new.shape)
print(dataset_new.columns)
dataset_new.head(5)

# @title reported asn vs port

from matplotlib import pyplot as plt
dataset_new.plot(kind='scatter', x='reported asn', y='
    port', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)

# @title category

from matplotlib import pyplot as plt
import seaborn as sns
dataset_new.groupby('category').size().plot(kind='barh',
    color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)

# @title type

```

```
from matplotlib import pyplot as plt
import seaborn as sns
dataset_new.groupby('type').size().plot(kind='barh',
   color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right',]].set_visible(False)

# @title port

from matplotlib import pyplot as plt
dataset_new['port'].plot(kind='hist', bins=20, title='
   port')
plt.gca().spines[['top', 'right',]].set_visible(False)

# @title reported asn

from matplotlib import pyplot as plt
dataset_new['reported asn'].plot(kind='hist', bins=20,
   title='reported asn')
plt.gca().spines[['top', 'right',]].set_visible(False)

study_set = dataset_new.groupby('category').size()
print(study_set)

"""# Export dataset_new to csv"""

dataset_new.to_csv('dataset_new_without_third_category.
   csv', index=False)
```

Listing 1. First part of feature engineering pipeline from Google Colab

```
# -*- coding: utf-8 -*-
"""Master_Thesis_Feature_Engineering_Part2.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1
       xmbPMggcD62JISkyyCSfOggEAjszB-u1

# Feature Engineering Part 2: Textual to Numerical
   Transformation

Here we take the machine learning ready dataset and
   transform all textual columns into numeric ones.
```

```
    Nominal Variable (Categorical). Variable comprises a finite set of discrete values with no relationship between values.
    Ordinal Variable. Variable comprises a finite set of discrete values with a ranked ordering between values.


A note on the dataset: In categorical data there can be ordinal variables for example:  place  variable with "first","second" e.t.c values has a natural ordering of values. This type of categorical variable is called an ordinal variable.

In this dataset we DO NOT have any ordinal variables.
For categorical variables where no such ordinal relationship exists, the integer encoding is not enough and we have to use one-hot encoding.
"""

import pandas as pd
## Display all the columns of the dataframe

pd.pandas.set_option('display.max_columns',None)

from google.colab import drive
drive.mount('/content/gdrive')

dataset = pd.read_csv('/content/gdrive/MyDrive/Masters_thesis/final_dataset_for_machine_learning.csv', low_memory = False)
print(dataset.head(5))
print(dataset.shape)
dataset.columns

dataset

print(dataset['description'].nunique())
description_array = dataset['description'].unique()
print(description_array)
dataset.dtypes

# Calculate word count for each description in description_array
word_count_per_description = [len(description.split()) for description in description_array]

```

```
# Display word count for each description
for i, description in enumerate(description_array):
    print("Description {}: Word Count = {}".format(
       description, word_count_per_description[i]))

"""# Applying NLP on description column

Here we apply tokenization, stop word removal, and
   lemmatization on the description column. Note that we
   have 91 unique descriptions for 87621 records. This
   process will eliminate unnecessary words from each
   description
"""

import pandas as pd
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer


df = pd.DataFrame(dataset)

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Tokenization, stop word removal, and lemmatization
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    tokens = word_tokenize(text.lower())  # Tokenization
       and lowercase
    filtered_tokens = [lemmatizer.lemmatize(token) for
       token in tokens if token.isalnum() and token not
       in stop_words]  # Remove stop words and non-
       alphanumeric tokens, lemmatize
    return ' '.join(filtered_tokens)

# Apply preprocessing to the 'description' column and
   replace it
df['description'] = df['description'].apply(
   preprocess_text)

```

```
# Display the DataFrame with replaced 'description' column
print(df)

#Make a copy
df_original = df.copy()

df['description']

df

print(df['description'].nunique())

"""# Applying custom path II encoding on IP/BGP and Network columns

We have three columns that have IP addresses in them. These are:
bgp prefix, ip, network range.

We write a function that converts them into numbers while maintaining the relationship between IP addresses and network addresses. An IP address has four octets each separated by dots. In this technique, each octet is first made a three digit number by appending zeros and the dots are dropped. This results in a standard 12 digits number representing each number.
"""

print(len(df.columns))
df.columns

def drop_subet(df, column_name):
    df[column_name] = df[column_name].str.split('/').str[0]
    return df

drop_subet(df, 'bgp prefix')
print(df['bgp prefix'])

drop_subet(df, 'network range')
print(df['network range'])

array = df['ip'].tolist()
print(array)
print(type(array))

```

```
# Create a list to store the parts
modified_array = []

for x in array:  # Iterate directly over elements in array
    z = ''
    part1 = x.split('.', 1)
    part2 = part1[1].split('.', 1)
    part3 = part2[1].split('.', 1)
    part4 = part3[1].split('.', 1)
    parts = [part1[0], part2[0], part3[0], part4[0]]  # Store only the first part before the dots
    for prt in range(4):  # Iterate over all parts
        if len(parts[prt]) == 3:  # Check the length of each part
            z += parts[prt]  # Concatenate if length is 3
        elif len(parts[prt]) == 2:
            z += '0' + parts[prt]  # Add leading '0' if length is 2
        else:
            z += '00' + parts[prt]  # Add leading '00' if length is 1
    modified_array.append(int(z))

# Print the modified array to verify the changes
print(modified_array)

def modify_ip_addresses(df, column_name):
    modified_array = []

    for index, x in enumerate(df[column_name]):
        z = ''
        part1 = x.split('.', 1)
        part2 = part1[1].split('.', 1)
        part3 = part2[1].split('.', 1)
        part4 = part3[1].split('.', 1)
        parts = [part1[0], part2[0], part3[0], part4[0]]
        for prt in range(4):
            if len(parts[prt]) == 3:
                z += parts[prt]
            elif len(parts[prt]) == 2:
                z += '0' + parts[prt]
            else:
                z += '00' + parts[prt]
        modified_array.append(int(z))
        # Update the value in the DataFrame
        df.at[index, column_name] = int(z)
```

```
161
162     return df
163
164 df_copy = df.copy()
165 modify_ip_addresses(df_copy, 'ip')
166 modify_ip_addresses(df_copy, 'bgp prefix')
167 modify_ip_addresses(df_copy, 'network range')
168 print(df_copy['ip'])
169
170 df_copy['network range']
171
172 print(df_copy)
173
174 #df_copy.to_csv('ip_processed_dataset.csv', index = False
    )
175
176 """# Plotting word distribution in different columns
177
178 Here we plot word distributions in the ip_address
    processed datasets by using pandas groupby on each
    function.
179 """
180
181 grouped = df_copy.groupby('transport protocol')
182 group_size = grouped.size()  # Number of elements in each
     group
183 print(group_size)
184
185 grouped1 = df_copy.groupby('feeder')
186 group_size1 = grouped1.size()  # Number of elements in
    each group
187 print(group_size1)
188
189 grouped2 = df_copy.groupby('feed type')
190 group_size2 = grouped2.size()  # Number of elements in
    each group
191 print(group_size2)
192
193 import seaborn as sns
194 import matplotlib.pyplot as plt
195
196 # Assuming you have already grouped the data
197 grouped = df_copy.groupby('transport protocol')
198 protocol_counts = grouped.size().reset_index(name='count'
    )
199
200 # Create horizontal bar plot using Seaborn
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='transport protocol', data=protocol_counts, orient='h')
plt.xlabel('Count')
plt.ylabel('Transport Protocol')
plt.title('Count of Transport Protocols')
plt.show()

def plot_horizontal_bar(df, column_name):
    grouped = df.groupby(column_name)
    counts = grouped.size().reset_index(name='count')

    plt.figure(figsize=(10, 6))
    sns.barplot(x='count', y=column_name, data=counts, orient='h')
    plt.xlabel('Count')
    plt.ylabel(column_name.capitalize())
    plt.title(f'Count of {column_name.capitalize()}')
    plt.show()

plot_horizontal_bar(df_copy, 'feed')
plot_horizontal_bar(df_copy, 'description')
plot_horizontal_bar(df_copy, 'type')
plot_horizontal_bar(df_copy, 'feed type')
plot_horizontal_bar(df_copy, 'category')

"""# Textual columns with greater number of categories and their processing

Following columns require more processing based on different categories that appeared in them after grouping by.

1. Feed
2. Description
3. Type

"""

print(df_copy['feed'].nunique())
print(df_copy['description'].nunique())
print(df_copy['type'].nunique())

# Calculate unique counts for each column
unique_counts = {
    'Feed': df_copy['feed'].nunique(),
    'Description': df_copy['description'].nunique(),
```

```
    'Type': df_copy['type'].nunique()
}

# Convert dictionary to DataFrame for plotting
unique_counts_df = pd.DataFrame(list(unique_counts.items()), columns=['Column', 'Unique Count'])

# Create vertical bar plot using Seaborn
plt.figure(figsize=(8, 5))
sns.barplot(x='Column', y='Unique Count', data=unique_counts_df, palette='viridis')
plt.xlabel('Column')
plt.ylabel('Unique category counts')
plt.title('Unique category counts in columns')
plt.show()

"""# Applying bag of words on columns with greater categories.

The columns are:
1. Feed
2. Description
3. Type

Description column will first be handled with bag of words and if that doesn't work, one-hot encoding!

Feed and type will be handeled by one-hot encoding

We are using df_copy dataset and make it's new copy for processing called: df_copy_new
"""

from sklearn.feature_extraction.text import CountVectorizer

def apply_bag_of_words(df, column_name):
    # Step 1: Tokenization
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(df[column_name])

    # Step 2: Vectorization
    feature_names = vectorizer.get_feature_names_out()
    bag_of_words_df = pd.DataFrame(X.toarray(), columns=feature_names)

```

```
    # Step 3: Replace the text column with the vectorized
        representation
    df.drop(columns=[column_name], inplace=True)
    df = pd.concat([df, bag_of_words_df], axis=1)

    return df

df_copy_new = df_copy.copy()
#apply_bag_of_words(df_copy_new, 'description')
df_with_bow = apply_bag_of_words(df_copy_new, '
   description')

df_with_bow
df_with_bow.columns

df_copy.columns

num_columns_not_in_copy = len(set(df_with_bow.columns) -
   set(df_copy))

print("Number of columns in df_with_bow not present in
   df_copy:", num_columns_not_in_copy)

print(type(df_original))  # Corrected from print(
   df_original.type())
print(df_original.size)   # Corrected from print(
   df_original.size())
print(df_original)        # This prints the DataFrame
   itself

df_original['description']

print(df_original['description'].value_counts())
print(df_original['description'].unique())

"""# Applying one-hot encoding!

One-hot encoding is a dumb way of converting text to
   numeric representation. Here we are doing it for
   following columns:
1. Feed
2. Type
3. Transport protocol
4. Feeder

Also, column feed type gets dropped because of
   univariative in nature.
```

```
319
320 We can use pd.get_dummies() function from pandas to one-
       hot encode the categorical columns. Alternate option
       is to use scikit learn library. Ref doc: https://www.
       geeksforgeeks.org/ml-one-hot-encoding/
321
322 We use df_with_bow here! It already has description
       processed with bag of words and has 251 features
323 """
324
325 one_hot_encoded_data = pd.get_dummies(df_with_bow,
       columns=['feed', 'type','transport protocol','feeder'
       ], dtype=float)
326
327 one_hot_encoded_data
328
329 one_hot_encoded_data = one_hot_encoded_data.drop('feed
       type', axis = 1)
330
331 one_hot_encoded_data.size
332 one_hot_encoded_data.columns
333
334 one_hot_encoded_data.to_csv('complete_numeric_dataset.csv
       ', index = False )
335
336 """Summary
337
338 1. Description column handled using bag of words. 1 > 239
339 2. Feed column handled using one-hot encoding. 1 > 86
340 3. Type column handled using one-hot encoding. 1 > 10
341 4. Transport protocol column handled using one-hot
       encoding. 1 > 2
342 5. Feeder column handled using one-hot encoding. 1 > 3
343
344 7 original columns, 1 dropped and rest of transformed
       ones. This means we got 340 columns from bag of words
       and one-hot encoding. CALCULATIONS ABOVE NEED
       CORRECTION!
345
346
347 """
```

Listing 2. Second part of feature engineering pipeline from Google Colab