Machine Learning Approaches for Enhanced Phishing Website Detection: Insights and Strategies

Hamza Osama AL-Risheq

May 2024

Abstract

Big data refers to large datasets, both structured and unstructured, that challenge typical data processing approaches. The computational analysis of big datasets reveals patterns and trends that are critical for directing decisions and comprehending challenging events, especially in fields that rely heavily on data-driven insights. One of the threats facing big data is phishing, which can lead to the compromise of sensitive data, affecting the organization's reputation and causing legal issues. Detecting phishing websites is a complex, challenging topic in the cybersecurity world, where innovative techniques and machine learning models are used to protect users from illegal attempts to steal sensitive information via phishing websites. This paper studies machine learning models aimed to enhance the accuracy and effectiveness of detecting phishing websites. The study focuses on testing and analyzing these models to identify which ones achieve the highest accuracy and F1 Score in detecting phishing websites. Our analysis recommends that the best models that can be used to detect phishing websites are the Neural Networks and SVM models before balancing, which achieve 89.2% accuracy and 94.3% F1 Score, and the CatBoost model after balancing, which achieves 89.8% accuracy and 90.3% F1 Score.

Keywords: Phishing Websites; Phishing Detection; Big Data; Machine Learning; Cyber Phishing; Neural Networks; SVM; CatBoost

1 Introduction

The paper focuses on how to detect phishing websites effectively. Phishing stands out as one of the most harmful security threats. It is a malicious cyber activity where attackers employ various techniques and tactics to craft counterfeit websites or emails that mimic legitimate ones [1]. Their goal; To trick unsuspecting individuals into sharing sensitive details like usernames, passwords, and financial information. Phishing is a type of social engineering attack that takes advantage of human trust and psychology to get access to personal information. Phishing techniques have changed throughout time, becoming more complex and difficult to detect.

Recent versions of phishing attacks, such as spear phishing and whale phishing, show significant shifts in attack techniques. As attackers develop their techniques, efforts, and strategies to detect phishing websites are improved. Today, several tools and services use AI, machine learning, and deep learning to detect and prevent phishing websites. On the other hand, claiming that we are completely protected from phishing websites remains illogical [1].

The goal here is to get better at detecting and catching them to protect users from getting scammed online. In my paper, I used an exited dataset [2] having features that are known to be good indicators of phishing websites. Then I used techniques and approaches to improve the results, and accuracy of the results of the dataset to get better insights. The goal is to make it easy to know between good and phishing websites while keeping everyone safe online.

Since the dawn of awareness about phishing websites, it has become an essential aspect of cybersecurity to understand and detect them due to their significant impact on both individuals and businesses. The lack of awareness about phishing websites can make normal people vulnerable, leading to potential attacks and loss of sensitive information.

However, the primary concern with phishing websites lies in their utilization as a steppingstone for more advanced attacks. Exploiting the unawareness of phishing websites, attackers can lead individuals into using fake websites and subsequently obtain their credentials, granting access to their devices. These individuals may include CEOs of companies, doctors in universities, medicine people, government officials, anyone, or maybe you. Hence, the importance and impact of phishing websites are evident, as they can result in damaged reputation, financial losses, data breaches, and loss of customers.

Figure 1 shows the financial losses caused by phishing attacks, highlighting the

economic significance.

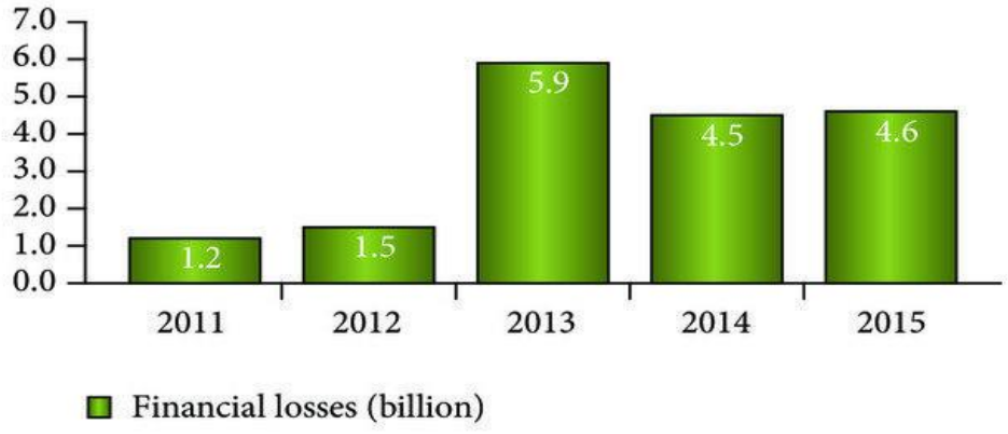


Figure 1: Financial loss due to phishing attacks [1]

2 Literature Review

Recent research focused on employing new technologies, such as deep learning and machine learning, to improve phishing detection. Ullah et al. (2024) [3] illustrated the effectiveness of deep learning algorithms, specifically Convolutional Neural Networks (CNNs), in correctly understanding phishing URLs. Using a vast dataset of millions of phishing and legitimate website URLs, the researchers created and tested deep learning models that detected phishing websites with high precision and recall percentages. The study demonstrated the effectiveness of modern AI-based strategies in combating the growing threat of phishing attacks, stressing the importance of technological advancements in cybersecurity.

In a similar way, Das Gupta et al. (2022) [4] illustrated the necessity of real-time phishing detection methods that were not dependent on other systems or technology. Their research looked at a number of detection tactics, including list-based, visual similarity-based, and machine learning-based methods. The study proposed a machine learning-based approach, which utilized datasets and classification algorithms to detect phishing websites using client-side features.

M. Mohammad et al. (2015) [2] addressed the main challenge of phishing detection on websites. They handled the challenge of not having much reliable training data, which is a common problem in this industry. They developed standards for distinguishing between legitimate and phishing websites by carefully evaluating the structure of URLs, such as URL patterns, URL length, URL depth, and the usage of the IFrame tool on the website. Their work went beyond simply identifying existing indications; they also presented new insights, expanding the number of strategies for detecting phishing. Furthermore, they analyzed real-world data from sites such as PhishTank and StopBadware, acquiring significant insights into the changing landscape of web-based threats.

The researchers planned to combine deep learning and machine learning to create real-time solutions for detecting phishing websites. These efforts were part of an expanded dedication to enhancing cybersecurity measures to protect individuals and organizations from phishing attempts.

Recently, new approaches such as Large Language Models (LLMs) have been used in classification cases, such as detecting phishing websites. These models require a huge amount of data to effectively train and achieve high performance. However, in this research, these models can't be applied because the dataset does not have a sufficient number of records. This limitation means that alternative methods or smaller models that require less data must be considered to address the problem effectively.

Based on the results above its imperative to design more effective phishing website detection techniques. This research comes to fill the gap by exploring more machine learning methods and techniques to effectively detect the phishing websites.

3 Research Approach and Methodologies

In this section, I will use the onion model, which is a research model illustrating the layers of research methodology applied in this paper.

3.1 Onion Model

Research methodology guides how research is conducted by defining philosophical beliefs, approaches, methods, and data collection strategies. The "research onion," proposed by Saunders et al. (2016), shows this process with layers reflecting philosophical foundations, research approaches, methods, and data gathering strategies. It guarantees that these parts are consistent, resulting in a well-designed

research plan that follows the research's aims and principles. It serves as a systematic framework for developing research methods. It leads researchers through a series of stacked layers, each contributing to the development of a successful research strategy [5]. The layers of onion model are presented in Figure 2.

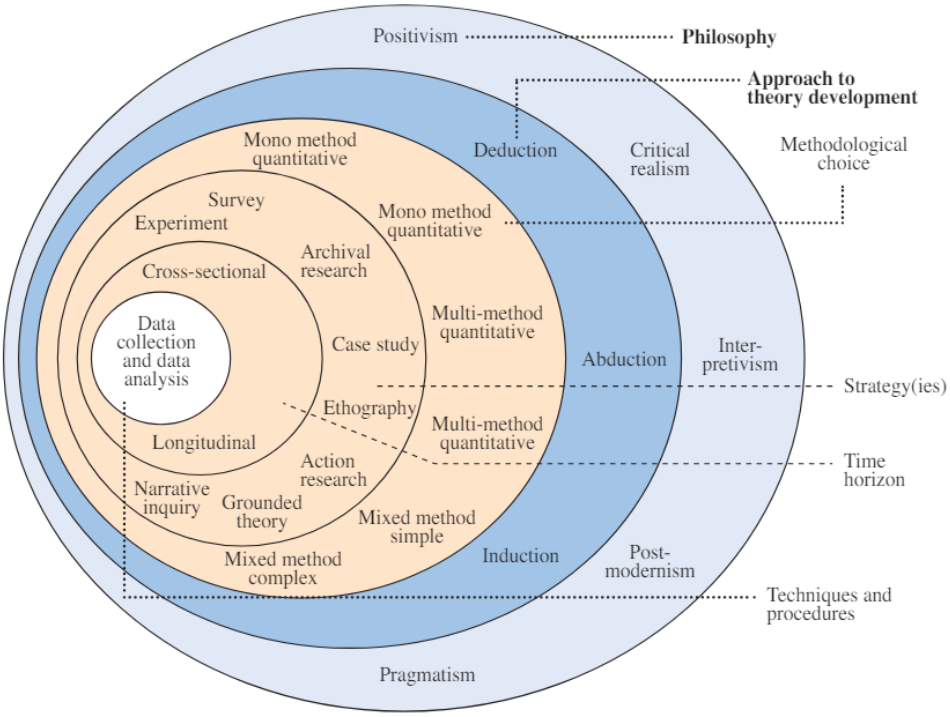


Figure 2: The structure of the Onion Research Model [5]

3.1.1 Philosophy

The research adopts a **positivist philosophy**, it relies on the idea that knowledge is derived from tangible and quantifiable facts. This philosophy is suitable for this study because it focuses on observed evidence derived from data to detect phishing websites. By employing a positivist philosophy, we can objectively evaluate concepts and validate our findings using statistical analysis. Because these are quantitative facts, they are objective and not open to several opinions.

3.1.2 Theory Development Approach

A **deductive approach** is utilized, starting with existing theories and concepts about phishing detection. This approach entails developing predictions based on these theories and evaluating them via statistical analysis. By using a deductive approach, we can systematically validate the effectiveness of different phishing detection methods and refine them based on the results. We started with broad theoretical concepts and worked up to down to specific details, as indicated by the results and discussion section.

3.1.3 Methodological Choice

Table 1: **Comparison of Quantitative and Qualitative Methods**

Aspects	***Quantitative Methods***	***Qualitative Methods***
Objective	Quantify and generalize findings through numerical data	Explore meanings, understand context, and uncover patterns.
Data Collection	Structured methods like surveys, and datasets.	Unstructured or semi-structured methods like interviews, observations, and surveys.
Data Type	Numerical data	Textual, descriptive data.
Data Type Result	Numerical data	Textual, descriptive data.
Analysis	Statistical analysis.	Interpretive analysis such as thematic analysis and content analysis.
Examples	Surveys with closed-ended questions. And analysis dataset that contain binary numbers.	Interviews, focus groups, observations
Outcomes	Quantifiable results, statistical relationships	Rich descriptions and deep ideas.

Key Differences:

- Nature of Data: Quantitative research deals with numerical data, while qualitative research deals with textual or descriptive data.
- Analysis Techniques: Quantitative methods use statistical analysis, whereas qualitative methods use interpretive and descriptive techniques.
- Purpose: Quantitative research aims for generalizability and establishing relationships, while qualitative research aims for understanding and exploring phenomena in-depth.

The research employs a **mixed-method simple approach**, combining both quantitative and qualitative methodologies to provide a comprehensive analysis of phishing website detection. This choice is appropriate for capturing the complexity of the research problem by integrating numerical data analysis with detailed insights from expert interviews.

Quantitative Methods

- Data Collection and Analysis: The quantitative component focuses on collecting and analyzing numerical data from an existing dataset as the one I will use [2]. This dataset includes features that are known to be indicative of phishing websites, such as URL patterns, domain age, and HTTPS usage.
- Machine Learning Algorithms: Implementing various machine learning algorithms, including logistic regression, decision trees, random forests, and support vector machines. These algorithms are used to classify websites as either legitimate or phishing based on their features.
- Model Training and Evaluation: Training the machine learning models on the dataset and evaluating their performance using metrics such as accuracy, precision, recall, and F1 score. Cross-validation techniques, such as k-fold cross-validation, are applied to ensure the models' robustness and generalizability.

Qualitative Methods

- Interviews: Conducting semi-structured interviews with cybersecurity experts such as interview with Dr.Safaa Hriez to gain deeper insights into the current challenges and practices in phishing detection. The interviews will explore the experts' experiences, opinions, and suggestions regarding effective strategies and tools for detecting phishing websites.

- The qualitative data from the interviews will be analyzed using thematic analysis to identify common themes and patterns. This analysis will provide context and support for the quantitative findings, highlighting practical considerations and expert perspectives that may not be evident from the numerical data alone.

3.1.4 Research Strategy

The research strategy combines both **experimental** and **narrative inquiry** strategies to provide a thorough analysis of phishing website detection.

- **Experimental**: Conducting controlled experiments to test different machine learning algorithms on the dataset. This involves training models, tuning hyperparameters, and assessing their performance using metrics such as accuracy, precision, recall, and F1 score.
- **Narrative inquiry:** Analyzing the qualitative data from the interviews using narrative inquiry.
  - Storytelling involves gathering detailed facts from participants about their experiences and thoughts on phishing detection.
  - Thematic analysis involves identifying common themes and patterns in facts in order to provide context and support for quantitative findings.
  - Findings integration is combining narrative insights with quantitative data analysis to build a thorough picture of phishing site detection.

3.1.5 Time Horizons

A **cross-sectional** study design is employed, where data is collected and analyzed. The research is independent of time and duration factors that could influence its outcomes. This approach is suitable for evaluating the effectiveness of different phishing detection techniques using the current dataset. It provides a snapshot view of the models' performance under existing conditions, offering insights into their immediate effectiveness in detecting phishing websites without any effects when the research and analysis are done.

3.1.6 Techniques and Procedures

Data Collection

Primary Data

Description of Primary Data

Interviews: Conducting semi-structured interviews with cybersecurity experts to gather qualitative primary data. These interviews aim to capture expert insights, opinions, and experiences related to phishing detection strategies, tools, and challenges. The primary data obtained from interviews will be analyzed using thematic analysis to identify key themes and patterns that contribute to understanding effective phishing detection methodologies.

Experience in Collecting Primary Data

Collecting primary data through interviews involved several steps:

- Participant Selection: Identifying and recruiting cybersecurity professionals with real-world experience in phishing detection.
- Interview Design: Develop a semi-structured interview guide to ensure consistency. Starting with generating comprehensive questions that cover all the important aspects of phishing websites. Then preparing and scheduling interviews with cybersecurity professionals.
- Data Collection: Conducting interviews in person, recording responses, and taking detailed notes.
- Data Analysis: Transcribing interview recordings, performing thematic analysis to identify recurring patterns and themes, and organizing findings.

Issues Faced in Collecting Primary Data

During the primary data collection process, I faced many challenges included:

- Participant Availability: Scheduling interviews with busy professionals within their available time slots.
- Ensure that interview questions produce comprehensive responses while keeping relevant to the research objectives.

- Ensuring that enough interviews were conducted to reach data, where no new significant information emerged from subsequent interviews.

Justification for Choosing Primary Data

Primary data in the form of qualitative interviews was chosen for several reasons:

- Depth of Insights: Interviews provide rich, detailed insights into complex topics like phishing detection, offering comprehensive perspectives that quantitative data alone may not capture.
- Contextual Understanding: Expert opinions and experiences provide contextual understanding of real-world challenges and practical solutions.
- Thematic analysis ensures systematic interpretation and validation of findings, enhancing the credibility of research outcomes.

Pros and Cons of Primary Data

- **Pros:**
  - Provides firsthand, in-depth insights into specific events.
  - Allows exploration of complex issues and contextual understanding.
  - Enhances research validity through direct engagement with participants and experts.
  - Learning and gaining vital information from experts' experience in the field.
  - The data is unique, and no one else will have access to it. Also, the collected information is up to date.
- **Cons:**
  - Needs more time and resources compared to secondary data collection.
  - Researchers may be biased in their interpretation and analysis.

– Analyzing primary data can be difficult and expensive, especially when using qualitative methods such as thematic analysis or grounded theory. Researchers may require specialized training or knowledge to correctly analyze and understand data, increasing overall research costs. Also, because it is primary data, you will be the first one to collect and analyze it.

Interview

In this section, I will present the questions asked in the interview, participations, interview records, and analysis of records as shown in Figure 3.

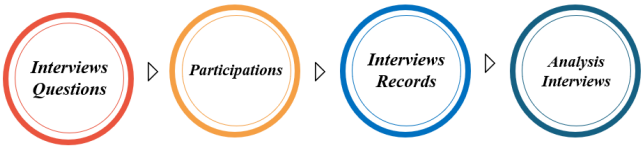


Figure 3: Flowchart of the interview analysis methodology

Interviews Questions

In this section, I will present the questions that I asked for cybersecurity specialist to further understanding phishing websites and techniques to detect phishing websites.

1. Can you describe some of the most common phishing techniques you've encountered in your experience?
2. What are the key indicators or red flags that cybersecurity professionals should look for to identify a potential phishing attempt?
3. In your opinion, what are the biggest challenges or obstacles in effectively detecting phishing attacks?
4. Could you share examples of successful strategies or tools you used before to detect and prevent phishing attacks?
5. How do you manage false positives in phishing detection?
6. How important is user awareness and education in combating phishing attacks?
7. How do you see the role of artificial intelligence (AI) and machine learning (ML) evolving in the detection and prevention of phishing attacks?

Participations

I conducted the interview with Dr. Safaa Hriez, an assistant professor at Al-Hussein Technical University. Her major is in cybersecurity, and she has extensive knowledge in phishing detection.

Interviews Records

You can find the interview record in this link:

Analysis Interviews

The answers of the questions:

1. The most common phishing techniques are whaling phishing, spear phishing, email-phishing, and SMS phishing.
2. If we take the email as an example, then we have to check the email sender and how you are familiar with this sender. Also, you can notice the content of the message if it is talking in the urgent tone. Also, if it requests sensitive data or has suspicious links.
3. It is professionally and dedicated targeting. Also, there are tools that can make the urls very similar to the targeted urls.
4. Email spam detection tools are successful tools in detecting the phishing attacks.
5. Using the era of AI with its effective models to detect the phishing and feed these models with huge amount of data since it is available so the accuracy of the models will be better.
6. The user awareness is the most important part in the phishing attacks because this attack relays on the lack of the awareness of the target.
7. As I said before, the AI and ML techniques will play a significant role in detecting this attack and to take action to prevent it.

Based on the interview, there are several important keys that are related to detect phishing attacks. Phishing techniques like whaling, spear phishing, email phishing, and SMS phishing are commonly used, each targeting individuals or organizations in different ways. When examining an email, it's crucial to check the sender's identity and familiarity, and to be aware of urgent language, requests for sensitive information, or suspicious links. Phishing attacks are often advanced and more structured, using tools to create URLs that are similar to legitimate ones. Effective tools such as email spam detection systems are successful in identifying these phishing attacks. Moreover, user awareness is vital part in detecting phishing attacks, as attackers exploit gaps in knowledge and awareness. At the end,

AI and ML technologies will continue to evolve and take a big part in both detecting and preventing phishing attacks.

Secondary Data

Description of Secondary Data

I utilize secondary data sources such as the dataset from M. Mohammad et al. (2015), which includes features indicative of phishing websites. Secondary data also includes relevant literature and research studies that provide insights into current trends, methodologies, advancements, and procedures in phishing detection. This data will be used to study, validate, and develop models that are effective in detecting phishing websites.

Pros and Cons of Secondary Data

- **Pros:**
  - Offers a broad overview of existing knowledge and research findings.
  - Provides historical and comparative data for analysis.
  - Cost-effective and time-efficient compared to primary data collection.
- **Cons:**
  - Potential for outdated or incomplete information.
  - May lack specific details, information, or context provided by primary data.
  - The quality and reliability of secondary sources can change.

Dataset Overview

The dataset contains a collection of both legitimate and phishing websites. It has 18 features, each representing a different part of the website URL, and there are 5000 zero labels (legitimate websites) and 5000 one labels (phishing websites). These features help to figure out if a website is fake or real. The last dataset feature is the class feature which is a label indicating whether each website is classified as legitimate, or phishing based on all the other features. As a result, this dataset serves as a critical dataset for training and evaluating machine learning models or statistical algorithms designed to determine between legitimate and phishing websites automatically [2].

Table 2: **Explaining Dataset Features**

Feature Name	***Description***
Domain	• The domain name of the website.
Have_IP	• Indicates whether the URL contains an IP address. • So, if an IP address is used instead of a domain name in the URL, it's classified as a phishing website.
Have_At	• Indicates whether the URL contains the "@" symbol. • Using "@" in the URL may indicate phishing attempts.
URL_Length	• This refers to the total number of characters in the URL of a webpage. • Phishers use long URLs to hide suspicious parts, but if the URL is too long (54 characters or more), it's classified as phishing.
URL_Depth	• This refers to the hierarchical level or depth of a specific webpage within the website's structure. (Measures how many subdirectories or levels deep a webpage is from the root domain).
Redirection	• Indicates whether the URL involves redirection to another website, often located on a different domain or server.
https_Domain	• Indicates whether the domain uses HTTPS.

Continued on next page

Table 2: continued from previous page

Feature Name	***Description***
TinyURL	• Indicates whether the URL is a TinyURL. • "TinyURL" is a URL shortening service that converts a long URL into a shorter, more manageable one.
Prefix/Suffix	• Indicates the presence of prefixes or suffixes in the URL. • "Prefix/Suffix" refers to additional strings or segments that come before or after the main domain name in the URL. • Phishers may add prefixes or suffixes separated by "-" to make the URL appear legitimate.
DNS_Record	• Indicates whether the domain has DNS records.
Web_Traffic	• "Web traffic" refers to the volume of data transmitted between a website and its visitors over a set period of time. It is a measure of the number of visitors to a website and how they engage with its content.
Domain_Age	• The age of the domain.
Domain_End	• Indicates the expiration date of the domain.

Continued on next page

Table 2: continued from previous page

Feature Name	***Description***
iFrame	• Indicates whether the website uses iFrames. • "iFrame" is a tool used in web development to embed content from one webpage into another. It's like a small window within a webpage that displays external content, such as videos, maps, or social media feeds.
Mouse_Over	• Indicates whether mouse-over events are present on the website. • "Mouse over" refers to an event that occurs when a user moves their mouse cursor over an element on a webpage.
Right_Click	• Indicates whether right click is enabled on the website. • "Right click" refers to the action of clicking the right mouse button while the cursor is positioned over an element on a webpage.
Web_Forwards	• Indicates whether the website uses web forwards. • "Web forwarding" refers to the server-side process of passing a request from one URL to another within the same application or server.
Label	• The label indicating whether the website is classified as phishing or not.

Concluded

Table 3: **Rule for Each Feature**

Feature Name	***Phishing (1)***	***Legitimate (0)***
Have_IP	1	0
URL's having "@" Symbol	1 (@ symbol present)	0
Long URL to Hide the Suspicious Part	1 (length >= 54)	0 (length < 54)
Redirecting using "//"	1 (position > 7)	0 (position <= 7)
Redirection	1 (Number of redirects >= 4)	0 (Number of redirects <= 1)
Using URL Shortening Services "TinyURL"	1 (TinyURL)	0
HTTPS (Hyper Text Transfer Protocol)	0 (trusted issuer & age >= 1 year)	1 (untrusted issuer or age < 1 year)
Adding Prefix or Suffix Separated by (-)	1 (dash symbol present)	0
DNS Record	1 (no DNS record)	0 (DNS record exists)
Website Traffic	1 (website rank > 100,000 or no traffic)	0 (website rank < 100,000)
Domain End	1 (Ends within 1 year)	0 (Ends after 1 year)
Age of Domain	1 (age < 6 months)	0 (age >= 6 months)
IFrame Redirection	1 (uses iframe)	0
Mouse Over	1 (Changes status bar)	0 (Doesn't change status bar)
Disabling Right Click	1 (right-click disabled)	0
Website Forwarding	1 (# of redirects >= 4)	0 (# of redirects <= 1)

Analysis

To enhance the results of the model, I suggest applying methods and techniques to improve the dataset results. Firstly, we can enhance the dataset by incorporating additional features. This entails adding extra layers of detail to provide our model with a more comprehensive understanding of phishing URLs.

Furthermore, we can conduct an in-depth analysis of the dataset to uncover hidden patterns or trends. For instance, exploring correlations between different features allows us to monitor the evolution of phishing websites. Such detailed examination provides valuable insights that can be utilized to refine our detection methods and proactively counter potential threats

Lastly, leveraging the concept of feature engineering allows us to creatively extract more useful information from the data. This involves devising innovative approaches to measuring URL similarity and identifying subtle indicators of phishing attempts. Through careful manipulation of features, we can enhance the intelligence and accuracy of our model.

This phase involves employing advanced techniques to extract additional valuable insights from the dataset. This may include developing unique methods for measuring URL similarity and identifying sophisticated phishing detection strategies. We aim to enhance the complexity and accuracy of our model by skillfully manipulating the features within the dataset.

The methodology of this research consists of several steps illustrated in figure 4. The following subsections discuss each step-in detail.

Figure 4: Flowchart of the data analysis methodology

Exploratory Data Analysis (EDA)

In this section, I perform a detailed exploration of the dataset to gain insights and prepare it for further analysis. The process involves six main steps as shown in Figure 5.

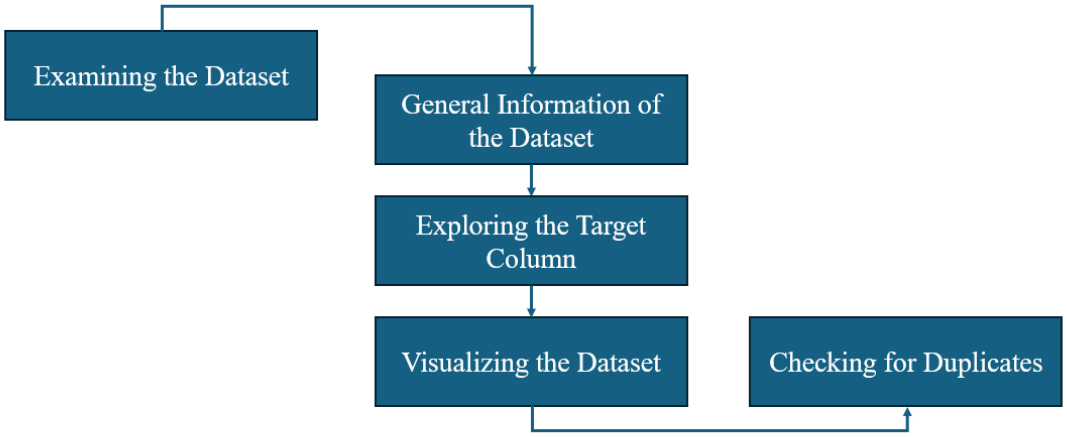


Figure 5: Dataset Exploration Flowchart

- Viewing the first 7 rows in the data (the head of the data) as shown in Figure 6.

```
data_frame.head(7)
```

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
0	graphicriver.net	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0
1	ecnavi.jp	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	0
2	hubpages.com	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0
3	extratorrent.cc	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
4	icicibank.com	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
5	nypost.com	0	0	1	4	0	0	1	0	0	1	1	1	0	0	1	0	0
6	kienthuc.net.vn	0	0	1	2	0	0	0	0	1	1	1	1	0	0	1	0	0

Figure 6: Dataset head

- Viewing the last 7 rows in the data (the tail of the data) as shown in Figure 7.

```
data_frame.tail(7)
```

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards
9993	sparkpassedecom.com	0	0	0	2	0	0	0	0	0	1	1	1	0	0	1	0
9994	info.lionnets.com	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0
9995	wvk12-my.sharepoint.com	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	0
9996	adplife.com	0	0	1	4	0	0	0	0	0	1	0	1	0	0	1	0
9997	kurortnoye.com.ua	0	1	1	3	0	0	1	0	0	0	1	1	1	0	1	0
9998	norcaltc-my.sharepoint.com	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	0
9999	sieck-kuehlsysteme.de	0	1	1	4	0	0	1	1	0	1	1	1	0	0	1	0

Figure 7: Dataset tail

- Viewing general information about the dataset to understand the data types of each feature and general statistics related to the data as shown in Figures 8 and 9.

```
# To check features data types and whether they have null values or not. The dataset is all numrical type and has no null values
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 17 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Have_IP         10000 non-null  int64
 1   Have_At         10000 non-null  int64
 2   URL_Length      10000 non-null  int64
 3   URL_Depth       10000 non-null  int64
 4   Redirection     10000 non-null  int64
 5   https_Domain    10000 non-null  int64
 6   TinyURL         10000 non-null  int64
 7   Prefix/Suffix   10000 non-null  int64
 8   DNS_Record      10000 non-null  int64
 9   Web_Traffic     10000 non-null  int64
 10  Domain_Age      10000 non-null  int64
 11  Domain_End      10000 non-null  int64
 12  iFrame          10000 non-null  int64
 13  Mouse_Over      10000 non-null  int64
 14  Right_Click     10000 non-null  int64
 15  Web_Forwards    10000 non-null  int64
 16  Label           10000 non-null  int64
dtypes: int64(17)
memory usage: 1.3 MB
```

Figure 8: Dataset Information

```
# Displays basic statistics for numeric columns such as count, mean, and other important things
data_frame.describe()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000	10000.000000	10000.00000	10000.00000	10000.000000	10000.000000
mean	0.005500	0.022600	0.773400	3.072000	0.013500	0.000200	0.090300	0.093200	0.100800	0.845700	0.413700	0.8099	0.090900	0.06660	0.99930	0.105300	0.500000
std	0.073961	0.148632	0.418653	2.128631	0.115408	0.014141	0.286625	0.290727	0.301079	0.361254	0.492521	0.3924	0.287481	0.24934	0.02645	0.306955	0.500025
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000	0.000000	0.00000	0.00000	0.000000	0.000000
25%	0.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.0000	0.000000	0.00000	1.00000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.0000	0.000000	0.00000	1.00000	0.000000	0.500000
75%	0.000000	0.000000	1.000000	4.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.0000	0.000000	0.00000	1.00000	0.000000	1.000000
max	1.000000	1.000000	1.000000	20.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000	1.000000	1.00000	1.00000	1.000000	1.000000

Figure 9: Dataset statistics

- Viewing the total observations in the target feature (the label feature) as shown in Figure 10.

```
data_frame.value_counts("Label")
```

```
Label
0    5000
1    5000
Name: count, dtype: int64
```

Figure 10: Label data

- Visualizes the distribution of the data frame's columns by displaying the outliers as shown in Figure 11.

```
#It visualizes the distribution of the data_frame's columns by displaying their quartiles (25th, median, and 75th percentiles) along with any outliers.
plt.figure(figsize=(12, 6))
sn.boxplot(data=data_frame)
plt.title("Boxplot of Data")
plt.xticks(rotation=45)
plt.show()
```

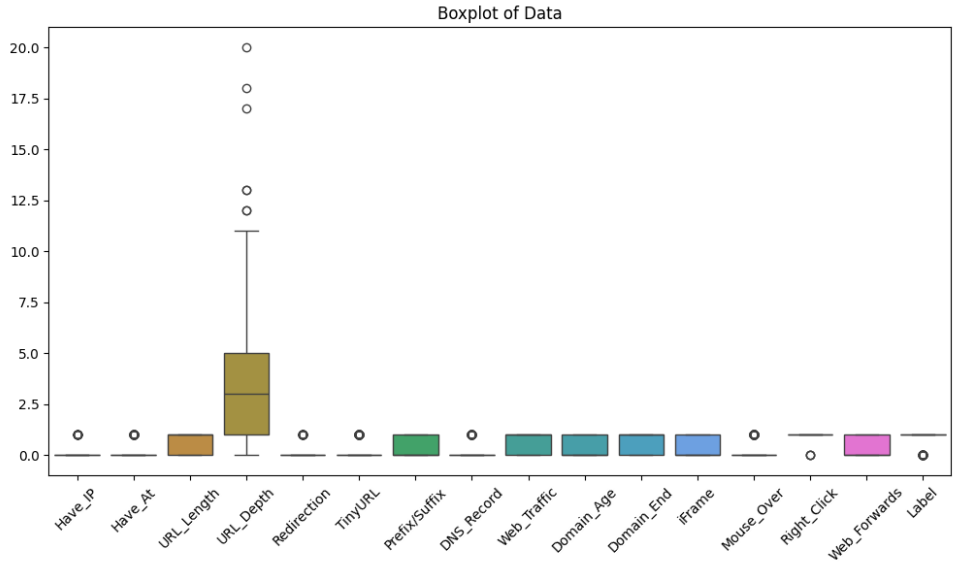


Figure 11: The outliers in the dataset

- Finally, viewing the number of duplicated rows in the dataset as shown in Figure 12.

```
# Checking whether the dataset's contains duplicated data or not. The dataset has too many instances that are duplicated
data_frame.duplicated().sum()
```

```
9120
```

Figure 12: Duplicated rows

Preprocessing

In this section, I focus on preparing the dataset for analysis by conducting preprocessing steps for further analysis. The process involves eight main steps, as shown in Figure 13.

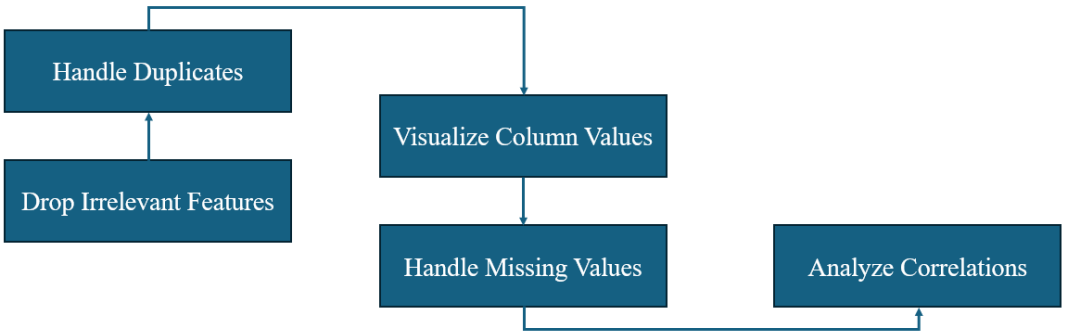


Figure 13: Preprocessing Flow Chart

- Firstly, since each domain name must be unique to distinguish between legitimate and phishing domains, I will drop the "Domain" feature to enhance modeling results, as shown in Figure 14.

```
"""
Since the domain is unique (each website will have a different domain),
it will be better to drop it from the dataset for better insights when we create the model.
"""
data_frame = data_frame.drop(["Domain"] , axis = 1)

# Print the dataframe after dropping "Domain" feature
data_frame
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0
1	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	0
2	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0
3	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
4	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	0	1
9996	0	0	1	4	0	0	0	0	0	1	0	1	0	0	1	0	1
9997	0	1	1	3	0	0	1	0	0	0	1	1	1	0	1	0	1
9998	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	0	1
9999	0	1	1	4	0	0	1	1	0	1	1	1	0	0	1	0	1

10000 rows × 17 columns

Figure 14: Dropping the "Domain" column

- Next, create a variable to store the duplicated data and view it as shown in Figure 15.

```
data_frame_dup = data_frame[data_frame.duplicated(keep=False)] # Create a variable to hold the data that are duplicated
data_frame_dup = data_frame_dup.reset_index(drop = True)       # Reset the indexes of the dataset for arranging matters
data_frame_dup                                                 # Print the new dataset that contains only the duplicated instances
                                                               # We conclude that 96% of the data is duplicated
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	0
1	0	0	1	1	1	0	0	0	0	1	1	1	0	0	1	0	0
2	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0
3	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
4	0	0	1	3	0	0	0	0	0	1	0	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9658	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1
9659	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	0	1
9660	0	0	1	4	0	0	0	0	0	1	0	1	0	0	1	0	1
9661	0	0	1	5	0	0	1	1	0	1	1	1	0	0	1	0	1
9662	0	1	1	4	0	0	1	1	0	1	1	1	0	0	1	0	1

9663 rows × 17 columns

Figure 15: Dropping the "Domain" column

- Drop all the duplicated rows as shown in Figure 16.

```
data_frame = data_frame.drop_duplicates(keep=False) # Dropping all the duplicated row's
data_frame = data_frame.reset_index(drop = True)    # Reset the indexes of the dataset for arranging the dataset again
data_frame                                          # Print the dataframe
# We conclude that we only have 337 that are unique (each has a special case)
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
0	0	1	1	8	0	0	0	0	0	1	1	1	0	0	1	0	0
1	0	0	1	4	0	0	0	0	1	1	1	1	0	0	1	1	0
2	0	0	1	5	0	0	0	0	0	1	0	1	1	0	1	0	0
3	0	0	1	5	0	0	0	0	0	0	1	1	0	0	1	0	0
4	0	0	1	2	0	0	1	0	1	1	1	1	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
332	0	1	1	4	0	0	0	0	0	0	1	1	1	1	1	1	1
333	0	0	0	2	0	0	0	1	1	1	1	1	0	0	1	0	1
334	0	0	1	2	0	0	0	0	1	0	1	1	1	1	1	1	1
335	0	0	1	1	1	0	0	1	0	1	0	1	0	0	1	0	1
336	0	1	1	3	0	0	1	0	0	0	1	1	1	0	1	0	1

337 rows × 17 columns

```
data_frame.duplicated().sum()
```

```
0
```

Figure 16: Dropping the "Domain" column

- Ensure that each column has unique values for better modeling insights by visualizing each column's values in bar charts as shown in Figures 17, 18, 19, and 20.

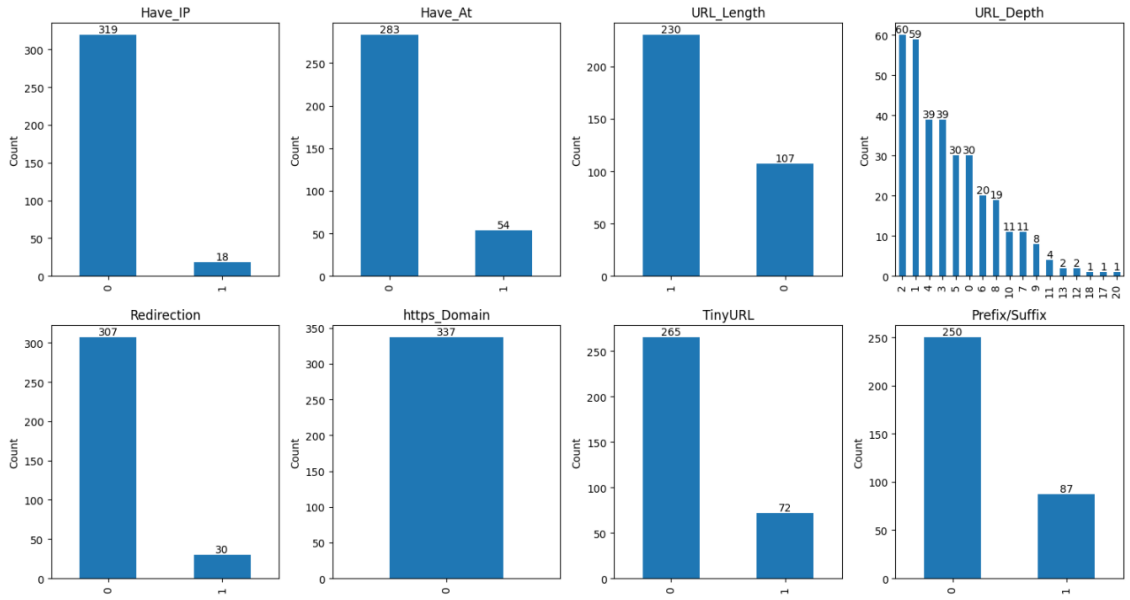


Figure 17: Columns values

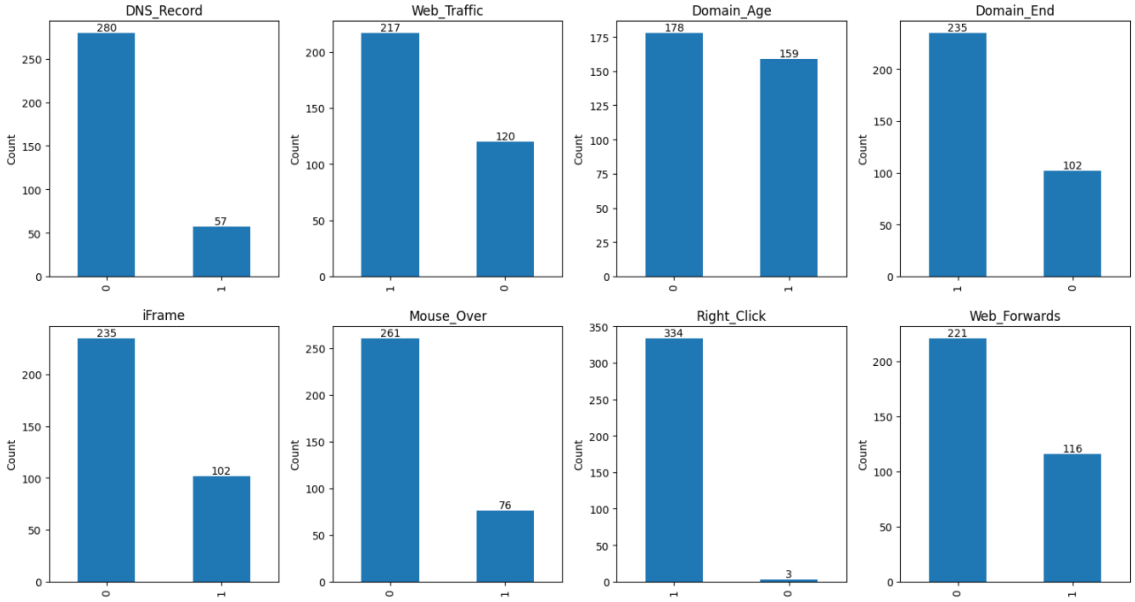


Figure 18: Columns values

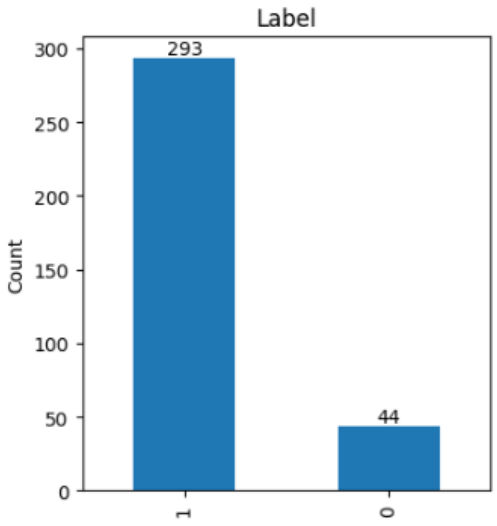


Figure 19: Columns values

```
for x in data_frame.columns:
    print(f"Unique Values In '{x}': {data_frame[x].nunique()}")
# Printing the unique values of each column
# Concluding that the feature "https_Domain" has only one unique value, which means that the variance is 0
# The "https_Domain" feature has no weight since all the instances have the same value. Thus, we will remove it from the dataset
```

```
Unique Values In 'Have_IP': 2
Unique Values In 'Have_At': 2
Unique Values In 'URL_Length': 2
Unique Values In 'URL_Depth': 17
Unique Values In 'Redirection': 2
Unique Values In 'https_Domain': 1
Unique Values In 'TinyURL': 2
Unique Values In 'Prefix/Suffix': 2
Unique Values In 'DNS_Record': 2
Unique Values In 'Web_Traffic': 2
Unique Values In 'Domain_Age': 2
Unique Values In 'Domain_End': 2
Unique Values In 'iFrame': 2
Unique Values In 'Mouse_Over': 2
Unique Values In 'Right_Click': 2
Unique Values In 'Web_Forwards': 2
Unique Values In 'Label': 2
```

Figure 20: Columns values

- Based on the insights gained, two actions are taken: firstly, drop the `https_Domain` column due to having only one value; secondly, prepare for model balancing using algorithms. Figure 21 shows the dropping `https_Domain` column.

```
sn.countplot(x = 'https_Domain' , data = data_frame)
data_frame = data_frame.drop(['https_Domain'] , axis = 1)
# Visualizaing the feature
```

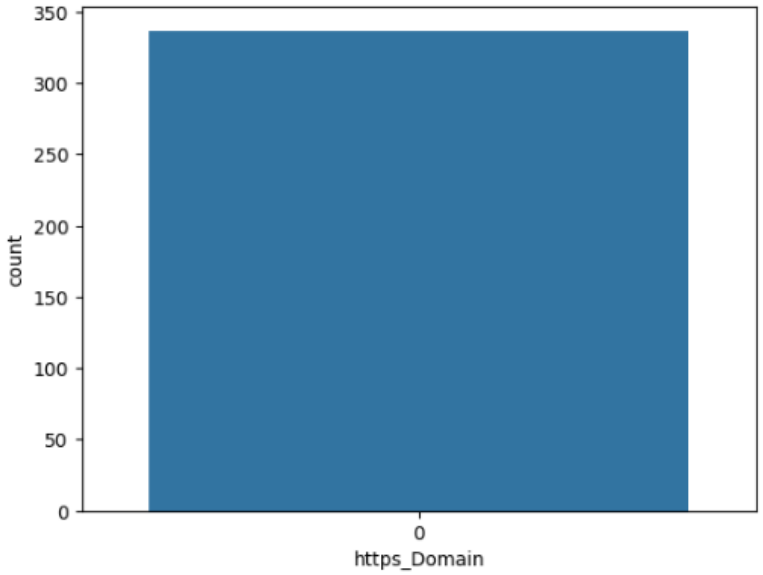


Figure 21: Columns values

- Checking for missing values in each row to fill them, but my dataset has no missing values as shown in Figure 22.

```
# Check missing values in each row
missing_in_rows = data_frame.isnull().sum(axis=1)
print("\nMissing values in each row:")
print(missing_in_rows)
```

```
Missing values in each row:
0      0
1      0
2      0
3      0
4      0
      ..
332    0
333    0
334    0
335    0
336    0
Length: 337, dtype: int64
```

Figure 22: Checking missing values in each row

- Checking for missing values in each column to fill them, but my dataset has no missing values as shown in Figure 23.

```
# Check missing values in each column
missing_in_columns = data_frame.isnull().sum()
print("Missing values in each column:")
print(missing_in_columns)
```

```
Missing values in each column:
Have_IP          0
Have_At          0
URL_Length       0
URL_Depth        0
Redirection      0
TinyURL          0
Prefix/Suffix    0
DNS_Record       0
Web_Traffic      0
Domain_Age       0
Domain_End       0
iFrame           0
Mouse_Over       0
Right_Click      0
Web_Forwards     0
Label            0
dtype: int64
```

Figure 23: Checking missing values in columns

- After cleaning and processing the data, examine the correlation between features using correlation matrices and correlation heatmap as shown in Figures 24 and 25.

```
# Calculate the correlation matrix and put in varaible called corr_matrix
corr_matrix = data_frame.corr()
data_frame.corr('pearson')
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label
Have_IP	1.000000	-0.103763	-0.121473	-0.100322	-0.074256	-0.123818	-0.140130	-0.071973	0.121525	-0.039457	0.070322	-0.041596	0.029703	0.022513	-0.033216	0.092052
Have_At	-0.103763	1.000000	0.176300	-0.003579	-0.051333	0.028871	-0.035872	-0.089196	0.054540	-0.056355	-0.099586	-0.146923	-0.138937	0.041399	-0.214316	0.145267
URL_Length	-0.121473	0.176300	1.000000	0.405979	0.056525	-0.033273	-0.034622	-0.015339	0.118468	0.057253	0.022398	-0.077899	-0.104778	0.138950	-0.123021	-0.264314
URL_Depth	-0.100322	-0.003579	0.405979	1.000000	-0.058661	-0.112206	-0.166302	-0.065982	0.110674	-0.051439	-0.018008	-0.160545	-0.172275	0.082547	-0.206179	-0.160353
Redirection	-0.074256	-0.051333	0.056525	-0.058661	1.000000	-0.086676	-0.089166	-0.141042	0.058374	-0.024094	0.047180	-0.092542	-0.068954	0.029626	-0.094886	-0.064424
TinyURL	-0.123818	0.028871	-0.033273	-0.112206	-0.086676	1.000000	-0.075891	0.073809	0.070121	-0.028574	-0.034790	-0.170067	-0.160016	0.049400	-0.133837	0.008608
Prefix/Suffix	-0.140130	-0.035872	-0.034622	-0.166302	-0.089166	-0.075891	1.000000	0.095591	-0.014455	0.012938	0.004905	0.024613	0.071061	0.055908	-0.056324	0.148102
DNS_Record	-0.071973	-0.089196	-0.015339	-0.065982	-0.141042	0.073809	0.095591	1.000000	0.087559	0.477386	0.297252	0.064573	0.040634	0.042761	0.006328	0.033881
Web_Traffic	0.121525	0.054540	0.118468	0.110674	0.058374	0.070121	-0.014455	0.087559	1.000000	0.094558	0.103588	-0.022655	0.000924	0.061472	0.017030	-0.122637
Domain_Age	-0.039457	-0.056355	0.057253	-0.051439	-0.024094	-0.028574	0.012938	0.477386	0.094558	1.000000	0.454464	0.076019	0.073141	0.089573	0.103467	-0.092453
Domain_End	0.070322	-0.099586	0.022398	-0.018008	0.047180	-0.034790	0.004905	0.297252	0.103588	0.454464	1.000000	0.068502	0.108231	0.075090	0.069466	-0.101940
iFrame	-0.041596	-0.146923	-0.077899	-0.160545	-0.092542	-0.170067	0.024613	0.064573	-0.022655	0.076019	0.068502	1.000000	0.664519	0.062439	0.447134	0.178622
Mouse_Over	0.029703	-0.138937	-0.104778	-0.172275	-0.068954	-0.160016	0.071061	0.040634	0.000924	0.073141	0.108231	0.664519	1.000000	0.051142	0.610325	0.166964
Right_Click	0.022513	0.041399	0.138950	0.082547	0.029626	0.049400	0.055908	0.042761	0.061472	0.089573	0.075090	0.062439	0.051142	1.000000	0.068663	-0.036727
Web_Forwards	-0.033216	-0.214316	-0.123021	-0.206179	-0.094886	-0.133837	-0.056324	0.006328	0.017030	0.103467	0.069466	0.447134	0.610325	0.068663	1.000000	0.021233
Label	0.092052	0.145267	-0.264314	-0.160353	-0.064424	0.008608	0.148102	0.033881	-0.122637	-0.092453	-0.101940	0.178622	0.166964	-0.036727	0.021233	1.000000

Figure 24: Correlation matrix

```
# Plotting the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sn.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

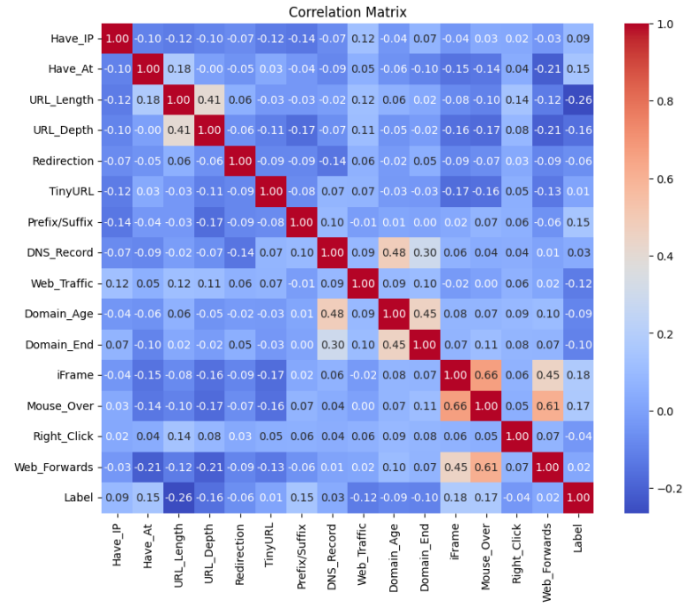


Figure 25: Correlation heatmap

- Finally, calculate the Spearman rank correlation matrix for the dataset and print correlations above 60% to identify significant relationships between variables, as shown in Figure 26.

```
corr = data_frame.corr(method='spearman')
print("Spearman Rank Correlation Matrix (above 60%):")
for i in range(len(corr)):
    for j in range(i+1, len(corr)):
        correlation_coefficient = corr.iloc[i, j]
        if correlation_coefficient > 0.6:
            print(f"Correlation between {corr.columns[i]} and {corr.columns[j]}: {correlation_coefficient}")
```

```
Spearman Rank Correlation Matrix (above 60%):
Correlation between iFrame and Mouse_Over: 0.6645188711541609
Correlation between Mouse_Over and Web_Forwards: 0.610324675107632
```

Figure 26: Correlation Spearman

Feature Selection

In this section, the focus is on identifying and selecting the most relevant features from the dataset to improve the effectiveness of the analysis and modeling.

- Utilizing SelectKBest with a chi-squared test to select and rank the top features based on their importance scores, improving feature selection for analysis and modeling, as shown in Figure 27.

```python
# Assuming 'label' is the name of your target column
X = data_frame.drop('Label', axis=1)
y = data_frame['Label']

# Perform feature selection using SelectKBest with chi-squared test
k = 10  # Number of top features to select
selector = SelectKBest(score_func=chi2, k=k)
X_selected = selector.fit_transform(X, y)

# Get the selected feature indices
selected_feature_indices = selector.get_support(indices=True)

# Get the names of the selected features
selected_feature_names = X.columns[selected_feature_indices]

# Get the importance scores of the selected features
feature_scores = selector.scores_[selected_feature_indices]

# Sort features based on their scores
sorted_indices = feature_scores.argsort()[::-1]

print("Selected Features and their Importance Rank:")
for rank, idx in enumerate(sorted_indices):
    print(f"{rank + 1}. {selected_feature_names[idx]} (Score: {feature_scores[idx]})")

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_indices)), feature_scores[sorted_indices], color = "purple", align='center')
plt.yticks(range(len(sorted_indices)), [selected_feature_names[idx] for idx in sorted_indices])
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Feature Importance Rank')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest score on top
plt.show()
```

```
Selected Features and their Importance Rank:
1. URL_Depth (Score: 23.610314756989897)
2. iFrame (Score: 7.497885905836117)
3. URL_Length (Score: 7.475233714200916)
4. Mouse_Over (Score: 7.275932034554275)
5. Have_At (Score: 5.972008193424576)
6. Prefix/Suffix (Score: 5.483575308219299)
7. URL_Complexity (Score: 2.9459992559267256)
8. Have_IP (Score: 2.703071672350486)
9. Web_Traffic (Score: 1.8047876652687838)
10. Domain_Age (Score: 1.521472045459423)
```

Figure 27: SelectKBest results

- Visualize SelectKBest results in a bar chart as shown in Figure 28.

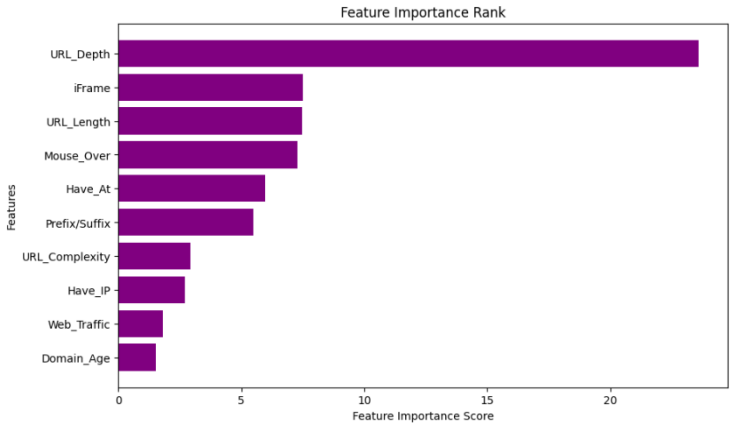


Figure 28: Visualizing SelectKBest

- Based on the feature selection, I conducted further analysis on URL_Depth to further understand this feature as shown in Figure 29.

```
plt.figure(figsize=(12, 6))
sn.countplot(x = 'URL_Depth', hue = 'Label', data = data_frame)
plt.show()
# According to the data, We conclude that when a url depth is 7, 11, 12, 13, 20 ... the website is fake (phishing website)
# When a url depth is 17 and 18, the website is safe (legimate website)
```

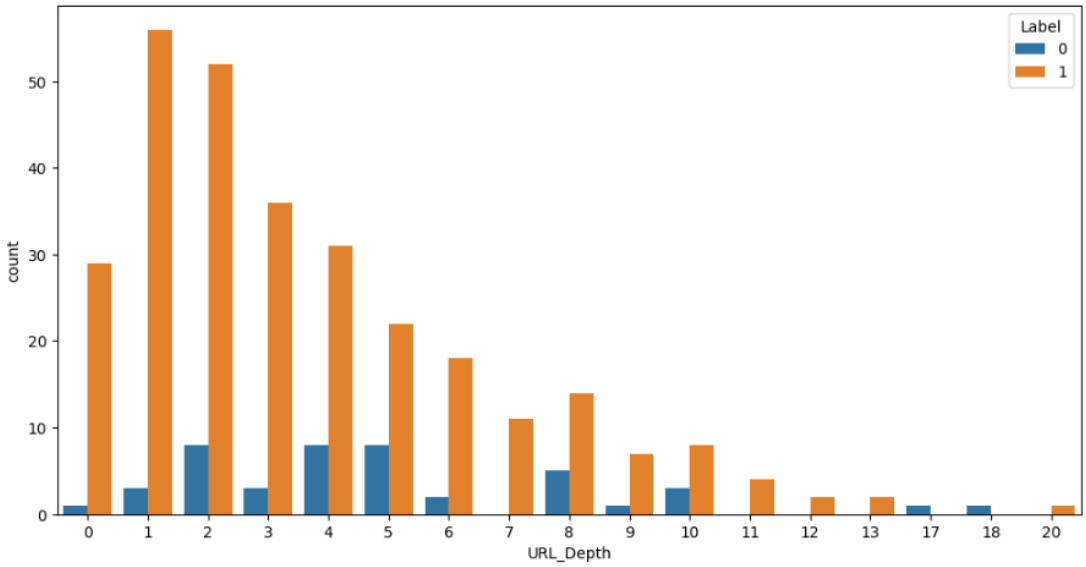


Figure 29: Visualizing URL_Depth

Based on the feature selection results, we observe large differences in feature rankings, with only a small number considered important compared to the total set of features (18 in total). Therefore, I will retain all features for modeling without dropping any additional ones.

Feature Engineering

In this section, I focus on enhancing the predictive insights and performance of the models through strategic feature engineering.
I have created a new feature for the dataset by combining the features URL_Depth and URL_Length to improve the results in the models. I chose these two features because they are related to each other and can be used to create an equation that combines them. Firstly, I extracted the maximum value of each feature, then normalized them, and finally multiplied them together. This approach was necessary because the URL_Length values are either 0 or 1, which would cause an error if I simply tried to divide URL_Depth by URL_Length. View Figures 30, 31, and 32.

```
"""
This code calculates the maximum depth and length of URLs in the dataset, normalizes these features,
combines them to create a new feature representing URL complexity,
"""
# Normalize URL depth and URL length
max_depth = data_frame['URL_Depth'].max()
max_length = data_frame['URL_Length'].max()

data_frame['Normalized_Depth'] = data_frame['URL_Depth'] / max_depth

# Combine the normalized features to create a new feature representing complexity
data_frame['URL_Complexity'] = data_frame['Normalized_Depth'] * data_frame['URL_Length']
```

Figure 30: Feature Engineering Code

```
data_frame.head()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label	Normalized_Depth	URL_Complexity
0	0	1	1	8	0	0	0	0	1	1	1	0	0	1	0	0	0.40	0.40
1	0	0	1	4	0	0	0	1	1	1	1	0	0	1	1	0	0.20	0.20
2	0	0	1	5	0	0	0	0	1	0	1	1	0	1	0	0	0.25	0.25
3	0	0	1	5	0	0	0	0	0	1	1	0	0	1	0	0	0.25	0.25
4	0	0	1	2	0	1	0	1	1	1	1	0	0	1	0	0	0.10	0.10

Figure 31: Dataset head after adding the new feature

```
data_frame.tail()
```

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards	Label	Normalized_Depth	URL_Complexity
332	0	1	1	4	0	0	0	0	0	1	1	1	1	1	1	1	0.20	0.20
333	0	0	0	2	0	0	1	1	1	1	1	0	0	1	0	1	0.10	0.00
334	0	0	1	2	0	0	0	1	0	1	1	1	1	1	1	1	0.10	0.10
335	0	0	1	1	1	0	1	0	1	0	1	0	0	1	0	1	0.05	0.05
336	0	1	1	3	0	1	0	0	0	1	1	1	0	1	0	1	0.15	0.15

Figure 32: Dataset tail after adding the new feature

Modeling

In this section, the focus shifts to implementing and evaluating six distinct models to analyze and predict outcomes based on the dataset. Each model brings its unique approach and strengths, aiming to uncover patterns, make predictions, and optimize performance across various metrics. The models are shown in Figure 33.

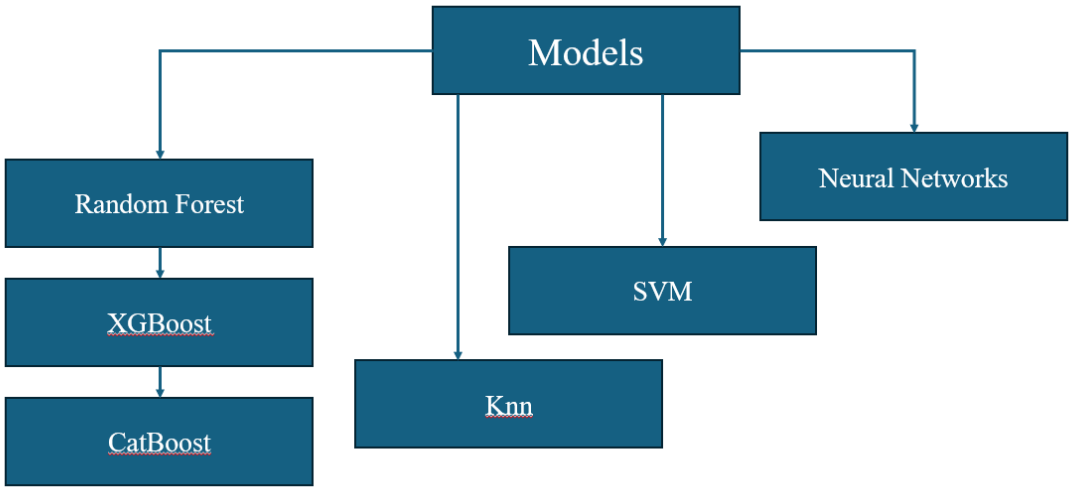


Figure 33: Models flowchart

Random Forest

Overview of Random Forest:

The idea is to select a feature in each node based on its potential to most effectively split the data. The decision of which features to use is dependent on whether one gives the best division of the data into subsets while maximizing the accuracy of each subset. If the same features are used in each tree, the trees may provide similar outcomes because the accuracy is calculated in the same way. Therefore, in a random forest, each tree is given a random subset of features to consider for splitting. This randomness ensures that each tree will be different from the others. When making predictions, each tree in the forest assigns a label to a given data point, and the final label is determined by the majority vote of all the trees in the forest. If, for example, 80 trees predict a label of 0 and 20 trees predict a label of 1 for a particular data point, the random forest will predict a label of 0 for that data point.

Figure 34 shows how Random Forest Algorithm generates the final output by combining the outputs of numerous (randomly generated) Decision Trees.

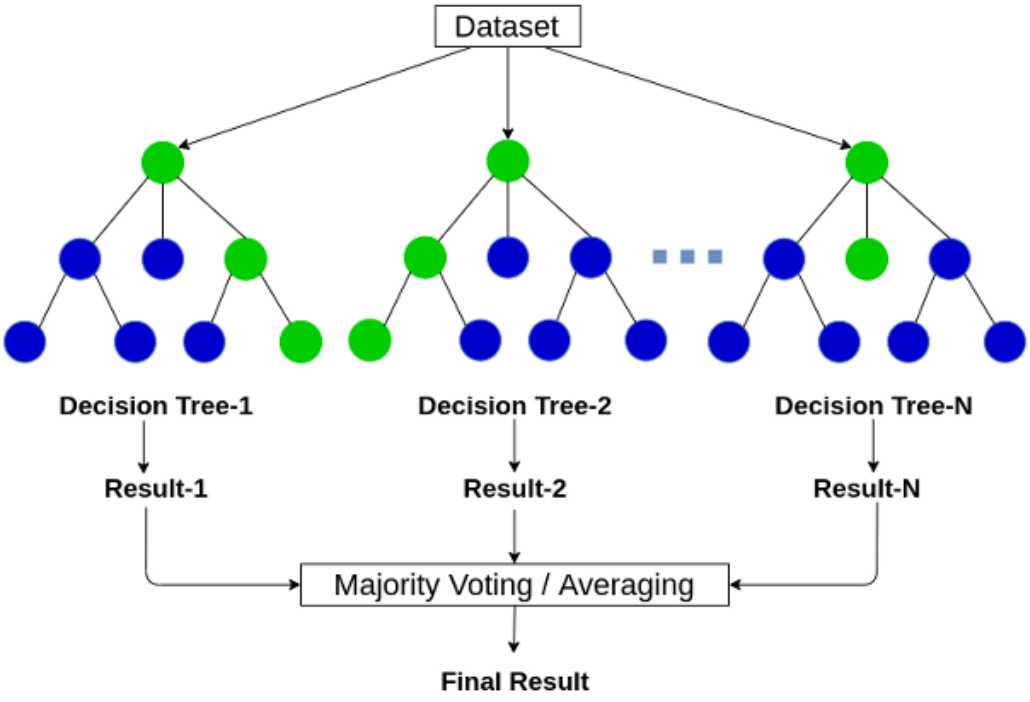


Figure 34: Random Forest Algorithm [6]

Pros of Random Forest:

- High Accuracy: Random Forest uses a group of decision trees to reduce individual tree differences and improve forecast accuracy via aggregation. This grouping technique often improves single decision tree models, particularly across various datasets.
- Random Forest is robust to noise since it collects predictions from several decision trees. The ability of the model to ignore outliers and noisy data points guarantees reliable performance, making it ideal for datasets including noise or outliers.
- Random Forest makes no assumptions about how the data should look. It can handle a wide range of forms of data without needing additional rules.
- It can figure out which parts of the data are most essential for making predictions. This helps us to focus on what features matter when we are trying to understand the data. [7]

Implement Randon Forest

In this section, I will implement the Random Forest model with and without balancing the dataset as shown in Figure 35 and 36.

Before Balancing the Dataset

```
RNF = RandomForestClassifier()

x = data_frame.drop(['Label'], axis=1)
y = data_frame['Label']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
RNF.fit(x_train, y_train)

predictions = RNF.predict(x_test)
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Figure 35: Implementing Random Forest before balancing the dataset

After Balancing the Dataset

```
[ ] # Initializing a RandomForestClassifier object
    RNF = RandomForestClassifier(class_weight="balanced")

[ ] # Extracting features (independent variables) and target (dependent variable) from the DataFrame
    x = data_frame.drop(['Label'], axis=1) # Features (all columns except 'Label')
    y = data_frame['Label'] # Target variable ('Label' column)

[ ] # Splitting the data into training and testing sets
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

[ ] RNF.fit(x_train, y_train) # Training the RandomForestClassifier model
    predictions = RNF.predict(x_test)

[ ] accuracy = accuracy_score(y_test, predictions)
    precision = precision_score(y_test, predictions)
    recall = recall_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)

    # This shows us how often the model is correct. A higher accuracy means the model is working correctly
    print("Accuracy:", accuracy)

    # This shows us how often the model is accurate in predicating phishig websites
    print("Precision:", precision)

    # Is the ratio of correctly predicted positive observations to total positives
    print("Recall:", recall)

    # This is the precenatge of combination between precision and recall
    print("F1 Score:", f1)
```

Figure 36: Implementing Random Forest after balancing the dataset

SVM (Support Vector Machine)

Overview of SVM

Support Vector Machine (SVM) is an adaptable machine learning algorithm primarily utilized for both linear and nonlinear classification and regression tasks. SVM works by identifying the optimal hyperplane that best separates data points belonging to different classes in the feature space. Therefore, when new data is inputted into the model based on its features, the algorithm decides which side of the hyperplane will be set.

The selection of the hyperplane is predicated on maximizing the margin, which refers to the distance between the hyperplane and the nearest data points. Furthermore, support vectors, which represent the closest data points to the hyperplane, play a vital role in determining the position of the hyperplane and the margin it creates. These support vectors basically fix the hyperplane in the feature space, changing its direction and increasing the gap between classes. [8] [9]
Figure 37 shows the structure of the SVM model.

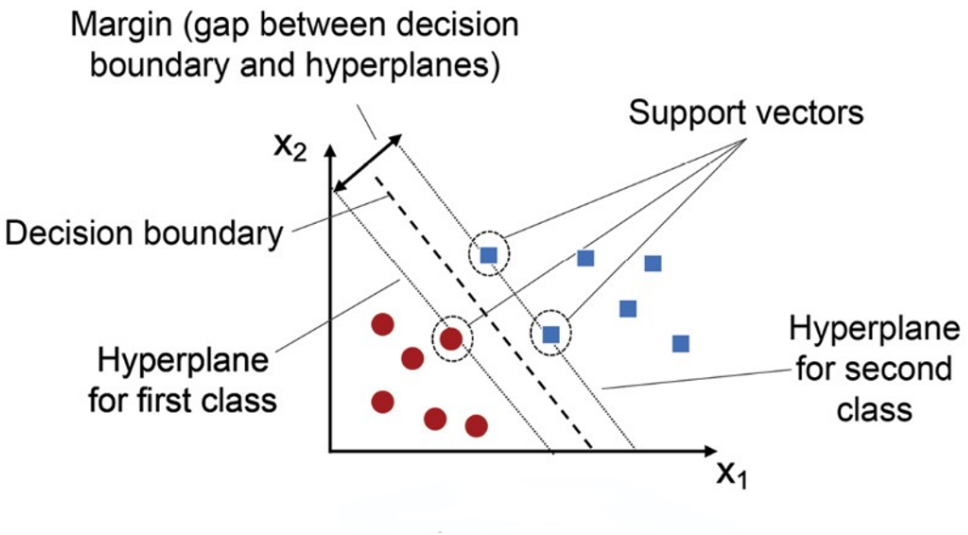


Figure 37: Main Components of SVM [10]

Additional parts in SVM are Kernels, which are functions used to transform data from the input space into a higher-dimensional feature space, helping SVM to better handle the non-linear data. Common kernel func-

tions are linear, polynomial, radial basis function (RBF), and sigmoid.

Pros of SVM:

- Handling complex relationships between data due to its kernel features.
- Accurate predictions with small data. SVM can learn from small datasets without the need for a large dataset to achieve high accuracy.
- Performs well at classifying non-linear data due to its kernels.
- SVM is memory-efficient because it uses a subset of training data points called support vectors, which significantly reduces the memory footprint required for training and prediction [11].

Implement SVM

In this section, I will implement the SVM model with and without balancing the dataset as shown in Figure 38 and 39.

```
kernels = ['linear', 'poly', 'rbf']
results = []

for kernel in kernels:

    svm = SVC(kernel=kernel)
    svm.fit(x_train, y_train)

    # Make predictions on imbalanced test data
    predictions_imbalanced = svm.predict(x_test)

    # Calculate evaluation metrics for imbalanced data
    accuracy_imbalanced = accuracy_score(y_test, predictions_imbalanced)
    precision_imbalanced = precision_score(y_test, predictions_imbalanced)
    recall_imbalanced = recall_score(y_test, predictions_imbalanced)
    f1_imbalanced = f1_score(y_test, predictions_imbalanced)

    results.append({
            'K': kernel,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1
    })

    print(f"Evaluation metrics for SVM with {kernel} kernel:")
    print("Accuracy:", accuracy_imbalanced)
    print("Precision:", precision_imbalanced)
    print("Recall:", recall_imbalanced)
    print("F1 Score:", f1_imbalanced)
    print("\n")

    # Create confusion matrix
    cm = confusion_matrix(y_test, predictions_imbalanced)

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title('Confusion Matrix')
    plt.show()
```

Figure 38: Implementing SVM before balancing the dataset

```python
# Apply Random Oversampling to balance the training data
oversampler = RandomOverSampler()
x_train_resampled, y_train_resampled = oversampler.fit_resample(x_train, y_train)
```

```python
kernels = ['linear', 'poly', 'rbf']
results_balanced = []


for kernel in kernels:
    svm = SVC(kernel=kernel)
    svm.fit(x_train_resampled, y_train_resampled)  # Use balanced training data

    # Make predictions on balanced test data
    predictions_balanced = svm.predict(x_test)

    # Calculate evaluation metrics for balanced data
    accuracy_balanced = accuracy_score(y_test, predictions_balanced)
    precision_balanced = precision_score(y_test, predictions_balanced)
    recall_balanced = recall_score(y_test, predictions_balanced)
    f1_balanced = f1_score(y_test, predictions_balanced)

    results_balanced.append({
        'K': kernel,
        'Accuracy': accuracy_balanced,
        'Precision': precision_balanced,
        'Recall': recall_balanced,
        'F1 Score': f1_balanced
    })

    print("\n")
    print(f"Evaluation metrics for SVM with {kernel} kernel:")
    print("Accuracy:", accuracy_balanced)
    print("Precision:", precision_balanced)
    print("Recall:", recall_balanced)
    print("F1 Score:", f1_balanced)
    print("\n")

    # Create confusion matrix
    cm = confusion_matrix(y_test, predictions)

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title('Confusion Matrix')
    plt.show()
```

Figure 39: Implementing SVM after balancing the dataset

KNN (K-Nearest Neighbors)

Overview of KNN

The KNN algorithm is a simple efficient approach used for classification and regression purposes. KNN relies on the principle of similarity, it makes predictions for new data points based on the majority class or average value of its 'k' nearest neighbors in the feature space. The idea behind the KNN model is based on comparing the data to its neighbors' majority class. We use 'k' to choose the number of neighbors to compare the data with, always ensuring that 'k' is an odd number to ensure a decisive result based on the majority percentage of the comparison (to prevent ties). KNN identifies the 'k' nearest neighbors based on a distance metric, such as Euclidean or Manhattan distance. For classification purposes, KNN chooses the class label that has the greatest popularity among its 'k' nearest neighbors. For

regression, a new data point's projected value is computed by averaging the target values of its 'k' nearest neighbors. Figure **??** shows an example of KNN predictions.

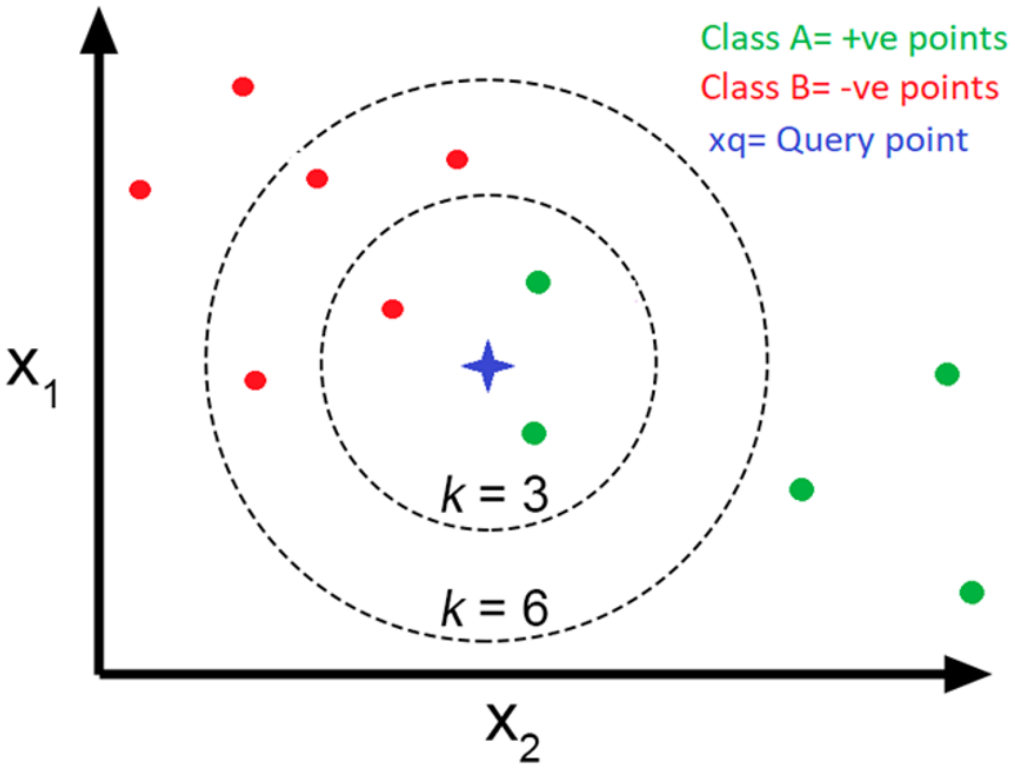


Figure 40: KNN Predictions [12]

Pros of KNN:

- Simple model easy to understand and implement.
- Effective with small datasets.
- KNN is a "lazy learner," it doesn't need an extensive training phase. Rather, it directly bases its predictions on the whole training dataset, which is stored. This can save time and computational resources compared to algorithms that require training.
- Complex interactions between variables can be effectively captured by KNN without the need to create a separable statistical model

(without coefficients and weights) [13] .

Implement KNN

In this section, I will implement the KNN model with and without balancing the dataset as shown in Figure 41 and 42.

Before balancing the Data

```
results = []
# Perform cross-validation for each value of K
for k in range(21):
 if k % 2 != 0:  # Check if k is odd
        knn = KNeighborsClassifier(n_neighbors=k)

        # Train the KNN model
        knn.fit(x_train, y_train)
        # Make predictions
        predictions = knn.predict(x_test)

        # Calculate evaluation metrics
        accuracy = accuracy_score(y_test, predictions)
        precision = precision_score(y_test, predictions)
        recall = recall_score(y_test, predictions)
        f1 = f1_score(y_test, predictions)

        # store evaluation metrics in a dictionary
        results.append({
            'Model': 'Knn_BeforeBalacing',
            'K': k,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1
        })

        # Print evaluation metrics
        print(f"K vaules equals {k}")
        print("Accuracy:", accuracy)
        print("Precision:", precision)
        print("Recall:", recall)
        print("F1 Score:", f1)
        print("##############################\n")

        # Create confusion matrix
        cm = confusion_matrix(y_test, predictions)
        # Plot confusion matrix
        plt.figure(figsize=(8, 6))
        sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
        plt.xlabel('Predicted labels')
        plt.ylabel('True labels')
        plt.title('Confusion Matrix')
        plt.show()
# Sort the results based on accuracy in descending order
sorted_results = sorted(results, key=lambda x: x['Accuracy'], reverse=True)
sorted_results = sorted(results, key=lambda x: x['Precision'], reverse=True)
sorted_results = sorted(results, key=lambda x: x['Recall'], reverse=True)
sorted_results = sorted(results, key=lambda x: x['F1 Score'], reverse=True)

# Print the top three results
print("Top three results:")
for i in range(3):
  print(f"{i+1}. K value: {sorted_results[i]['K']}, Accuracy: {sorted_results[i]['Accuracy']}, Precision: {sorted_results[i]['Precision']},"
                                        f"Recall: {sorted_results[i]['Recall']}, F1 Score: {sorted_results[i]['F1 Score']}")
```

Figure 41: Implementing KNN before balancing the dataset

After Balancing the Data

```
# Apply SMOTE to balance the dataset
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)
results = []
# Perform cross-validation for each value of K
for k in range(21):
 if k % 2 != 0:  # Check if k is odd
        knn = KNeighborsClassifier(n_neighbors=k)

        # Train the KNN model
        knn.fit(X_train, y_train)
        # Make predictions
        predictions = knn.predict(x_test)

        # Calculate evaluation metrics
        accuracy = accuracy_score(y_test, predictions)
        precision = precision_score(y_test, predictions)
        recall = recall_score(y_test, predictions)
        f1 = f1_score(y_test, predictions)

        results.append({
            'K': k,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1
        })
        # Print evaluation metrics
        print(f"K vaules equals {k}")
        print("Accuracy", accuracy)
        print("Precision:", precision)
        print("Recall:", recall)
        print("F1 Score:", f1)
        print("##############################\n")

        # Create confusion matrix
        cm = confusion_matrix(y_test, predictions)
        # Plot confusion matrix
        plt.figure(figsize=(8, 6))
        sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
        plt.xlabel('Predicted labels')
        plt.ylabel('True labels')
        plt.title('Confusion Matrix')
        plt.show()

# Sort the results based on accuracy in descending order
sorted_results = sorted(results, key=lambda x: x['Accuracy'], reverse=True)
sorted_results = sorted(results, key=lambda x: x['Precision'], reverse=True)
sorted_results = sorted(results, key=lambda x: x['Recall'], reverse=True)
sorted_results = sorted(results, key=lambda x: x['F1 Score'], reverse=True)
# Print the top three results
print("Top three results:")
for i in range(3):
  print(f"{i+1}. K value: {sorted_results[i]['K']}, Accuracy: {sorted_results[i]['Accuracy']}, Precision: {sorted_results[i]['Precision']},"
                                        f"Recall: {sorted_results[i]['Recall']}, F1 Score: {sorted_results[i]['F1 Score']}")
```

Figure 42: Implementing KNN after balancing the dataset

Neural Networks

Overview of Neural Networks

Neural networks work like the human brain. They have layers, each with specific "neurons" that connect the layers together. The connections between neurons have associated weights, which determine the strength of the connection and influence the output neuron. View Figure 42 to understand why it looks like the human brain. Everything is connected. Neural networks have 3 layers:

- **Input layer:** The input layer receives the raw data or features that are fed into the neural network. Each neuron in this layer represents one feature of the input data.
- **Hidden layer:** Between the input and output layers, there are one or many hidden layers. Each hidden layer consists of neurons that perform computations on the input data. These computations involve multiplying the input values by the weights of connections, summing the results, and applying activation functions to produce the output of each neuron.

- **Output layer:** The output layer produces the final predictions based on the computations performed in the hidden layers. View figure **??** for better understanding.

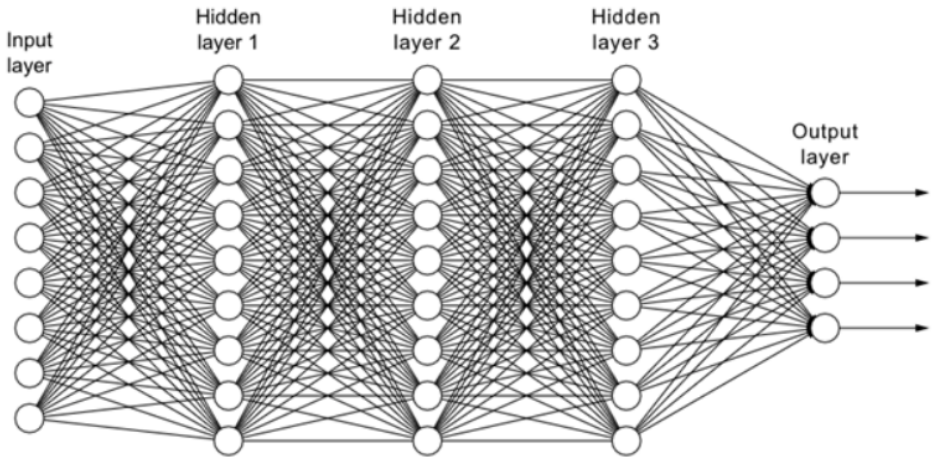


Figure 43: Neural Network Layers[14]

The weights are initialized with random values, serving as starting points for the optimization process. The neurons take the initialized weights, sum them with the previous neuron's weights, and pass them to the next neuron. This process is called forward propagation. Also, each neuro is connected with all the neurons in the next layer.

Pros of Neural Network

- Neural networks do well with huge datasets, utilizing huge quantities of data to improve accuracy and robustness.
- Neural networks can be trained and executed on parallel processing units like GPUs and TPUs, significantly speeding up computation.
- Compared to traditional machine learning models, which often require significant manual feature engineering, neural networks can create features from raw data. This minimizes the necessity for domain-specific feature extraction.

Implement Neural Network

In this section, I will implement the Neural Network model with and without balancing the dataset as shown in Figure 43 and 44.

Before Balancing the Dataset

```
# Load your dataset and split it into features (X) and target variable (y)
X = data_frame.drop(['Label'], axis=1)
y = data_frame['Label']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define the neural network architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

"""
Compile the model:
The optimizer is the algorithm used to update the weights of the neural network during training in order to minimize the loss function.
The loss function is a measure of how well the model is performing during training. It quantifies the difference between the predicted output and the true output.
The metrics are used to monitor the performance of the model during training and evaluation.
"""
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose = 0)


predictions = model.predict(X_test)
predictions = (predictions > 0.5).astype(int)

accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Create confusion matrix
cm = confusion_matrix(y_test, predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 44: Implementing Neural Network before balancing the dataset

After Balancing the Dataset

```
# Apply Random Oversampling to balance the training data
oversampler = RandomOverSampler()
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)

# Define the neural network architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train_resampled.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train_resampled, y_train_resampled, epochs=10, batch_size=32, validation_data=(X_test, y_test), verbose=0)

# Make predictions
predictions = model.predict(X_test)
predictions = (predictions > 0.5).astype(int)

# Calculate evaluation metrics
accuracy_balanced_1 = accuracy_score(y_test, predictions)
precision_balanced_1 = precision_score(y_test, predictions)
recall_balanced_1 = recall_score(y_test, predictions)
f1_balanced_1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy_balanced_1)
print("Precision:", precision_balanced_1)
print("Recall:", recall_balanced_1)
print("F1 Score:", f1_balanced_1)

# Create confusion matrix
cm = confusion_matrix(y_test, predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sn.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

Figure 45: Implementing Neural Network after balancing the dataset

XGBoot (Extreme Gradient Boosting)

Overview of XGBoost

XGBoost is an advanced model of decision tree, an improvement over the GBoost model. GBoost works by generating a series of decision trees sequentially, with each new tree learning from the mistakes of the previous one, creating an ensemble of weak learners. The newest tree is trained using residuals (the difference between actual and predicted values) of the previous ensemble. Thus, GBoost adjusts the weights of the new tree to correct previous mistakes and stops creating trees when a predefined number is reached or when the best performance is achieved and cannot be improved further. XGBoost, an improved model of GBoost, utilizes both parallel and sequential approaches. This means that XGBoost can create multiple trees at a time, unlike GBoost creates one tree at a time and then learns from these trees sequentially. XGBoost employs a technique called pruning to reduce the size of decision trees by removing parts that do not significantly contribute to improving model performance. There are two methods of pruning: pre-pruning and post-pruning. Pre-pruning involves stopping the growth of a tree based on certain conditions, while post-pruning involves removing nodes after the tree has been fully grown.

Pros of XGBoost

- High speed and scalability are achieved due to the parallelization techniques it uses.
- To combat overfitting, XGBoost has built-in regularization methods, including L1 and L2 regularization, which improve the accuracy of the model.
- XGBoost implements advanced tree pruning techniques to reduce overfitting and improve model performance.
- XGBoost provides insights into the importance of features.

Implement XGBoost

In this section, I will implement the XGBoost model with and without balancing the dataset as shown in Figure 46 and 47.

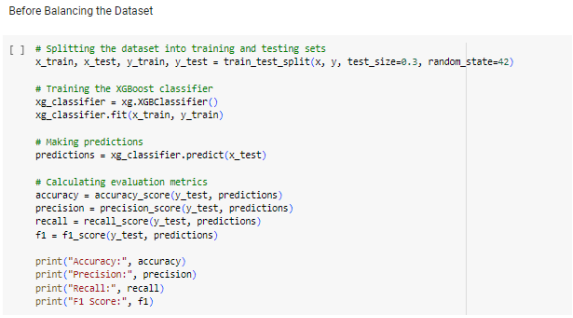


Figure 46: Implementing XGBoost before balancing the dataset

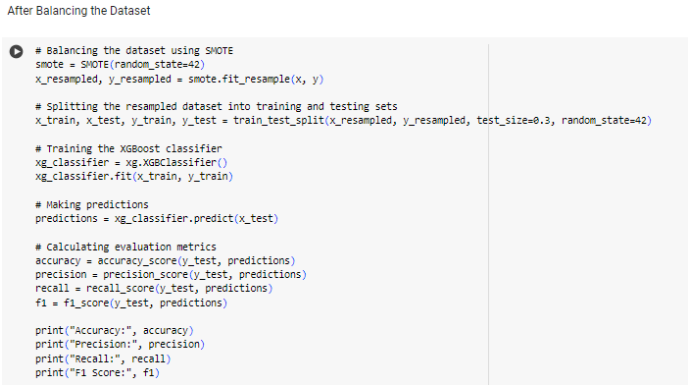


Figure 47: Implementing XGBoost after balancing the dataset

CatBoost (Categorical Boosting)

Overview of CatBoost

CatBoost is an advanced approach that offers special features over other techniques such as GBoost and XGBoost. Similar to the GBoost model, CatBoost operates by generating a series of decision trees as XGBoost using paralleling techniques and learning sequentially.

The main idea and feature of CatBoost is its ability to handle categorical features seamlessly. Unlike traditional models, which require one-hot encoding, Label Encoder, or other algorithm steps for categorical variables to

convert categorical features into numerical features, CatBoost can directly handle categorical data, eliminating the need for manual feature engineering and improving model performance. Also, CatBoost can handle missing values in the dataset to reduce the overfitting and improve the accuracy of model using algorithm called symmetric weighted quantile sketch (SWQS).

Furthermore, CatBoost employs a new gradient boosting technique that adapts to complicated data structures and automatically finds the best split points for category features during training.

Pros of CatBoost

- Automatically handling missing values.
- Automatically handling categorical data.
- High speed and scalability are achieved due to the parallelization techniques it uses.
- CatBoost provides insights into the importance of features.

Implement CatBoost

In this section, I will implement the CatBoost model with and without balancing the dataset as shown in Figure 48 and 49.

Catboost Before Balancing the Dataset

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the CatBoostClassifier model
model = CatBoostClassifier(verbose = False)
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Calculating evaluation metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Figure 48: Implementing CatBoost before balancing the dataset

CatBoost After Balancing the Dataset

```
# Balancing the dataset using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Create and train the CatBoostClassifier model
model = CatBoostClassifier(verbose = False)
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Calculating evaluation metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Figure 49: Implementing CatBoost after balancing the dataset

Measures

In this section, I focus on key performance metrics used to evaluate the effectiveness and efficiency of the used models. These metrics provide a comprehensive understanding of the model's accuracy, precision, recall, and F1 Score.

Accuracy

This metric measures the proportion of correct predictions out of the total number of predictions.It is calculated as shown in equation 1.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}$$

Precision

Precision is the ratio of true positive predictions to the total number of positive predictions made by the model (true positives and false positives). It indicates how many of the predicted positive instances are positive. High precision means that the model has a low false positive rate. It is calculated as shown in equation 2.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{2}$$

Recall

Recall is the sensitivity or true positive rate, measures the ratio of true positive predictions to the actual number of positive instances in the dataset (true positives and false negatives). It shows how well the model captures all the positive instances. High recall indicates that the model has a low false negative rate.It is calculated as shown in equation 3.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \tag{3}$$

F1 Score

F1 Score is the harmonic mean of precision and recall, providing a single metric that balances both precision and recall. It is especially useful when dealing with imbalanced datasets. The F1 Score combines the strengths of both precision and recall, giving a better measure of the model's performance in cases where there is a significant class imbalance. It is calculated as shown in equation 4.

$$\text{F1 Score } = 2 \text{ x } (\frac{\text{Precision x Recall}}{\text{Precision + Recall}}) \tag{4}$$

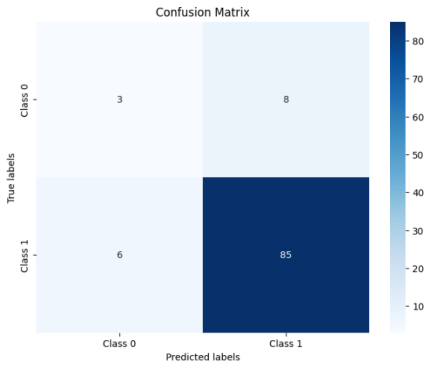
4 Results and Discussion

In this section, I will present and analyze the results received from the models. By examining the performance metrics, I will discuss the insights and key findings, highlighting the strengths and weaknesses of each model. This analysis will provide a deeper understanding of the model's behavior, its predictive capabilities, and the overall effectiveness of the applied methods.

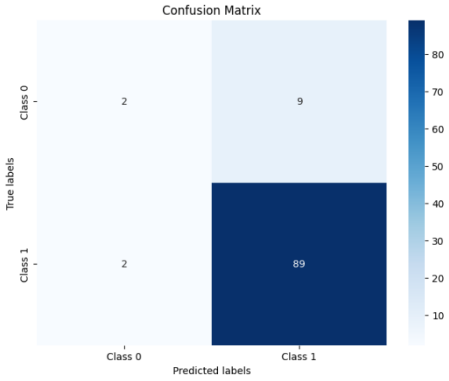
4.1 Confusion Matrix

In this section, I will present the confusion matrix for all the models before and after balancing the dataset.

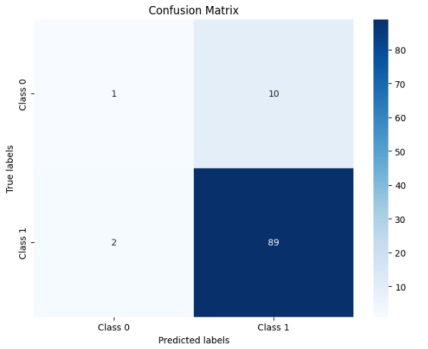
4.1.1 Model's Confusion Matrix Before Balancing the Dataset



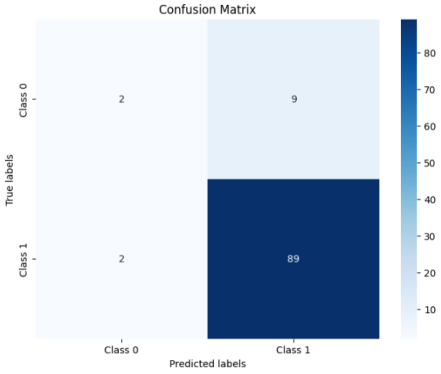
(a) Confusion Matrix RandomForest



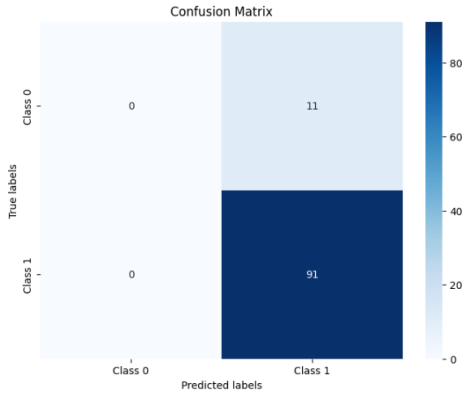
(b) Confusion Matrix Knn_13



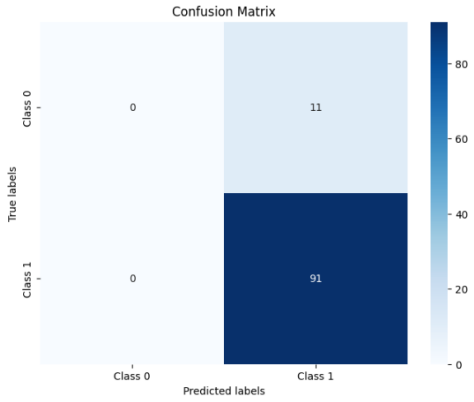
(c) Confusion Matrix Knn_15



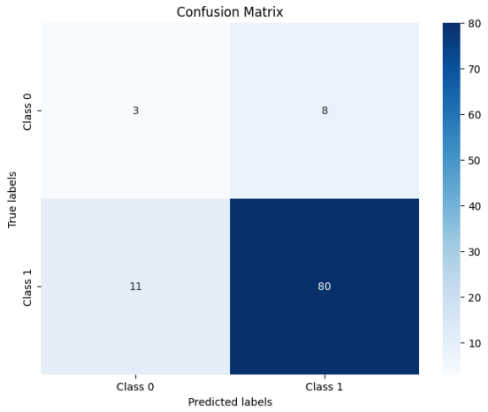
(d) Confusion Matrix Knn_17



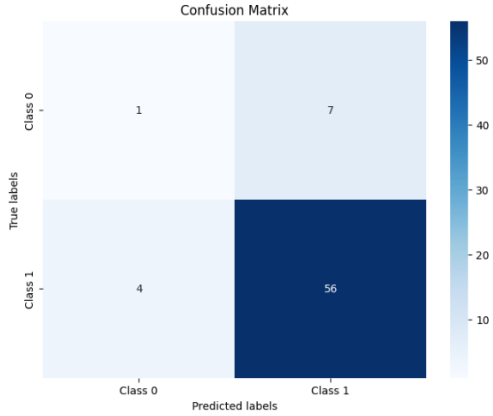
(e) Confusion Matrix SVM



(f) Confusion Matrix Neural Networks



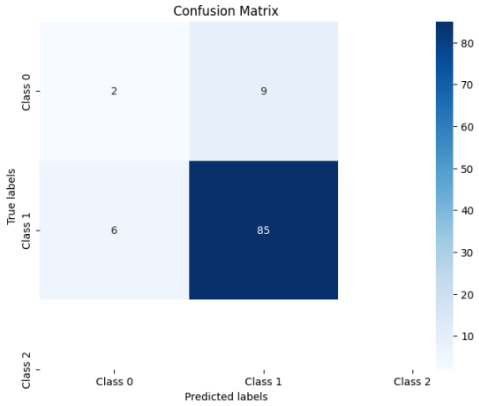
(g) Confusion Matrix XGBoost



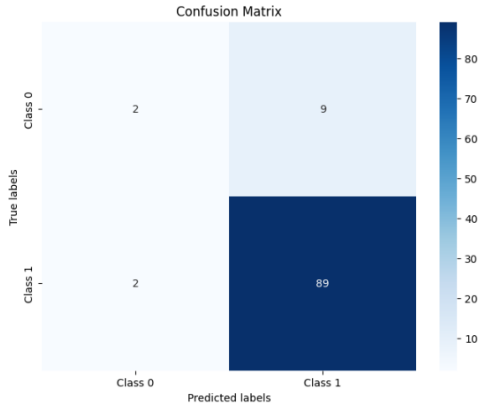
(h) Confusion Matrix CatBoost

Figure 50: Models' Confusion Matrices Before Balancing the Dataset

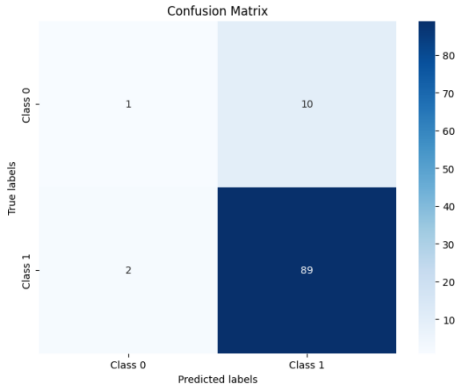
4.1.2 Model's Confusion Matrix After Balancing the Dataset



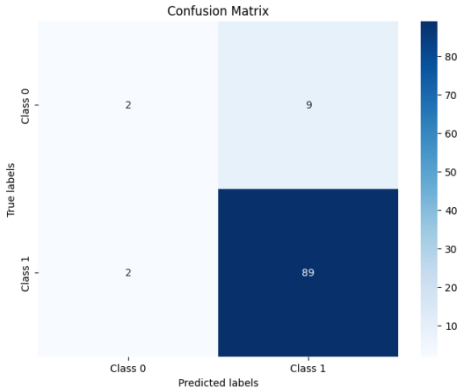
(a) Confusion Matrix RandomForest



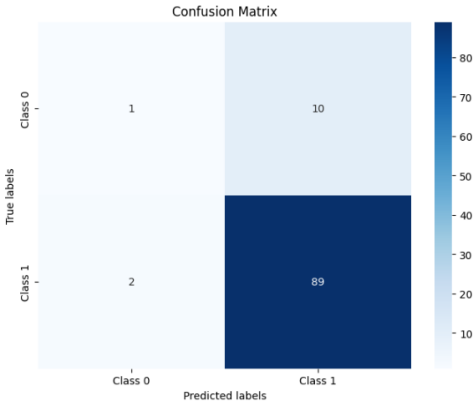
(b) Confusion Matrix Knn_13



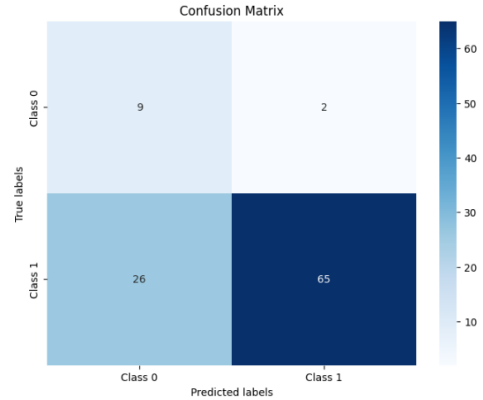
(c) Confusion Matrix Knn_15



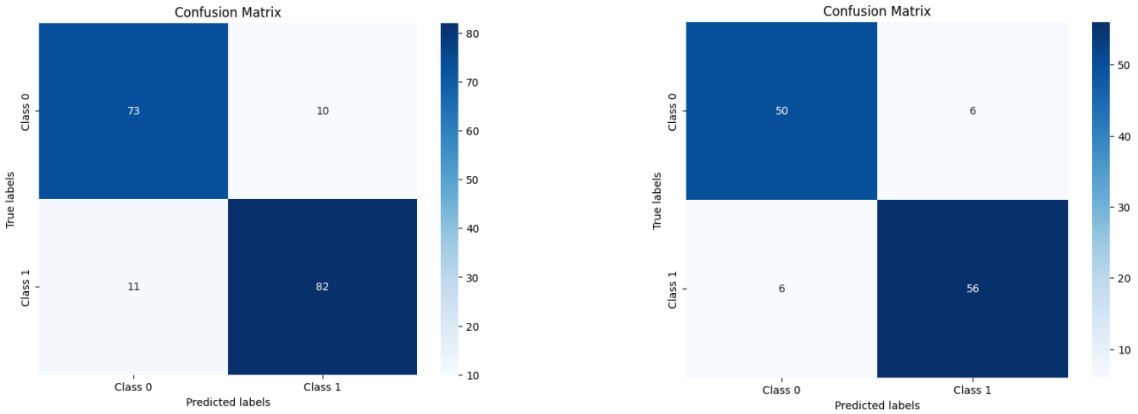
(d) Confusion Matrix Knn_17



(e) Confusion Matrix SVM



(f) Confusion Matrix Neural Networks



(g) Confusion Matrix XGBoost (h) Confusion Matrix CatBoost

Figure 51: Models' Confusion Matrices After Balancing the Dataset

4.2 Models Results

In this section, I present the results of the models and conduct a comparative analysis. By evaluating various performance metrics. This comparison will offer insights into which model performs best for the given dataset and problem context.

Table 4: **Models results**

Model	***Accuracy***	***Precision***	***Recall***	***F1 Score***
RandomForestClassifier_BeforeBalancing	0.852	0.913	0.923	0.918
RandomForestClassifier_AfterBalancing	0.852	0.904	0.934	0.918
KNN_BeforeBalancing_K_13	0.892	0.908	0.978	0.941
KNN_BeforeBalancing_K_15	0.882	0.898	0.978	0.936
KNN_BeforeBalancing_K_17	0.892	0.908	0.978	0.941
KNN_AfterBalancing_K_13	0.892	0.908	0.978	0.941
KNN_AfterBalancing_K_15	0.882	0.898	0.978	0.936
KNN_AfterBalancing_K_17	0.892	0.908	0.978	0.941
SVM_BeforeBalancing	0.892	0.892	**1**	**0.943**
SVM_AfterBalancing	0.823	0.974	0.824	0.892
NeuralNetworks_BeforeBalancing	0.892	0.892	**1**	**0.943**
NeuralNetworks_AfterBalancing	0.715	**1**	0.681	0.810
XGBoost_BeforeBalancing	0.813	0.909	0.879	0.893
XGBoost_AfterBalancing	0.880	0.891	0.881	0.886
CatBoost_BeforeBalancing	0.838	0.888	0.933	0.910
CatBoost_AfterBalancing	**0.898**	0.903	0.903	0.903

RandomForestClassifier shows robust performance with an accuracy of 85.2% before balancing and 85.2% after balancing. With a little fall in accuracy, recall improves from 92.3% to 93.4% after balancing, indicating that the model becomes better at detecting true positives, but at the cost of a minor decrease in precision (from 91.3% to 90.4%). This implies a small conflict between precision and recall after balancing, but the model remains good overall.

K-Nearest Neighbors (KNN) model was evaluated with different values of K (13, 15, and 17) both before and after balancing. The KNN model with K=13 and 17 consistently achieves high accuracy 89.2% and recall 97.8%, resulting in a high F1 score 94.1% in both scenarios, making it a reliable and

good choice. Models with K= 15 also perform well, with minor differences in precision and recall. For instance, the recall stays high across all K values, demonstrating the model's great ability to detect true positives.

Support Vector Machine (SVM) shows excellent performance before balancing, with an accuracy of 89.2% and a recall of 100%. However, after balancing, its accuracy drops to 82.3%, and recall to 82.4%, while precision increases greatly to 97.4%. This suggests that while the model becomes extremely precise, it struggles to detect all true positives after balancing, leading to a lower recall and indicating a higher rate of false negatives.

The Neural Networks model shows outstanding performance before balancing, with an accuracy of 0.892 and a perfect recall of 1.00%. This results in a high F1 score of 94.3%, showing that the model effectively identifies all true positives without sacrificing precision. However, after balancing, its performance drops significantly, with an accuracy of 71.5%, perfect precision 100%, but a reduced recall (68.1%). This suggests that the model, while precise, misses more true positives after balancing. This performance of Neural Networks models across balanced and unbalanced datasets highlights the model's robustness and reliability only before balancing the dataset.

The XGBoost model shows significant improvement after balancing. Before balancing, it achieves an accuracy of 81.3% with a recall of 87.9%. Post-balancing, the accuracy improves to 88.0%, with balanced precision 89.1% and recall 88.1%. This indicates that balancing the dataset enhances the model's overall performance, making it a strong competitor for predictive tasks.

The CatBoost model shows significant improvement after balancing. Initially, it achieves an accuracy of 83.8% with a recall of 93.3%. After balancing, the accuracy increases to 89.8%, with balanced precision and recall both at 90.3%. This balanced performance across all metrics indicates that the model is well-suited for handling the dataset, effectively managing both false positives and false negatives.

In summary, the Neural Networks and SVM achieve the best recall result, with a perfect score of 1.000 before balancing. Neural Networks performance decreases significantly after balancing. The CatBoost model is the most balanced performer after balancing, with high accuracy (89.8%) and balanced precision and recall (90.3%). The Neural Networks and SVM before balancing the dataset have the highest F1 score of 94.3%, reflecting their balanced precision and recall, making them highly reliable in identifying true positives while minimizing false positives and negatives. Precision is highest in the Neu-

ral Networks after balancing the dataset, with a score of 100%, highlighting its exceptional capability to minimize false positive predictions, thus ensuring that the positive predictions it makes are highly accurate.

Overall, the best models that can be used to detect phishing websites are the Neural Networks and SVM models before balancing which achieve 89.2% accuracy and 94.3% F1 Score, and CatBoost model after balancing achieves 89.8% accuracy and 90.3% F1 Score.

5 Conclusion

This study aimed to improve the detection of phishing websites by employing modern approaches and techniques. To increase the detection model's accuracy, a new feature was added to the dataset, and extensive analyses were performed. Furthermore, it involved implementing Machine Learning models to better detect phishing websites, such as KNN, SVM, Random Forest, Neural Networks, CatBoost, and XGBoost. These algorithms analyzed website data and features to discover patterns indicating a phishing website. In conclusion, the most effective models for detecting phishing websites were identified as the Neural Networks and SVM models before balancing, which achieved an accuracy of 89.2% and an F1 score of 94.3%, and the CatBoost model after balancing, which achieved an accuracy of 89.8% and an F1 score of 90.3%.

For future development of this study, I would recommend doing a combination of the used models to create an efficient and accurate machine to detect phishing websites. Consider developing a hybrid model that combines the strengths of Neural Networks, SVM, and CatBoost. This approach could potentially lead to even higher accuracy and robustness in detecting phishing websites. Since phishing techniques evolve rapidly, it's important to continuously update the dataset with new features and data. Furthermore, testing and training new models will keep the detection system current and effective against new phishing websites.

6 Reflections

In undertaking the research on detecting phishing websites, a critical cybersecurity challenge, I utilized an established dataset [2] and conducted primary interviews with cybersecurity experts. This approach aimed to enhance detection accuracy through machine learning models, recognizing the evolving

nature of phishing techniques and the persistent threat they pose to online security.

6.1 Selected Research Methodology

The decision to use secondary data analysis and primary interviews was vital. It enabled an in-depth analysis of phishing detection techniques, employing both quantitative insights from the dataset and qualitative thoughts from industry experts. This methodology allowed an improved comprehension of the complicated factors involved in detecting and combating phishing websites. While the approach provided useful insights, it was difficult to write a professional literature review while keeping a scientific tone. These problems highlighted the importance of improving academic writing abilities and using an organized approach to literature synthesis in future research projects.

6.2 Alternative Research Methodologies

Exploring different methods, such as controlled studies to test machine learning model performance, may provide additional insights. Using more extensive qualitative data gathering approaches, such as focus groups or longitudinal studies with end users, may help to better understand user behaviors and perceptions of phishing attempts. The main findings include the need to conduct early-stage interviews in completely guiding dataset selection. Future study could benefit from a more iterative strategy which includes qualitative feedback into model development procedures, increasing the relevance and application of detection algorithms.

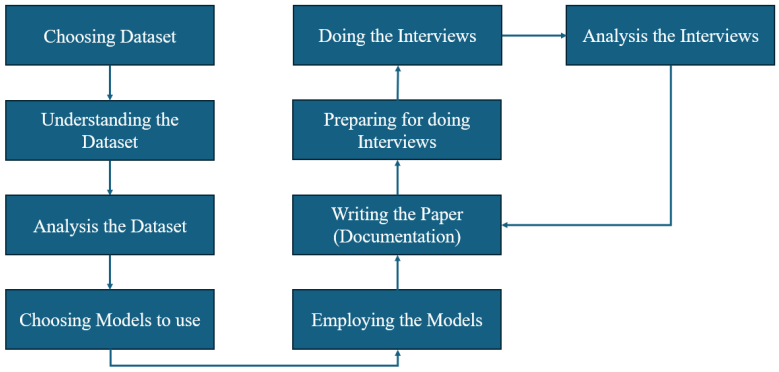
6.3 Recommended Actions and Future Considerations

Reflecting on this work helps to inform future research strategies. Improving interview design and questions to capture varied expert opinions, as well as improving data analysis approaches, will improve methodological quality. Continuous learning in advanced machine learning models, such as neural networks and CatBoost, is critical for staying ahead of new phishing methods. Furthermore, increasing engagement with teams from different cybersecurity fields and AI industry partners might improve research outcomes by validating results in real-world contexts. This collaborative approach ensures that detection solutions are not only technically sound, but also practical and adaptable

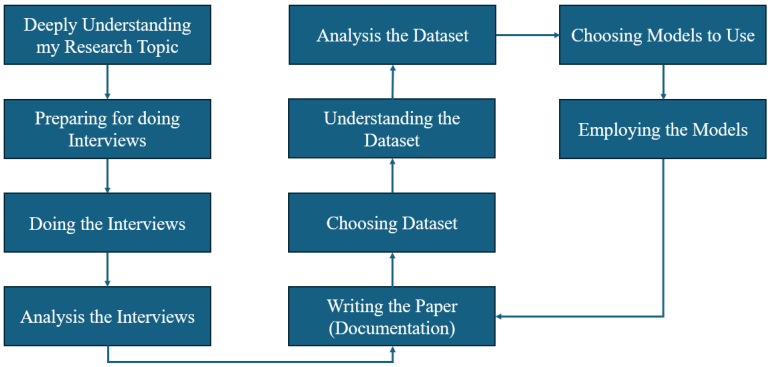
to new cybersecurity threats and techniques.

6.4 Recommended Methodology

The following flowchart is my approach used to do the research.



The following flowchart shows how would be my approach for the next researches.



References

[1] M. S. Kheruddin, M. A. E. M. Zuber, and M. M. M. Radzai, "Phishing attacks: Unraveling tactics, threats, and defenses in the cybersecurity landscape," *Authorea Preprints*, 2024.

[2] R. M. Mohammad, F. Thabtah, and L. McCluskey, "Phishing websites features," *School of Computing and Engineering, University of Huddersfield*, 2015.

[3] A. Ullah, R. A. Shah, S. A. Nawaz, N. Ahmad, and M. H. Malik, "Enhancing phishing detection, leveraging deep learning techniques," *Journal of Computing & Biomedical Informatics*, 2024.

[4] S. Das Guptta, K. T. Shahriar, H. Alqahtani, D. Alsalman, and I. H. Sarker, "Modeling hybrid feature-based phishing websites detection using machine learning techniques," *Annals of Data Science*, vol. 11, no. 1, pp. 217–242, 2024.

[5] A. Melnikovas, "Towards an explicit research methodology: Adapting research onion model for futures studies." *Journal of futures Studies*, vol. 23, no. 2, 2018.

[6] A. Sharma. (2024) Random forest vs decision tree — which is right for you? [Online]. Available: https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

[7] R. Kumar. (2024) What are the advantages and disadvantages of random forest? [Online]. Available: https://www.geeksforgeeks.org/what-are-the-advantages-and-disadvantages-of-random-forest/

[8] V. Jakkula, "Tutorial on support vector machine (svm)," *School of EECS, Washington State University*, vol. 37, no. 2.5, p. 3, 2006.

[9] A. M. Deris, A. M. Zain, and R. Sallehuddin, "Overview of support vector machine in modeling machining performances," *Procedia Engineering*, vol. 24, pp. 308–312, 2011.

[10] A. Kumar. (2023) Support vector machine (svm) python example. [Online]. Available: https://vitalflux.com/classification-model-svm-classifier-python-example/

[11] Anon. (2023) Support vector machine (svm) algorithm. [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-algorithm/

[12] D. Rawat. (2022) Knn algorithm - a geometric intuition. [Online]. Available: https://medium.com/@vagadro/knn-algorithm-a-geometric-intuition-c54cfbb3ef98

[13] Anonymous. (n.d.) Advantages and disadvantages of knn. [Online]. Available: https://app.myeducator.com/reader/web/1421a/11/q07a0/

[14] H. Ashtari. (2022) What is a neural network? definition, working, types, and applications in 2022. [Online]. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/