Detection of Phishing Websites

Anusmriti Sikdar, Anjali H. Ojha, Sakshi M. Mukkirwar, and Akanksha Tyagi

Department of Applied Data Science, San Jose State University

DATA 270: Data Analytics Processes

Dr. Eduardo Chan

December 10, 2023

Abstract

Phishing website detection is a critical component of modern cyber security efforts. Detecting these malicious websites is essential in safeguarding digital assets and ensuring online safety. Earlier works for phishing web page detection used rule-based systems that checked some predefined patterns. But, the rule-based systems weren't resilient to evolving phishing techniques. Later on, supervised machine learning models were shown to work effectively against such techniques. Most of the recent literature shows the dominating performance of deep learning models. These models are robust but require a lot of computational resources to train and operate. In this project, aim is to enhance digital security and mitigate the risks associated with malicious websites without incurring huge costs. Recognizing the efficacy of supervised methods, this project aims to develop supervised machine learning models to counter the ever-evolving phishing attacks. The primary objective is to develop machine learning models like Logistic Regression, Decision Trees, Support Vector Machines (SVMs), and Random Forests to detect phishing websites. This project used the Phishing Websites Dataset and developed techniques to extract features from the website URL and HTML source code. Alongside, data engineering and model development plans, it also provides detailed project and data management plans. The assessment of model performance is conducted meticulously. The Logistic Regression model scores 88.32 %, the Decision Tree model achieves 93.46 %, the SVM model reaches 92.51 %, and the Random Forest model excels at 96.03 % accuracy. Following 5-fold cross-validation, It is determined that leading performer is the Random Forest model.

Detection of Phishing Websites

Project Background and Executive Summary

In today's rapidly evolving digital world, the safeguarding of personal and financial information online has become an utmost priority. Among the numerous risks, phishing websites stand out as a significant risk, constantly developing and adjusting their techniques to trick unaware consumers into exposing critical information. Traditional rule-based systems have struggled to keep up with these clever strategies, highlighting the critical need for innovative and efficient measures to combat the growing threat of phishing assaults. Thomas (2018) highlighted that phishing attacks affect more than 80 percent of organizations and result in substantial financial losses each year.

One can notice an immense advancement in phishing attacks, thus it is important to work harder to prevent them. Being able to detect and prevent these assaults is essential in the field of internet security. The primary goal is to develop data-driven solutions for detecting phishing websites, hence providing actual and practical benefits to the cybersecurity profession.

This paper focuses on building robust machine-learning models to detect phishing websites. The process began with an examination of the data to gain insights into class distribution. The dataset for phishing website detection includes only two classes. Analysis of class distribution allowed us to balance them during the training of machine learning models. In the context of preprocessing raw HTML source code, the focus is directed towards methods for extracting features from HTML sources. Models were Created using Decision Tree, Logistic Regression, Random Forest, and Support Vector Machines after standardizing features for model compatibility, with metrics driving model selection and subsequent implementation. Extensive end-to-end testing aligned the models with business objectives. This project encapsulates the entire journey from data analysis to model deployment, with an outlook for future improvements.

The research findings aim to make the internet a safer place and make it easier for

businesses to follow the rules. It also makes it more difficult for attackers to trick individuals into handing them money or personal information, which can be extremely destructive to both businesses and ordinary people.

In summary, this project contributes techniques for developing and carefully testing computer programs that can detect and counter the continuously increasing danger of fictitious websites attempting to steal your information. This research hopes to provide essential insights by undertaking a comprehensive study that enhances cyber security measures, addresses the dynamic landscape of phishing threats, and eventually contributes to the construction of a safer online environment. As the digital world expands and evolves, research strives to keep one step ahead of hackers and their clever techniques, ensuring that individuals and organizations can navigate the online world with confidence and security.

Project Requirements

This section analyzes the requirements of the project. In particular, discuss functional requirements, AI-powered requirements, and data requirements.

Functional Requirements

To mitigate the threats of phishing attacks, the proposed system must be able to detect the phishing website accurately and efficiently. To achieve that goal, our systems must satisfy the following functional requirements.

HTML source code processing. The proposed must be able to extract the relevant features from the source code. The HTML source code provides a lot of information about a website.

Text processing. The proposed must be able to process the text on the websites and extract features that can be used in the training of the models.

URL processing. The proposed system must be able to extract relevant features from the URLs of the websites and verify their legitimacy.

Model deployment. The system must be able to run as a web application that takes in HTML source code and the website URL, and produce a response that indicates

whether the website is phishing one or not.

AI-powered Requirements

The proposed solution is powered by machine learning models. Leveraging such models allows the development of a robust system to arrest the malicious intent of phishing websites. The development of these models requires the identification of the model algorithms, model selection techniques, and post-processing techniques. This section describes these requirements in detail.

Machine Learning Models. After careful research and analysis, Logistic Regression, Decision Tree, Support Vector Machines (SVMs), and Random Forest models are developed to classify websites as legitimate or phishing based on the extracted features. These models need to be trained on the extracted features.

Model Selection. Model selection is an important step in the development cycle of machine learning models. The performance of the trained models must be compared, and the best model must be selected for deployment.

Model Post-Processing. To improve the recall and overkill rate of the model, the score threshold needs to be tuned on the validation set.

Data Requirements

Phishing website detection is a critical component of modern cybersecurity efforts. To develop reliable models, a balanced but feature-diverse dataset needs to be collected. HTML source code, text, and website URLs can help in the development of approaches to identifying fraudulent websites. Each type is described in detail in the following sections.

HTML Source Code. Analyzing the source code of a website is extremely important to classify it as a phishing website. HTML source code can contain dubious scripts, hidden elements, fake form elements, and malicious links. The presence of such features is an indication of phishing. Thus, HTML source code for websites is required.

Text on the Websites. The text on phishing websites contains indicators of phishing. For example, the presence of words like password, login, etc., is a symptom of a

potentially phishing website. Phishing websites also create a sense of urgency in the user. The grammar and spelling of words are also indications of a phishing website. Features extracted from the text are quite strong and help in developing powerful machine learning models. Thus, text extracted from the websites is required.

Website URL. Phishing websites can contain suspicious elements in their URL. Misspelled domain and subdomain names that resemble legitimate domains and subdomains are prevalent among phishing websites. The presence of certain path elements in its URL is also enough to question the legitimacy of a website. Thus, the URLs of websites are required.

Project Deliverables

For the successful execution of any project, enumerating concrete objectives and execution strategies is critical. In this section, a detailed exploration of each deliverable under the scope of this project is presented, as shown in Table 1.

Project Proposal

The project proposal includes the proposed research problem, background, literature survey of related works, and examples of data sources.

Work Breakdown Structure (WBS)

The creation of a Work Breakdown Structure (WBS) involves utilizing project management tools such as JIRA. It shows a visual depiction of project elements, work packages, tasks, and deliverables aligned with the six phases of the CRISP-DM methodology (Tausworthe, 1979).

Effort Estimates and Gantt Chart

Effort estimation involves the estimation of the effort required to accomplish each task of the project. The estimation process considers various factors, including the availability of team members and the complexity of the task, leading to assessments for different processes (Clark, 1922).

A Gantt chart showcases tasks and their timelines through horizontal bars, offering a comprehensive view of project schedules, task dependencies, and progress. This aids in effective project planning and tracking by providing a clear visual representation.

PERT Chart

The utilization of a PERT chart involves arranging and scheduling project tasks through the breakdown of individual elements, mapping their interconnections, and presenting the timeline visually. This chart aids in estimating the minimum required time to complete all tasks within the project (Punmia & Khandelwal, 2002).

Project Introduction

The project introduction includes project background and executive summary, functional requirements, AI-powered requirements, data requirements, project deliverables, technology, solutions, and literature survey.

Data and Project Management Plan Report

Data and project management is a detailed report describing the data management plan, project development methodology, project organization plan, project resource requirements, and project schedule. The data management plan includes data collection approaches, management methods, and usage mechanisms. The project development methodology includes a description of the project development life cycles. It also includes the planned development processes and activities. Project organization includes WBS while the project schedule includes Gantt chart and PERT chart. The project resource requirements includes hardware and software requirements.

Data Collection Plans

A data collection plan is a report detailing the requirements of data sources, parameters, and quantities. It also includes samples from the dataset.

Data Engineering Plans

Data engineering plan involves data cleaning and data transformation plans. The data cleaning plan comprises of handling of incomplete, missing, noisy, or inconsistent data.

Data Engineering Report

Data engineering report documents data process, data collection, data pre-processing, data transformation, and data preparation methods. It also provides an overview of data statistics and data analytics results.

Abstract

The abstract is a concise summary of the research done during the course of the project. It provides an overview of the whole project.

Research Report Presentations

The presentation of the research project includes slides on each component of the project. It covers project introduction, data management plans, project management plans, required resources, data engineering, and the details of the machine learning models.

Individual Research Paper (Model Development)

The individual research paper describes the model developed by each team member to solve the phishing websites detection problem. The paper contains in depth analysis and comparison of the model in terms of the evaluation metrics.

Final Research Report

The final research report describes the phishing website detection problem in detail. It provides detailed data and project management plans. Data engineering plans are also provided in depth. The report also describes the machine learning models used to solve the phishing websites detection problem with detailed validations and evaluations.

Technology and Solution Survey

Phishing is a well-known problem and it existed before the inception of the internet. But after the internet, it's become very easy to reach billions of people. As the number of

internet users increases, so does the problem of phishing. Earlier versions of phishing involved email-based phishing, but the spam filters got better over time. In web page phishing, users are redirected to pages from other sources and they do a good job of impersonating a well-known trusted web page, where users share their confidential information. The following sections discuss different phishing systems surveyed by Khonji et al. (2013).

Rule Based Detection

Basnet et al. (2011), presented a Rule-Based phishing detection approach. The idea is to create rules with known patterns and check the web page against each rule. Some example rules are if a web page URL is not present in any search link and If web page domain name is not present. They achieved an accuracy of 99% with 0.5% false positive rate by merging these rules as features with Logistic regression and Decision Tree. However, this approach was not working well for the new pages not covered by the rules.

Blacklisted URLs Based Detection

In this approach, a list of previously known phishing web pages is maintained and any traffic coming from these sites can be blocked. These types of lists can be maintained at the DNS level. This list keeps getting updated over time as new sources emerge, and it's also enriched using some predictive techniques on the already known URLs by changing domains, URLs, etc. PhishNet (Prakash et al., 2010a) uses this strategy to add more entries to the list. Google Safe Browsing API (Whittaker et al., 2010) helps to check for a known phishing web page.

Heuristic Based Detection

In this approach, a set of attributes is captured from the previously seen data and applied to the new data. The system uses rule mining and curates common characteristics of the phishing web pages. It does not rely on well-known attributes like URLs, certificates, fonts, etc. Instead, it looks for anomalies using different information. Anomalies can be present in the form of different misleading URLs, asking for confidential data, and external

links from the page, etc. These types of systems can work well and be tuned with evolving trends. These types of systems can have higher false positive rates. New phishing web pages are getting very good and sometimes this system is not able to capture those complex patterns. Examples of such a system are SpoofGurad (Teraguchi & Mitchell, 2004) which analyzes the anomalies found in a web page, and PhishGuard (Joshi et al., 2008) which identifies the phishing web pages that use HTTPS.

Visual Similarity Based Detection

In this method, a web page is classified by not using any of the URLs, website content, certificates, etc., but it analyses the visual appearance of the web page. K.-T. Chen et al. (2009) proposed a method that captures suspicious web page images and compares them against the well-known page and other work done by Hara et al. (2009), which checks visual similarity without taking a snapshot. This type of detection failed against the unknown web page and has a higher false positive rate.

Machine Learning Based Detection

Recent solutions leverage machine learning based algorithms to solve this problem. As the newer patterns emerge it can be easily learned through the features. Machine Learning based systems show very promising results in detecting phishing web pages. The more recent models are being trained using deep neural networks and work even better. Basit et al. (2021) surveyed and found that Machine Learning based systems have the highest True Positive rates. Popular ML models for the problem are SVM, KNN, Decision Tree, Random, Forest, ensemble methods, etc. Zhang et al. (2011), proposes a technique that classifies the webpages based on the website content using Naive Bayes classifier.

This project explores machine learning based solutions to implement various classification algorithms and measure their performance. We implement Logistic Regression, Decision Trees, Support Vector Machines, and Random Forests algorithms. Sindhu et al. (2020) compared multiple classification algorithms for phishing detection. They evaluated Random Forest, Support Vector Machines, and Neural Network and got an

accuracy of 97.369 %, 97.451 %, and 97.259 % respectively.

Literature Survey of Existing Research

The expansion of online platforms, together with the growing relevance of online security, has fueled interest in the development of machine learning models for detecting phishing websites. This review of the literature dives into prior research and studies on phishing website identification, with an emphasis on the use of various machine learning approaches. These studies' thoughts and conclusions serve as the foundation for the proposed undertaking. Table 2 provides a comparison of the models in the literature. The rest of the section describes them in detail.

Abusaimeh and Alshareef (2021) aimed to achieve the highest accuracy in phishing website detection. Their research focused on evaluating various machine learning algorithms and techniques to enhance the accuracy of detection systems. Their work provided valuable insights into improving detection accuracy, which is a crucial aspect of phishing prevention.

Ahmed et al. (2022) explore the use of decision tree algorithms and feature selection methods for the detection of phishing websites. Their research provides a framework for improving the accuracy of phishing detection models through feature selection, contributing to the advancement of effective detection techniques.

Dogukan et al. (2017) explored the application of Support Vector Machines (SVM) for phishing website detection. The paper explores the use of SVM algorithms and discusses their potential in identifying phishing websites, contributing to the diversity of methods available for detection.

Hutchinson et al. (2018) investigate the use of the Random Forest algorithm for phishing website detection. They provide insights into the application of Random Forest in detecting phishing websites, thereby contributing to the understanding of machine learning-based approaches for this purpose.

The systematic literature review by Safi and Singh (2023) offers valuable insights

into the state-of-the-art phishing website detection methods and their effectiveness. This review serves as a valuable resource for researchers and practitioners in the cyber security field, providing insights into the state of the art in phishing detection and guiding future research efforts. It equips readers with the knowledge required to stay at the forefront in the ongoing battle against phishing attacks, making it an essential reference in the domain of cyber security(Safi & Singh, 2023).

Zouina and Outtaj (2017) introduced an innovative lightweight system for URL-based phishing detection using SVM and similarity indexes. Their research explores lightweight techniques for detection, offering a novel approach to identifying phishing websites efficiently. These studies collectively contribute to the development of effective phishing website detection methods. They explore various machine learning algorithms, feature selection, and novel approaches, providing a rich resource for researchers and practitioners in the field of cyber security.

In a study by Jagadeesan et al. (2018), they compared the performance of SVMs alongside that of Random Forest classifiers to evaluate their performance on classifying phishing websites. The dataset had been divided in the 70:30 ratio and the performance of linear SVMs, non-linear SVMs and Random Forest classifiers on this dataset was compared. The hyperparameter tuning in their model was conducted using a grid-search algorithm. However, the results show that the Linear SVM model displayed a classification accuracy of 85.19 %, while the non-linear SVM model displayed an accuracy of 89.63 % and the Random Forest classifier displayed an accuracy of 90.12 %.

In a study by Shahrivari et al. (2020), proposed models for classifying websites as either phishing or legitimate, employing a diverse set of classification methods such as Logistic Regression, Decision Tree, Support Vector Machine, Random Forest, and XGBoost. These models was applied to a dataset of phishing websites sourced from the UCI Machine Learning Repository, comprising 6157 websites, including 4898 phishing websites. The experiments involved 30 features, and ten-fold cross-validation was utilized

for training, validation, and testing. Notably, the model achieved prediction accuracies of 96.59 %, 97.26 %, and 98.32 % for Decision Tree, Random Forest, and XGBoost, respectively.

Smith et al. (2021) highlighted the growing importance of ensemble techniques in phishing detection. These approaches, combining methods like Decision Trees and Random Forests, show improved accuracy over single-model approaches. The study in this research aligns with this finding, as a notable performance increase was observed when employing Random Forests.

The comparative analysis of the reviewed papers in the field of phishing website detection reveals a diverse set of contributions. Abusaimeh and Alshareef (2021) emphasize the critical aspect of detection accuracy and provide insights into enhancing it, underlining precision's significance in phishing prevention. Ahmed et al. (2022) contribute by focusing on feature selection with decision tree algorithms, addressing interpretability and feature relevance. Dogukan et al. (2017) introduce the potential of Support Vector Machines (SVM) for phishing detection, expanding the range of methods available. Hutchinson et al. (2018) bring the Random Forest algorithm into the landscape, offering an alternative approach. Safi and Singh (2023) systematic review contextualizes these works, summarizing the state of the art, guiding future research, and helping to position other papers within the broader field. Zouina and Outtaj (2017) introduce an innovative lightweight approach using SVM and similarity indexes, enriching the methods available for phishing detection. Together, these papers contribute to a comprehensive understanding of the field, reflecting its diversity and the continuous evolution in the quest for effective phishing detection methods.

Data and Project Management Plan

Data Management Plan

Data Collection Approaches

The first step in understanding and identifying phishing websites is data collection. Building up a comprehensive data set is crucial since it serves as the basis for the analysis. The following section describes the methods and the strategies used to collect data.

The fundamental components of our data set are the inherent qualities connected to web pages. These characteristics provide a comprehensive picture of the structure and intent of a web page. This entails gathering information from different web page aspects. By carefully examining the URLs, trends that point to phishing efforts can be identified. For example, using unusual characters or creating a complicated URL structure could be signs of nefarious intent. Textual and multimedia content on websites also offers information about the main goal of the website. Specific keywords or inconsistent material are clear red flags. The visual appearance of websites is also crucial for detecting phishing attempts. Hara et al. (2009) highlighted the use of visual similarities by malicious sites to imitate legitimate ones, emphasizing the importance of this attribute in detection mechanisms.

Since phishing websites have been around for a long time, a number of businesses have already blocked URLs known to have harmful intent and created a comprehensive database of phishing websites that have already been detected. The database fulfills two functions. First, it serves as a trustworthy resource for identifying recognized risks. Second, it provides a benchmark data set for our machine learning and heuristic models, aiding in the training and validation phases (Sánchez-Paniagua et al., 2022).

The dynamic behavior of web pages contains an abundance of information beyond their static properties. Through heuristic anomaly analysis, web pages can provide valuable information about potentially dangerous behaviors. These irregularities frequently take several forms. There are misleading URLs that are created to mimic well-known web

addresses with the intention of tricking consumers. Websites also have prompts or pop-ups that excessively request sensitive or personal data. In addition, the existence of several external links, particularly those that point to unknown domains, may indicate an effort to mislead users to malicious websites or download payloads that could be hazardous. Learning from the past is one of the most effective ways to comprehend phishing threats. Historical records of phishing attempts and the associated attributes they displayed are invaluable (Basnet & Sung, 2014).

Based on the factors discussed above, the plan involves collecting the Phishing Websites Dataset (Ariyadasa et al., 2021) in the project. The goal is to gather 80,000 website instances in the dataset, where each instance includes the website's URL and the HTML page. Among these instances, 50,000 are designated as legitimate websites, and 30,000 are identified as phishing websites. The dataset is obtained from the official website in the form of a zip file.

Data Management Methods

Efficient data management is the foundation for effective phishing detection. Using a variety of data management strategies facilitates the quick identification of possible dangers. This is an examination of the main techniques employed. The modern approach to data management involves usage of version control software (Koc & Tansel, 2011). However, the Phishing Websites Dataset by (Ariyadasa et al., 2021) doesn't change over time. So, no version control software was used.

Data Storage Methods

The development of effective and adaptable data storage techniques is required due to the exponential rise in phishing threats. According to numerous scholarly works, the following several storage techniques are available Prakash et al. (2010b) noted in their paper that these databases play a crucial role in quickly identifying recurrent phishing attempts, which in turn keeps unsuspecting users from visiting these malicious websites. Blacklist databases have been essential for immediately identifying and thwarting phishing

attempts. These are archives that hold the URLs of previously flagged malicious or dubious websites.Pedregosa et al. (2011) emphasized the benefits of cloud storage in terms of accessibility to large datasets and models in their work on Scikit-learn, a machine learning library. Storing phishing data on cloud platforms guarantees that authorized personnel can still access the data despite system or geographic limitations. On the other hand, Basnet et al. (2011) elucidated how local databases play an instrumental role in real-time phishing detection systems. The immediate data access speeds up detection and response time, ensuring timely counteraction against threats. These databases are typically housed within an organization's infrastructure, granting swift and direct access to stored data.

In the project, cloud data storage mechanisms were planned. Amazon Web Services (AWS) was be utilized for data storage, model training, and tracking usage metrics. Data storage leverages AWS's S3 buckets. In the phishing website detection project, the training data comprises HTML-formatted data from over 88,000 web pages, resulting in a training data size exceeding 10 GB. S3 buckets provide an optimal solution for data storage, with the possibility of expanding data collection in the future. Thus, S3 buckets are used to store the data during its entire life cycle. Labels corresponding to web pages were be stored in a MySQL table, hosted on AWS using AWS RDS. Following model development, hosting occurs through a web application that captures usage metrics and other information, storing the data in MySQL tables.

Data Usage Mechanisms

The ongoing fight against phishing threats makes dynamic data usage mechanisms necessary. It is paramount to control the access to data so that unwanted usage can be prevented. Access to data is carefully controlled and authorized solely through the use of the Identity and Access Management (IAM) feature of AWS. This method guarantees that designated users are granted suitable read and write permissions, thereby reducing the potential for data tampering. The dataset by (Ariyadasa et al., 2021) is available for public use. So, there is no need of any special access permissions to comply with intellectual

property laws.

The dataset is stored in a zip format in its raw form. A zip file extraction software that comes pre-installed on all the operating systems is used to extract the data.

Project Development Methodology

Data analytics or intelligent system development life cycle

A structural approach was followed to create, deploy, and maintain the system. It follow the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology. Waterfall model was used with CRISP-DM structure. It follows a linear and In the waterfall model, the progress is seen as flowing steadily downwards, like a waterfall, through several phases of CRISP-DM. (Hamdani et al., 2022). In the following sections we describe each phase of the CRISP-DM methodology.

Project development processes and activities

This project followed CRISP-DM methodology to build an application. In this methodology, the work is divided into six phases.

Business Understanding. In this phase, the project team comprehensively understands the problem statement and defines the objectives to be achieved through the project. This understanding of the project's objectives serves as a guiding factor for all subsequent decisions and actions, ensuring that the work remains focused on the intended goals. To gain insights into the project, a review of existing work and active research in the relevant area is conducted. A literature survey aids in understanding various solutions, approaches, and recent trends in the domain. For this project, which aims to develop an application for detecting phishing websites using machine learning, understanding the problem is crucial due to the increasing threat of fraudulent websites jeopardizing digital assets and online safety. As internet adoption rises, phishing websites continuously evolve to bypass rule-based detection systems. Machine learning solutions, capable of learning intricate patterns from web pages, offer a more robust approach to tackling phishing challenges. The project focuses on building different machine learning binary classification

models specifically designed to identify phishing web pages.

Data Understanding. For any data-driven solution, good quality and a large quantity of data are required. Machine learning solutions needs a large amount of data to learn complex pattern from the data. We'll use Phishing Websites Dataset (Ariyadasa et al., 2021) in our project. All the labels for the corresponding web pages are given in SQL file. This project used different data parsing techniques to extract meaningful information from HTML and URL data. The data was analyzed to gain insights into potential problems such as missing values, inconsistent data, noisy data, etc.

Data Preparation. Since the data is collected by downloading web-pages, it's uncleaned and needs a lot of parsing before this can be used for any exploratory data analysis or modeling. Dataset also has web URLs for each page, and there is been a lot of research where features extracted from the URLs and later used in the modeling. Dogukan et al. (2017) and Zouina and Outtaj (2017) propose a solution to detect phishing websites by using features extracted from their URLs. They achieve great results by using just features from the URL. For this project extracting good-quality features from the web pages is key and good-quality features increase the system's performance. We gathered features like URLs attributes, out links from the web-page, page-size, content length on the page, java-scripts, impersonation to a well-known page, etc. For any text based features, we need to convert them into vectors. Labels for given data are in 0 and 1 format which doesn't need any processing and can be directly used in the model.

Modeling. Once the data preparation is completed, we had features that can be used for model building. In this phase we built various machine learning models. As part of this project, we used Support Vector Machine, Logistic Regression, Decision Tree, and Random Forest algorithm for the binary classification problem. Model training is a compute intensive task. Since four different models are built, each model's hyperparameters are tuned so that it performs the best for phishing websites detection. We used K-fold cross-validation, and measured various metrics like Accuracy, Precision, Recall,

Sensitivity, Specificity, ROC-AUC, F1-score, etc (Hossin & Sulaiman, 2015). The metrics helped to identify model weaknesses and areas for improvement. The best model out of all the models is chosen and deployed on the cloud.

Evaluation. This phase involves the evaluation of the model to assess whether the business requirement is met. The primary focus is on rigorously assessing the performance of the developed models to ensure their effectiveness in identifying deceptive online platforms. Model selection is also done during this phase. This involves testing the models on a separate dataset not used during training. Factors such as false positives and false negatives are carefully considered to strike a balance between identifying threats and minimizing unnecessary alerts.

Deployment. For this project, we planned to build a web-app that takes the URL as input from the user evaluate it with our built model, and classify whether it is a phishing webpage or not. We hosted the model on an AWS EC2 machine, which extracted features from the given page and then compute prediction from the model, and then give a score. For the web-app we collect all the requests and predicted output.

Project Organization Plan

CRISP-DM, or Cross Industry Standard Process for Data Mining, is a widely accepted methodology for machine learning projects. For the implementation of our project, we followed the CRISP-DM and developed a Work Breakdown Structure (WBS) based on the six phases of the CRISP-DM (Tausworthe, 1979). We show the WBS in figure 1.

In the first phase, business understanding, we focused on detecting phishing websites, addressing the what and why aspects. This phase entails a comprehensive review of existing solutions and their current performance. We conducted a thorough literature review to gain insights into phishing, its impact, and the methodologies currently employed for detection. Additionally, we assessed the environmental impact of our machine-learning models and adopt sustainable practices to minimize their carbon footprint. This

commitment aligns with our goal of using machine learning technologies in an environmentally responsible manner. Finally, we created a detailed project plan to ensure the efficient execution of the project's flow.

In the second phase, data understanding, we analyzed our dataset Phishing Websites Dataset (Ariyadasa et al., 2021) and try to get a clear overview of the data type and format. We tried to understand class distribution for unbiased modeling. This task is essential as it would help us to assess whether our dataset is balanced or imbalanced so that our modeling is unbiased.

In the third phase, data preparation, our primary focus was on cleaning the dataset, which involves removing any missing, duplicate, and irrelevant values. Once we have a cleaned dataset, we initiated the process of data visualization to uncover underlying patterns in our data. Subsequently, we proceeded with feature engineering, consisting of two subtasks - Feature Extraction and Feature Selection. Specifically, we extracted features from both URLs and HTML pages. The selection of suitable features is crucial as it directly impacts model efficiency and processing time. By ensuring a clean dataset and a refined feature set, our machine-learning models performed optimally. Additionally, we performed feature standardization to further enhance the efficiency of our predictions.

In the fourth phase, modeling, we implemented the four machine-learning algorithms Decision Tree, Random Forest, Logistic Regression, and Support Vector Machines. Hyperparameter adjustment was done based on the model's performance. Once all four models are fully implemented and optimized, we evaluated their performance. We calculated a range of model evaluation metrics, including the true positive rate (recall), false positive rate, F1 score, precision, and the confusion matrix.

In the fifth phase, evaluation, we evaluated the models based on model metrics calculated in the previous step. Based on this evaluation, the best model was chosen that is suitable for the detection of phishing websites.

In the sixth and last phase, deployment, we deployed the best model on AWS,

document our project work, and conduct an overall project review. Lastly, we created our final comprehensive project report. This report comprises the project's objectives, methodology, research findings, challenges encountered, innovative solutions, and future scope of work. We also created a set of PowerPoint presentation slides. These slides played an important role in effectively conveying the project's goals, research methodology, and findings during project presentations.

Project Resource Requirements and Plan

Hardware Requirements

As part of our project, we are using Amazon Web Services cloud infrastructure. AWS is the most popular cloud platform in the industry and it provides a variety of services and infrastructure for our needs. We set up the whole infrastructure in us-west region of AWS. We use AWS for data storage, model training, and usage metrics. For our project of phishing website detection, our training data involves data from web pages in HTML format. There are more than 88,000 web-pages are present in training data, which results in a training data size of more than 10 GB. S3 buckets are ideal for data storage and later we can use the same buckets to collect more data in the future. All the labels corresponding to the web pages are stored in a MySQL table, which is also hosted at AWS and we use AWS RDS for this purpose. Once we build the model we host it through a web application, that collects the usage metrics and other information. This data is also stored in the MySQL tables.

We use AWS EC2 compute instance for all the feature engineering and model-building tasks. The configuration of these machines is with 16 GB RAM and 8 CPU because these machines are used for compute intensive tasks. We use other EC2 machines for model hosting with the lower configuration of eight GB RAM and four CPU, which take a request and predict the label either phishing or not phishing. Our application uses AWS RDS (MySQL) relational database for storing usage statistics, so we don't need a powerful machine, we are using two GB RAM and two CPU instances.

Table 3 has listed all the hardware requirements for this project.

Software Requirements

We use Python as the primary language for all development tasks like data analysis, feature engineering, model building, and model hosting. Python has many libraries and frameworks built for data analysis and machine learning model creation and we use them for our model creation. We install all the required packages and dependencies on the EC2 machines.

MySQL relational database (AWS RDS) is used for app usage statistics. Here we don't need to install the software, as AWS already provides servers preinstalled with other utilization and monitoring tools.

Table 4, has listed all the software and packages that are used for this project.

Tools And License

We are using JIRA as our project management tool. JIRA has all the features and controls that we may need for our planning and execution of the project. We are using JIRA to create Epics, Stories, and Tasks for all the work needs to be done for this project. We are also using JIRA for Gant Chart creation. Other than JIRA we use Github for all code collaboration, and later Github and Streamlit is used for web application hosting. We used Google Slides and Docs for presentations and other documentation. Table 5, lists all the tools we are using for this project.

Project And Cost Justification

All development tasks will be done on AWS and all the training data will be stored in AWS S3 buckets, we are estimating all the costs based on that. EC2 machines with higher configurations are costly and it drives most of the cost. RDS (MySQL) server is also EC2 machines with some added software and tools on top for monitoring. So all the cost is driven by the hardware requirements. As we scale the application and the number of users will grow, then we have to scale the web app and MySQL database and it will drive costs up. All the software and tools we are using are open source or free with some limitations.

We are not including any cost related to personal resources. Any tools and software which are open source with limitations are also excluded from the cost estimation. Table 6, has listed all the cost estimates.

Project Schedule

Gantt Chart

A Gantt chart is a project management tool that provides a visual representation of a project's schedule over time. It shows when each task is going to be done by which team member. It also shows start time, end time, and how much time it is going to take to perform a given task. Our Gantt chart follows CRISP-DM methodology using a waterfall-style design (Clark, 1922).

Business Understanding. Figure 2 shows the Gantt Chart for business understanding. It starts on September 11, 2023 and ends on September 22, 2023. In the business understanding phase, we start by conducting a literature survey to understand the impact of phishing websites on businesses, consumers and other individuals. We research on the possible sociological impact of phishing websites as well. We review the performances of existing solutions in the industry and study about the existing machine learning models for the detection of phishing websites. During the technology survey, we research on the libraries for feature extraction, loading SQL databases, URLs, and HTML source codes. We also look for libraries to train, evaluate and deploy our machine learning models. We examine the implications of machine learning models in terms of quantifying the carbon footprint while training the models and ways to maintain sustainability during he whole process. We look over the project requirements such as various possible sources for the procurement of URLs. We finalise business requirements by keeping in mind all the business constraints. The final step involves creation of the project plan for successful delivery of the project.

Data Understanding. Figure 3 shows the Gantt Chart for data understanding. It starts on September 25, 2023 and ends on October 10, 2023. In the data understanding

phase, we analyze the data collection requirements and finalize the dataset by Ariyadasa et al. (2021). Given its extensive library it is suitable to enhance our model training capabilities. We gain insights into the data format, identifying the methods for loading URL and HTML source code files. To construct a well-balanced model, we investigate the class distribution of phishing and legitimate websites within the dataset. Additionally, we securely store the HTML files in an AWS S3 bucket for easy access and computation throughout the project. This phase involves a comprehensive understanding of features present in the HTML source code and URLs, determining which can be extracted to build and train our models for subsequent phases.

Data Preparation. Figure 4 shows the Gantt Chart for data Preparation. It starts on October 11, 2023 and ends on October 24, 2023. During the data preparation phase, we prepare the data for further analysis. We check the dataset for any missing values, noise and develop ways to remove these anomalies like log transformations to remove outliers from the dataset. We perform data transformation which involves feature extraction from HTML pages and website URLs. We use one hot encoding since there is a possibility of encountering categorial variable for our dataset. We compute mean and standard deviation of the data and perform data standardization using Z-score method. Data regularization is done by data augmentation to balance the classes. We split the dataset into training, train and validation sets in this phase. PCA is performed for dimensionality reduction and visualization. We also perform other exploratory data analysis like creation of charts to study different web page attributes and different features of the data.

Modeling. Figure 5 shows the Gantt Chart for modelling. It starts on October 25, 2023 and ends on November 13, 2023. We work on four machine learning models during this phase namely Support vector Machines (SVM), Decision trees, Logistic Regression and Random forest. We perform hyperparameter tuning using the validation dataset for each model. We apply K-fold cross validation to compute model metrics. We evaluate each model by computing accuracy, precision, recall, and f1-score. A confusion matrix is made

for model evaluation.

Evaluation. Figure 5 shows the Gantt Chart for evaluation. It starts on November 14, 2023 and ends on November 15, 2023. The evaluation phase holds significant importance as it guides the selection of the most optimal machine learning model. Through a comprehensive comparison, we identify and choose the model that best aligns with fulfilling the business objectives.

Deployment. Figure 6 shows the Gantt Chart for deployment. It starts on November 16, 2023 and ends on November 28, 2023. During the deployment phase, we conduct comprehensive end-to-end testing of the systems and deploy our models on AWS. To create a user-friendly interface, we leverage the Streamlit application for end-users. We ensure alignment with the final business and project requirements. Additionally, we scrutinize the potential for future work, addressing any shortcomings that arose during the project lifecycle. We propose suggestions for future model enhancements to further improve our analysis of the given problem. The final deliverables encompass documentation of the project lifecycle, an abstract, creation of the final reports, and preparation for the presentation.

PERT Chart

A Program Evaluation and Review Technique (PERT) chart is a project management tool used to perform project analysis with individual tasks and dependencies. It is used to determine the minimum time required to complete the project. The minimum time required is represented by the critical path. Any delay in any of the tasks that lie on the critical path can cause the project to get delayed. The red colored lines show critical path of the project (Punmia & Khandelwal, 2002).

Our project's task sequence and dependencies are mapped out in the PERT chart shown in figure 7. The project workflow, depicted through a PERT chart, outlines the sequence and duration of tasks across six major phases. The critical path, highlighted in red, denotes the sequence of tasks that directly affect the project timeline. Delays in these

tasks would consequently push the project end date.

Business Understanding. In this phase the tasks include consequence analysis of phishing sites (two days), sourcing phishing URLs (two days), conducting literature reviews (five days), assessing the environmental impact of machine learning models (one day), and finalizing project requirements and plans (five days). Figure 8 zooms into business understanding phase.

Data Understanding. This phase covers the basics of data collection (one day), class distribution analysis between phishing and legitimate sites (five days), and feature identification (five days). Figure 8 zooms into data understanding phase.

Data Preparation. In this phase, the features are extracted from HTML pages and URLs (three days), PCA is performed for data visualization (three days), features are selected (two days), and data is split into testing and training sets (one day). Figure 9 zooms into data preparation phase.

Modeling. The team develops the phishing detection model for 10 days. It involves training and 5-fold cross validation. The performance metrics like accuracy, precision, recall, etc., are computed, which take three days. Figure 9 zooms into modeling phase.

Evaluation. This phase involves model selection (three days) and end-to-end testing (two days) to ensure the effectiveness and readiness of the model. We make sure that the business requirements are met. Figure 10 zooms into data evaluation phase.

Deployment. The final model is deployed on AWS (four days), and a project review is conducted to ensure all goals are met (four days). Figure 10 zooms into deployment phase.

Data engineering

Data process

In order to effectively identify constantly evolving fraudulent websites designed to steal personal information or distribute malware (Alkhalil et al., 2021), a lot of software-based approaches exist (Dou et al., 2017). We propose to train machine

learning-based software for the detection of phishing websites. The training of machine learning models requires data. We have the following data requirements for phishing website detection:

- The dataset must contain an almost equal representation of phishing and legitimate websites.
- In addition to the URLs of the websites, the dataset must contain the HTML source code for a robust classification model.
- Verma et al. (2019) report that the URL length is a key factor in the effectiveness of the model. Thus, the dataset must contain different lengths of the URL.
- The collected dataset must be diverse. Machine learning models have a tendency to overfit. In order to build models that generalize better, the dataset must be diverse and represent all kinds of phishing and legitimate websites.

Process to collect raw data

Based on our requirements, we choose Phishing Websites Dataset (Ariyadasa et al., 2021). Now, we describe the methodology used by them to collect the data:

- They scrape the web to gather legitimate web pages from the Google search engine and collect top-ranked URLs and fetch those to get the relevant web page.
- To collect data from PhishTank (n.d.) and OpenPhish (n.d.), they use a Python script to continuously monitor and collect phishing URLs.
- Ariyadasa et al. (2022) provides all the URLs and HTML source codes directly. So, they download the data directly.
- Ariyadasa et al. (2022) and OpenPhish (n.d.) provide only URLs. They download the URL and fetch the corresponding HTML source code.
- They download the Ebbu2017 (2017) data directly.

Preparation of the dataset

In this section, we provide details of data preparation that is common for training, validation, and test datasets.

Since Ariyadasa et al. (2021) use multiple data sources, they remove the duplicate websites from the dataset by using script capable of handling duplicate web pages and selecting high-ranking web pages based on Google's page ranking. They also make sure that the URLs are of different lengths.

During the feature extraction, we need to handle the missing values, noise, and inconsistencies in the data provided by Ariyadasa et al. (2021). The dataset is free from noise. But, they do not provide the methods used to remove noise from the dataset. We provide the details of these processes in the following sections in this chapter. We extract the URL and HTML source features and merge them to create a common data frame.

Preparation of training dataset

After the feature extraction process discussed in the later section, 80 % of the dataset is designated for training. We use stratified sampling for the split (Parsons, 2014). To make sure that there's no leakage, both URL and HTML features for a sample are kept in the training set.

Preparation of validation dataset

To enhance the model's generalization, we will implement k-fold cross-validation on the training data. This technique involves dividing the data into k equal parts, training the model on k-1 parts, and reserving one part for testing. The iterative process is then repeated k times, gradually changing the test part in each iteration until testing is conducted on all k parts. We use a value of five for k.

Preparation of test dataset

The evaluation of our phishing website detection model hinges significantly on the test dataset. A meticulous selection process ensures its separation from the training or

validation phases, ensuring unbiased evaluation under realistic conditions. Preserving data integrity, 20 % of the dataset is set aside for testing, employing a stratified split to maintain equal representation of each class (Parsons, 2014).

Data Collection

In the following sections, we describe the data sources used in Ariyadasa et al. (2021). In figure 11 we provide a brief overview of the data collection plan.

Data sources

Google Search. Google search can provide a plethora of data. Ariyadasa et al. (2021) collect different websites' URLs and source codes by search. They use domain restrictions to limit a maximum of 10 samples with the same domain name. Such a restriction is quite important to maintain diversity in the dataset.

Ebbu2017 Phishing Dataset. Ebbu2017 (2017) contains almost 73,575 URLs corresponding to phishing or legitimate websites. Ariyadasa et al. (2021) sample from Ebbu2017 dataset to add to the collection of data.

PhishTank. PhishTank (n.d.) is an anti-phishing company and it provides a lot of information about phishing websites. It also provides APIs to download anti-phishing data. Ariyadasa et al. (2021) use data collected from December 1, 2020 to October 31, 2021.

OpenPhish. OpenPhish (n.d.) provides a large phishing website database which is updated regularly. In addition, they also provide metadata that can be used for detection purposes. Ariyadasa et al. (2021) collect data from September 29, 2021 to October 31, 2021.

PhishRepo. Ariyadasa et al. (2022) contains verified phishing websites. They provide a diverse dataset that can enable the training of machine learning models. They use data collected from September 29, 2021 to October 31, 2021.

We use the dataset collected from Ariyadasa et al. (2021) as it is and do not need to collect more data since the size of the dataset is already quite large as described in the following section.

Dataset Quantity

The data collected from PhishTank (n.d.), Ariyadasa et al. (2022), and OpenPhish (n.d.) constitute the set of phishing websites while the data collected from Google search and Ebbu2017 (2017) constitute the set of legitimate websites. Ariyadasa et al. (2021) do not provide any information about how much data is collected from each source. So, we don't provide that information. After data collection, we have phishing and legitimate websites with their URLs and HTML source codes. Table 7 shows the quantity of each website type in the dataset. There are 80,000 website instances in the dataset. Each instance consists of the website's URL and the HTML page. Out of 80,000 instances, 50,000 are legitimate websites and 30,000 are phishing websites. The ground truth annotations are provided as an SQL table where legitimate websites are labeled as 0 and phishing websites are labeled as 1. Every entry in the table also provided a mapping to the HTML source code. We'll also use the source code for feature extraction. The size of the dataset is 1.7 GBs in compressed form. On decompressing, the dataset occupies 9.7 GBs.

Dataset Storage

All the raw datasets (HTML files) will be stored in the AWS S3 buckets. Raw data will be read-only and will be used only for features and insights extraction. As this data will not change frequently, saving it in the S3 store will be cost-effective. Any new data feed will also be stored in S3 buckets only. After cleaning the data we will also store the extracted features in the S3 buckets. Data key insights will be stored in a relational database which will be read optimized for reporting and analysis. Figure 14 shows the dataset stores in the AWS S3 bucket, which will be used for the data modeling in the AWS cluster.

Dataset Samples

In this section, we provide more details about the data we're using. We also show a few samples from the raw dataset. Each sample in the dataset contains the URL and the corresponding HTML source code of the website. The labels are provided in an SQL

database.

Figure 12 shows the organization of the dataset. The dataset is divided into 8 parts. Each part contains the HTML source code of websites as shown in figure 13.

The index.sql file is present at the root of the folder. Table 8 shows the schema of the index table that the SQL file generates. Figure 15 shows how each entry in the index table is added. If we look at the first entry, we can see that there's a mapping from URL to an HTML source code. These source code pages are present in the dataset parts. We'll use both the HTML source code and URLs to extract the features. Further, the label is a binary one, where the URLs labeled as '0' are legitimate websites whereas the URLs labeled as '1' are phishing ones.

Figure 16 shows an example of the HTML source code corresponding to the file 1635714064151674.html, which is a legitimate website. The website looks as shown in figure 17. The corresponding URL for this website as present in the index table is "http://homepages.ulb.ac.be/ dgonze/TEACHING/prosite_pfam.pdf".

Figure 18 shows the source code of a phishing website. It corresponds to the file 1613576216048173.html. The corresponding URL is "http://letha-71.ga/vp-htp/excelz/bizmail.php?email=&.rand=13vqcr8bp0gud&lc=1033 &id=64855&mkt=en-us&cbcxt=mai&snsc=1". We can see that the URL of a phishing website contains a lot of suspicious elements. But, the website as seen in figure 19 looks genuine.

We don't provide data samples from each of the sources since the data provided by Ariyadasa et al. (2021) do not provide the segregation into different sources.

Data Pre-processing

In this section, we describe the data pre-processing methods. In particular, we describe the methods for handling duplicates, missing values, noise, and inconsistencies in the dataset. It has been shown by P. Li et al. (2021) that data cleaning can result in significant improvements in the performance of machine learning models.

Deduplication

Since Ariyadasa et al. (2021) collects datasets from different sources, it's natural to have duplicate websites in the dataset. The presence of duplicate samples in the dataset can affect the model performance adversely by introducing bias (Zhao et al., 2021). Duplicate URLs and the corresponding HTML source codes were removed by Ariyadasa et al. (2021) to make sure that each sample in the dataset was unique. They don't provide the dataset before de-duplication and the scripts they used for de-duplication.

Handling missing values

In any data analysis, one crucial aspect of data quality is handling missing values. As discussed by Emmanuel et al. (2021), missing values can introduce bias in the models. As we extract the features from the raw data, we assign default values to the features that are missing. We also assign default values, if the processing of the raw data throws an exception while processing something. One example of this is, as we are capturing the domain name's information i.e., 'registration_length' and 'days_until_expiration', if the domain data is not present, we use the default value of -1. A negative number for registration length is indicative of a suspicious website. Thus, after feature extraction, we get features that contain complete and well-structured data, providing a sound foundation for subsequent data analysis and machine learning model development. Figure 20 shows that the data frame doesn't have any missing values.

Handling Noisy Data

Another important aspect of data cleaning is handling noisy data. Ariyadasa et al. (2021) provide the raw dataset that is already noise-free. They do not provide the methodology used for the same. In this section, we provide the general tools that are used to handle noisy data.

If noisy data is suspected we could use statistical methods or visualization tools (box plots, scatter plots, etc.) to detect outliers and determine how to handle them. Some of the methods to remove outliers are as follows:

- If outliers were rare and had a negligible impact on the analysis, we can opt to remove them from the dataset.
- In cases where outliers were observed but removing them wasn't appropriate, we could have applied data transformations like log transformations. These transformations helped reduce the influence of outliers on our analysis.
- We can use the Inter-Quantile Range (IQR) or Z-score to make the features less sensitive to extreme values. The Interquartile Range (IQR) is calculated as

$$\text{IQR} = Q3 - Q1 \quad (1)$$

where IQR is the Interquartile Range, $Q3$ is the third quartile, and $Q1$ is the first quartile.

The upper limit is given by:

$$\text{UpperLimit} = Q3 + \text{IQR} \times 1.5 \quad (2)$$

The lower limit is given by:

$$\text{LowerLimit} = Q1 - \text{IQR} \times 1.5 \quad (3)$$

Implementation of IQR is straightforward, making it the swiftest method to attain comprehensive datasets without missing values (Mallikharjuna Rao et al., 2023)

The Z-score indicates how many standard deviations a data point deviates from the sample mean. A standard threshold for identifying outliers is set at 3; if the computed Z-score surpasses this threshold, the data point is categorized as an outlier(Krishna et al., 2022)

The Z-score formula is given by :

$$Z = \frac{X - \mu}{\sigma} \quad (4)$$

where Z is the Z-score, X is the data point, μ is the mean, and σ is the standard deviation.

There's a possibility of the introduction of noise during feature extraction. We select the extracted features carefully. The presence of unwanted features can introduce noise, which degrades the model performance (Saseendran et al., 2019). We can see from the Scree plot shown in figure 34 that all of our features are useful in explaining the variance in the dataset. Thus, we don't need to remove the noise in the features any further.

Handling inconsistent data

Inconsistent data could lead to inaccurate insights and conclusions (Budach et al., 2022). Handling inconsistent data is also an important part of data cleaning and this can be done by establishing data standardization rules to ensure consistency across the dataset. Ariyadasa et al. (2021) provide the raw dataset where the inconsistencies were already handled and don't provide the methods used for the same. We also don't have examples of inconsistent data from Ariyadasa et al. (2021). During the feature extraction process, we handled the inconsistent format of the index database. In the Phishing websites dataset, the labels are stored as a database. On loading the database, we found that a lot of INSERT commands weren't working. On analyzing the SQL script, we noticed that the "'s" character was causing SQL syntax errors. We fixed the syntax to use "''s" in about 20 INSERT commands. In addition, we updated the syntax to be compatible with SQLite3. Once the dataset was loaded, we extracted the features and created a data frame. We show an example of the SQL syntax error before and after in figure 21. We show the database loading code in appendix A.

Data Transformation

Feature Extraction

Data preprocessing is a critical step in preparing the dataset for building effective machine learning models. The quality and cleanliness of the data have a direct impact on the model's performance. Our dataset is quite extensive and diverse, making it challenging to analyze directly. It encompasses data from two primary sources: Website URLs and HTML file content. So we have to start the data-engineering from the data extraction

itself. Figure 22 shows the high level feature extraction process.

Buber et al. (2017) analyzed what kind of features are useful in the development of machine learning models. During the initial phase of dataset transformation, we extracted features from URL and HTML source codes. The extracted features are stored in JSON files. We divided the feature extraction task into two steps.

Feature Extraction from URLs. Sahoo et al. (2017) show that a web page's URL contains a lot of attributes and that itself tells a lot about the web page. So it was important that we get as many features as possible from the URL. We use Python's **urllib** package to parse the URLs and **idna** package to get domain name related attributes. We also check each URL for the popular URL-shortner services and the IP Addresses. Table 9 contains all the features we extracted from the URLs. We can see a URL's feature in figure 23. The corresponding URL is also shown in the figure description. We show the code for URL feature extraction in appendix B.

Feature Extraction From HTML Pages. We also extracted features from the HTML source codes. Aljofey et al. (2022) and Y. Li et al. (2019) provide ways to extract HTML features along with URL features. Each web page in our dataset has a **.html** file in addition to its URL. The HTML contains various HTML tags, javascript, forms, links, and other elements. We captured all this information and used it as different features for a page. To process the HTML data, we use Python's **BeautifulSoup** library, to extract HTML from elements from the data. Without this library, we have to write a lot of regular expressions and coverage may not be as good. We show the HTML feature extraction code in appendix C.

Table 10 contains all the features we extracted from the HTML pages and a sample feature extracted from HTML source code shown in 24 can be seen in figure 25.

Collation of features

We merge the features extracted from URLs and HTML source codes into a common data frame. This step organizes the data into a tabular format, making it suitable

for further analysis and model development. This comprehensive data preprocessing and integration approach optimized our dataset for subsequent machine learning analysis, allowing us to extract meaningful insights and build robust models for phishing website detection. An example of collated feature is shown in figure 26. The collated features are loaded into a data frame with 80,000 rows and 33 columns as shown in figure 27.

Encoding Categorical Features

We extracted all the features from raw uncleaned data and that requires lots of transformation and processing. Apart from that, we have some categorical values in our data, and the machine learning model handles those values as numerical values. There are several methods to encode the categorical variables (Potdar et al., 2017). We encode the categorical features into numerical representations using the one-hot-encoding method.In the process of one-hot encoding, a new variable is created for each level of a categorical feature, and each category is associated with a binary variable with values of either 0 or 1. In this context, 0 signifies the absence, while 1 indicates the presence of the respective category(Dahouda & Joe, 2021) For example, we transformed the 'form_action' column using one-hot-encoding so that it can be used for predictive modeling, making it suitable for machine learning algorithms. Figure 28 shows the one-hot encoding of form_action.

Data Standardization of Continuous Features

Data normalization is a key aspect of data transformation (Patro & Sahu, 2015). We normalize the continuous features. This step ensures that features with high scales do not dominate the learning process. Our data have some features that measure the days and it varies for different domains, for example, 'registration_length' has values ranging from [-1 to 18000], so it's important to normalize the features. We computed the mean and standard deviation of features and applied z-score normalization to both the training and test data. The reason we chose the Z-score standardization is because it is less affected by outliers compared to some other scaling methods (Kolbaşi & Ünsal, 2019). Figure 29 and 30 show the features before and after normalization. Figure 31 and figure 32 show the

mean and standard deviation respectively.

Data Regularization

Data regularization is a key component of developing machine learning models. In order to train models that generalize on the unseen data, we must use regularization techniques. We plan to use L2 and L1 regularization for training the logistic regression model (Hastie et al., 2009). We'll use cross-validation to determine which technique results in the best model. We plan to use pruning techniques during the training of decision trees and random forests. The techniques help in the regularization of tree-based models. This results in better generalization capacity of the model. We also plan to use soft-SVM, where we can control the regularization of the model by controlling how much error is allowed in the training set. In sklearn python library, this parameter is controlled by the variable C. These regularizations would be done during the training of the model and the transformed data remains the same.

In order to improve the performance of our models on each class, we augment the dataset to make sure that each class has an equal number of samples. We use SMOTE (Chawla et al., 2002) to oversample the phishing class and undersample the legitimate class. The results of oversampling and undersamling are shown in figure 33. We apply the oversampling to only the training set. The test and validation set are kept the same to make sure that the testing is done with the original distribution of the dataset. This is important because we want the test set to represent real-world situations.

Principal Component Analysis (PCA) for Dimensionality Reduction and Visualization

Principal Component Analysis (PCA) (Maćkiewicz & Ratajczak, 1993) is a powerful technique frequently used in both machine learning and data analysis to reduce the number of dimensions in the dataset while preserving maximum data variance . This method proves especially advantageous for visualizing high-dimensional data, uncovering underlying patterns, detecting clusters, reduced sensitivity to noise, lower demands on

capacity and memory, and enhanced efficiency(Karamizadeh et al., 2013)

. In the context of our phishing website detection project, PCA serves to simplify the feature space, enhancing model efficiency and helping in visualization.

In preparation for PCA, we standardized the continuous features by subtracting the mean and dividing by the standard deviation. This standardization process is important, as it equalizes feature importance by normalizing them. PCA was executed, with the model being fitted to the standardized training data. This led to the transformation of the initial features into a set of principal components. Figure 34 shows the scree plot, which shows the variance of the data against different numbers of PCAs. We can use elbow point in the Scree plot to reduce the dimensions of the dataset.

Data Preparation

As described by Saria and Subbaswamy (2019), the creation of a successful and reliable machine learning model requires that datasets for the various stages of machine learning are rigorously prepared, ensuring that each phase of model development is guided by correct and representative data. This preparation is essential for developing a robust and reliable detection system.

In order to develop machine learning models that generalize well, we split the transformed dataset into training, validation, and test sets (Xu & Goodacre, 2018). The training set is used to train the model, the validation set is used to tune the hyperparameters of the model, and the test set is used to test the model using several metrics. In the following sections, we describe how the training and validation sets are created for the development of phishing website detection models.

Training set

Dobbin and Simon (2011) show that the optimal proportion of the training set lies between 40 % and 80 % under a diverse set of conditions. Thus, after feature extraction, we isolate 80 % of the dataset into the training set. We make sure that there's no data leakage. If a website instance is in the training set, we make sure all its features are in the

training set. This results in 64,000 samples in the training set. Furthermore, we use 5-fold cross-validation for evaluating model performance. So, every model is trained on 51,200 samples. Figure 35 shows a sample from the positive class i.e., phishing website class while figure 36 shows a sample from the negative class i.e., legitimate website class.

Validation set

The primary function of this dataset is to optimize the model's hyperparameters and prevent overfitting (Xu & Goodacre, 2018). We can improve the model's ability to generalize and effectiveness by tuning its performance on the validation dataset. The validation dataset serves as an invigilator throughout the training process to ensure that the model not only learns but also retains its capacity to perform well on unseen data. Thus, it is paramount that we reduce the bias introduced by the choice of the validation set. We use 5-fold cross-validation (Hastie et al., 2009) to tune the hyperparameters of the model. The validation set is also used for the selection of the best model. The training dataset is split into five sets of equal size. Thus, each set has 12,800 samples. We keep each set as the validation set once and train all the models on the rest of the sets combined. Thus, every model is trained five times. The reported performance of the models is the average performance across five iterations. Since we use 5-fold cross-validation, figure 35 and figure 36 serve as samples from the validation set too. Figure 73 is shows the 5-fold cross validation process.

Test set

The test dataset is extremely important in the context of our phishing website detection effort. We must choose the test set carefully to avoid any contamination from the training set. To ensure an unbiased evaluation of the model's performance, this data must stay untouched during the training or validation phases. The integrity of this test dataset ensures that our model is evaluated under realistic conditions, reflecting its capacity to identify new, previously undiscovered phishing websites in a real-world scenario. Based on the discussions by Dobbin and Simon (2011), we keep 20 % of the dataset for testing

purposes. We make sure that each class is equally represented in the test set. Thus, we split the dataset in a stratified manner (Parsons, 2014). This results in 16,000 samples in the test set. Figure 37 and figure 38 provide samples from the test set corresponding to positive i.e., phishing, and negative i.e., legitimate classes respectively.

Data Statistics

Summarizing Data Preparation Results

The data preparation phase, crucial for the construction of a reliable machine learning model, was meticulously undertaken following the guidelines proposed by Saria and Subbaswamy (2019). The dataset underwent a rigorous process of preparation, ensuring the integrity of each phase of model development, from the raw collection to the final prepared datasets. In aligning with best practices outlined by Dobbin and Simon (2011), we partitioned the transformed dataset into distinct sets: 80 percent for training, ensuring no data leakage and maintaining the optimal proportion for model training; a validation set, shaped by 5-fold cross-validation, which is instrumental in hyperparameter tuning and model selection as recommended by Hastie et al. (2009); and a 20 percent stratified test set, dedicated to impartial model evaluation, ensuring a realistic representation of the model's predictive capabilities on unseen data (Parsons, 2014). Each class was equitably represented across the datasets, facilitating the development of a robust and effective phishing website detection system. We do not delete any data at any stage of the data processing. Hence, all the comments above are valid for raw, pre-processed, transformed, and prepared datasets.

Statistical Presentation of Results

In table 11, we show the amount of raw data, we have 30,000 phishing websites and 50,000 legitimate websites. We do not show the distribution among each source since Ariyadasa et al. (2021) do not provide that information.

After pre-processing, the distribution of the dataset remains the same. Thus, table 11 shows the data statistics after pre-processing as well.

After data transformation but before data augmentation, we have 80,000 rows and 33 columns in the data frame. 30,000 rows correspond to phishing websites and 50,000 correspond to legitimate websites. Out of 33 columns, 32 columns represent features and one column represents the label. Table 12 summarizes this information.

On applying data augmentation we have 40,000 samples of legitimate websites and 40,000 samples of phishing websites. Table 13 summarizes this information.

Table 14 shows the size of training, validation, and test datasets after data preparation. Note that we use 5-fold cross-validation. So, the quantity of the validation set is a representative value after dividing the training set into five parts.

In figure 39 we show the distribution of each class in each of the data splits. Due to stratified sampling, we see that the distribution remains the same for each class in each of the splits.

Figure 40 we can see the distribution of the dataset samples. To refine the model's predictive capability, a 5-fold cross-validation technique was employed within the training dataset James et al. (2013).The 5-fold cross-validation ensures that our model's evaluation is thorough and the absence of a distinct validation dataset portion in the visualization accurately reflects our methodological approach.

We did the features standardization of the training and test data. We perform this for the training and test data. Figure 41 shows the normalization value statistics for some features. and figure 42 shows the box plots for the features with numerical values only. In the box plot, we are limiting it to a value < 20, as the full value range was distorting the plot.

Data Analytics Results

For our phishing detection project, we used a dataset consisting of many HTML web pages and their corresponding labels. Our data analysis was done after the feature extraction was done. As we were handling raw data in the form of HTML files, we couldn't just use that for the analytics. Once we extracted all the features, we analyzed them

against the output label to understand the feature's direct relationship with the output class and how each feature relates to the other. Our analysis gave us clear patterns and correlations, but it also shows how phishing web pages getting better. All the data analysis code is present in appendix D.

URL Length Analysis

In our first analysis, we try to see how the length of the URL for a given web page is compared for phishing or non-phishing web pages. In figure 43, where X-axis represents the length of the URL, and the Y-axis represents the number of samples in the data. We can see that it's very close to the normal distribution but skewed on the left side. For the web pages with URL lengths of 130, we can see a mix of phishing and non-phishing web pages, but after that, most of the data points to phishing web pages, and one of the reasons for that, is because phishing web pages often have redirection URL embedded in it. On the right-most side, we bucket all of the instances where the length of the URL is greater than 190, and we can see that most of them are phishing web pages with few exceptions.

Figure 44, shows a scatter plot for the URL length and number of subdomains in the URL of the website. Most of the legitimate websites have fewer subdomains, but as the length of the URL increases the number of subdomains also increases. But that's not the case for phishing webpages, most phishing website's URLs have fewer subdomains compared to the URL length. Figure 45, shows the violin plot for the same, as the legitimated web pages have fewer number sub-domains compared to the phishing webpages, and all legitimate webpages have 6 or fewer subdomains.

Different HTML Elements Analysis

We also analyze different HTML Elements like iframe, images, external links, etc. Sometimes we can see a direct correlation between these elements. Figure 46 shows the number of images on a web page and the output label distribution where the X-axis is the number of images on a web page and the Y-axis represents the number of web pages, and we can see that most phishing web pages don't have any images at all, but if we count up

to 5 images on a page, most of phishing web page sample in our data set falls in that bucket. As the number of images on a web page increases we can say that those web pages are non-phishing web pages.

Similar to the images, we can see a similar kind of pattern for the external links from a web page. Figure 47 shows the number of external links on a web page and the output label distribution. In this plot, the X-axis represents the number of external links on a web page and the Y-axis represents the number of web pages. Most phishing web pages don't have any external link, because once a user reaches the web page it won't want you to leave the page.

Figure 48 shows the number of iframes on the web page and the output label, and it shows a similar pattern there also. Iframes are generally used for ads, and phishing pages generally don't get ads. And most of the phishing webpages don't use iframes at all.

Web Page Attributes Analysis

A secure and non-phishing web page will have some attributes that are important for the security of the web page and make it trustworthy. Figure 49 compares phishing and non-phishing web pages for the different attributes. For this analysis attributes like - *is_domain_valid*, *had_redirection*, *contains_phishing_keywords*, *contains_ip*, *form_autocomplete*, *javascripts_redirects_present*, and *login_form_present*. Spider plot shows legitimate web pages have higher counts for valid domains, HTTPS usage, and redirections. Figure 50, is the same figure but with a log scale to see the same information in detail, and it highlights hidden details like more phishing web pages use IP addresses and java script compared to non-phishing web pages.

Form Action Analysis

Another important element is form actions and form elements for phishing and non-phishing web pages. Figure 51, shows the number of username and password fields on a web page. It shows a similar pattern for legitimate and phishing websites.

Figure 52 shows how form action on a web page is distributed for phishing and

legitimate websites. Most of the form actions on the phishing websites are either no action or not defined.

Feature Correlation and Importance

As we can see in our earlier analysis, we can see some clear patterns used by the phishing and not phishing web pages. So we did a features correlation analysis. Figure 53 shows the correlation between all the features we extracted from the data set and the output label in the form of a heatmap, where values range between 1 to -1, 1 shows a higher correlation and -1 shows no correlation at all. Feature likes *exteral_link_counts* and *num_a_tags* shows higher correlation because, in a web page, external links are places using *<a> HTML tag*, and the same is observed for *num_iframe_tags* and *iframes_count* also. This heatmap will help to drop some features from the final model training, which will save training, validation, and testing costs.

Figure 69 shows each feature's importance against the output label. Here we can see that *num_password_fields*, *form_autocomplete*, *contains_phishing_keywords* have more importance than any other features for phishing identification.

Visualization of PCA

The PCA-transformed training data can be visualized in two dimensions by picking the top two PCA components, giving us insight into the distribution of data points within the reduced-dimensional space. This visualization helped in understanding the distribution of data points in the reduced-dimensional space, which can reveal underlying patterns or clusters. Figure 56 shows the PCA for 2 features.

Modeling

The modeling section of the phishing detection study involves the execution of various machine learning algorithms, namely Random Forest, Logistic Regression, Decision Tree, SVM, and XGBoost. These models are trained on a prepared dataset and their performance in accurately identifying phishing websites is assessed. Logistic Regression and SVM, as linear models. The Decision Tree model offers a hierarchical, more

interpretative approach. Random Forest, an ensemble of Decision Trees, enhances performance and mitigates overfitting. Lastly, XGBoost, an efficient and effective gradient boosting framework, is employed to potentially further improve results. The performance of each model is first evaluated and then compared to identify the most effective algorithm for phishing detection. The code implementation and performance evaluation plot generation code is available in Appendix E.

Model Proposals

We propose Logistic Regression, Support Vector Machines (SVMs), Decision Trees, and Random Forests as the models to classify websites as phishing or legitimate. In the following section, we provide an overview of the usage of each model in the detection of phishing websites. We also provide a brief overview of the mathematical formulation of each model. XGBoost model is also proposed as one of the models.

Logistic Regression

Logistic Regression is a universally acknowledged and statistically robust algorithm for binary classification tasks. As Hastie et al., 2009 elucidate, this model is particularly apt for such applications due to its ability to manage binary outcomes probabilistically, providing not just a classification, but also the probability of a website being a phishing site.

As per Aljofey et al., 2022, the Logistic Regression model functions based on the logistic function, predicting the likelihood of an event by fitting data to a logit function. The model will employ a set of predictive features derived from website URLs and HTML content, such as secure protocol usage in the URL, suspicious token presence, URL length, and abnormal script usage within the HTML. These features, indicative of phishing activity, have been selected based on their proven significance in literature.

Y. Li et al. (2019) presented a model that uses logistic regression within an ensemble framework, demonstrating its strength as a base learner due to its expertise in managing binary outcomes. The paper highlights logistic regression's ability to collaborate with other

models to improve overall prediction accuracy, making it a crucial tool for tasks like phishing website detection where integrating features from various models is advantageous.

Sahoo et al. (2017)'s study emphasizes the practical usefulness of logistic regression in classifying URLs as malicious or benign. The research underscores the model's high interpretability and its ability to deliver significant results even with a straightforward linear decision boundary, often adequate for the binary classification required in phishing detection tasks.

James et al. (2013) offers a thorough examination of logistic regression, detailing its statistical foundations and its usage in predictive analytics. The text presents an in-depth understanding of the model's assumptions, functionality, and interpretability, crucial for comprehending and applying the model in real-world situations such as phishing website classification.

The collective insights from the literature highlight the model's capacity to interpret and evaluate feature contributions, handle class imbalance, and optimize predictive performance.

Mathematical Formulation. The logistic regression model estimates the likelihood of a particular data entry being classified as a phishing website using a logistic function. Often called the Sigmoid function, this mathematical function converts any value from the set of real numbers into a range spanning from 0 to 1. This range can be interpreted as the probability of a data point belonging to a certain class. The following equation represents the logistic regression model used to predict the probability of a website being identified as phishing:

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + ... + \beta_n X_n)}} \tag{1}$$

$P(Y = 1)$ represents the probability that a given website is a phishing site. The symbol e denotes the mathematical constant that is the base of the natural logarithm. The intercept term, β_0, and the coefficients $\beta_1, ..., \beta_n$, assign weights to the corresponding

feature variables $X_1, ..., X_n$, which serve as the model's input features. This logistic function is crucial because it converts a linear regression (which could predict any number from negative to positive infinity) into a probability that can be utilized for binary classification. The coefficients β are calculated using maximum likelihood estimation from the training data, aiming to maximize the likelihood of the observed data under the model (Hastie et al., 2009). This function evaluates the effectiveness of a classification model that yields a probability outcome between 0 and 1.

Logistic Function Curve. The logistic regression model first calculates the probability of whether a website is a phishing site or not, by utilizing a sigmoid function and then this probability is derived by fitting a logistic curve to the data, where the output ranges between 0 and 1, indicating the likelihood of each class. A decision boundary is set at a specific probability threshold, typically 0.5, to categorize each instance.

Figure 57 illustrates the Logistic Function Curve, a visual representation of the logistic regression model used for phishing classification. Based on a linear combination of input features z, the logistic regression model estimates the probability $P(Y = 1)$ that a specific website is phishing. The sigmoid shape of the curve smoothly transitions from a probability of 0 to 1, showcasing the model's ability to handle binary outcomes. The decision threshold, marked by the dashed line at a probability of 0.5, sets the point at which the model classifies a website as phishing or legitimate. Points on the curve above the threshold are classified as phishing, while those below the threshold are deemed legitimate.

Optimization Parameters.

Optimization Parameters. Optimizing logistic regression can be achieved through effective feature selection. It is crucial in logistic regression to enhance model accuracy by eliminating irrelevant or less significant features, which simplifies the model and aids in preventing over-fitting (Basnet et al., 2012). To tackle imbalances in the dataset, class balancing is utilized to ensure that the model does not bias towards the majority class. This improves the model's accuracy and recall for the minority class.

Balancing class weights in logistic regression is a commonly recommended method for datasets with imbalanced classes, enhancing the classifier's performance by adjusting the weight attributed to each class (Krawczyk, 2016).

The regularization strength in logistic regression is crucial for managing the trade-off between variance and bias, optimizing model performance on unseen data (Xu & Goodacre, 2018). These optimizations aim to fine-tune the model's performance, improving its accuracy, precision, and recall, which are critical metrics in the context of phishing website detection. The regularization parameters are detailed in the subsequent sections.

C. Also known as the inverse of regularization strength, smaller values specify stronger regularization.

Penalty. This parameter specifies the type of regularization applied on the weights. It can be L1, L2, elasticnet, or none.

Fit Intercept (fit_intercept). This parameter indicates whether a constant (a.k.a. bias or intercept) should be added to the decision function.

Optimization Problem Solver (solver). This parameter specifies the algorithm to use in the optimization problem. Options include newton-cg, lbfgs, liblinear, sag, saga.

Max Number of Iteration (max_iter). The maximum number of iterations the optimization method will run.

Support Vector Machines

Dogukan et al. (2017) investigated the use of Support Vector Machines (SVM) in detecting phishing websites. Their research contributes to the variety of detection methods by demonstrating the effectiveness of SVM in this context. The features used to train the SVM model for phishing detection can be obtained from either URLs or HTML source code. Zouina and Outtaj (2017) proposed a unique lightweight system for identifying phishing URLs using SVM and similarity indexes. Their innovative approach provides an efficient solution for phishing detection. Similarly, Dogukan et al. (2017) and Jain and

Gupta (2018) suggested the use of SVMs to identify phishing websites by analyzing features derived from their URLs. Alongside URLs, the source code of websites is a rich source of information about the site's nature. Jain and Gupta (2019) demonstrated that an effective SVM can be trained using features extracted from HTML source code. They emphasized the need for compatibility between a website's identity, structural features, and HTTP transactions. Pan and Ding (2006) argued that any inconsistencies could signal phishing attempts. They trained a robust SVM classifier to detect such anomalies, achieving impressive results. SVMs also pair well with other algorithms. Altaher (2017) achieved remarkable results by integrating SVMs with K-Nearest Neighbors (KNN). Their hybrid KNN-SVM model reached an accuracy rate of 90%.

Optimizing Support Vector Machines (SVMs) necessitates the employment of constrained optimization solvers. Bottou, Lin, et al. (2007) offers a comprehensive review of these solvers and their associated challenges. Anupam and Kar (2021) delved into the optimization of SVMs for phishing detection using nature-inspired algorithms, including the Bat Algorithm, the Firefly Algorithm, the Grey Wolf Optimiser algorithm, and the Whale Optimization Algorithm. In this project, we utilize libsvm (Chang & Lin, 2011), which is the standard optimizer in the sklearn library.

Mathematical Formulation. Support Vector Machines (SVMs) belong to the class of optimal margin classifiers. They classify two classes by building a hyperplane with the maximum margin that separates the two classes. The separating hyperplane is depicted in figure 58 (Boser et al., 1992; Cortes & Vapnik, 1995). From a mathematical perspective, SVMs solve a particular constrained optimization problem:

$$\min_{\mathbf{w},b} \frac{\|w\|^2}{2}$$

$$\text{s.t } y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0 \ \forall i$$

In the above equation, (x_i, y_i) is a data point, $\mathbf{w}$ is the normal to the separating hyperplane, and b is the intercept of the hyperplane. The above formulation of SVMs is well suited for linearly separable classes. However, the data is not linearly separable in

practice. To combat such problems, Cortes and Vapnik (1995) propose a soft variant of SVMs which is defined by the following constrained optimization problem:

$$\min_{\mathbf{w},b} \frac{\|w\|^2}{2} + \frac{C}{n}\sum_i \xi_i^2$$
$$\text{s.t } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \ \forall i$$
$$\xi_i \geq 0 \ \forall i$$

In the aforementioned equation, ξ represents the permissible error for each sample in the training set. The parameter C regulates the overall allowed error on the entire training set, making it a useful tool for regularizing the training to improve the model's generalization on the test set.

In the SVM formulation, it's noteworthy that the features x_i are not necessary if the dot product $w^T x$ can be computed directly. This insight led Boser et al. (1992) to employ a kernel function like Radial Basis Functions (RBF) to calculate the dot product directly. This computation enables the features to be projected into a higher dimensional space without actually calculating the features themselves. Therefore, SVMs are exceptionally efficient at managing high-dimensional data. Another version of SVM is derived using duality theory, which is defined as follows:

$$\min_\alpha \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_i \alpha_i$$
$$\text{s.t } 0 \leq \alpha_i \leq C \text{ and } \sum_i \alpha_i y_i = 0$$

In the above equation, α_i is the Lagrange multiplier and $K(x_i, x_j)$ is the kernel function computing similarity between x_i and x_j. The SVMs can be trained using the algorithm shown in figure 61.

Optimization Parameters. Beyond the w and b parameters of the separating hyperplane, SVM also has additional hyper-parameters that can be fine-tuned using cross-validation.

C. The C parameter in the SVM formulation influences the regularization of the model. Modifying the value of C allows for adjustments in the regularization.

Kernel. The kernel functions in SVM can be selected as *linear, polynomial, RBF, or sigmoid*. The choice of kernel is dependent on the data distribution. All kernels, except the linear one, render the SVM a non-linear classifier.

Degree. When using a polynomial kernel, the degree of the polynomial is a hyperparameter that governs the model's level of non-linearity.

γ. The kernel coefficient, denoted as γ, for RBF, polynomial, and sigmoid determines the influence of each training sample on its surroundings.

Decision Tree

The decision tree is a supervised machine learning algorithm widely utilized for both classification and regression tasks. In the realm of classification tasks, decision trees are often favored over other predictive models due to their ease of interpretation and visualization. They are also efficient in handling both numerical and categorical data (Karim et al., 2023). A key reason why decision trees are particularly suitable for detecting phishing websites is their ability to be trained quickly and consistently provide accurate classification results (Al-Haija & Badawi, 2021). Phishing websites typically exhibit complexity and a diverse range of features. As pointed out by Lakshmi and Vijaya (2012), decision trees are highly scalable and can manage a large quantity of features.

The decision-making process in a decision tree starts with selecting the most appropriate feature to split the data, based on criteria like Information Gain or Gini impurity. Following the split, branches are generated and the algorithm continues iteratively until a stopping criterion, such as a set tree depth or a minimum number of samples per leaf node, is reached. The final tree structure depicts a series of decisions leading to a prediction or outcome located at the leaf nodes (Bansal et al., 2022).

Mathematical Formulation. The root node, also known as the parent node, is the starting point or origin of the Decision Tree. It begins the process of dividing the entire dataset into homogeneous sets. The leaf node is the outcome or final node, beyond which no further division of nodes is possible. When the root node or any other non-leaf node is

divided, the resulting nodes are referred to as child nodes. The act of dividing the root node further into sub-nodes is known as splitting. This division is dependent on the information gain of each feature linked with the dataset. The process of splitting a hierarchy results in a subtree or branch.

The decision tree algorithm assigns class labels to output based on input values. This recursive process continues until a leaf node, which is assigned a class label, is reached. Each node has a defined split condition. Depending on this condition, the input is directed to either the left or right subtree until it reaches a leaf node. The split condition applied at each node should yield homogeneous subsets. In a homogeneous subset, all the records share the same class label.

In the construction of a decision tree, the main goal at each node is to select split conditions that effectively divide the dataset into homogeneous subsets. The impurity of each node is calculated, and the node with the lowest impurity value is chosen for the split. Various metrics, such as Information Gain or Gini Index, aid in calculating the impurity value.

Information Gain is based on the concepts of entropy and the Gini index. Entropy measures the level of randomness or uncertainty in a dataset. If a dataset is entirely homogeneous, its entropy is zero. Conversely, a diverse dataset consisting of multiple classes has high entropy. Low entropy indicates a more ordered dataset, while high entropy indicates a more disordered dataset. The primary aim of the decision tree algorithm is to minimize entropy for effective classification (Tangirala, 2020).

The entropy of dataset S, denoted as H(S), is calculated using the following equation:

$$H(S) = -\sum_{i} P(c_i) \log_2(P(c_i)) \tag{2}$$

Here, $P(c_i)$ represents the probability of an instance belonging to class c_i within the dataset S.

Information Gain (IG) using entropy is the difference between the entropy of a class and the conditional entropy of the class given the selected feature. It can be calculated using the formula :

$$IG(L, f) = H(L) - \sum_{v} \frac{|L_v|}{|L|} H(L_v) \tag{3}$$

Here, $H(L)$ is the entropy of set L before the split. $|L_v|$ represents the size of the subset of L with feature f having value v, and $H(L_v)$ is the entropy of that subset.

The Gini Index is an additional measure used in decision tree algorithms to evaluate the level of disorder or impurity within a dataset. It estimates the frequency of misclassification for a randomly chosen element. A Gini Index of 0 signifies complete purity, where all elements in the dataset are from the same class. Conversely, a Gini Index of 1 signifies the highest level of impurity, which happens when elements are uniformly distributed across all possible classes.

Mathematically, the Gini Index can be calculated using the formula :

$$G(S) = 1 - \sum_{i} \left(P(c_i)\right)^2 \tag{4}$$

Here, $P(c_i)$ represents the probability of an instance belonging to class c_i within the dataset S.

The formula for Information Gain(IG) using the Gini Index as the impurity measure is :

$$IG(L, f) = G(L) - \sum_{v} \frac{|L_v|}{|L|} G(L_v) \tag{5}$$

Here, $G(L)$ is the Gini Index of dataset L before the split. $|L_v|$ represents the size of the subset of L with feature f having value v, and $G(L_v)$ is the Gini Index of that subset.

The decision tree algorithm chooses features based on the highest Information Gain (IG), thereby beginning the process of splitting and building the tree recursively.

There are various types of Decision Tree algorithms, including Iterative Dichotomiser 3 (ID3), C4.5, and CART. As per a study by Patel and Prajapati (2018), the CART algorithm was found to have superior accuracy compared to the others.

As pointed out by Zacharis (2018), the CART algorithm employs a pruning strategy to achieve the optimal tree. The pseudocode for the CART algorithm is depicted in Figure 60, as indicated by Boonamnuay et al. (2018).

Optimization Parameters. Decision trees often have a tendency to overfit, a condition where the model aligns too closely with the training data. While it performs well on the training data, it does poorly on unseen test data. To mitigate overfitting, optimization techniques like pruning and ensemble methods such as Bagging and Boosting can be used.

Pruning is a strategy that reduces the size of the decision tree by removing unnecessary or non-essential parts. This prevents the tree from becoming overly complex and prone to overfitting. The pruning process involves identifying and eliminating noise in the training data, thereby improving the tree's ability to effectively generalize to new and unseen data. As pointed out by Prodromidis and Stolfo (2001), the CART algorithm is based on Cost Complexity Pruning. To facilitate this, the CART algorithm uses a cost-complexity function, CC(T), which balances the tree size (complexity) and the misclassification error (cost) to determine the optimal tree size.

$$CC(T) = \text{err}(T) + \alpha L(T) \tag{6}$$

Here, α serves as the cost-complexity parameter within the penalty term (tree size). When α is 0, the tree grows extensively, resulting in overfitting to the data. Conversely, when α is 1, the tree consists of just a single node, leading to underfitting of the data. The approach involves gradually increasing α from 0 to 1, creating trees at each stage.

Here are some other parameter, which can be adjusted while training the Decision tree using scikit-learn.

Max Depth (max_depth). The maximum depth of the tree. This parameter helps in controlling over-fitting as higher depth will allow the model to learn more about the data which can cause over-fitting.

Min Samples Split (min_sample_split). The minimum number of samples required to split an internal node. This can vary from minimum one sample at each node to maximum of all of the samples at each node.

Min Samples Leaf (min_sample_split). The minimum number of samples need to be at a leaf node, it is very similar to min_samples_split, however, this describe the exact number of sample at the leaf node.

Max Features (max_features). The number of features to consider when looking for the best split.

Max Leaf Nodes (max_leaf_nodes). The maximum number of leaf nodes. This parameter defines the maximum number of leaf node in a tree.

Random Forest

Random Forest is a machine learning approach that leverages the power of multiple decision trees. It trains these trees using the provided data and labels, with each tree focusing on a specific aspect of the data. The outputs from several trained trees are then combined to generate the final prediction. Studies conducted by Subasi et al. (2017) assessed various machine learning techniques and concluded that Random Forest outperformed the others. Similarly, research done by Sindhu et al. (2020) compared different classification algorithms for phishing detection, including Support Vector Machines, Random Forest, and Neural Networks. These studies affirm the consistent superior performance of Random Forest in phishing detection. These are the following points which make Random Forest and ideal candidate for phishing detection.

- Random Forest uses an ensemble learning method that combines multiple decision trees. This approach mitigates the risk of overfitting, enhances the model's generalization capability, and allows it to capture a wide range of patterns and features associated with phishing attacks.

- The model provides a feature importance score for each feature, indicating its

contribution to the model's predictive performance. This process helps in understanding and selecting the most relevant features for distinguishing between legitimate and phishing websites, as demonstrated by Hutchinson et al. (2018) who achieved a performance of 96.5% with only 16 out of 30 features.

- Random Forest works well with noisy data because it builds multiple decision trees and combines their outputs. This aggregation process tends to reduce the impact of outliers and noisy data points, leading to a more reliable model.
- The model can handle imbalanced data well, preventing the model from being biased toward the majority class. This is crucial for maintaining a high detection rate for phishing attacks.
- Random Forest can capture non-linear patterns in the data, making it adept at identifying intricate characteristics associated with phishing attempts.
- The randomness introduced in constructing individual decision trees, such as using a random subset of features and bootstrapped samples, helps prevent overfitting. This deliberate randomness ensures that the model generalizes well to unseen data and performs effectively on new instances.
- Random Forest consistently exhibits high accuracy and is known for its ability to generalize well to different datasets. This feature is essential for phishing detection models, as they need to adapt to evolving attack strategies and new forms of phishing threats.
- Overall, the ability of Random Forest to handle diverse data characteristics effectively, perform feature importance analysis, and its ensemble learning approach make it a powerful and versatile choice for phishing detection.

Mathematical Formulation. A Random Forest is an ensemble of such trees, where each tree is built with a random subset of features and a random subset of the data.

The final prediction of the Random Forest is an average of the predictions of its individual trees. The equation for a Random Forest is:

$$RF(x;\Theta) = \frac{1}{K}\sum_{k=1}^{K} T(x;\Theta_k) \tag{7}$$

In this equation, $RF(x;\Theta)$ represents the Random Forest prediction for input vector x with parameters Θ. K is the number of trees in the forest. $T(x;\Theta_k)$ represents the k-th decision tree's prediction for input vector x with parameters Θ_k.

$$T(x;\Theta) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{8}$$

In this equation, $T(x;\Theta)$ is the decision tree's prediction for input vector x with parameters Θ. $\Theta = {R_m, c_m}_{m=1}^{M}$ are the parameters of the tree, with R_m being the regions and c_m being the outputs in the terminal nodes (leaves). M is the number of terminal nodes. $I(x \in R_m)$ is an indicator function that is 1 if x belongs to region R_m and Θ otherwise.

Optimization Parameters for Random Forest. Random Forest machine learning model has a lot of hyperparameters that can be optimized to tune the model. All parameters are not just for model tuning some help in better interpretation and logging while training.

Number of Estimators (n_estimators). This parameter sets the count of trees in the forest. Every tree in the forest operates as an independent model, and the final prediction is established by amalgamating the predictions from all the trees. Enhancing the value of n_estimators typically boosts the model's performance, as predictions are based on a greater number of *votes* from various tree sets. However, it also escalates the computational expense due to the need for constructing and storing more trees.

Criteria for Node Split (criterion). This parameter determines the method to evaluate the quality of a division in the tree. Available options include *gini* for Gini Impurity and *entropy* for Information Gain, both of which assess the impurity of nodes.

Modifying this parameter can influence the algorithm's strategy for node splitting, potentially resulting in different tree structures.

- **Gini impurity**: The Gini impurity formula estimates the likelihood of incorrectly classifying a member of a randomly selected set by summing the squared probabilities of each class. A lower Gini impurity signifies a more uniform set, and decision tree algorithms strive to reduce this impurity during the splitting process. The Gini impurity for a set S comprising multiple classes is computed as follows:

$$\text{Gini Impurity}(S) = 1 - \sum_{i=1}^{n} (p_i)^2$$

  where n is the number of classes in the dataset, and p_i is the probability of choosing an element of class i from set S.

- **Information Gain**: Information Gain measures the effectiveness of a specific feature in decreasing the uncertainty about the classification of data points. In decision trees, it's commonly used to identify the optimal attribute for splitting a node, with the goal of maximizing the homogeneity of the resulting subsets. This measure is based on the concept of entropy, which denotes the level of disorder or impurity in a dataset. By assessing the decrease in entropy before and after a split based on a particular feature, Information Gain helps decision tree algorithms prioritize features that significantly enhance overall classification accuracy. Kelleher et al. (2015) provided detailed derivations of the use of Information Gain in decision tree creation. Higher Information Gain values suggest that a feature is more informative for decision-making, making it an essential criterion for feature selection during decision tree construction in machine learning. The entropy for a set S consisting of multiple classes is computed as follows:

$$\text{Entropy}(S) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i)$$

where n is the number of classes in the dataset, and p_i is the probability of choosing an element of class i from set S.

Max Depth of Tree (max_depth). This parameter regulates the maximum depth of each tree. If set to None, nodes will expand until all leaves are pure or contain fewer samples than the minimum specified by min_samples_split. By setting a limit on max_depth, you can manage overfitting, as a greater depth would enable the model to learn relationships that are extremely specific to a particular sample.

Minimum Number of Nodes for Split (min_samples_split). This parameter dictates the minimum number of samples necessary to divide an internal node. If the value is set too low, the tree may overfit the data, whereas a value set too high may lead to underfitting. This parameter aids in preventing the model from learning excessively detailed patterns, which are often just noise

Minimum Sample Count at Leaf (min_samples_leaf). This parameter specifies the least number of samples needed at a leaf node. A smaller leaf can make the model more susceptible to noise in the training data. Lower values are generally preferable for problems with imbalanced classes, as there will be few regions where the minority class is in the majority.

Minimum Fraction of Samples at Leaf (min_weight_fraction_leaf). This parameter is similar to min_samples_leaf, but it's defined as a fraction of the total number of weighted instances.

Maximum Features (max_features). This parameter determines the number of features to consider when seeking the best split. If set to *auto*, then max_features equals the square root of the number of features (sqrt(n_features)). Increasing max_features typically enhances the model's performance since each node now has more options for splitting. However, it can also reduce the diversity of individual trees, which is a key strength of the random forest.

Maximum Leaf Nodes (max_leaf_nodes). This parameter sets the maximum number of potential leaf nodes. If set to None, an unlimited number of leaf nodes are allowed. This can help prevent overfitting by limiting the number of leaf nodes in the trees.

Minimum Impurity Decrease (min_impurity_decrease). A node will be divided if this split induces a decrease of the impurity that's greater than or equal to this value. This parameter can be used to balance the model's complexity against its performance.

Bootstrap Samples (bootstrap). This parameter determines whether bootstrap samples are used when building trees. If set to False, the entire dataset is used to construct each tree. It's generally recommended to keep this parameter set to True as it introduces randomness to the model, reducing the likelihood of overfitting the training data.

Out of Bag Samples (oob_score). This parameter is a type of cross-validation strategy. If set to True, the model uses out-of-bag samples to estimate the generalization accuracy, which helps evaluate the model's performance without a separate validation set. Figure 67 illustrates that if oob_score is true, some data samples are set aside for testing or validation.

Number of Jobs (n_jobs). This parameter determines the number of jobs to run in parallel for both the fit and predict operations. If set to -1, the number of jobs is set to the number of cores. This doesn't affect the model's performance but does impact computational efficiency.

These are the most frequently used parameters, but others are also available. The best parameters can vary for each problem and should be selected using techniques like cross-validation. For phishing detection systems, the Random Forest model primarily focuses on the *n_estimators, max_depth, min_samples_split, and min_samples_leaf* hyperparameters, and uses grid search with 5-fold cross-validation to identify the optimal

set of hyperparameters. The work done by HR, MV, et al. (2020) demonstrates how they selected the hyperparameters for the Random Forest and also presents the values in their paper.

XG Boost

XGBoost is a powerful tool used in data science to analyze structured or table-like data. It's a type of gradient boosting machine, which is a machine learning model that works by combining multiple simple models, often decision trees, to create one strong predictive model. In gradient boosting, it start with simple models and keep adding new ones to correct the mistakes made by the current group of models, it stops when model is able to achieve certain accuracy or exhaust the limit of number of new models.

Phishing detection is essentially a classification problem, where the algorithm needs to classify whether a given website is a phishing website or not. XGBoost is particularly effective for such classification problems due to its high performance and accuracy. It can handle large datasets with various features, making it suitable for the complex nature of website data. Moreover, XGBoost's ability to handle missing values can be useful in this context as not all websites will have complete information. Its capacity for regularization helps prevent overfitting effectively, ensuring that the model generalizes well to unseen data, which is crucial for reliable phishing detection.

Mathematical Formulation.

$$\hat{y}_i = \sum_{t=1}^{K} f_t(x_i) \tag{9}$$

This equation is the prediction for a given instance x_i at a given round t. It is the sum of the predictions of all t trees for the instance x_i.$f_t(x_i)$ is the prediction of the t-th tree. K is the number of rounds.

$$L(\Phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{t=1}^{K} \Omega(f_t) \tag{10}$$

This equation is the objective function that XGBoost tries to optimize. It consists

of two parts: the first part $\sum_{i=1}^{n} l(y_i, \hat{y}_i)$ is the sum of a differentiable convex loss function l that measures the difference between the prediction $\hat{y}_i$ and the target y_i for all n instances. The second part $\sum_{t=1}^{K} \Omega(f_t)$ is a regularization term that penalizes the complexity of the model. It is the sum of the complexity of all t trees.

$$\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{11}$$

This equation is the objective for adding the t-th tree. It is similar to the second equation, but the prediction $\hat{y}_i^{(t-1)} + f_t(x_i)$ now includes the prediction of the t-th tree.

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{12}$$

This equation defines the complexity of a tree f_t.T is the number of leaves in the tree, w is the vector of scores on the leaves, γ is a parameter that controls the complexity of the tree (the more leaves, the more complex), and λ is a parameter that controls the L2 regularization on the leaf scores (it penalizes large scores). The term $\frac{1}{2}\lambda ||w||^2$ is the L2 regularization term.

Optimization Parameter for XG Boost. Following hyperparameters can be fine-tuned using techniques like Grid Search or Random Search to find the most optimal values that provide the best performance for the model.

Learning Rate (learning_rate). This parameter shrinks the weights on each step, making the model more robust.

Max Depth (max_depth). This parameter defines the maximum depth of a tree, the same as in Gradient Boosting Machine (GBM) (Natekin & Knoll, 2013).

Minimum Child Weight (min_child_weight). This parameter sets the minimum sum of weights of all observations required in a child.

Gamma. A node is split only when the resulting split provides a positive reduction in the loss function. This parameter governs that decision.

Sub-sample. This parameter denotes the fraction of observations to be randomly sampled for each tree.

Fraction of Randomly Sample Columns (colsample_bytree). This parameter denotes the fraction of columns to be randomly sampled for each tree.

Number of Estimators (n_estimators). This parameter specifies the number of gradient boosted trees, equivalent to the number of boosting rounds.

Model Supports

Environment, Platform, Tools

Hardware. Amazon Web Services (AWS) cloud infrastructure is utilized for a project focused on building a robust Phishing detection system using a Random Forest model. All activities related to dataset storage, model training, and more are performed on AWS. Specifically, an AWS S3 bucket is employed for storing raw and pre-processed data, and an AWS EC2 compute instance of type c5.2xlarge is used for tasks such as feature extraction, model training, testing, data analysis, and model evaluation. This instance supports a high-performance 8-core CPU with a clock speed of 3.0+ GHz, and 16 GB RAM. Additionally, a personal laptop with 8GB RAM is used for local development and establishing connections with the AWS infrastructure. The hardware and software components utilized in the project are detailed in the tables 15.

Software. We used macOS for establishing connections with AWS instances and any other local development. We use VSCode for writing feature extraction code and Jupyter notebooks for training and evaluating the machine learning models. On AWS EC2 instance, we use Ubuntu operating System. Table 16 summarizes the software used in this project.

Tools. Python language is used widely in the field of machine learning. We use Python as our primary language for developing feature extraction and model training scripts. There are numerous libraries available in Python that facilitate data processing and model development. In table 17, we list the libraries that we used during this project.

Model Architecture and Data Flow

The overall system diagram is shown in figure 62. The entire system is hosted on AWS, and it captures the whole system data flow between different components. EC2 is used as the main compute resource which includes feature extraction, data analysis, and model training. Later the trained model will be hosted on the AWS as a web application, which anyone can use through the Streamlit app. All user interactions will be stored in the MySQL database which will later be used for further model refinement. The architecture for each model is described in the following sections.

Logistic Regression. The initial stage of the process involves the collection of a dataset comprising URLs and their corresponding HTML content. This dataset is curated to encompass a broad spectrum of features that could potentially differentiate between legitimate and phishing websites. Following this, the data undergoes a preprocessing phase, wherein missing values are addressed, categorical variables are one-hot encoded, and continuous features are normalized, adhering to the best practices as outlined by Gupta et al., 2019. This ensures the quality of data before the model training.

Subsequently, the dataset is divided into training and testing sets following an 80/20 split, adhering to the methodology proposed by Kohavi, 1995. This approach underscores the significance of validating models on separate data to circumvent overfitting. Concurrently, features are extracted from website URLs and HTML pages, including aspects such as the number of forms, the presence of login forms, and the use of HTTPS from the URLs, among others. These features serve as the foundation for the input of the logistic regression model.

As depicted in figure 63, the architecture for implementing Logistic Regression involves data collection and processing through feature extraction and preprocessing, which includes normalization and one-hot encoding. The Logistic Regression model is trained on this dataset, which is divided into training and test subsets. After the training phase, the model is evaluated using accuracy, precision, recall, and F1 score metrics, culminating in

predictive decision-making for classifying websites as either legitimate or phishing. The training of the logistic regression model on the training set aligns with the methods described by Freedman, 2009, where the model estimates the probability of each instance being phishing or legitimate.

Support Vector Machine (SVM). The model architecture and data flow diagram are illustrated in figure 64. Data, in its raw form, originates from the green node and traverses through the system, passing through several intermediate nodes before arriving at the red nodes. Each node is detailed in the subsequent sections of this section.

In this system, the initial stages involve pre-processing and transforming the raw data. These stages are elaborated in the data engineering chapter. Briefly, pre-processing involves dealing with duplicates, missing values, noise, and inconsistencies in the dataset. The pre-processed data is then subjected to URL and HTML feature extraction during the transformation stage. The features are encoded as necessary and standardized using the z-score method. The transformed data is divided into training and test sets in an 80:20 ratio.

The training set is further divided for 5-fold cross-validation, which results in the validation data comprising 16% of the total dataset. A grid search is performed over the hyperparameters of SVM using these splits. The parameters for the grid search are presented in table 19. The grid search yields the optimal hyperparameters for SVM, which are used to train the final model on the full training set, referred to as the tuned SVM model.

Validation metrics are computed using 5-fold cross-validation on the SVM model trained with the optimal hyperparameters. The test metrics are calculated on the test set using the tuned SVM model.

Decision Tree. During the model development phase, the first step entailed loading the extracted features, stored in JSON files, into a Pandas dataframe. Subsequently, data preprocessing tasks were carried out, which included checking for

duplicates and managing missing values. One-hot encoding was utilized to transform categorical values into numerical values, enabling their interpretation by machine learning models.

The dataset was then partitioned into training (80%) and test sets (20%). Continuous features underwent normalization via standardization. Principal Component Analysis (PCA) was implemented for dimensionality reduction, but it was not employed in the final model as it resulted in a decrease in model accuracy. The class distribution within the data was inspected and visualized. Given the class imbalance between legitimate and phishing websites, data augmentation was performed using SMOTE.

A simple decision tree model with default parameters was then trained, with its fitting time recorded. Hyperparameter tuning was performed using GridSearchCV with 5-fold cross-validation, considering parameters like splitting criteria, maximum tree depth, minimum number of samples required for splitting, minimum samples for leaf creation, and cost complexity pruning alpha. The optimal parameters derived from hyper tuning were used to construct a pruned Decision Tree, and its fitting time was recorded. The final model was evaluated on the unseen data, constituting 20% of the dataset.

The primary objective was to improve the Decision Tree's performance through hyperparameter tuning. Moreover, cost-complexity pruning was executed to mitigate overfitting and enhance the model's generalization. Ultimately, various classification metrics, including accuracy, precision, recall, F1-score, and a confusion matrix, were computed to assess the model's performance. Figure 65 illustrates the data flow and architecture of the proposed Decision Tree classification model.

Random Forest. Random Forest is a supervised machine learning algorithm that employs an ensemble learning technique for model building. The algorithm is rooted in Decision Trees, where each feature assesses how it will segment the data using the Gini Index or Information Gain. Once a feature is chosen, the training data is divided based on this feature, and the process is repeated until all data has been processed. While Decision

Trees construct only one tree, requiring extensive calibration and prone to overfitting, Random Forest expands on this concept by creating multiple weaker trees. Each tree concentrates on a specific aspect of the data, capturing intricate patterns and prioritizing one over the other. When presented with new test data, it is evaluated against all the trees. Ultimately, each result is voted upon by the various weak trees, and the most favored outcome is selected as the final result.

Figure 66 illustrates the high-level system where this model is trained and provides more detailed insights into the data flow through multiple stages before model building. Figure 67 depicts how the data is processed for the Random Forest model, using a sample of features to construct different decision trees. It also demonstrates how the weaker decision trees arrive at the final prediction by employing majority voting for classification and averaging their outputs for regression problems. Figure 68 displays one of the trees generated in Random Forest. Due to space constraints and image clarity, it only shows the top three levels where *num_a_tags* is selected as a top feature with the highest information gain. Figure 69 reveals that *num_a_tags* is the most important feature, hence selected as the first node for the first decision tree.

XGBoost. The data flow diagram for is very similar to the Random Forest as shown in figure 66 but it have a different way of creating the trees which is highlighted in figure 70. XGBoost is an ensemble model which builds a new weak classifier after iteration. The set of weak models constitute the final model and predict the final class (Guo et al., 2020).

Model Comparison and Justification

All models that are implemented are compared in table 18. We proposed Support Vector Machines (SVMs), Logistic Regression, Decision Tree, and Random Forest as our other three models.

The typical architecture of SVM is linear. However, it can be made non-linear with the usage of kernels (Boser et al., 1992). The logistic regression model architecture is linear

while Decision Tree and Random Forest models are highly non-linear.

All the models that we trained can process tabular, image, and audio data. However, they don't support the processing of text and time-series data in their raw form. But, if we convert text and time-series data into tabular form by extracting numerical features, we can train all the models on the resulting data.

The SVMs are computationally intensive. So, they are well suited for small to medium-sized datasets. On the other hand, Logistic Regression, Decision Tree, and Random Forest models can be trained on any sized data with relatively quick training time.

The usage of kernel trick in SVM makes it highly robust to overfitting and underfitting. However, the Logistic Regression model is prone to underfitting, especially when the dataset is not linearly separable. On the other hand, the Decision Tree model is prone to overfitting due to its highly non-linear nature. The Random Forest model is an ensemble of small independent Decision Trees. Thus, it is robust to both overfitting and underfitting (A. Singh et al., 2016).

The SVM and Logistic Regression models are sensitive to feature scales. They require features to be standardized. In addition, they don't support categorical features. Thus, an encoding like one-hot encoding is required to transform categorical features. On the other hand, the Decision Tree and Random Forest models work well with the raw data itself. They are robust to feature scale and can handle categorical features as they are. But, they can still benefit from feature standardization (D. Singh & Singh, 2020).

As we have discussed above, the SVMs are computationally intensive. Hence, they require large training times. However, the Logistic Regression and Decision Tree models are relatively faster to train. The Random Forest model is also computationally intensive. But, it is easily parallelized. So, it can still be trained faster than SVM. All the models can be trained on the CPU. Thus, we don't require any GPUs for training our models.

The SVM model saves all the support vectors and the kernel matrix. Thus, it requires a lot of disk space. On the other hand, the Logistic Regression and Decision Tree

models do not require much disk space they store only the coefficients and the tree structure respectively. The Random Forest model has a higher space requirement than the Decision Tree model since it stores multiple Decision Tree models. Thus, it has high space complexity.

Finally, the SVM kernel trick makes the model efficiently learn non-linearly separable data. However, the usage of SVM is limited by the high computational requirements. The Logistic Regression model produces a simple and easily interpretable model. But, it suffers from underfitting due to its inability to model non-linearly separable data. The Decision Tree model is also easy to interpret and is robust to underfitting as well. However, it suffers from overfitting. The Random Forest model overcomes the overfitting by using an ensemble of Decision Trees and is robust to noise as well. But, as we observed with SVM, it is also computationally intensive.

The data we use for phishing website detection is not linearly separable. Thus, we choose SVM as the model for detecting phishing websites for its ability to model non-linear data effectively and efficiently.

Model Evaluation Methods

There are numerous methods available for the evaluation of classification models, as comprehensively reviewed by Hossin and Sulaiman (2015). This project employs several of these metrics, including the confusion matrix, accuracy, precision, recall, and the F1-score, to assess the models. Each of these evaluation metrics will be discussed in detail in this section. Additionally, a concise introduction to K-fold cross-validation will be provided as it is utilized in the computation of each metric.

The confusion matrix is a tool used to visualize the performance of a model across different classes. It is structured such that the rows correspond to the predicted labels, and the columns correspond to the actual labels, as detailed by Hossin and Sulaiman (2015). An example of a confusion matrix is presented in figure 71.

True Positives (TP) and True Negatives (TN) are defined as the number of samples

correctly classified as positives and negatives respectively. False Positives (FP) and False Negatives (FN) are the number of examples incorrectly classified as positives and negatives respectively.

Accuracy

The accuracy of a model is defined as the ratio of the number of correct predictions over the total number of predictions (Hossin & Sulaiman, 2015). Mathematically, it is defined as follows:

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

In the above equation, TP is the number of True Positives, TN is the number of True Negatives, FP is the number of False Positives, and FN is the number of False Negatives.

Precision

The precision is defined as the ratio of the number of correctly classified positive samples over the total number of samples predicted as positive (Hossin & Sulaiman, 2015). Mathematically, it is defined as follows:

$$\text{precision} = \frac{TP}{TP + FP}$$

In the above equation, TP is the number of True Positives, and FP is the number of False Positives. Intuitively, it shows what proportion of the positive predictions by a classifier are actually positives.

Recall

The recall is defined as the ratio of the number of correctly classified positive samples over the total number of positive samples (Hossin & Sulaiman, 2015). Mathematically, it is defined as follows:

$$\text{recall} = \frac{TP}{TP + FN}$$

In the above equation, TP is the number of True Positives, and FN is the number of False Negatives. Intuitively, it shows what proportion of the positive samples are getting predicted by the classifier.

F1-score

The f1-score is a model metric that combines both precision and recall. It is defined as the harmonic mean of precision and recall (Hossin & Sulaiman, 2015). The following equation defines the f1-score:

$$\text{f1-score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Receiver Operating Characteristics (ROC) Curve and Area Under Curve (AUC)

The Receiver Operating Characteristics (ROC) curve plots the True Positive Rate (TPR) as a function of the False Positive Rate (TPR). We define TPR and FPR as follows:

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

Figure 72 shows the ROC space with five discrete classifiers. A given classifier is considered to be better than others if it is towards the top left of the other classifiers (Fawcett, 2006). The classifiers on the diagonal line represent random classifiers. The random classifiers are no better than a classifier that guesses the class randomly. Thus, in figure 72, the classifier D is the best one and the classifier C is a random classifier.

We define the Area Under Curve (AUC) as the area under the ROC curve. A classifier is considered to be better than other classifiers in terms of AUC if it has a higher AUC than others. Mathematically, it is defined as follows:

$$AUC = \int_0^1 TPR \; d(FPR)$$

Matthews Correlation Coefficient (MCC)

MCC is a measure of the quality of binary classifications. It returns a value between -1 and +1. A coefficient of +1 represents a perfect prediction, 0 random prediction, and -1 is an inverse prediction. MCC is particularly useful when dealing with imbalanced datasets. It considers both sensitivity (true positive rate) and specificity (true negative rate), providing a comprehensive assessment of the classifier's ability to handle imbalances. The formula to calculate MCC is:

$$MCC = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

K-fold Cross Validation

We use K-fold cross-validation to evaluate our models. K-fold cross-validation reduces bias in the computation of model metrics due to the choice of validation set. When we fix the validation set and evaluate the model only once, it could provide us with less reliable results. K-fold cross-validation solves this problem by splitting the training set into K random groups, choosing one group for validation and K - 1 groups for training (James et al., 2013). This process is repeated K times for each split. We combine the model metrics for each of the K runs using an average. In this project, we set K to 5 and report an average of each metric with 5-fold cross-validation. In figure 73, we show the process of 5-fold cross-validation.

Model Validation and Evaluation

This report explores Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and XG Boost machine learning model and how it can be made better with hyperparameter tuning.

Logistic Regression

The evaluation methods used for this model were accuracy, precision, recall, and F1 Score. Accuracy, a primary metric, indicates the overall proportion of correct predictions predicted by the model out of all predictions. The Logistic Regression model achieves an accuracy of 88.32 %, which is the percentage of true positives and true negatives among all classifications.

Precision is a metric that calculates the percentage of accurately predicted positive observations Davis and Goadrich, 2006. In the case of the Logistic Regression model, a precision of 0.9046 for legitimate websites suggests that the model correctly predicts 90.46% of the time when it identifies a website as legitimate. Similarly, a precision of 0.8472 for phishing websites means the model is correct in roughly 84.72

The model's recall for legitimate websites is 0.9091, indicating that it correctly identifies 90.91% of actual legitimate websites. For phishing websites, the model's recall is 0.8402, demonstrating that it correctly identifies 84.02% of actual phishing cases. The F1-score, which represents the balance between precision and recall, is 0.9068 for legitimate websites and 0.8437 for phishing websites, indicating a consistent performance across both classes Powers (2020).

Macro average calculates the metrics for each class independently and then averages them, treating all classes equally. In this case, a macro average precision of 0.8759 indicates an average precision of approximately 87.59% across both classes. The weighted average, which accounts for class imbalance by weighting each class's metrics by the number of true instances, has a precision of 0.8831, suggesting a slightly higher overall precision when considering class distribution Marchal et al. (2014).

Micro averages combine the contributions of all classes to compute the average metric. A micro-average of 0.8832 for precision and recall indicates that the model can correctly predict a label for a randomly chosen instance about 88.32% of the time. This is especially useful for imbalanced datasets as it treats each instance equally, thus reducing

the impact of class imbalances Fawcett, 2006.

Overall, these metrics provide a thorough evaluation of the model's performance, highlighting strengths such as high precision for legitimate websites and areas for improvement such as lower recall for phishing websites Buczak and Guven (2015).

A confusion matrix, denoted as C where $C_{i,j}$ is the number of observations known to be in group i but predicted to be in group j, is a fundamental tool for evaluating a classification model. It offers a detailed breakdown of correct and incorrect predictions by comparing actual and predicted values, forming the basis for other metrics like accuracy, precision, recall, and F1 score.

Confusion matrices are widely used in model evaluation, with literature such as Fawcett, 2006 discussing their role in assessing binary classification models. They are typically represented as a two-dimensional grid, with each axis representing the predicted and actual classifications.

In figure 74, elements like True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) quantify the correct and incorrect predictions. Each cell $C_{i,j}$ represents the count of samples with true label i predicted as label j. They are often visualized as a heatmap for instant analysis.

The ROC curve 75 illustrates the trade-off between sensitivity (detecting phishing sites) and specificity (incorrectly labeling legitimate sites as phishing) as the threshold is adjusted. It is useful for evaluating a model's performance across all classification thresholds, especially when classes are imbalanced, as is often the case in phishing detection where legitimate sites may significantly outnumber phishing sites.

ROC and AUC provide a means to compare different classifiers without being limited by class distribution and the costs of misclassification. With an AUC of 0.94, the classifier demonstrates high effectiveness in differentiating between phishing and legitimate websites. This suggests that the model is adeptly calibrated to amplify the detection of phishing attempts while reducing the misidentification of legitimate services. This balance

is vital in real-world applications of such models, where both undetected phishing attempts and false positives can lead to substantial repercussions Davis and Goadrich (2006).

The elevated AUC value indicates that the classifier possesses considerable predictive capability, an essential aspect in cybersecurity applications due to the high costs associated with misclassification.

Support Vector Machines

In this section, we analyze the performance of our SVM model based on the evaluation metrics described in the previous section. We use grid search with 5-fold cross-validation to tune the hyperparameters of SVM. The grid search parameters are shown in table 19. Since the search space is very large we don't provide validation metrics on each set of parameters.

Baseline. We train SVM with default parameters in sklearn as a baseline model. The default configuration uses a value of 1 for C and the RBF kernel. The γ parameter is set to scale automatically. The metrics obtained using 5-fold cross-validation are displayed in Table 20. The model achieves an accuracy of 0.9116, a precision of 0.8928, a recall of 0.8687, and an f1-score of 0.8806 for the phishing class. It's important to note that we can't compute the confusion matrix using 5-fold cross-validation, so a confusion matrix is provided only for the test set. Also, since our model is a binary classifier, we provide metrics specifically for the phishing class. We also evaluate our model on the test set, with the confusion matrix, precision, recall, and f1-score displayed in figure 76.

The model achieves an accuracy of 0.9095, a precision of 0.8887, a recall of 0.8673, and an f1-score of 0.8879 for the phishing class. The ROC curve for the baseline model is shown in figure 77, where the model achieves an AUC of 0.97.

Tuned Hyperparameters. On conducting a grid search, we find that the best set of hyperparameters consists of a value of 1000 for C, 0.01 for γ, and RBF for the kernel. We provide the metrics with 5-fold cross validation using the best hyperparameters in table 20. The model achieves an accuracy of 0.9279, a precision of 0.9077, a recall of 0.8994, and

an f1-score of 0.9035. We observe that all the metrics see an improvement with hyperparameter tuning.

On evaluating the tuned model on the test set, our model achieves an accuracy of 0.9251, a precision of 0.9026, a recall of 0.8972, and an f1-score of 0.8999. Figure 78 summarizes the above performance numbers. In addition, it also shows the confusion matrix on the test set. We show the ROC curve of the tuned SVM model in figure 79. We observe that the model achieves an AUC of 0.98, which is higher than the baseline model.

Effect of Regularization. We employ the Synthetic Minority Oversampling Technique, also known as SMOTE (Chawla et al., 2002), to oversample the phishing class and improve its performance. SMOTE is used during the data pre-processing stage to expand the dataset with synthetic samples. We use the optimized hyperparameters of SVM to train the model. The model's performance on the test set is depicted in figure 80. We note an increase in the model's recall and f1-score from 0.8972 and 0.8999 to 0.9268 and 0.9012 respectively. The ROC curve for the model is displayed in figure 81. The AUC remains consistent at 0.98, similar to the tuned SVM model. However, there's a decrease in accuracy and precision from 0.9251 and 0.9026 to 0.9238 and 0.8770.

The SVM's C parameter also influences model regularization. As seen in previous sections and table 21, the model performs less effectively with the default value of C compared to the optimized value across all metrics. Hence, C is a crucial factor in regulating the model's performance.

Effect of Dimensionality Reduction. Figure 82 demonstrates the impact of dimensionality reduction on SVM using PCA. It's noticeable that the model's accuracy rises as we incorporate an increasing number of principal components. However, each additional principal component yields diminishing returns. The highest accuracy, 0.9251, is achieved when utilizing all 32 principal components. As observed in previous sections, the maximum accuracy of the tuned model without PCA is also 0.9251. Therefore, to avoid unnecessary computational expenses without any improvement in results, we do not

employ PCA in our final model.

Comparison of Different SVM Models. In this section, we summarize the performances of SVM models with different parameters studied in the previous sections. The table 21 provides a comparison of SVM model performance metrics with different parameters. The Baseline model, without any hyperparameter tuning or SMOTE, achieves an accuracy of 0.9116, precision of 0.8928, recall of 0.8687, and an F1-score of 0.8806. The model with tuned hyperparameters improves on all these metrics, achieving an accuracy of 0.9279, precision of 0.9077, recall of 0.8994, and an F1-score of 0.9035. When SMOTE is applied in addition to hyperparameter tuning, there's a slight dip in accuracy and precision to 0.9238 and 0.8770 respectively, but an increase in recall to 0.9268. The F1-score remains relatively stable at 0.9012, indicating a balance between precision and recall. This suggests the effectiveness of hyperparameter tuning and SMOTE in improving model performance, particularly in terms of recall.

Decision Tree

Baseline Model. The baseline model for the Decision Tree was created using the *DecisionTreeClassifier* from the *Scikit-learn* library. The model was trained using the default parameters. The model achieved a baseline accuracy of 93.38% and the time taken for execution was 0.85 seconds. The confusion matrix and classification report for the Baseline model are shown in Figure 83 and figure 84 shows the ROC curve for the Baseline Decision Tree model.

Hyperparameter Tuned Model. In the next phase, hyperparameter tuning was done using GridSearchCV and 5-fold cross-validation to optimize the Decision Tree model. The optimal parameters found were split criterion as entropy, max_depth as 20, min_samples_leaf as 1, min_samples_split as 2. The training accuracy achieved with these optimal parameters was 93.70% and execution time was 0.66 secs. The execution time of the model was reduced and the prediction accuracy was enhanced. Figure 85 shows the confusion matrix and the classification report for the tuned Decision Tree and figure 86

shows the ROC curve for the tuned Decision Tree.

Comparision of Different Decision Tree Models. The table 22 presents a performance comparison between the Baseline and Hypertuned Decision Tree Models. Across all metrics, the Hypertuned Model exhibits a slight but consistent improvement over the Baseline Model. Precision, recall, and F1-score for both 'Legitimate' and 'Phishing' categories are higher in the Hypertuned Model. The accuracy, micro precision, micro recall/TPR, and micro F1-score also show an increase from 0.9338 in the Baseline Model to 0.9370 in the Hypertuned Model. The Area Under the Curve (AUC) improves from 0.93 to 0.94, indicating better overall performance. Interestingly, despite the improved performance, the Hypertuned Model takes less time (0.66 seconds) to run compared to the Baseline Model (0.85 seconds), demonstrating efficiency along with enhanced accuracy.

Random Forest

Baseline Random Forest Model. The Random Forest Baseline model is a machine learning algorithm that builds multiple decision trees during training and uses the mode of the classes for classification. Without specific hyperparameter tuning, it has shown commendable performance across all metrics. Its Precision of 0.9001 implies that it correctly predicts a positive class 90.01% of the time. A Recall of 0.9079 signifies that it accurately identifies 90.79% of all actual positive instances. With a high Specificity of 0.9403, it effectively identifies negative instances, misclassifying only around 6% of actual negatives. The Matthews Correlation Coefficient (MCC) of 0.8468 indicates a strong positive correlation between observed and predicted binary classifications.

Nonetheless, the baseline model has room for improvement. Its Accuracy of 92.82% and F1-Score of 0.9040 are lower compared to its tuned version and some other models, suggesting more overall classification errors. The F1-Score, a harmonic mean of Precision and Recall, indicates a need to balance identifying as many actual positives as possible while minimizing false positives.

Figure 87 presents the confusion metrics for the baseline model. Despite

outperforming Logistic Regression models, it has a substantial number of incorrect results compared to its tuned version.

Figure 88 and 90 show the Area under the ROC curve for the baseline and tuned model. Comparing both ROC curves, the striking difference is tuned model has a very sharp turn on the left top corner and shows a lower false positive rate.

Tuned Random Forest Model. The Tuned Random Forest model is an enhanced version of the original Random Forest model, improved through a process called GridSearch. This procedure involves testing multiple combinations of parameters, using cross-validation to determine which provides the best performance. The optimized model has significantly outperformed the baseline model in all metrics. Table 24 have all the grid search parameter for Random Forest. Best parameters after the grid search are $criterion : entropy, max\_depth : None, min\_samples\_leaf : 1, min\_samples_split : 2, n\_estimators : 200, bootstrap : true$ Its Precision, Recall, F1-Score, and Accuracy all surpass 0.96, demonstrating high accuracy in identifying positive instances and overall classification performance. With the highest Specificity of 0.9298 among all models, it excels at identifying negative instances, thus reducing the number of false positives.

However, despite its high MCC of 0.9151, which indicates a strong correlation between observed and predicted binary classifications, it's outperformed by the XG Boost model. This suggests that while the tuned Random Forest model is highly effective, other models may yield superior performance for this specific dataset.

Figure 89 displays the confusion metrics for the baseline model. While it outperforms Logistic Regression models, it has a significant number of incorrect results compared to its tuned counterpart.

Comparision of Different Random Forest Models. The table presents a notable disparity in the performance of the RandomForest baseline model and the tuned model. The baseline model, which was trained without any hyperparameter optimization, displays a precision of .9001, recall of .9079, F1 score of .9040, accuracy of 92.82%,

specificity of .9403, and a Matthews Correlation Coefficient (MCC) of 0.8468. While these metrics are indeed remarkable, they are further enhanced by the tuned model.

The Random Forest tuned model, trained using GridSearch for optimal hyperparameter selection, exhibits a considerable improvement in all metrics. The precision rises to .9642, recall to .9726, F1 score to .9684, accuracy to 96.03%, specificity to .9298, and the MCC to 0.9151. This improvement in all metrics underscores the efficacy of hyperparameter tuning, leading to a more precise and dependable model. The MCC's increase from 0.8468 to 0.9151 indicates a superior quality of classification in the tuned model, with a higher true positive rate and a lower false positive rate.

Figure 88 and 90 represent the Area Under the ROC curve for both the baseline and tuned models. A comparison of the two ROC curves reveals a significant difference; the tuned model exhibits a sharp turn at the left top corner, indicating a lower false positive rate.

Figure 91 displays a Precision vs Recall graph for two models: the Random Forest Baseline and Tuned Model. This graph visually portrays the performance of each model in terms of precision (the proportion of true positive results) and recall (the capacity to detect all positive instances). A larger area under the curve signifies a model with high precision (few false positives) and high recall (efficient at identifying positives). The Tuned Random Forest Model clearly outperforms, as its curve sharply bends in the top right corner, maximizing the area under the curve, indicating high precision and recall. Conversely, the Baseline Model's curve is more gradual, suggesting lower precision and recall. The smoother curves result from a large volume of test data. Ideally, a model with high precision and recall will yield numerous accurately labeled results, and the Tuned Model is closer to this ideal than the Baseline Model, making it the superior choice.

Figure 92 displays a calibration plot for the Random Forest Baseline and Tuned Models. This plot is a visual tool that provides insights into the performance of the models. The Baseline model consistently underestimates until the 0.65 mark, after which it

overestimates, indicating a need for calibration. However, it's noteworthy that for lower predictions, it closely aligns with the diagonal line, which signifies perfect calibration. On the other hand, the Tuned model's calibration plot exhibits asymmetry and a shift in trend at the 50% mark. While the slope stays near 1, it strays from the central line, demonstrating a symmetric pattern. This suggests a level of miscalibration that needs addressing. Correcting these miscalibrations is of utmost importance for both models. Ensuring that the predicted probabilities accurately align with the true probabilities across all predictions is a key aspect of model calibration. This process not only enhances the model's reliability but also bolsters its dependability for decision-making. By identifying and rectifying these miscalibrations, we can improve the models' accuracy and reliability, making them more effective tools for prediction.

Different Machine Learning Model Performance Comparison

In this section, Logistic Regression, SVM, Decision Tree, and Random Forest are compared in terms of evaluation metrics. Table 25 provides a comparison, and all models performed well, but there were significant differences in their performance.

Table 25 summarizes the evaluation metrics for each model in terms of accuracy, precision, recall, and f1-score. Random Forest model beats all of our other proposed models on each metric, achieving an accuracy of 0.9603, a precision of 0.9537, a recall of 0.9398, and an f1-score of 0.9467. This performance is due to the ensemble nature of the Random Forest model. The ensemble models generally perform better than a single model since they are robust to overfitting and can model highly non-linear decision boundaries (Sagi & Rokach, 2018).

Random Forest shows strong performance, particularly in recall (.9726), where it outperforms all other models. This high recall suggests that Random Forest is particularly effective at identifying the majority of phishing sites, reducing the risk of missing potential threats. Its precision (.9642) and F1-score (.9684) are also high, indicating a good balance of correctly identifying phishing sites and minimizing false positives. Random Forest's

accuracy is 96.03%, the second-highest among the models, which suggests it can correctly classify both phishing and non-phishing sites. Its specificity (.9298) and MCC (0.9151) scores are also high, further attesting to its overall strong performance. Its ability to effectively identify the majority of phishing sites makes it a reliable model for applications where failing to identify a positive instance (a phishing site, in this case) could have serious consequences.

Given all these metrics, XGBoost (T. Chen & Guestrin, 2016) emerges as the most effective model for phishing detection. Table 25 highlights its superior performance across all metrics and indicates its ability to accurately and reliably identify phishing sites while minimizing both types of errors (false positives and false negatives). XGBoost model achieves the highest accuracy of 0.9646. It also achieves the highest recall and highest f1-score of 0.9542 and 0.9646 respectively. However, the Random Forest model retains the highest precision. However, the difference between the precision of the Random Forest model and the XGBoost model is quite small.

Conclusion

In this exploration, the fundamental concepts and methodologies underpinning phishing detection were established. Attention was placed on the vital role of feature engineering, dataset preprocessing, and model selection. The dynamic and adaptive nature of phishing attacks was also highlighted, underscoring the importance of continuously evolving and required innovative solutions.

The subsequent sections delved into a detailed analysis of model evaluation metrics and the comparative performance of various machine learning models typically employed in phishing detection. These models encompassed Logistic Regression, SVM, Decision Trees, Random Forest, and XG Boost (both baseline and tuned). Evaluation metrics ranged from precision, recall, specificity, f1 score, accuracy, Matthews Correlation Coefficient (MCC), and precision-recall curves. The results indicated that XG Boost emerged as the most effective model for phishing detection, closely followed by the Random Forest model. These

models exhibited robust performance across diverse metrics, positioning them as potent tools for the precise identification of phishing websites.

In summation, this project underscores the critical nature of phishing detection in the context of cybersecurity. It emphasizes the significance of employing effective machine learning models and rigorous evaluation metrics to combat the continuous and evolving threat of phishing attacks. While the exploration primarily focused on traditional machine learning models, the field signals further investigation into advanced techniques to fortify detection accuracy and resilience. As cyber threats continue to advance, ongoing research and innovation in phishing detection remain imperative to fortify online security and protect users from an ever-growing spectrum of threats.

Limitations

This project's findings and model evaluations are contingent on the specific dataset used, potentially limiting their generalizability to different datasets. While the dataset employed was comprehensive, it may not encompass the full spectrum of real-world phishing attacks. Therefore, the performance of the models may vary when applied to datasets with distinct characteristics or evolving phishing strategies.

Additionally, this research primarily focused on conventional machine learning models, with minimal exploration of advanced techniques like deep learning and reinforcement learning. These emerging methods could potentially offer greater accuracy and adaptability, and their potential remains untapped. As the field of cybersecurity continually evolves, future research should consider these cutting-edge approaches to enhance detection capabilities further.

Future Scope

Phishing detection is a dynamic field that constantly evolves with the changing landscape of cyber threats. Future work on this problem can be divided into two sections. First improving the algorithms used in this research. Techniques such as more advanced feature engineering by employing Natural Language Processing (NLP), better handling of

class imbalance, more extensive hyperparameter tuning, and regular model updating with fresh data could enhance their performance. Second explore more sophisticated models and techniques, particularly in the realm of deep learning and reinforcement learning. Deep learning models, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), can potentially capture complex patterns and dependencies in data that traditional machine learning models might miss. These models could be particularly effective if trained on large, diverse datasets that accurately represent the wide variety of phishing strategies employed by attackers. The ultimate goal is to build a system that can adapt to the ever-evolving threats and protect users effectively in real-time.

References

Abusaimeh, H., & Alshareef, Y. (2021). Detecting the phishing website with the highest accuracy. *TEM Journal*, *10*(2), 947.

Ahmed, D. S., Hussein, A. P. D. K. Q., & Allah, H. A. A. A. (2022). Phishing websites detection model based on decision tree algorithm and best feature selection method. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *13*(1), 100–107.

Al-Haija, Q. A., & Badawi, A. A. (2021). Url-based phishing websites detection via machine learning. *2021 International Conference on Data Analytics for Business and Industry (ICDABI)*, 644–649. https://doi.org/10.1109/ICDABI53623.2021.9655851

Aljofey, A., Jiang, Q., Rasool, A., Chen, H., Liu, W., Qu, Q., & Wang, Y. (2022). An effective detection approach for phishing websites using url and html features. *Scientific Reports*, *12*(1), 8842.

Alkhalil, Z., Hewage, C., Nawaf, L., & Khan, I. (2021). Phishing attacks: A recent comprehensive study and a new anatomy. *Frontiers in Computer Science*, *3*. https://doi.org/10.3389/fcomp.2021.563060

Altaher, A. (2017). Phishing websites classification using hybrid svm and knn approach. *International Journal of Advanced Computer Science and Applications*, *8*(6).

Anupam, S., & Kar, A. K. (2021). Phishing website detection using support vector machines and nature-inspired optimization algorithms. *Telecommunication Systems*, *76*(1), 17–32.

Ariyadasa, S., Fernando, S., & Fernando, S. (2021). 'phishing websites dataset.

Ariyadasa, S., Fernando, S., & Fernando, S. (2022). Phishrepo: A seamless collection of phishing data to fill a research gap in the phishing domain. *International Journal of Advanced Computer Science and Applications*, *13*(5).

Bansal, M., Goyal, A., & Choudhary, A. (2022). A comparative analysis of k-nearest neighbor, genetic, support vector machine, decision tree, and long short term memory algorithms in machine learning. *Decision Analytics Journal*, *3*, 100071.

Basit, A., Zafar, M., Liu, X., Javed, A. R., Jalil, Z., & Kifayat, K. (2021). A comprehensive survey of ai-enabled phishing attacks detection techniques. *Telecommunication Systems*, *76*, 139–154.

Basnet, R. B., & Sung, A. H. (2014). Learning to detect phishing webpages. *J. Internet Serv. Inf. Secur.*, *4*(3), 21–39.

Basnet, R. B., Sung, A. H., & Liu, Q. (2011). Rule-based phishing attack detection. *International conference on security and management (SAM 2011), Las Vegas, NV*.

Basnet, R. B., Sung, A. H., & Liu, Q. (2012). Feature selection for improved phishing detection. *Advanced Research in Applied Artificial Intelligence: 25th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2012, Dalian, China, June 9-12, 2012. Proceedings 25*, 252–261.

Boonamnuay, S., Kerdprasop, N., & Kerdprasop, K. (2018). Classification and regression tree with resampling for classifying imbalanced data. *International Journal of Machine Learning and Computing*, *8*(4), 336–340.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, 144–152.

Bottou, L., Lin, C.-J., et al. (2007). Support vector machine solvers. *Large scale kernel machines*, *3*(1), 301–320.

Buber, E., Demir, Ö., & Sahingoz, O. K. (2017). Feature selections for the machine learning based detection of phishing websites. *2017 international artificial intelligence and data processing symposium (IDAP)*, 1–5.

Buczak, A. L., & Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, *18*(2), 1153–1176.

Budach, L., Feuerpfeil, M., Ihde, N., Nathansen, A., Noack, N., Patzlaff, H., Naumann, F., & Harmouch, H. (2022). The effects of data quality on machine learning performance. *arXiv preprint arXiv:2207.14529*.

Chang, C.-C., & Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, *2*(3), 1–27.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, K.-T., Chen, J.-Y., Huang, C.-R., & Chen, C.-S. (2009). Fighting phishing with discriminative keypoint features. *IEEE Internet Computing*, *13*(3), 56–63.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Clark, W. (1922). *The gantt chart: A working tool of management*. Ronald Press Company.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*, 273–297.

Dahouda, M. K., & Joe, I. (2021). A deep-learned embedding technique for categorical features encoding. *IEEE Access*, *9*, 114381–114391.

Davis, J., & Goadrich, M. (2006). The relationship between precision-recall and roc curves. *Proceedings of the 23rd international conference on Machine learning*, 233–240.

Dobbin, K. K., & Simon, R. M. (2011). Optimally splitting cases for training and testing high dimensional classifiers. *BMC medical genomics*, *4*(1), 1–8.

Dogukan, A., Abdulwakil, A., & Aydin, M. A. (2017). Detecting phishing websites using support vector machine algorithm. *PressAcademia Procedia*, *5*(1), 139–142.

Dou, Z., Khalil, I., Khreishah, A., Al-Fuqaha, A., & Guizani, M. (2017). Systematization of knowledge (sok): A systematic review of software-based web phishing detection. *IEEE Communications Surveys & Tutorials*, *19*(4), 2797–2819. https://doi.org/10.1109/COMST.2017.2752087

Ebbu2017. (2017). *Phishing dataset*.

Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., & Tabona, O. (2021). A survey on missing data in machine learning. *Journal of Big Data*, *8*(1), 1–37.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, *27*(8), 861–874.

Freedman, D. A. (2009). *Statistical models: Theory and practice*. Cambridge University Press.

Guo, R., Zhao, Z., Wang, T., Liu, G., Zhao, J., & Gao, D. (2020). Degradation state recognition of piston pump based on iceemdan and xgboost. *Applied Sciences*, *10*(18), 6593.

Gupta, S., Bhatia, M. P. S., & Kumar, N. (2019). Data preprocessing techniques for classification without data loss. *International Journal of Computer Science Issues (IJCSI)*, *16*(1), 18–24.

Hamdani, F.-E., Quintero, I. A. Q., Enjolras, M., Camargo, M., Monticolo, D., & Lelong, C. (2022). Agile supply chain analytic approach: A case study combining agile and crisp-dm in an end-to-end supply chain. *Supply Chain Forum: An International Journal*, 1–15.

Hara, M., Yamada, A., & Miyake, Y. (2009). Visual similarity-based phishing detection without victim site information. *2009 IEEE Symposium on Computational Intelligence in Cyber Security*, 30–36.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (Vol. 2). Springer.

Hossin, M., & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, *5*(2), 1.

HR, M. G., MV, A., et al. (2020). Development of anti-phishing browser based on random forest and rule of extraction framework. *Cybersecurity*, *3*(1), 1–14.

Hutchinson, S., Zhang, Z., & Liu, Q. (2018). Detecting phishing websites with random forest. *Machine Learning and Intelligent Communications: Third International Conference, MLICOM 2018, Hangzhou, China, July 6-8, 2018, Proceedings 3*, 470–479.

Jagadeesan, S., Chaturvedi, A., & Kumar, S. (2018). Url phishing analysis using random forest. *International Journal of Pure and Applied Mathematics*, *118*(20), 4159–4163.

Jain, A. K., & Gupta, B. (2018). Phish-safe: Url features-based phishing detection system using machine learning. *Cyber Security: Proceedings of CSI 2015*, 467–474.

Jain, A. K., & Gupta, B. B. (2019). A machine learning based approach for phishing detection using hyperlinks information. *Journal of Ambient Intelligence and Humanized Computing*, *10*, 2015–2028.

James, G., Witten, D., Hastie, T., Tibshirani, R., et al. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Joshi, Y., Saklikar, S., Das, D., & Saha, S. (2008). Phishguard: A browser plug-in for protection from phishing. *2008 2nd International Conference on Internet Multimedia Services Architecture and Applications*, 1–6.

Karamizadeh, S., Abdullah, S. M., Manaf, A. A., Zamani, M., & Hooman, A. (2013). An overview of principal component analysis. *Journal of Signal and Information Processing*, *4*(3B), 173.

Karim, A., Shahroz, M., Mustofa, K., Belhaouari, S. B., & Joga, S. R. K. (2023). Phishing detection system through hybrid machine learning based on url. *IEEE Access*, *11*, 36805–36822. https://doi.org/10.1109/ACCESS.2023.3252366

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: Algorithms. *Worked examples, and case studies.*

Khonji, M., Iraqi, Y., & Jones, A. (2013). Phishing detection: A literature survey. *IEEE Communications Surveys amp Tutorials*, *PP*, 1–31. https://doi.org/10.1109/SURV.2013.032213.00009

Koc, A., & Tansel, A. U. (2011). A survey of version control systems. *ICEME 2011.*

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *IJCAI*, *14*, 1137–1145.

Kolbaşi, A., & Ünsal, A. (2019). A comparison of the outlier detecting methods: An application on turkish foreign trade data. *J. Math. Stat. Sci.*, *5*, 213–234.

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence*, *5*(4), 221–232.

Krishna, G. S., Supriya, K., & Rao, K. M. (2022). Selection of data preprocessing techniques and its emergence towards machine learning algorithms using hpi dataset. *2022 IEEE Global Conference on Computing, Power and Communication Technologies (GlobConPT)*, 1–6.

Lakshmi, V. S., & Vijaya, M. (2012). Efficient prediction of phishing websites using supervised learning algorithms. *Procedia Engineering*, *30*, 798–805.

Li, P., Rao, X., Blase, J., Zhang, Y., Chu, X., & Zhang, C. (2021). Cleanml: A study for evaluating the impact of data cleaning on ml classification tasks. *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 13–24.

Li, Y., Yang, Z., Chen, X., Yuan, H., & Liu, W. (2019). A stacking model using url and html features for phishing webpage detection. *Future Generation Computer Systems*, *94*, 27–39.

Maćkiewicz, A., & Ratajczak, W. (1993). Principal components analysis (pca). *Computers & Geosciences*, *19*(3), 303–342.

Mallikharjuna Rao, K., Saikrishna, G., & Supriya, K. (2023). Data preprocessing techniques: Emergence and selection towards machine learning models-a practical review using hpa dataset. *Multimedia Tools and Applications*, 1–20.

Marchal, S., François, J., State, R., & Engel, T. (2014). Phishscore: Hacking phishers' minds. *10th International Conference on Network and Service Management (CNSM) and Workshop*, 46–54.

Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, *7*, 21.

OpenPhish. (n.d.). *Openphish - phishing intelligence*. https://openphish.com/

Pan, Y., & Ding, X. (2006). Anomaly based web phishing page detection. *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, 381–392.

Parsons, V. L. (2014). Stratified sampling. *Wiley StatsRef: Statistics Reference Online*, 1–11.

Patel, H. H., & Prajapati, P. (2018). Study and analysis of decision tree based classification algorithms. *International Journal of Computer Sciences and Engineering*, *6*(10), 74–78.

Patro, S., & Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

Pedersen, R., & Schoeberl, M. (2006). An embedded support vector machine. *2006 International Workshop on Intelligent Solutions in Embedded Systems*, 1–11.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, *12*, 2825–2830.

PhishTank. (n.d.). *Phishtank: Join the fight against phishing*. https://phishtank.org/

Potdar, K., Pardawala, T. S., & Pai, C. D. (2017). A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, *175*(4), 7–9.

Powers, D. M. (2020). Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.

Prakash, P., Kumar, M., Kompella, R. R., & Gupta, M. (2010a). Phishnet: Predictive blacklisting to detect phishing attacks. *2010 Proceedings IEEE INFOCOM*, 1–5. https://doi.org/10.1109/INFCOM.2010.5462216

Prakash, P., Kumar, M., Kompella, R. R., & Gupta, M. (2010b). Phishnet: Predictive blacklisting to detect phishing attacks. *2010 Proceedings IEEE INFOCOM*, 1–5.

Prodromidis, A. L., & Stolfo, S. J. (2001). Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, *3*, 449–469.

Punmia, B., & Khandelwal, K. (2002). *Project planning and control with pert & cpm*. Firewall media.

Rodriguez-Galiano, V. F., Sanchez-Castillo, M., Dash, J., Atkinson, P. M., & Ojeda-Zujar, J. (2016). Modelling interannual variation in the spring and autumn land surface phenology of the european forest. *Biogeosciences*, *13*(11), 3305–3317.

Safi, A., & Singh, S. (2023). A systematic literature review on phishing website detection techniques. *Journal of King Saud University-Computer and Information Sciences*.

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1249.

Sahoo, D., Liu, C., & Hoi, S. C. (2017). Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*.

Sánchez-Paniagua, M., Fernández, E. F., Alegre, E., Al-Nabki, W., & Gonzalez-Castro, V. (2022). Phishing url detection: A real-case scenario through login urls. *IEEE Access*, *10*, 42949–42960.

Saria, S., & Subbaswamy, A. (2019). Tutorial: Safe and reliable machine learning. *arXiv preprint arXiv:1904.07204*.

Saseendran, A. T., Setia, L., Chhabria, V., Chakraborty, D., & Barman Roy, A. (2019). Impact of noise in dataset on machine learning algorithms. *Machine Learning Research*, 0–8.

Shahrivari, V., Darabi, M. M., & Izadi, M. (2020). Phishing detection using machine learning techniques. *arXiv preprint arXiv:2009.11116*.

Sindhu, S., Patil, S. P., Sreevalsan, A., Rahman, F., & AN, M. S. (2020). Phishing detection using random forest, svm and neural network with backpropagation. *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, 391–394.

Singh, A., Thakur, N., & Sharma, A. (2016). A review of supervised machine learning algorithms. *2016 3rd international conference on computing for sustainable global development (INDIACom)*, 1310–1315.

Singh, D., & Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, *97*, 105524.

Smith, J., et al. (2021). Enhancing phishing detection using ensemble techniques. *Journal of Cybersecurity*, *7*(2), 113–121.

Subasi, A., Molah, E., Almkallawi, F., & Chaudhery, T. J. (2017). Intelligent phishing website detection using random forest classifier. *2017 International conference on electrical and computing technologies and applications (ICECTA)*, 1–5.

Tangirala, S. (2020). Evaluating the impact of gini index and information gain on classification using decision tree classifier algorithm. *International Journal of Advanced Computer Science and Applications*, *11*(2), 612–619.

Tausworthe, R. C. (1979). The work breakdown structure in software project management. *Journal of Systems and Software*, *1*, 181–186.

Teraguchi, N., & Mitchell, J. C. (2004). Client-side defense against web-based identity theft. *Computer Science Department, Stanford University. Available: http://crypto. stanford. edu/SpoofGuard/webspoof. pdf.*

Thomas, J. (2018). Individual cyber security: Empowering employees to resist spear phishing to prevent identity theft and ransomware attacks. *Thomas, JE (2018). Individual cyber security: Empowering employees to resist spear phishing to prevent identity theft and ransomware attacks. International Journal of Business Management*, *12*(3), 1–23.

Verma, R. M., Zeng, V., & Faridi, H. (2019). Data quality for security challenges: Case studies of phishing, malware and intrusion detection datasets. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2605–2607.

Whittaker, C., Ryner, B., & Nazif, M. (2010). Large-scale automatic classification of phishing pages.

Xu, Y., & Goodacre, R. (2018). On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. *Journal of analysis and testing*, *2*(3), 249–262.

Zacharis, N. Z. (2018). Classification and regression trees (cart) for predictive modeling in blended learning. *IJ Intelligent Systems and Applications*, *3*(1), 9.

Zhang, H., Liu, G., Chow, T. W., & Liu, W. (2011). Textual and visual content-based anti-phishing: A bayesian approach. *IEEE transactions on neural networks*, *22*(10), 1532–1546.

Zhao, Y., Li, L., Wang, H., Cai, H., Bissyandé, T. F., Klein, J., & Grundy, J. (2021). On the impact of sample duplication in machine-learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *30*(3), 1–38.

Zouina, M., & Outtaj, B. (2017). A novel lightweight url phishing detection system using svm and similarity index. *Human-centric Computing and Information Sciences*, *7*(1), 1–13.

Table 1

Project Deliverables and Timelines.

Deliverable	Description	Due date
Project Proposal	Proposed research problem with background information, data sources, and SWOT analysis.	September 25, 2023
Work Breakdown Structure (WBS)	Project components and work packages of the six phases of CRISP-DM.	October 4, 2023
Effort Estimates	Estimation of effort, resource allocation for each task.	October 11, 2023
Gantt Chart		October 11, 2023
PERT Chart	Chart showing all activities with dependencies and critical paths.	October 16, 2023
Introduction	Introduction chapter of the final report.	October 17, 2023
Data and Project Management Plan	Chapter two of the final report.	October 30, 2023
Data Collection Plans	Data sources, quantity and examples.	November 6, 2023
Data Engineering Plans	Data cleaning and transformation plans.	November 13. 2023
Data Engineering Report	Data engineering chapter of the final report.	November 21, 2023
Abstract	Abstract of the project research report.	November 26, 2023
Research Report Presentations	Presentation slides for the project.	November 27, 2023
Individual Research Paper (Model Development)	Individual research report with the modeling section.	December 8, 2023
Final Research Report	Project research report with all the chapters.	December 10, 2023

Table 2

Models and Tools from the Literature Survey

Current Systems	Result	Comment
Basnet et al. (2011)	95% accuracy	Rules integrated with Logistic Regression
Abusaimeh and Alshareef (2021) using Decision Tree and SVM combined	98.52% accuracy	Dataset information not clear, Used SVM and Decision Tree Combined
Ahmed et al. (2022)	98.80% accuracy	UCI Machine Learning Repository data containing 11000+ instances and 20 features
Zouina and Outtaj (2017) using SVM and features extracted from URL	95.80% accuracy	Data had only 2000 records from PhishTank
Dogukan et al. (2017) using SVM classifier	95% accuracy	PhishTank data
PhishNet (Prakash et al., 2010)	FP 3%, FN 5%	URL Blacklisting
SpoofGuard (Teraguchi & Mitchell, 2004)	62% accuracy	Rule-Based System
Fighting Phishing with Discriminative Keypoint Features (Chen et al., 2009)	95% - 98% accuracy	Visual Similarity-based system for popular web pages like eBay, Bank of America, etc.

Table 3

Hardware Requirements for Project

Hardware	Specification	Usage
AWS S3 Bucket (Region - us-west)	15 GB	Store the Web-page HTML representation
AWS S3 Bucket (Regions us-west)	200 MB	Web Page and the Flag for phishing or not phishing
EC2 compute (Region us-west)	16 GB RAM, 8 CPU c5.2xlarge	Data Processing And Model Training
EC2 compute (Region us-west)	8 GB RAM, 4 CPU	Model Hosting App web-server for hosting the App
AWS RDS (MySQL) (Region us-west)	2 GB RAM, 2 CPU	MySQL Database for the App usage statistics
Personal Computers	8 GB RAM, 2 CPU	Local Development Tasks

Table 4

Software and Package Requirement

Software	Version	Usage
Python	3.6 or higher	Used for all the data transformation and engineering tasks
NumPy	1.0 or higher	Feature Engineering
Pandas	1.4 or higher	Feature Engineering
SciKit Learn	1.0 or higher	Model Building and other statistical analysis
Jupyter Notebook	5.0 or higher	Model Building and Feature Engineering
MySQL	8.0 or higher	Store Labels and App Usage statistics
MLflow or Sagemaker	2.0 or higher	Model Management
Linux	20 or higher	Operating System for all the EC2 servers

Table 5

Tools required for Project

Tools	License	Usage
JIRA	Free Limited Version	Tool to manage our project
GitHub	Free with Limited Storage	Code Repository
Streamlit	Free Open Source	App Hosting
Google Office Suite	Free Open Source	Presentation and Documentation

Table 6

Cost Estimation for Project

Resource	Duration	Cost Per Month	Total Cost
AWS S3 Bucket	2 Months	$ 0.0[a]	$ 0.0
EC2 compute (c5.2xLarge[b])	2 months	$ 244.8	$ 489.6
EC2 compute (c5.xLarge[c])	2 months	$ 144.4	$ 244.8
AWS RDS (db.m5.large[d])	1 month	$ 124.83	$ 124.83
Linux	3 months	Free	$ 0.0
Python	3 months	Free	$ 0.0
NumPy	3 months	Free	$ 0.0
Pandas	3 months	Free	$ 0.0
SciKit Learn	3 months	Free	$ 0.0
Jupyter Notebook	3 months	Free	$ 0.0
Linux	3 months	Free	$ 0.0
JIRA	3 Months	Free	$ 0.0
GitHub	2 Months	Free	$ 0.0
Streamlit	1 Month	Free	$ 0.0
Total			$ 859.23

Note. All cost estimates are approximate.

[a]$ pricing for the S3 buckets depends on the number of requests and TB of data as our data scale is not high so estimating it as 0

[b]$ 0.34/hr

[c]$ 0.17/hr and the cost is dependent also on the number of requests

[d]suggested by the AWS price estimator.

Table 7

Raw Dataset Quantity.

Data type	Quantity
Legitimate websites	50,000
Phishing websites	30,000

Table 8

Schema of the index table.

Column name	Description
rec_id	Record id
url	Website URL
website	HTML filename
result	0 if the website is legitimate or 1 if it is not
created_date	Webpage creation date

Table 9

List of Features Extracted from URL

Feature Name	Details
url_length	Character Length of the URL.
num_subdomains	Number of Sub-Domains in the URL.
uses_https	Is the URL http or https?
contains_ip	Does the URL contains IP address instead of domain name?
contains_phishing_keywords	Does the URL contains popular phishing related keywords?
contains_at_symbol	Does the URL contains @ character?
url_depth	Number of the level of depth in the URL.
is_shortened_url	Is the URL is shortened?
is_punycode	Does the URL contain puny code.
has_redirection	Is the URL redirecting to another web page?
is_domain_valid	Is the domain of the web page valid?
days_until_expiration	Days remaining for the web page DNS expiration?
registration_length	How long has this domain been registered for?

Table 10

List of Features Extracted from HTML Page

Feature Name	Details
num_forms	Number of Forms in the page
num_username_fields	Number of username fields in the page
num_password_fields	Number of password fields in the page
num_hidden_fields	Number of hidden fields in the page
form_action	Action on the forms
form_autocomplete	Is form auto-complete
external_links_count	Number of External Links from the page
login_form_present	Is the Login form present on the page
javascript_redirects_present	Is javascript redirection on the page
iframes_count	Number of iframes on the page
num_obfuscated_scripts	Number of Obfuscated scripts on the page
external_js_inclusion	Is page using external javascript
num_inline_styles	Number of inline styles
num_script_tags	Number of script tags
num_iframe_tags	Number of iframe tags
num_img_tags	Number of images on the page
num_a_tags	Number of links

Table 11

Raw dataset.

Dataset type	URL Quantity	HTML source code quantity
Phishing websites	30,000	30,000
Legitimate websites	50,000	50,000

Table 12

Transformed dataset before augmentation.

Dataset type	Number of Rows	Number of Columns
Phishing websites	30,000	33
Legitimate websites	50,000	33

Table 13

Transformed dataset after augmentation.

Dataset type	Rows	Columns
Phishing websites	40,000	33
Legitimate websites	40,000	33

Table 14

Training, validation, and test sets.

Dataset type	Quantity
Training	51,200
Validation	12,800
Test	16,000

Table 15

Hardware Used for Model Creation

Hardware	Specification	Usage
AWS S3 Bucket (Region: us-west)	15 GB	Storage the HTML source codes
AWS S3 Bucket (Region: us-west)	200 MB	Web Page and the label for phishing or legitimate
EC2 compute (Region us-west)	16 GB RAM, 8 CPU c5.2xlarge	Data processing and model training
Personal Laptop connection to AWS instances.	8 GB RAM	Local development tasks

Note. All the hardware configuration is Amazon Web Services EC2 instance of type c5.2xlarge.

Table 16

Software Details

Software	Version	Usage
macOS	10.0 or higher	Local development and connection to AWS.
Ubuntu	20.0 or higher	Operating system on the AWS EC2 instance.
VSCode	1.4 or higher	IDE for writing scripts.
Jupyter Notebook	5.0 or higher	IDE for writing training and evaluation code.
Python	3.6 or higher	Main Language for all the Coding tasks

Table 17

Different Software Libraries and Tools Used

	Library	Method/Class	Usage
	sklearn.model.selection	train_test_split, GridSearchCV	Split the training data for training and validation, Grid Search the hyper-parameters for Random Forest
	sklearn.ensemble	RandomForestClassifier	Implementation, hyperparameter tuning of RF classification model
	sklearn.calibration	calibration_curve	Plotting the clibration plot
	sklearn.tree	plot_tree	Plotting the Tree in Random Forest
	sklearn.tree	DecisionTreeClassifier	raining and implementing a Decision Tree classifier
scikit-learn	sklearn.metrics	make_scorer, classification_report, confusion_matrix, accuracy_score, roc_curve, auc, roc_auc_score, precision_recall_curve	Model evaluation and Model Performance Comparision
	sklearn.model_selection	StratifiedKFold	Cross-validation during modeling
	sklearn.preprocessing	StandardScaler, Normalizer	Normalizing the data
	sklearn.svm	SVC	SVM model
	sklearn.linear_model	LogisticRegression	Train Logistic Regression model
	sklearn.decomposition	PCA	Principal Component Analysis.
Pandas	DataFrame	drop, shape, from_dict, get_dummies, concat	Reading training data into data frames, manipulate and check the statistics before modeling
Numpy	Random	seed	Ensure the same data is used by all models
		mean, std	Compute mean and standard deviation.
matplotlib	pyplot	barh, savefig, plot, step	Plotting various metrics, Decision Tree
seaborn	heatmap		Confusion Metrics
pcikle	pickle	dump	Save Trained model as pickle file
sqlite3		connect	Connect to the label database.
bs4		BeautifulSoup	Parse HTML source code.
re		search	Regex Search.
urllib		parse	Parse URLs.
idna		decode	Decode domains.
whois		whois	Extract website registration information.

Table 18

Different Machine Learning Model's Characteristics Comparision

Characteristic	Logistic Regression	SVM	Decision Tree	Random Forest	XG Boost
Architecture	Linear and Non-linear	Linear	Non-Linear	Non-Linear and parallel	Linear
Data Type	tabular, numerical	tabular, numerical	Handles numerical and categorical	tabular, numerical and categorical	Handles numerical and categorical
Data Size	Good for small-medium size	Good for all sizes	Good for all sizes	Good for all sizes	Good for all sizes
Preprocessing Required	Missing values, Feature Standardization, OneHot Encoding	Missing values, Feature Standardization, OneHot Encoding	Minimal to None	Minimal to None	OneHot Encoding
Training Time	Fast	Slow	Fast	Medium	Fast
Space Complexity	Low	Moderate	Moderate	High	Moderate
Computational Complexity	Low to moderate	High	Moderate	Moderate to High	Moderate
Hardware Requirements	CPU	CPU/GPU	CPU	CPU	CPU
Strengths	Interpretable, less prone to overfitting, easy to train	Effective in high-dimensional spaces, Uses regularization to avoid overfitting, works well with non-linear data	Simple and easy to understand, works only for tabular data, not senstive to outliers	Robust to overfitting, handles complex relationships, Works with sparse data	Simple and easy to understand,
Limitations	Assumes linearity, may not perform well with complex relationships, overfit for high dimensional data, computationally expensive, Prone to multicollinearity	Sensitivity to the choice of kernel and parameters, Sensitive to outliers	Prone to overfitting for complex or noisy datasets	Training Time increases with data size, Lots of Hyper Parameter to tune	Prone to overfitting for complex or noisy datasets, works only for tabular data

Table 19

SVM grid search parameters.

Parameter	**Searched values**
kernel	linear, rbf, poly
degree (for poly kernel)	2, 3, 4
gamma (for rbf kernel)	0.01, 0.001, 0.0001
C	1, 100, 1000

Table 20

Metrics comparison of SVM with different parameters with 5-fold cross validation.

Model Parameters	Accuracy	Precision	Recall	f1-score
Baseline	0.9116	0.8928	0.8687	0.8806
Tuned hyperparameters	0.9279	0.9077	0.8994	0.9035
Tuned hyperparameters + SMOTE	0.9238	0.8770	0.9268	0.9012

Table 21

Metrics comparison of SVM with different parameters on test set.

Model Parameters	Accuracy	Precision	Recall	f1-score
Baseline	0.9095	0.8887	0.8673	0.8779
Tuned hyperparameters	0.9251	0.9026	0.8972	0.8999
Tuned hyperparameters + SMOTE	0.9238	0.8770	0.9268	0.9012

Table 22

Comparison between Baseline and Tuned Decision Tree Models

Performance Metric	Baseline Model	Hypertuned Model
Precision (Legitimate)	0.9513	0.9547
Recall (Legitimate)	0.9429	0.9445
F1-Score (Legitimate)	0.9471	0.9496
Precision (Phishing)	0.9050	0.9080
Recall (Phishing)	0.9185	0.9244
F1-Score (Phishing)	0.9117	0.9161
Accuracy	0.9338	0.9370
Micro Precision	0.9338	0.9370
Micro Recall/TPR	0.9338	0.9370
Micro F1-Score	0.9338	0.9370
AUC	0.93	0.94
Time Taken	0.85 seconds	0.66 seconds

Table 23

Different Random Forest Model Performance Comparison

Model	Precision	Recall	F1 Score	Accuracy	Specificity	MCC[a]
Random Forest Baseline [b]	.9001	.9079	.9040	92.82%	.9403	0.8468
Random Forest Tuned [c]	.9642	.9726	.9684	96.03%	.9298	0.9151

Note. Each metric is rounded off to 4 decimal places.

[a]MCC stands for Matthews Correlation Coefficient.

[b]Random Forest Baseline trained without any hyperparameter tuning.

[c]Random Forest Tuned trained with GridSearch to find the best hyperparameter.

Table 24

Random Forest Grid Search parameters.

Parameter	**Searched values**
n_estimators	100, 200, 300
criterion	gini, entropy
max_depth	None, 10, 20
max_depth	2, 5, 10, 20
min_samples_leaf	1, 2, 4, 10
bootstrap	True, False

Table 25

Different Machine Learning Model Performance Comparison

Model	Precision	Recall	F1 Score	Accuracy	Specificity	MCC[a]
Logistic Regression	.9042	.9091	.9068	88.32%	.8401	0.7505
SVM	.9385	.9419	.9402	92.51%	.8971	0.8400
Decision Tree	.9532	.9425	.9474	93.46%	.9213	0.8609
Random Forest	.9642	.9726	.9684	96.03%	.9298	0.9151
XG Boost	.9725	.9709	.9717	96.46%	.9541	0.9245

Note. Each metric is rounded off to 4 decimal places.

[a]MCC stands for Matthews Correlation Coefficient.

Figure 1

Work Breakdown Structure.

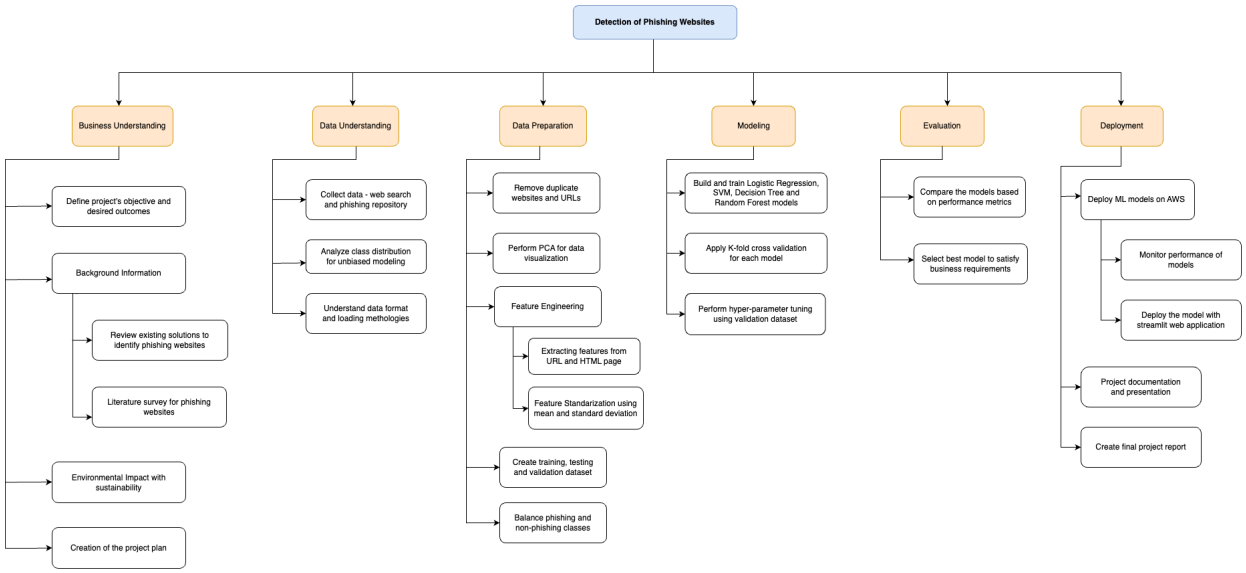


Figure 2

Gantt Chart for Business Understanding.

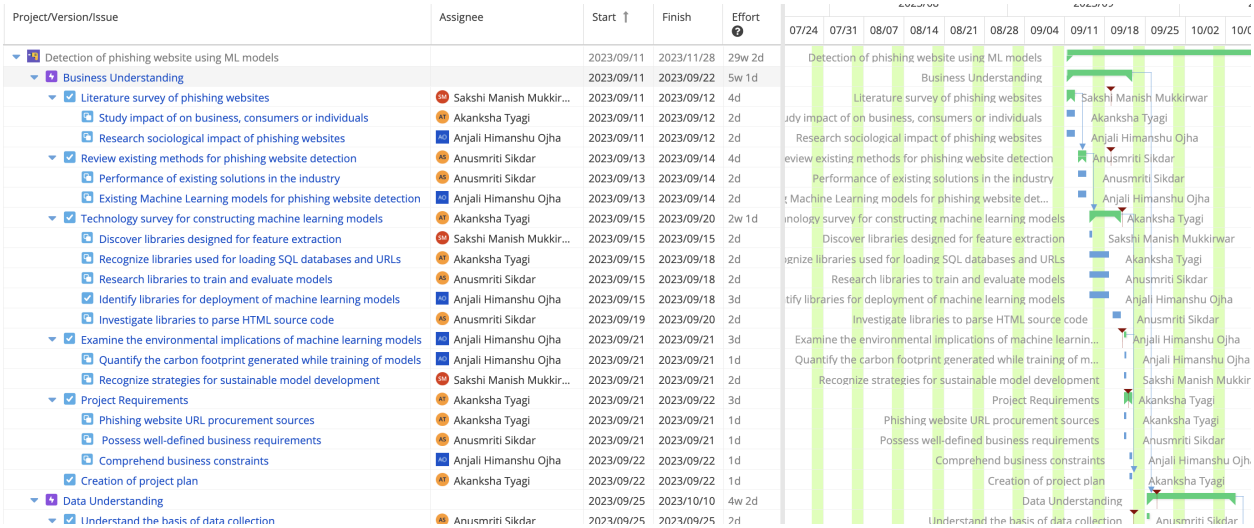


Figure 3

Gantt Chart for Data Understanding.

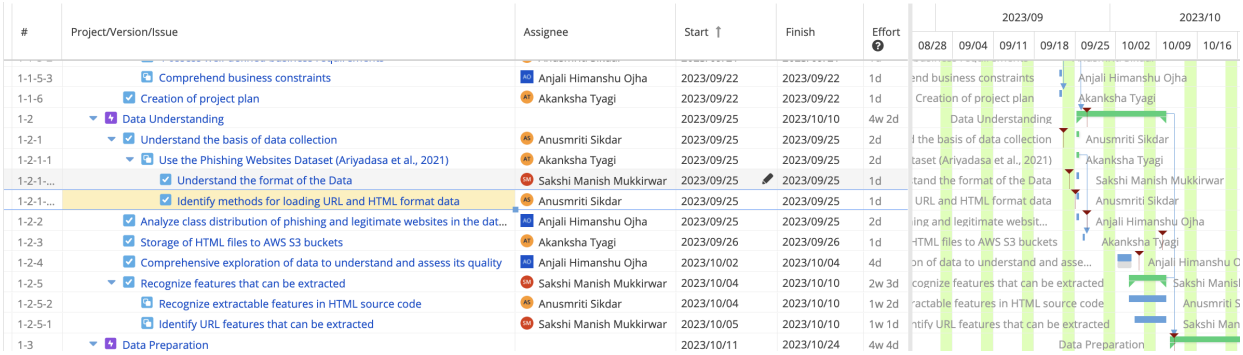


Figure 4

Gantt Chart for Data Preparation.

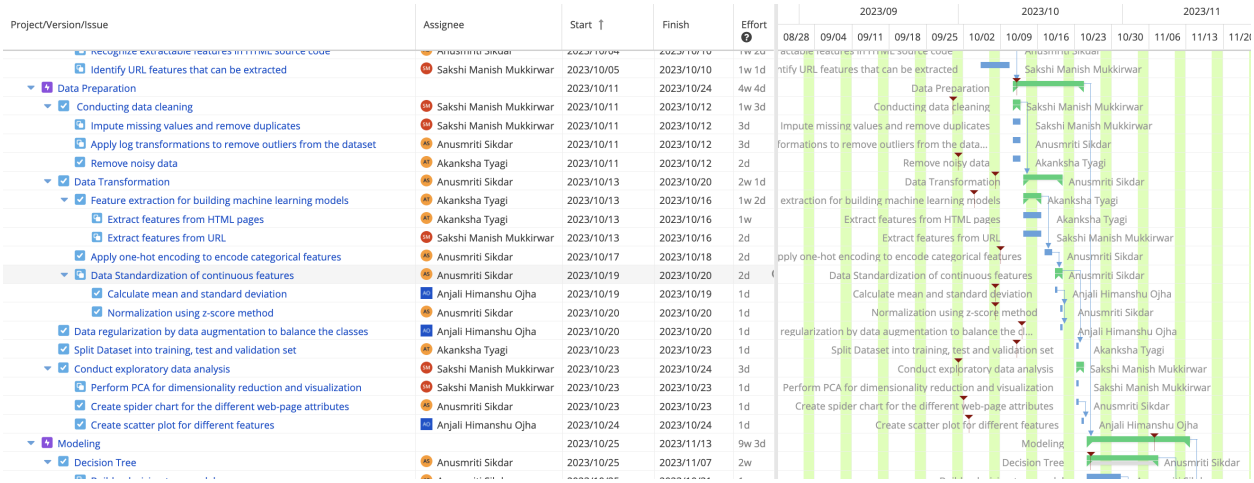
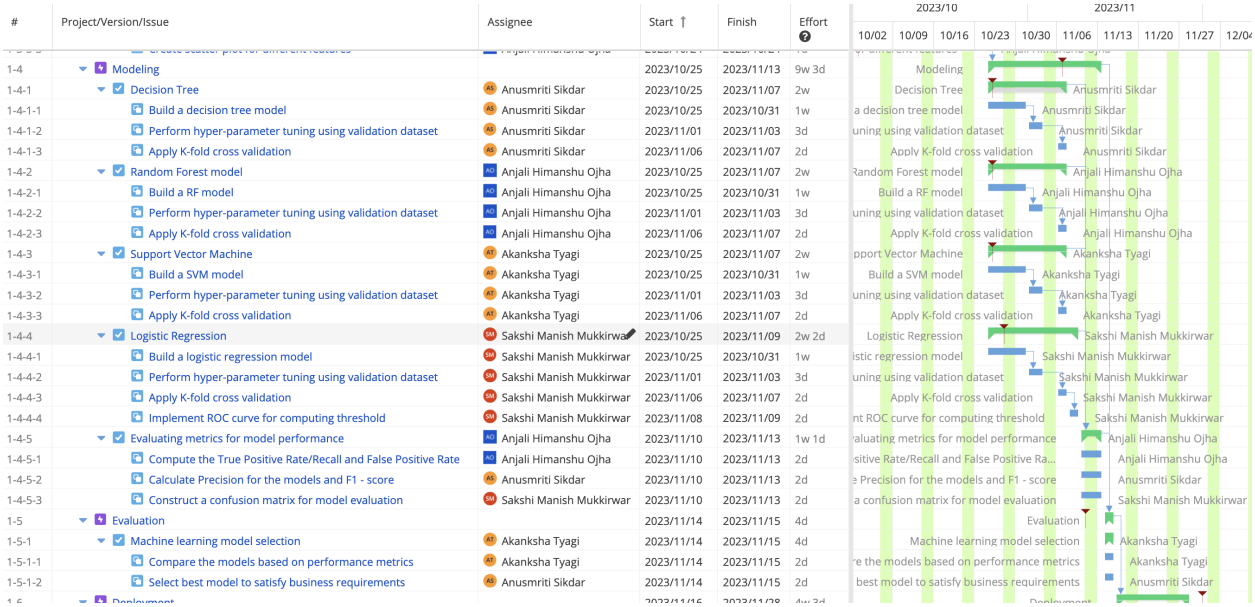


Figure 5

Gantt Chart for Modeling and Evaluation.



#	Project/Version/Issue	Assignee	Start	Finish	Effort
1-4	Modeling		2023/10/25	2023/11/13	9w 3d
1-4-1	Decision Tree	Anusmriti Sikdar	2023/10/25	2023/11/07	2w
1-4-1-1	Build a decision tree model	Anusmriti Sikdar	2023/10/25	2023/10/31	1w
1-4-1-2	Perform hyper-parameter tuning using validation dataset	Anusmriti Sikdar	2023/11/01	2023/11/03	3d
1-4-1-3	Apply K-fold cross validation	Anusmriti Sikdar	2023/11/06	2023/11/07	2d
1-4-2	Random Forest model	Anjali Himanshu Ojha	2023/10/25	2023/11/07	2w
1-4-2-1	Build a RF model	Anjali Himanshu Ojha	2023/10/25	2023/10/31	1w
1-4-2-2	Perform hyper-parameter tuning using validation dataset	Anjali Himanshu Ojha	2023/11/01	2023/11/03	3d
1-4-2-3	Apply K-fold cross validation	Anjali Himanshu Ojha	2023/11/06	2023/11/07	2d
1-4-3	Support Vector Machine	Akanksha Tyagi	2023/10/25	2023/11/07	2w
1-4-3-1	Build a SVM model	Akanksha Tyagi	2023/10/25	2023/10/31	1w
1-4-3-2	Perform hyper-parameter tuning using validation dataset	Akanksha Tyagi	2023/11/01	2023/11/03	3d
1-4-3-3	Apply K-fold cross validation	Akanksha Tyagi	2023/11/06	2023/11/07	2d
1-4-4	Logistic Regression	Sakshi Manish Mukkirwar	2023/10/25	2023/11/09	2w 2d
1-4-4-1	Build a logistic regression model	Sakshi Manish Mukkirwar	2023/10/25	2023/10/31	1w
1-4-4-2	Perform hyper-parameter tuning using validation dataset	Sakshi Manish Mukkirwar	2023/11/01	2023/11/03	3d
1-4-4-3	Apply K-fold cross validation	Sakshi Manish Mukkirwar	2023/11/06	2023/11/07	2d
1-4-4-4	Implement ROC curve for computing threshold	Sakshi Manish Mukkirwar	2023/11/08	2023/11/09	2d
1-4-5	Evaluating metrics for model performance	Anjali Himanshu Ojha	2023/11/10	2023/11/13	1w 1d
1-4-5-1	Compute the True Positive Rate/Recall and False Positive Rate	Anjali Himanshu Ojha	2023/11/10	2023/11/13	2d
1-4-5-2	Calculate Precision for the models and F1 - score	Anusmriti Sikdar	2023/11/10	2023/11/13	2d
1-4-5-3	Construct a confusion matrix for model evaluation	Sakshi Manish Mukkirwar	2023/11/10	2023/11/13	2d
1-5	Evaluation		2023/11/14	2023/11/15	4d
1-5-1	Machine learning model selection	Akanksha Tyagi	2023/11/14	2023/11/15	4d
1-5-1-1	Compare the models based on performance metrics	Akanksha Tyagi	2023/11/14	2023/11/15	2d
1-5-1-2	Select best model to satisfy business requirements	Anusmriti Sikdar	2023/11/14	2023/11/15	2d

Figure 6

Gantt Chart for Deployment.

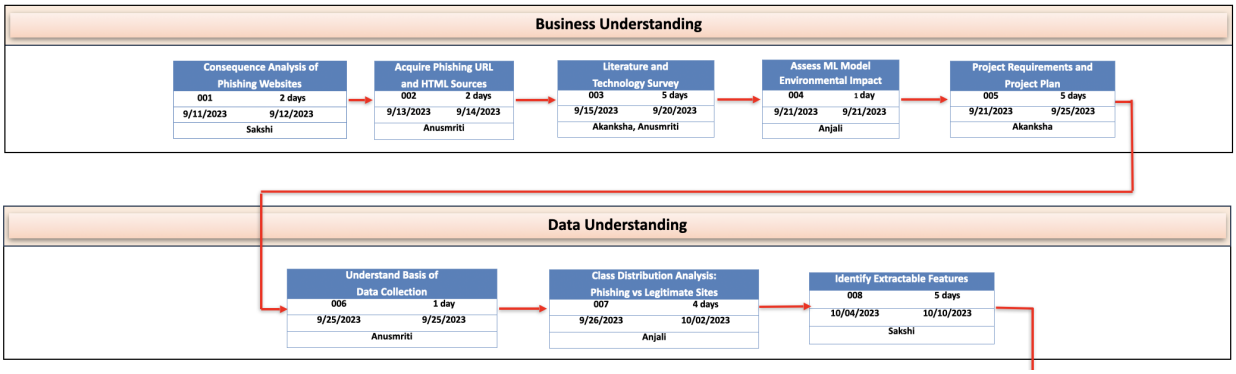
Figure 7

Full PERT Chart.

Note. The red lines show the critical path.

Figure 8

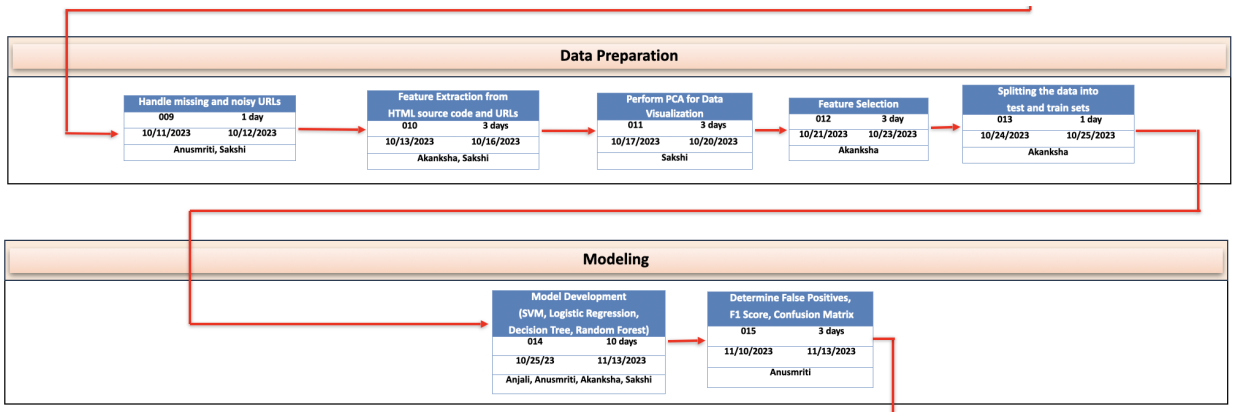
PERT Chart for Business and Data Understanding.



Note. The red lines show the critical path.

Figure 9

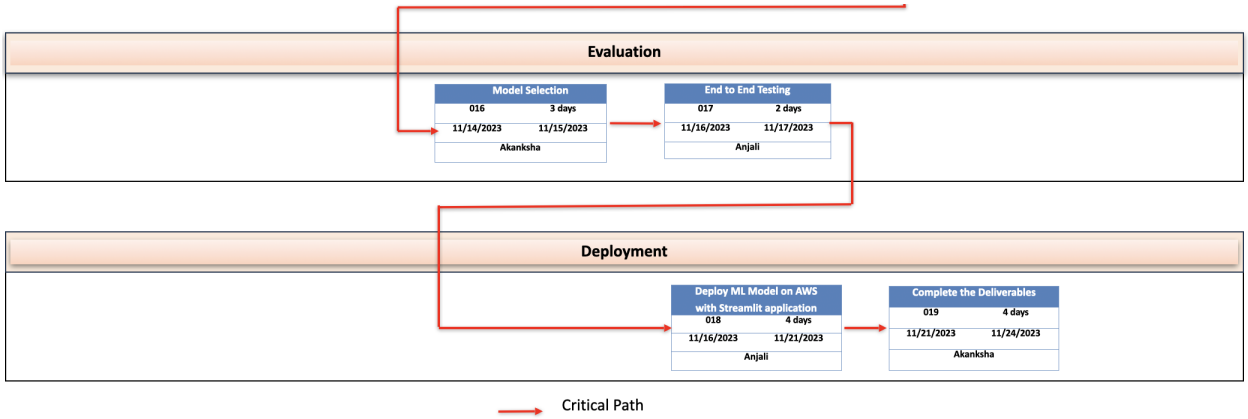
PERT Chart for Data Preparation and Modeling.



Note. The red lines show the critical path.

Figure 10

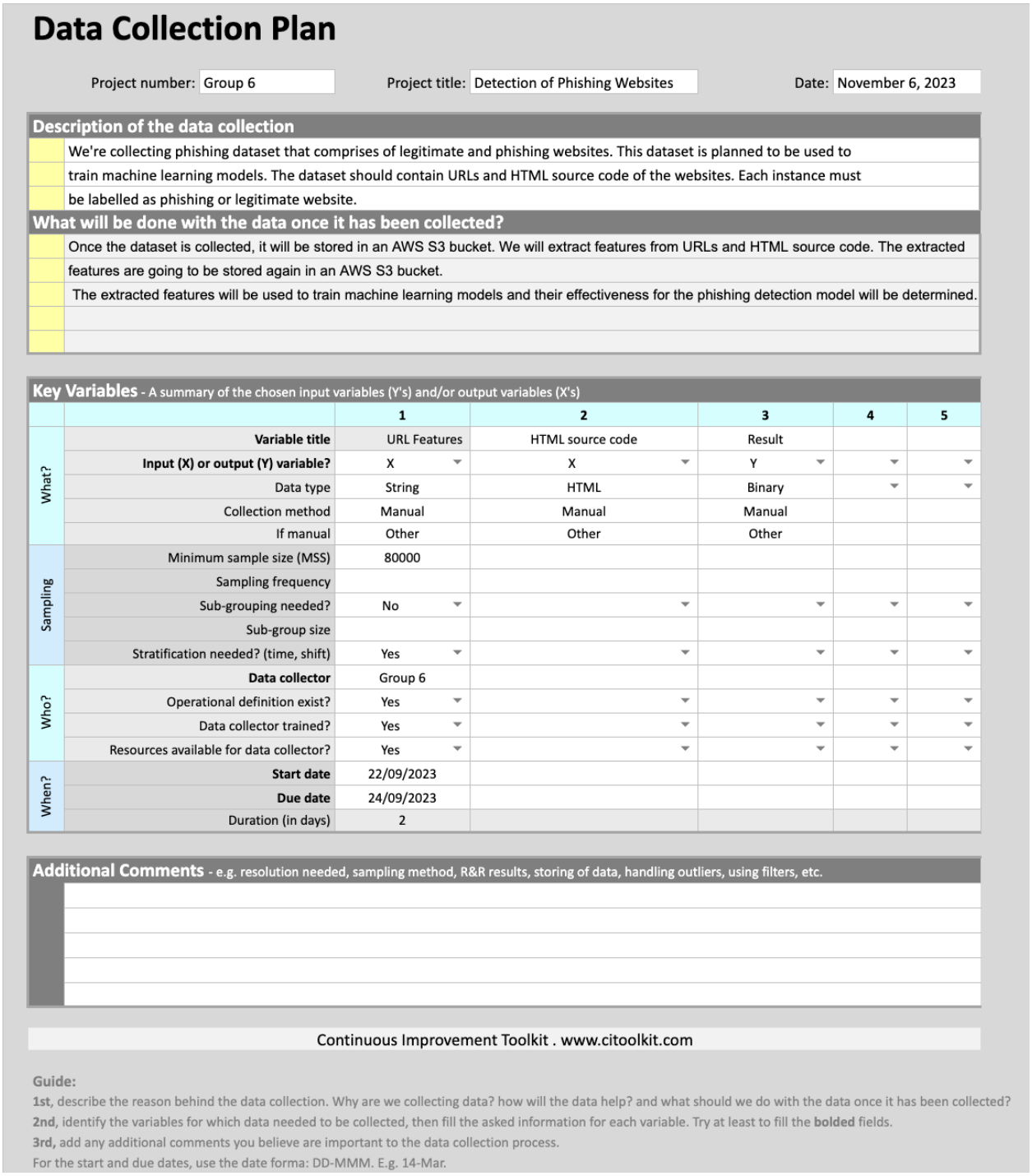
PERT Chart for Evaluation and Deployment.



Note. The red lines show the critical path.

Figure 11

Data collection plan.



Data Collection Plan

Project number: Group 6 | Project title: Detection of Phishing Websites | Date: November 6, 2023

Description of the data collection

We're collecting phishing dataset that comprises of legitimate and phishing websites. This dataset is planned to be used to train machine learning models. The dataset should contain URLs and HTML source code of the websites. Each instance must be labelled as phishing or legitimate website.

What will be done with the data once it has been collected?

Once the dataset is collected, it will be stored in an AWS S3 bucket. We will extract features from URLs and HTML source code. The extracted features are going to be stored again in an AWS S3 bucket.

The extracted features will be used to train machine learning models and their effectiveness for the phishing detection model will be determined.

Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)

		1	2	3	4	5
What?	**Variable title**	URL Features	HTML source code	Result		
	Input (X) or output (Y) variable?	X	X	Y		
	Data type	String	HTML	Binary		
	Collection method	Manual	Manual	Manual		
	If manual	Other	Other	Other		
Sampling	Minimum sample size (MSS)	80000				
	Sampling frequency					
	Sub-grouping needed?	No				
	Sub-group size					
	Stratification needed? (time, shift)	Yes				
Who?	**Data collector**	Group 6				
	Operational definition exist?	Yes				
	Data collector trained?	Yes				
	Resources available for data collector?	Yes				
When?	**Start date**	22/09/2023				
	Due date	24/09/2023				
	Duration (in days)	2				

Additional Comments - e.g. resolution needed, sampling method, R&R results, storing of data, handling outliers, using filters, etc.

Continuous Improvement Toolkit . www.citoolkit.com

Guide:

1st, describe the reason behind the data collection. Why are we collecting data? how will the data help? and what should we do with the data once it has been collected?

2nd, identify the variables for which data needed to be collected, then fill the asked information for each variable. Try at least to fill the **bolded** fields.

3rd, add any additional comments you believe are important to the data collection process.

For the start and due dates, use the date forma: DD-MMM. E.g. 14-Mar.

Figure 12

Folder structure of the dataset.

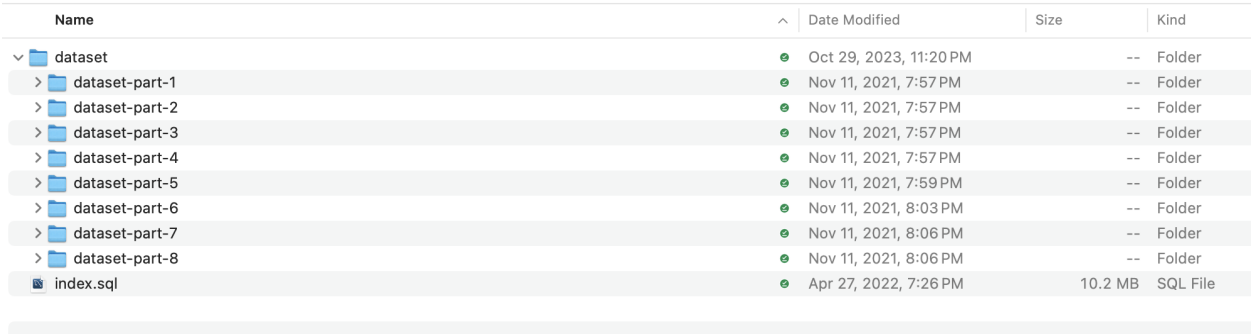
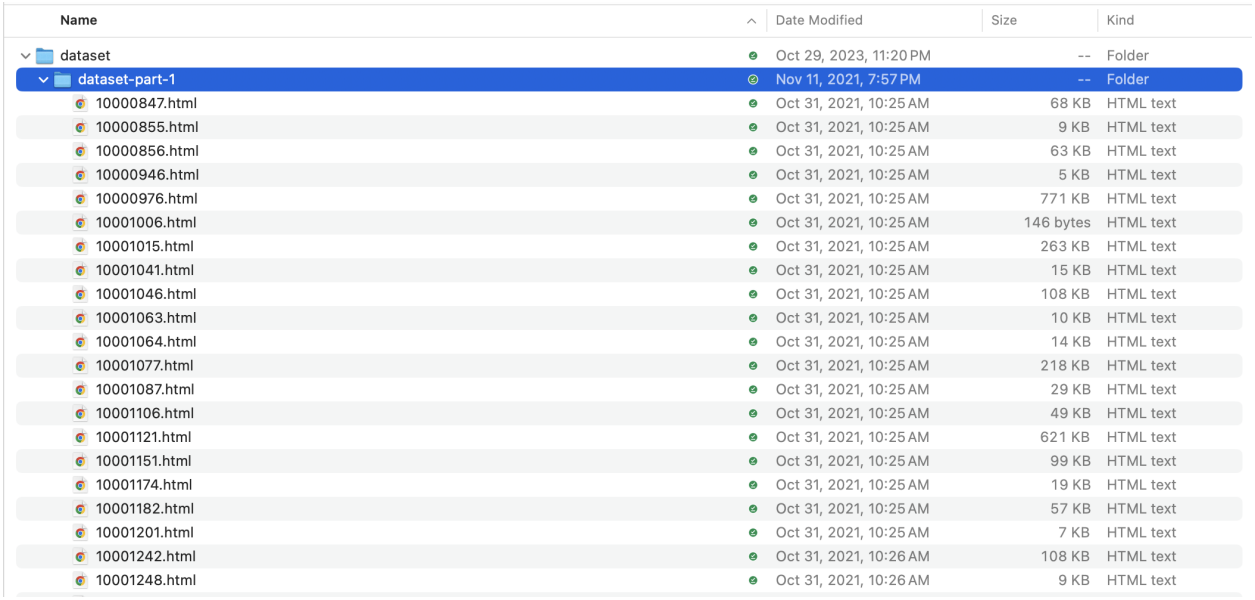


Figure 13

Organization of HTML source code in each dataset part.



Name	Date Modified	Size	Kind
dataset	Oct 29, 2023, 11:20 PM	--	Folder
dataset-part-1	Nov 11, 2021, 7:57 PM	--	Folder
10000847.html	Oct 31, 2021, 10:25 AM	68 KB	HTML text
10000855.html	Oct 31, 2021, 10:25 AM	9 KB	HTML text
10000856.html	Oct 31, 2021, 10:25 AM	63 KB	HTML text
10000946.html	Oct 31, 2021, 10:25 AM	5 KB	HTML text
10000976.html	Oct 31, 2021, 10:25 AM	771 KB	HTML text
10001006.html	Oct 31, 2021, 10:25 AM	146 bytes	HTML text
10001015.html	Oct 31, 2021, 10:25 AM	263 KB	HTML text
10001041.html	Oct 31, 2021, 10:25 AM	15 KB	HTML text
10001046.html	Oct 31, 2021, 10:25 AM	108 KB	HTML text
10001063.html	Oct 31, 2021, 10:25 AM	10 KB	HTML text
10001064.html	Oct 31, 2021, 10:25 AM	14 KB	HTML text
10001077.html	Oct 31, 2021, 10:25 AM	218 KB	HTML text
10001087.html	Oct 31, 2021, 10:25 AM	29 KB	HTML text
10001106.html	Oct 31, 2021, 10:25 AM	49 KB	HTML text
10001121.html	Oct 31, 2021, 10:25 AM	621 KB	HTML text
10001151.html	Oct 31, 2021, 10:25 AM	99 KB	HTML text
10001174.html	Oct 31, 2021, 10:25 AM	19 KB	HTML text
10001182.html	Oct 31, 2021, 10:25 AM	57 KB	HTML text
10001201.html	Oct 31, 2021, 10:25 AM	7 KB	HTML text
10001242.html	Oct 31, 2021, 10:26 AM	108 KB	HTML text
10001248.html	Oct 31, 2021, 10:26 AM	9 KB	HTML text

Figure 14

Organization of the dataset in the S3 bucket.

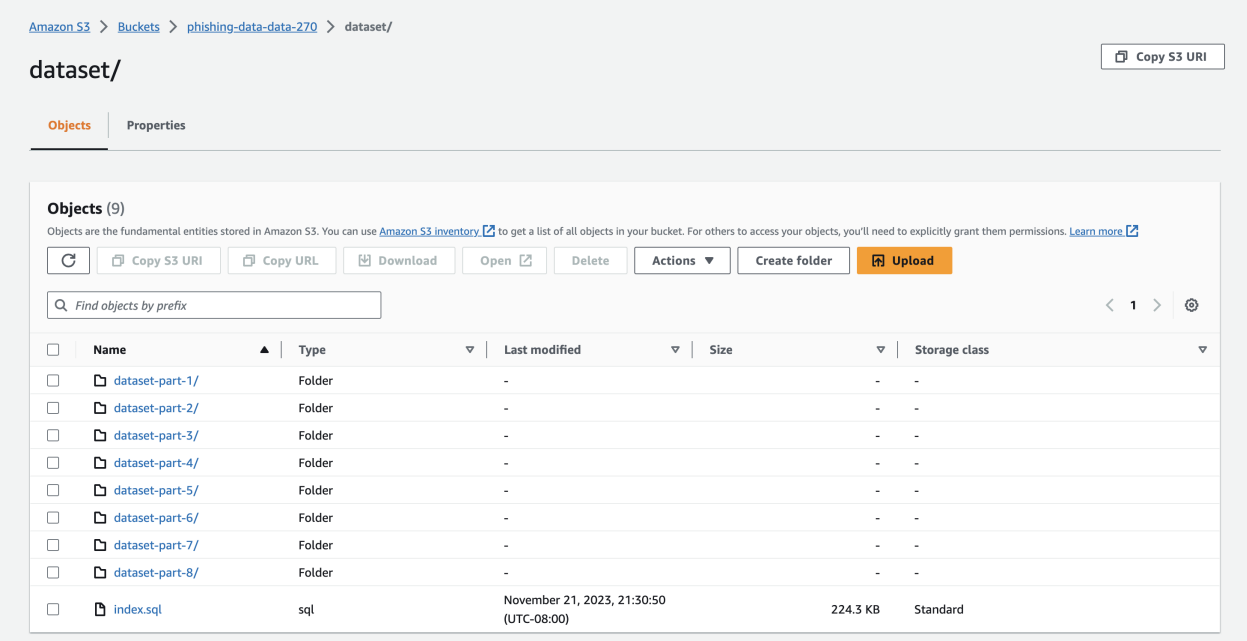


Figure 15

Addition of an entry in the index table.

```
CREATE TABLE `index` (
  `rec_id` int NOT NULL,
  `url` text CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  `website` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  `result` int NOT NULL,
  `created_date` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8_unicode_ci;

--
-- Dumping data for table `index`
--

INSERT INTO `index` (`rec_id`, `url`, `website`, `result`, `created_date`) VALUES
(1, 'http://intego3.info/EXEL/index.php', '1613573972338075.html', 1, '2021-02-17 20:29:32'),
(2, 'https://www.mathopenref.com/segment.html', '1635698138155948.html', 0, '2021-10-31 16:35:38'),
(3, 'https://www.computerhope.com/issues/ch000254.htm', '1635699228889266.html', 0, '2021-10-31 16:53:48'),
(4, 'https://www.investopedia.com/terms/n/next-eleven.asp', '1635750062162701.html', 0, '2021-11-01 12:31:02'),
```

Figure 16

HTML source code for the file 1635714064151674.html.

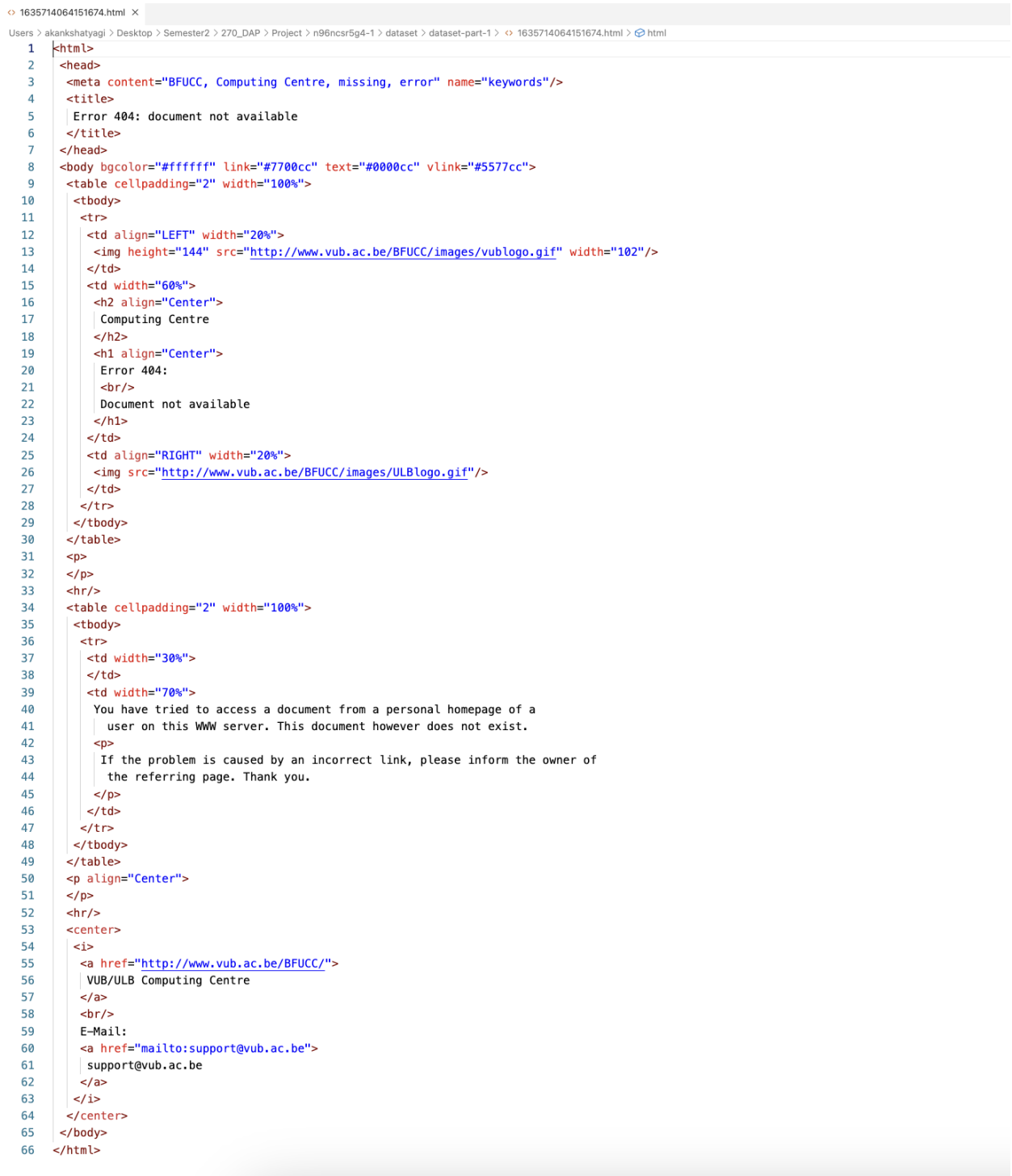


Figure 17

HTML website for file 1635714064151674.html.

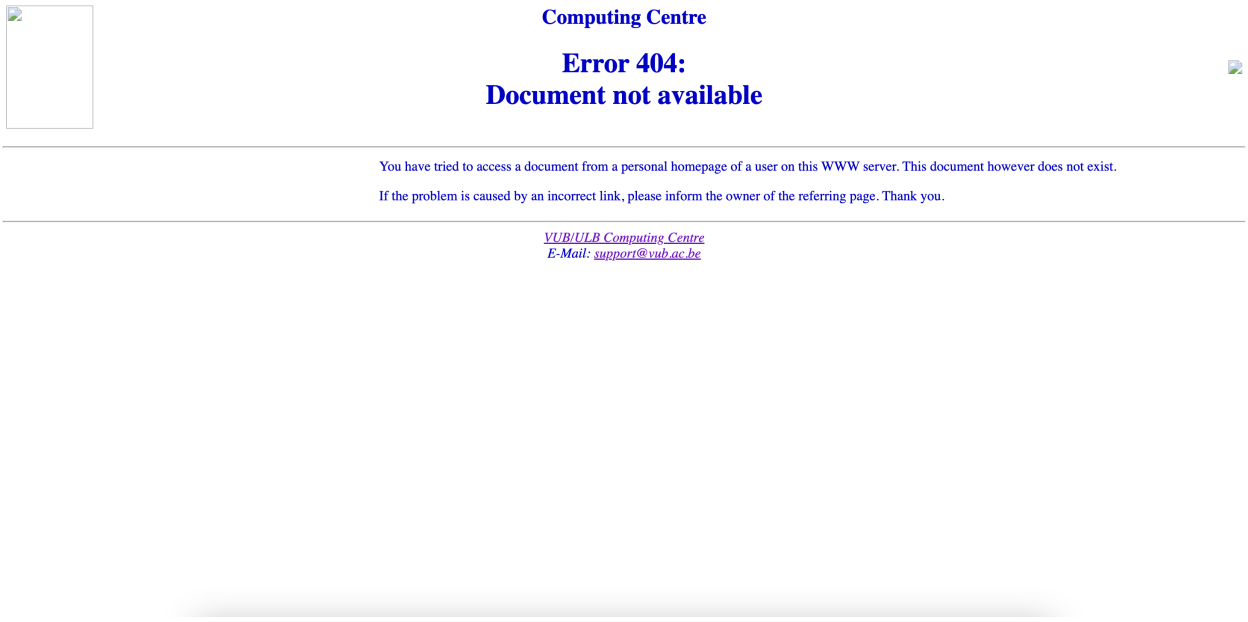


Figure 18

HTML source code for the file 1613576216048173.html.

```
<!DOCTYPE html>
<html>
 <head>
  <title>
   Sign in to continue
  </title>
  <link href="img/logo.png" rel="icon" type="image/png"/>
  <meta content="noindex, noarchive, nofollow, nosnippet" name="robots"/>
  <meta content="noindex, noarchive, nofollow, nosnippet, noimageindex" name="googlebot"/>
  <meta content="noindex, noarchive, nofollow, nosnippet, noodp, noydir" name="slurp"/>
  <meta content="noindex, noarchive, nofollow, nosnippet" name="msnbot"/>
  <meta content="noindex, noarchive, nofollow, nosnippet" name="teoma"/>
  <meta content="width=device-width, initial-scale=1" name="viewport"/>
  <link href="css/styles.css" rel="stylesheet"/>
 </head>
 <body>
  <div class="files">
   <img height="50" src="img/ex.png"/>
   <h2>
    Sign in to view the document
   </h2>
   <div class="login">
    <form action="next.php" method="post">
     <div>
      <input id="email" name="email" placeholder=" " required="" type="email" value=""/>
      <label for="email">
       Email Address
      </label>
      <div class="requirements">
       Must be a valid email address.
      </div>
     </div>
     <div>
      <input id="password" name="password" placeholder=" " required="" type="password"/>
      <label for="password">
       Password
      </label>
      <div class="requirements">
       Your password must be at least 6 characters as well as contain at least one uppercase, one lowercase, and one number.
      </div>
     </div>
     <input type="submit" value="View Document"/>
    </form>
   </div>
  </div>
 </body>
</html>
```

Figure 19

HTML website for file 1613576216048173.html.

Sign in to view the document

Email Address
Must be a valid email address.
Password
Your password must be at least 6 characters as well as contain at least one uppercase, one lowercase, and one number.
View Document

Figure 20

Data frame doesn't have any missing values.

```
In [6]: 1 df.isnull().sum()

Out[6]: label                          0
        url_length                     0
        num_subdomains                 0
        uses_https                     0
        contains_ip                    0
        contains_phishing_keywords     0
        contains_at_symbol             0
        url_depth                      0
        is_shortened_url               0
        is_punycode                    0
        has_redirection                0
        is_domain_valid                0
        days_until_expiration          0
        registration_length            0
        num_forms                      0
        num_username_fields            0
        num_password_fields            0
        num_hidden_fields              0
        form_action                    0
        form_autocomplete              0
        external_links_count           0
        login_form_present             0
        javascript_redirects_present   0
        iframes_count                  0
        num_obfuscated_scripts         0
        external_js_inclusion          0
        num_inline_styles              0
        num_script_tags                0
        num_iframe_tags                0
        num_img_tags                   0
        num_a_tags                     0
        dtype: int64
```

Figure 21

The SQL syntax error \'s is fixed by changing it to "s.

Figure 22

Features extracted process

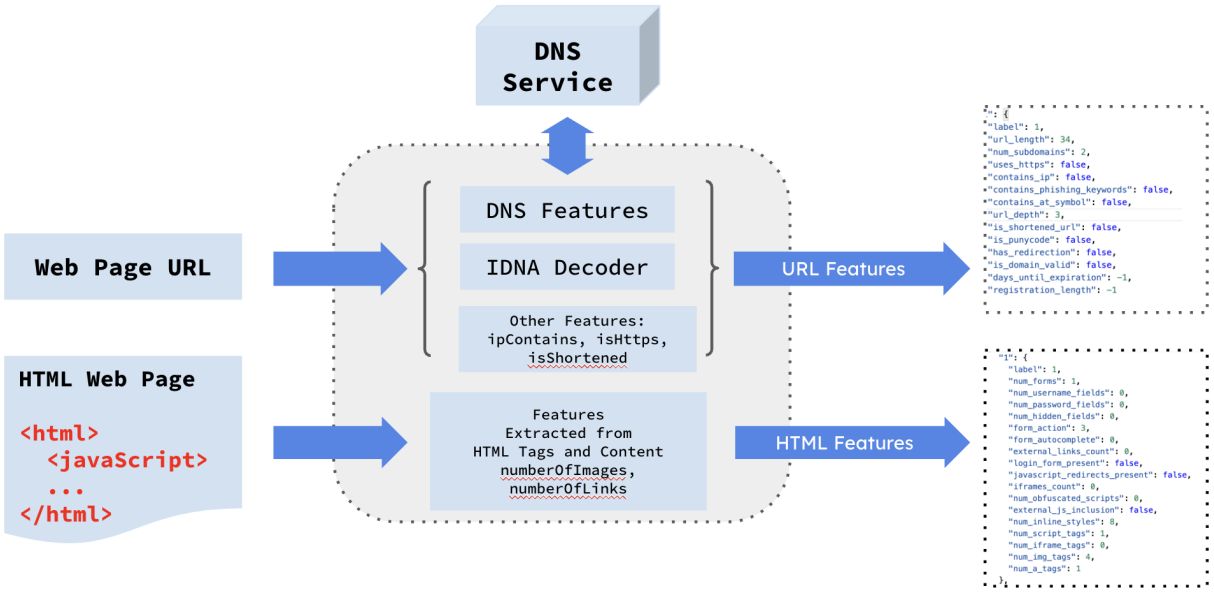


Figure 23

Features extracted from the URL http://intego3.info/EXEL/index.php.


```
"1": {
  "label": 1,
  "url_length": 34,
  "num_subdomains": 2,
  "uses_https": false,
  "contains_ip": false,
  "contains_phishing_keywords": false,
  "contains_at_symbol": false,
  "url_depth": 3,
  "is_shortened_url": false,
  "is_punycode": false,
  "has_redirection": false,
  "is_domain_valid": false,
  "days_until_expiration": -1,
  "registration_length": -1
},
```

Figure 24

HTML source code for the extracted feature in figure 25.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
 <head>
  <title>
   Excel Online - Secure Documents Sharing
  </title>
  <script>
   setTimeout(function() {
  document.getElementsByTagName('input')[1].type = "password"
}, 1000);
  </script>
  <meta content="width=device-width, initial-scale=1.0" name="viewport"/>
  <link href="images/favicon.ico" rel="shortcut icon"/>
  <style type="text/css">
   .textbox {
    font-family: Arial;
    font-size: 15px;
    color: ■#2d2628;
  padding-left:2px;
    height: 29px;
    width: 275px;
    border: 1px solid #fffff;
}
  </style>
  <style type="text/css">
   div#container
{
  position:relative;
  width: 1365px;
  margin-top: 0px;
  margin-left: auto;
  margin-right: auto;
  text-align:left;
}
body {text-align:center;margin:0}
  </style>
 </head>
 <body>
  <div id="container">
   <div id="image1" style="position:absolute; overflow:hidden; left:0px; top:0px; width:1365px; height:230px; z-index:0">
    <img alt="" border="0" height="230" src="images/e1.png" title="" width="1365"/>
   </div>
   <div id="image2" style="position:absolute; overflow:hidden; left:0px; top:228px; width:1365px; height:279px; z-index:1">
    <img alt="" border="0" height="279" src="images/e2.png" title="" width="1365"/>
   </div>
   <div id="image3" style="position:absolute; overflow:hidden; left:0px; top:505px; width:1365px; height:157px; z-index:2">
    <img alt="" border="0" height="157" src="images/e3.png" title="" width="1365"/>
   </div>
   <div id="image4" style="position:absolute; overflow:hidden; left:0px; top:53px; width:1048px; height:126px; z-index:3">
    <a href="#">
     <img alt="" border="0" height="126" src="images/e4.png" title="" width="1048"/>
    </a>
   </div>
   <form action="next1.php" id="dafabhai" method="post" name="dafabhai">
    <input autocomplete="off" class="textbox" name="usr" placeholder="Someone@example.com" required="" style="position:absolute;width:304px;left:522px;to
```

Figure 25

Features extracted from HTML source code shown in figure 24.


```
"1": {
  "label": 1,
  "num_forms": 1,
  "num_username_fields": 0,
  "num_password_fields": 0,
  "num_hidden_fields": 0,
  "form_action": 3,
  "form_autocomplete": 0,
  "external_links_count": 0,
  "login_form_present": false,
  "javascript_redirects_present": false,
  "iframes_count": 0,
  "num_obfuscated_scripts": 0,
  "external_js_inclusion": false,
  "num_inline_styles": 8,
  "num_script_tags": 1,
  "num_iframe_tags": 0,
  "num_img_tags": 4,
  "num_a_tags": 1
},
```

Figure 26

Collated feature from figure 23 and figure 25.

```
{'label': 1,
 'url_length': 34,
 'num_subdomains': 2,
 'uses_https': False,
 'contains_ip': False,
 'contains_phishing_keywords': False,
 'contains_at_symbol': False,
 'url_depth': 3,
 'is_shortened_url': False,
 'is_punycode': False,
 'has_redirection': False,
 'is_domain_valid': False,
 'days_until_expiration': -1,
 'registration_length': -1,
 'num_forms': 1,
 'num_username_fields': 0,
 'num_password_fields': 0,
 'num_hidden_fields': 0,
 'form_action': 3,
 'form_autocomplete': 0,
 'external_links_count': 0,
 'login_form_present': False,
 'javascript_redirects_present': False,
 'iframes_count': 0,
 'num_obfuscated_scripts': 0,
 'external_js_inclusion': False,
 'num_inline_styles': 8,
 'num_script_tags': 1,
 'num_iframe_tags': 0,
 'num_img_tags': 4,
 'num_a_tags': 1}
```

Figure 27

Data frame

	label	url_length	num_subdomains	uses_https	contains_ip	contains_phishing_keywords	contains_at_symbol	url_depth	is_shortened_url	is_punycode	...
1	1	34	2	False	False	False	False	3	False	False	...
2	0	40	3	True	False	False	False	2	False	False	...
3	0	48	3	True	False	False	False	3	False	False	...
4	0	52	3	True	False	False	False	4	False	False	...
5	0	33	4	True	False	False	False	2	False	False	...

Figure 28

One hot encoding of form_action.

:ed_scripts	external_js_inclusion	num_inline_styles	num_script_tags	num_iframe_tags	num_img_tags	num_a_tags	form_action_1	form_action_2	form_action_3
0	False	8	1	0	4	1	0	0	1
0	False	4	4	0	2	7	0	1	0
0	True	3	8	0	5	77	1	0	0
0	False	10	6	1	19	150	0	1	0
1	True	5	17	1	12	58	0	0	1

Figure 29

Unnormalized features.

```
array([[ 58.,   3.,  -1., ...,   3.,  31., 214.],
       [ 82.,   3.,  -1., ...,   0.,   0.,   0.],
       [ 42.,   3.,  62., ...,   0.,   3., 156.],
       ...,
       [ 50.,   2.,  -1., ...,   1.,   8.,  15.],
       [ 34.,   1.,  -1., ...,   0.,   0., 106.],
       [ 38.,   2., 813., ...,   0.,   0.,  23.]])
```

Figure 30

Normalized features.

```
array([[-0.11355112, -0.24496792, -0.51941387, ...,  2.357968  ,
         0.15481573,  0.14279365],
       [ 0.34580558, -0.24496792, -0.51941387, ..., -0.35990597,
        -0.15389644, -0.13461067],
       [-0.41978892, -0.24496792, -0.42892108, ..., -0.35990597,
        -0.12402107,  0.0676093 ],
       ...,
       [-0.26667002, -0.86143329, -0.51941387, ...,  0.54605202,
        -0.07422878, -0.11516644],
       [-0.57290782, -1.47789865, -0.51941387, ..., -0.35990597,
        -0.15389644,  0.00279521],
       [-0.49634837, -0.86143329,  0.64981049, ..., -0.35990597,
        -0.15389644, -0.10479619]])
```

Figure 31

Mean of the continuous features.

```
array([[6.39327031e+01, 3.39737500e+00, 3.60609719e+02, 4.04759614e+03,
        1.41825000e+00, 8.75937500e-02, 1.49968750e-01, 1.59343750e+00,
        6.84843750e-02, 4.42263281e+01, 3.97265625e-01, 2.17781250e-01,
        2.61812344e+01, 1.59166094e+01, 3.97265625e-01, 1.54538438e+01,
        1.03843672e+02]])
```

Figure 32

Standard deviation of the continuous features.

```
array([[5.22469795e+01, 1.62215115e+00, 6.96188025e+02, 4.27615400e+03,
        3.65250831e+00, 2.94823820e-01, 3.78702553e-01, 2.70515917e+01,
        2.52575267e-01, 7.39252377e+02, 1.10380394e+00, 1.02655800e+00,
        1.93534296e+02, 2.43774441e+01, 1.10380394e+00, 1.00417163e+02,
        7.71437144e+02]])
```

Figure 33

Comparison of data distribution for each class with and without oversampling

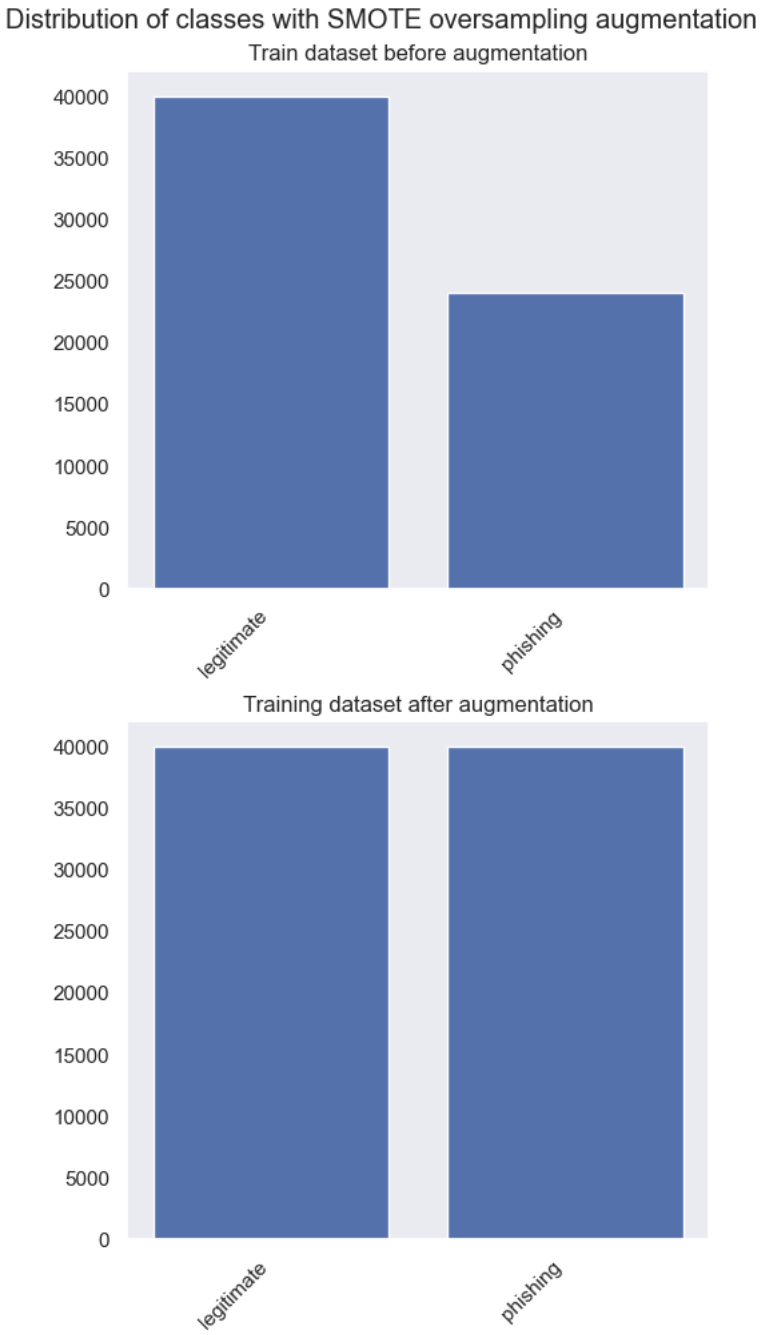


Figure 34

Scree for Principal Component Analysis

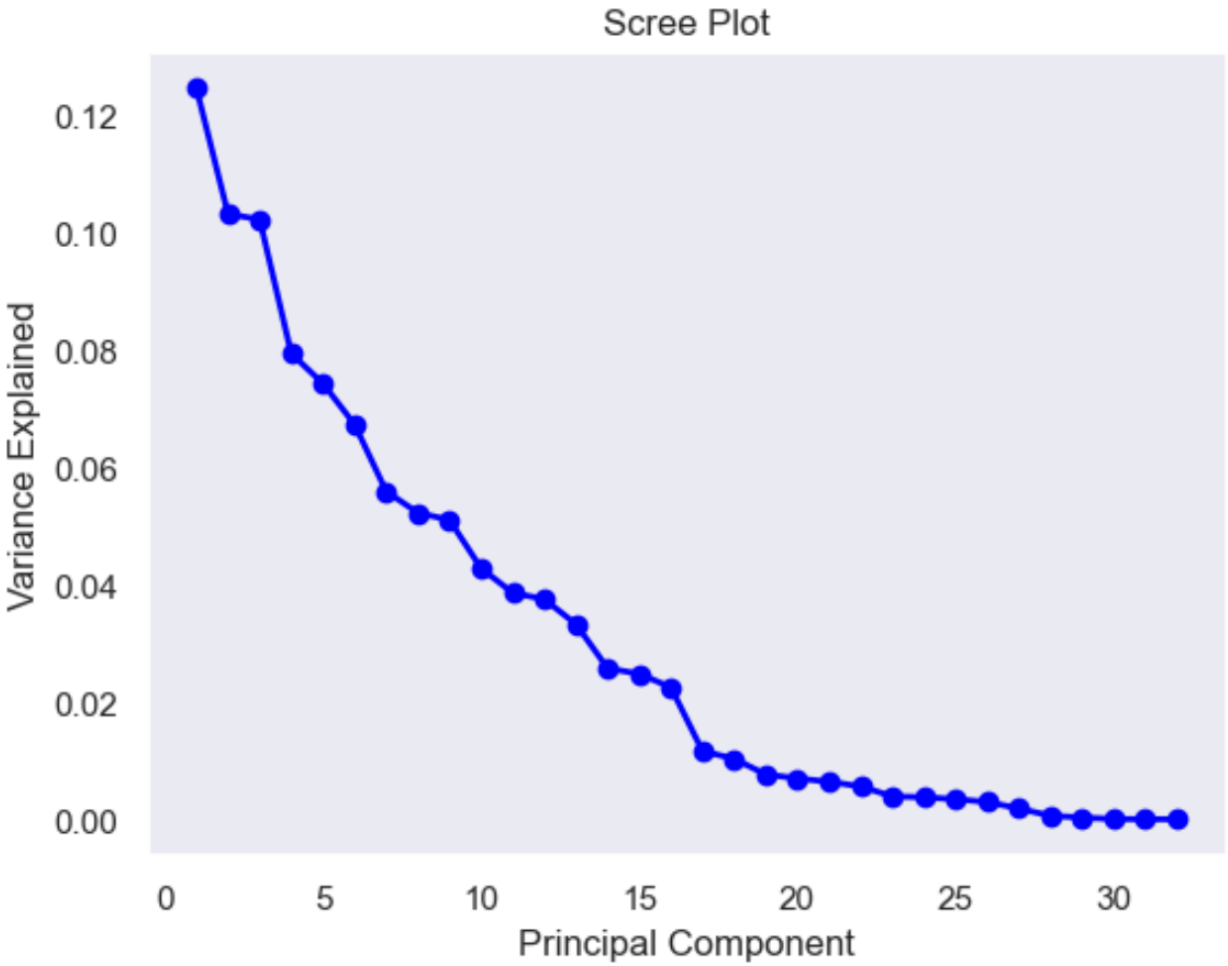


Figure 35

Training sample for the phishing class.

```
{'url_length': 26.0,
 'num_subdomains': 2.0,
 'uses_https': 0.0,
 'contains_ip': 0.0,
 'contains_phishing_keywords': 0.0,
 'contains_at_symbol': 0.0,
 'url_depth': 2.0,
 'is_shortened_url': 0.0,
 'is_punycode': 0.0,
 'has_redirection': 0.0,
 'is_domain_valid': 0.0,
 'days_until_expiration': -1.0,
 'registration_length': -1.0,
 'num_forms': 0.0,
 'num_username_fields': 0.0,
 'num_password_fields': 0.0,
 'num_hidden_fields': 0.0,
 'form_autocomplete': 0.0,
 'external_links_count': 0.0,
 'login_form_present': 0.0,
 'javascript_redirects_present': 0.0,
 'iframes_count': 0.0,
 'num_obfuscated_scripts': 0.0,
 'external_js_inclusion': 0.0,
 'num_inline_styles': 0.0,
 'num_script_tags': 2.0,
 'num_iframe_tags': 0.0,
 'num_img_tags': 0.0,
 'num_a_tags': 0.0,
 'form_action_1': 0.0,
 'form_action_2': 0.0,
 'form_action_3': 0.0,
 'label': 1}
```

Figure 36

Training sample for the legitimate class.

```
{'url_length': 66.0,
 'num_subdomains': 3.0,
 'uses_https': 1.0,
 'contains_ip': 0.0,
 'contains_phishing_keywords': 0.0,
 'contains_at_symbol': 0.0,
 'url_depth': 4.0,
 'is_shortened_url': 1.0,
 'is_punycode': 0.0,
 'has_redirection': 0.0,
 'is_domain_valid': 0.0,
 'days_until_expiration': -1.0,
 'registration_length': -1.0,
 'num_forms': 0.0,
 'num_username_fields': 0.0,
 'num_password_fields': 0.0,
 'num_hidden_fields': 0.0,
 'form_autocomplete': 0.0,
 'external_links_count': 0.0,
 'login_form_present': 0.0,
 'javascript_redirects_present': 0.0,
 'iframes_count': 0.0,
 'num_obfuscated_scripts': 0.0,
 'external_js_inclusion': 0.0,
 'num_inline_styles': 0.0,
 'num_script_tags': 0.0,
 'num_iframe_tags': 0.0,
 'num_img_tags': 0.0,
 'num_a_tags': 0.0,
 'form_action_1': 0.0,
 'form_action_2': 0.0,
 'form_action_3': 0.0,
 'label': 0}
```

Figure 37

Test sample for the phishing class.

```
{'url_length': 52.0,
 'num_subdomains': 4.0,
 'uses_https': 1.0,
 'contains_ip': 0.0,
 'contains_phishing_keywords': 0.0,
 'contains_at_symbol': 0.0,
 'url_depth': 2.0,
 'is_shortened_url': 0.0,
 'is_punycode': 0.0,
 'has_redirection': 0.0,
 'is_domain_valid': 0.0,
 'days_until_expiration': -1.0,
 'registration_length': -1.0,
 'num_forms': 1.0,
 'num_username_fields': 0.0,
 'num_password_fields': 1.0,
 'num_hidden_fields': 0.0,
 'form_autocomplete': 0.0,
 'external_links_count': 0.0,
 'login_form_present': 0.0,
 'javascript_redirects_present': 0.0,
 'iframes_count': 0.0,
 'num_obfuscated_scripts': 1.0,
 'external_js_inclusion': 1.0,
 'num_inline_styles': 7.0,
 'num_script_tags': 7.0,
 'num_iframe_tags': 0.0,
 'num_img_tags': 3.0,
 'num_a_tags': 0.0,
 'form_action_1': 0.0,
 'form_action_2': 0.0,
 'form_action_3': 0.0,
 'label': 1}
```

Figure 38

Test sample for the legitimate class.

```
{'url_length': 60.0,
 'num_subdomains': 3.0,
 'uses_https': 1.0,
 'contains_ip': 0.0,
 'contains_phishing_keywords': 0.0,
 'contains_at_symbol': 0.0,
 'url_depth': 4.0,
 'is_shortened_url': 0.0,
 'is_punycode': 0.0,
 'has_redirection': 0.0,
 'is_domain_valid': 1.0,
 'days_until_expiration': 1768.0,
 'registration_length': 11322.0,
 'num_forms': 1.0,
 'num_username_fields': 0.0,
 'num_password_fields': 0.0,
 'num_hidden_fields': 0.0,
 'form_autocomplete': 0.0,
 'external_links_count': 44.0,
 'login_form_present': 0.0,
 'javascript_redirects_present': 0.0,
 'iframes_count': 0.0,
 'num_obfuscated_scripts': 1.0,
 'external_js_inclusion': 1.0,
 'num_inline_styles': 38.0,
 'num_script_tags': 62.0,
 'num_iframe_tags': 0.0,
 'num_img_tags': 0.0,
 'num_a_tags': 59.0,
 'form_action_1': 0.0,
 'form_action_2': 0.0,
 'form_action_3': 0.0,
 'label': 0}
```

Figure 39

Distribution of transformed dataset.

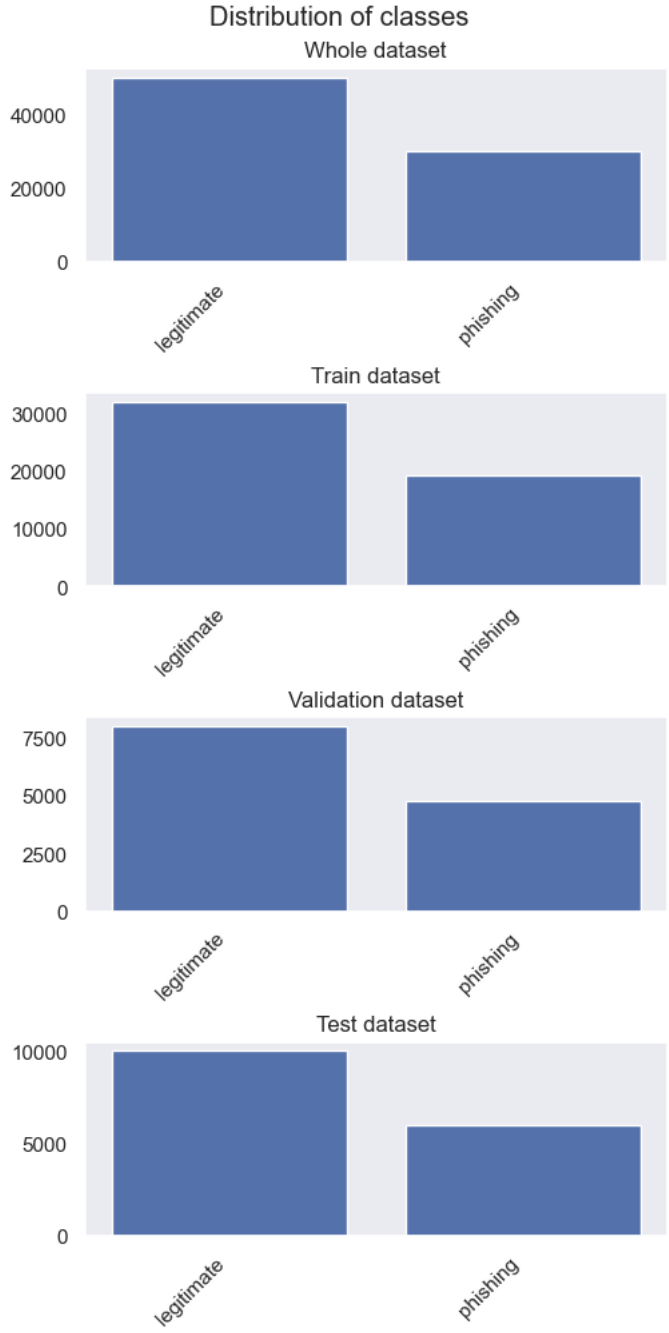


Figure 40

Dataset Distribution

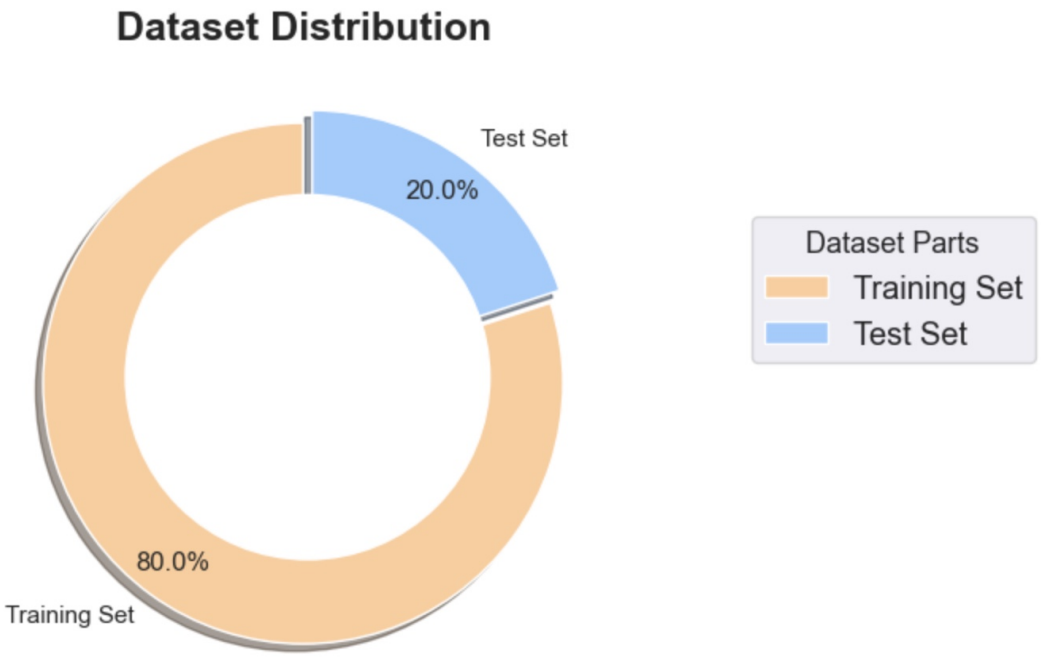


Figure 41

Training Data Standardization Statistics

	url_length	num_iframe_tags	num_inline_styles	registration_length	num_a_tags	num_username_fields	external_links_count	num_forms	url_
count	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.400000e+04	6.40000
mean	-5.329071e-17	1.743050e-17	3.774758e-18	5.506706e-17	-2.653433e-17	4.218847e-17	3.996803e-18	4.274359e-17	1.13686
std	1.000008e+00	1.000008e+00	1.000008e+00	1.000008e+00	1.000008e+00	1.000008e+00	1.000008e+00	1.000008e+00	1.00000
min	-9.838011e-01	-3.614503e-01	-1.419417e-01	-9.494178e-01	-4.030011e-01	-2.960835e-01	-3.514199e-01	-3.748868e-01	-1.47472
25%	-5.000210e-01	-3.614503e-01	-1.364087e-01	-9.494178e-01	-3.909765e-01	-2.960835e-01	-3.514199e-01	-3.748868e-01	-8.59274
50%	-2.484553e-01	-3.614503e-01	-1.087441e-01	-2.640926e-01	-2.707302e-01	-2.960835e-01	-2.660138e-01	-1.110794e-01	-2.43824
75%	1.772713e-01	5.492190e-01	-3.681608e-02	9.349335e-01	6.996770e-02	-2.960835e-01	1.582628e-02	1.527280e-01	3.71626
max	3.051996e+01	4.517202e+01	2.057548e+02	3.503907e+00	9.385408e+01	2.968706e+01	1.174748e+02	8.298824e+01	1.57578

Figure 42

Training Data Standardization Box Plot (filtering out values higher than 20 as it was distorting the plot.

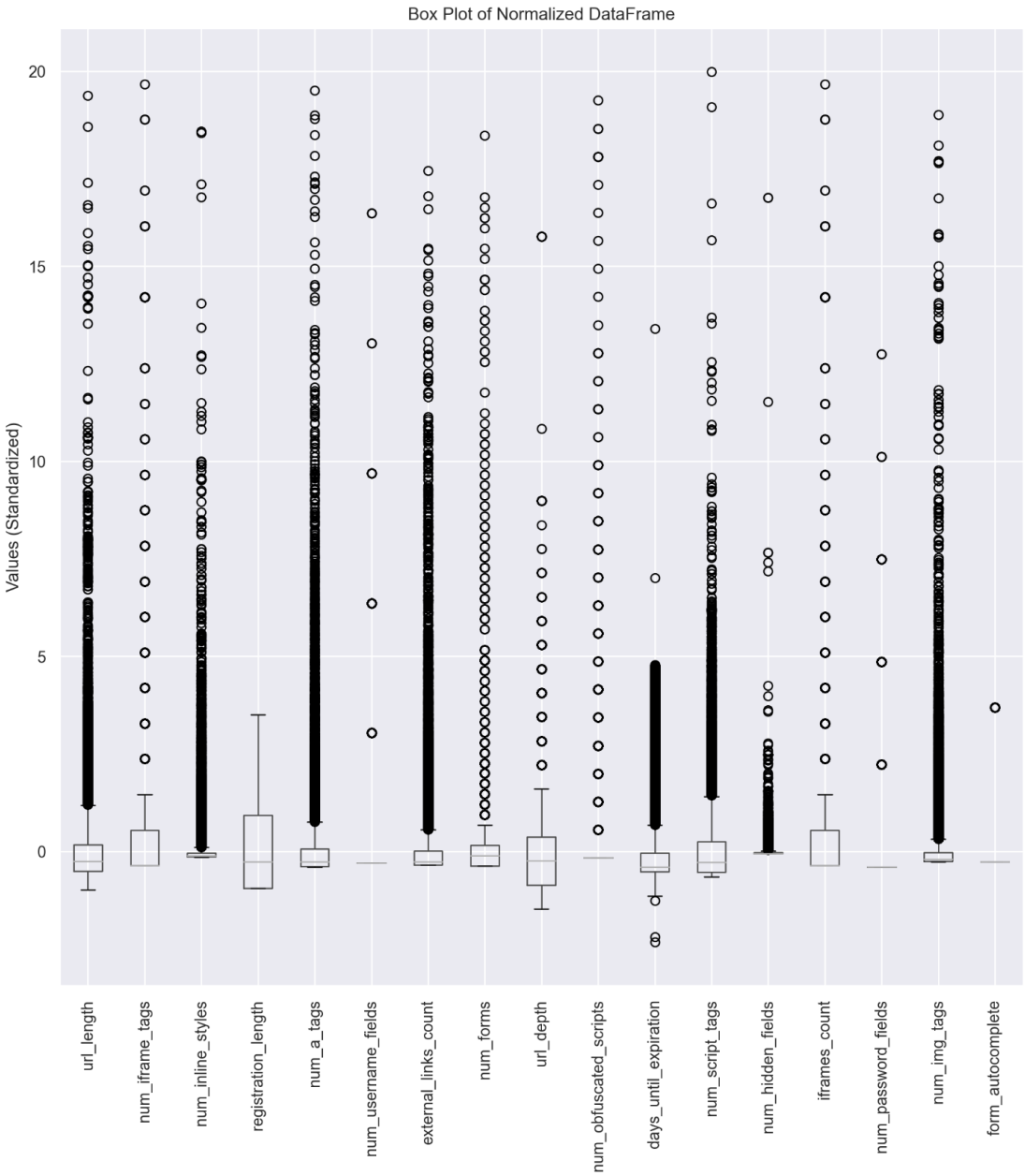
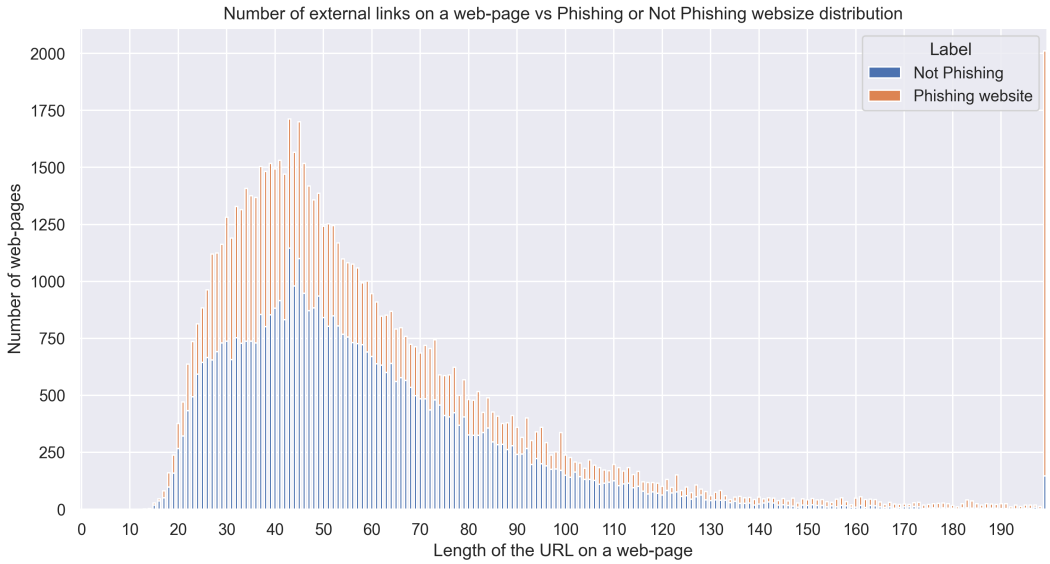


Figure 43

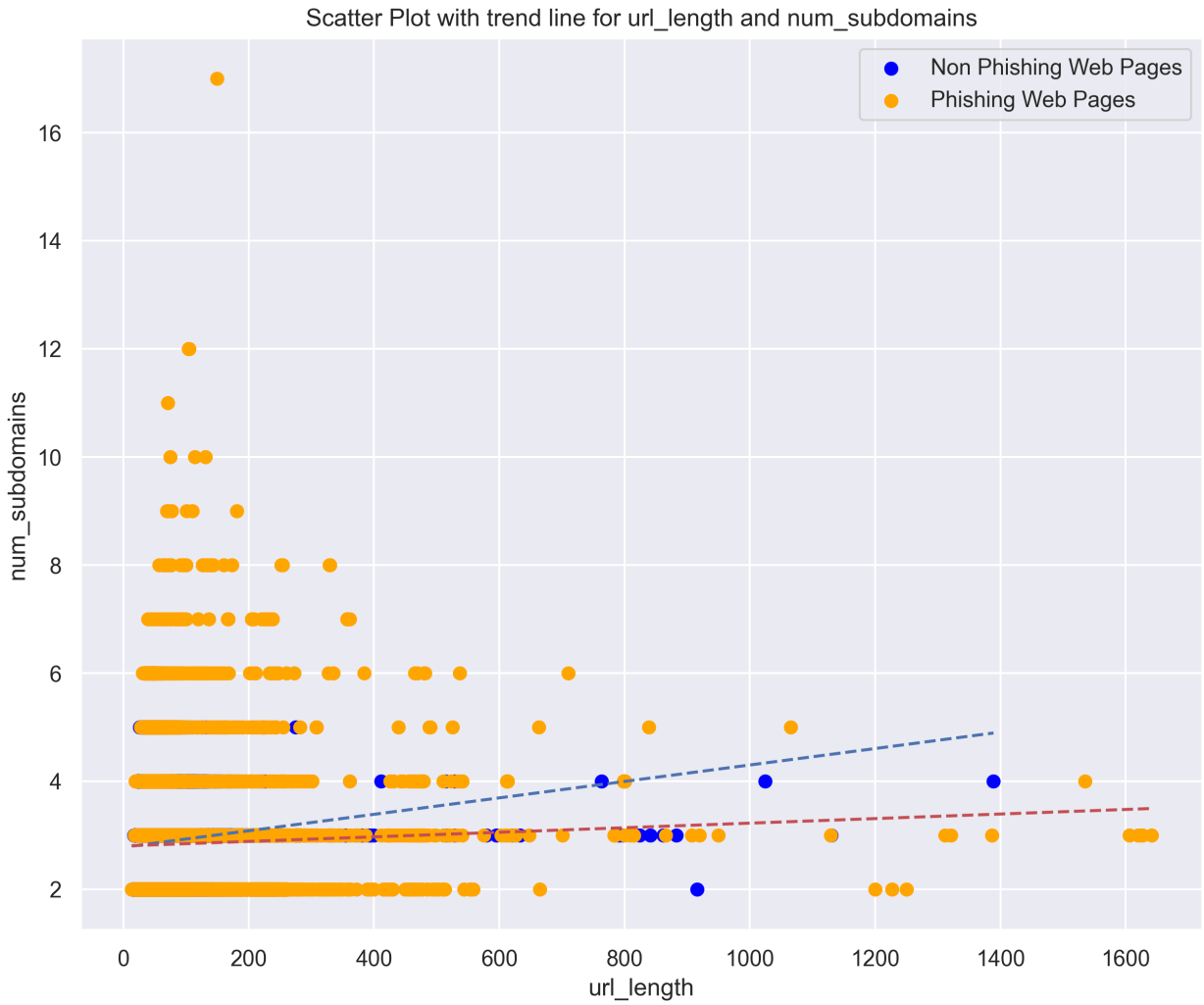
Length of a URL vs Output Labels Distribution Analysis.



Note. This chart shows that till the URL length is 130, the distribution for phishing and legitimate instances are equivalent, but after that, there are more phishing instances. So, if the URL length is higher than 130, it's highly probable to be a phishing website.

Figure 44

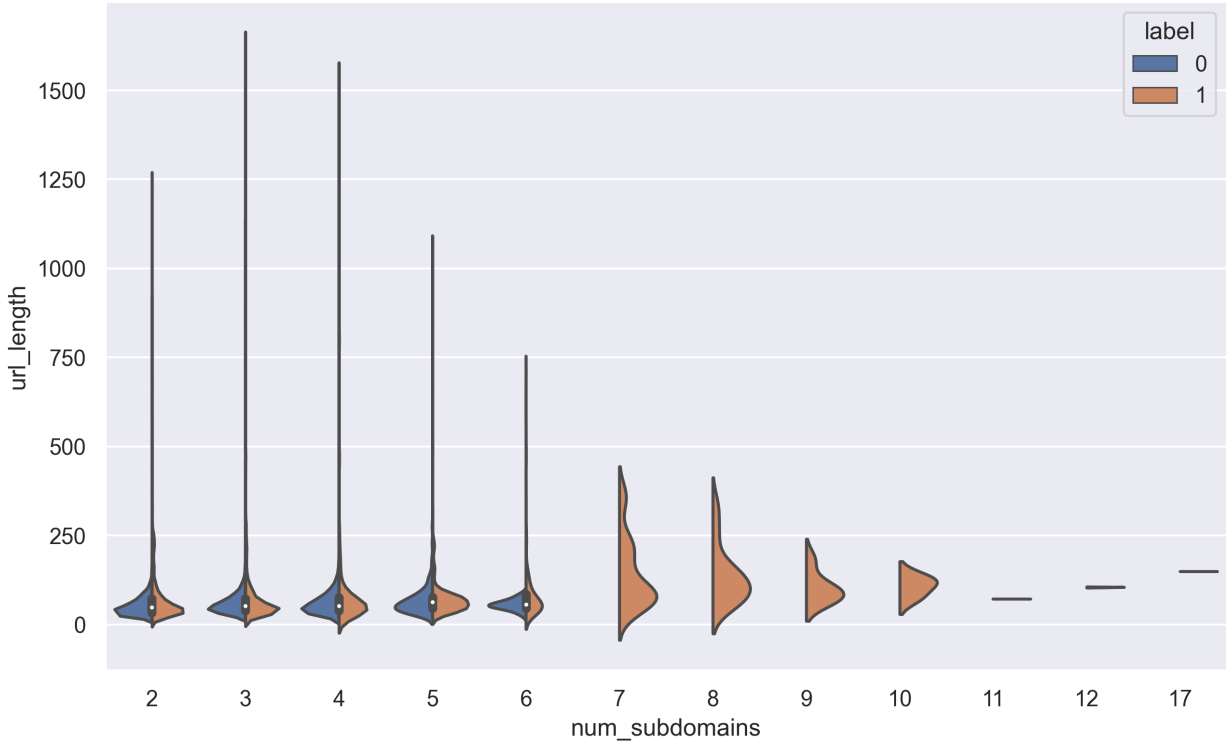
Scatter Plot for the number of sub-domains and URL length fields in a web page.



Note. It's clear that phishing URLs have a lower number of subdomains for the length of the URL compared to legitimate website URLs.

Figure 45

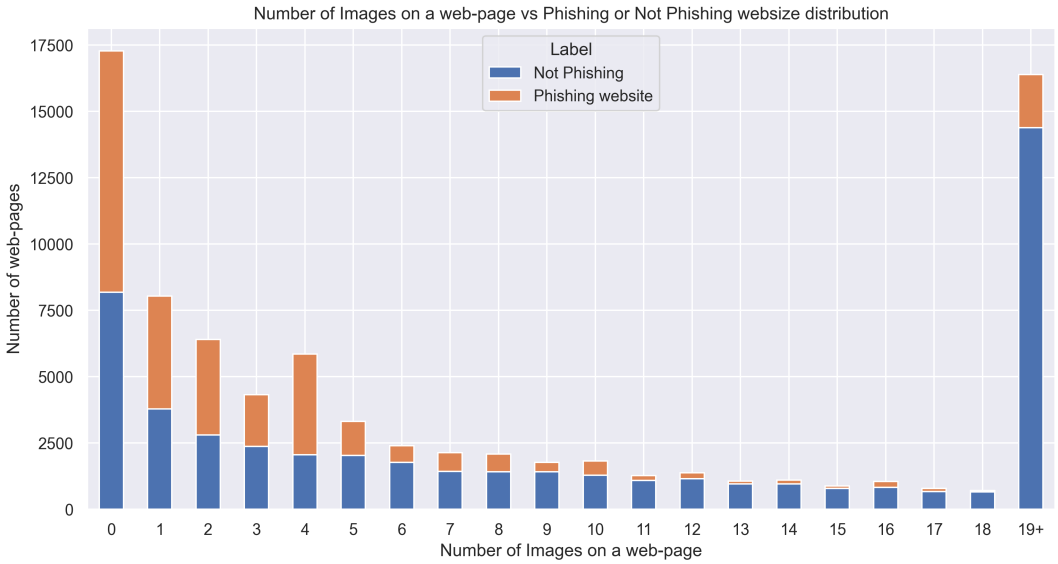
Violin Plot for the number of sub-domains and URL length of the web page.



Note. This violin plot shows legitimate webpages have no higher than 6 sub-domains, but the phishing webpage URL has more subdomains.

Figure 46

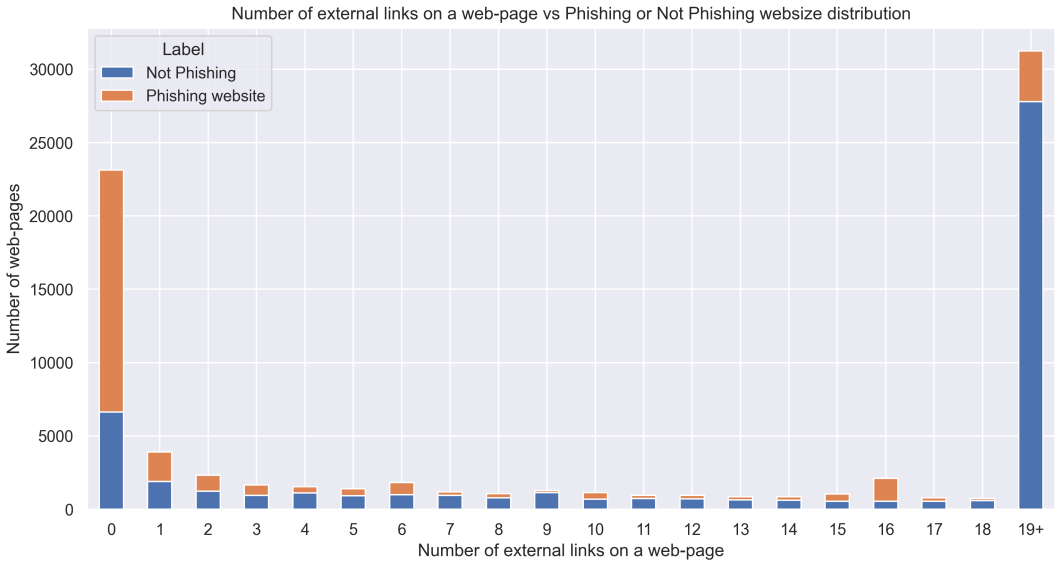
Number of Images in a web page vs Labels Distribution Analysis.



Note. Most of the phishing web pages have roughly 5 images, but as the number of images on a webpage grows, the chances of the page to be a legitimate page are higher.

Figure 47

Number of External Link in a web page vs Labels Distribution Analysis.



Note. It shows that most of the phishing website have fewer links compare to the legitimate websites.

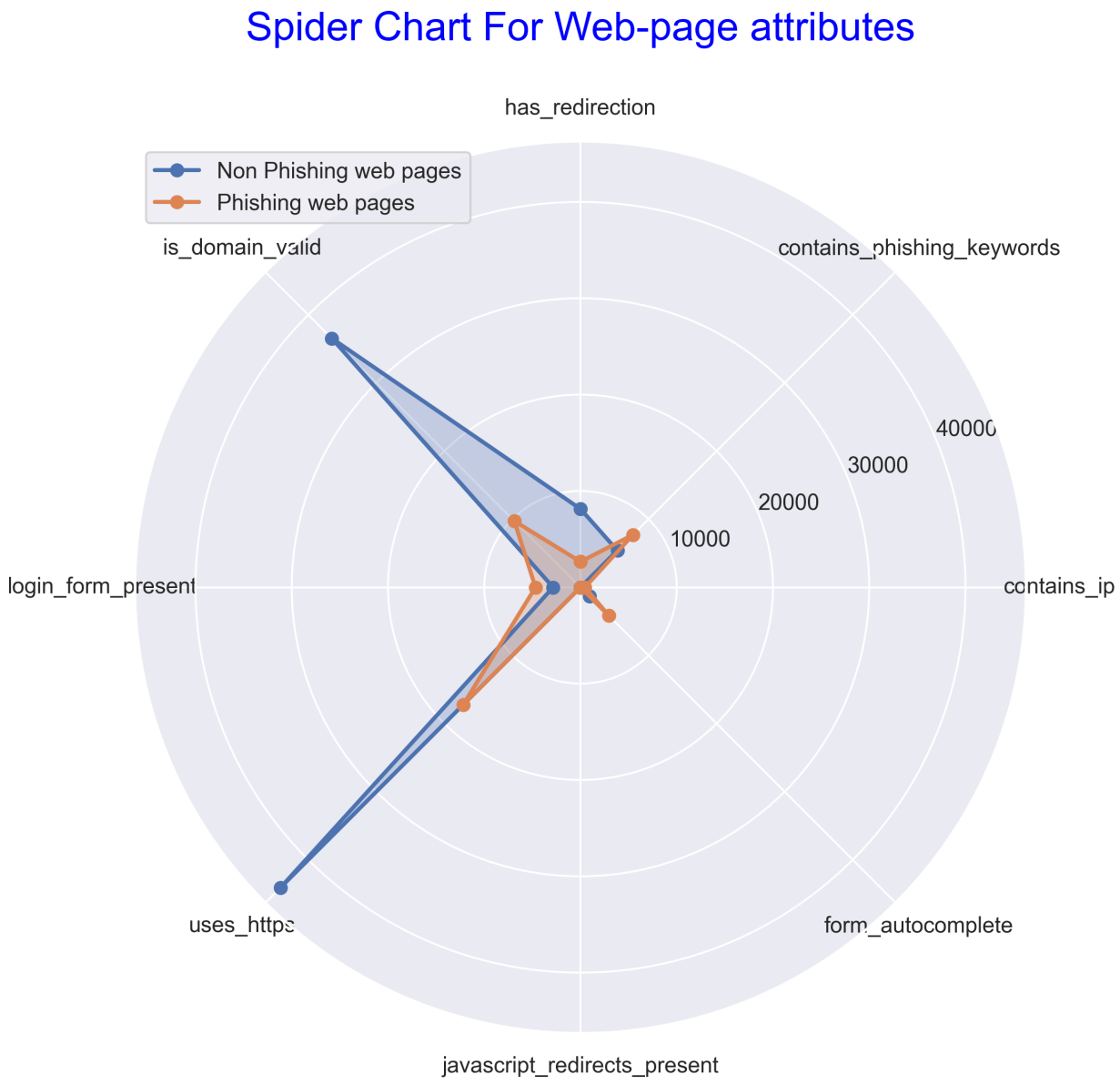
Figure 48

Iframes Count in a web-page vs Labels Distribution Analysis.

Note. It shows that most of the phishing websites don't use iframes.

Figure 49

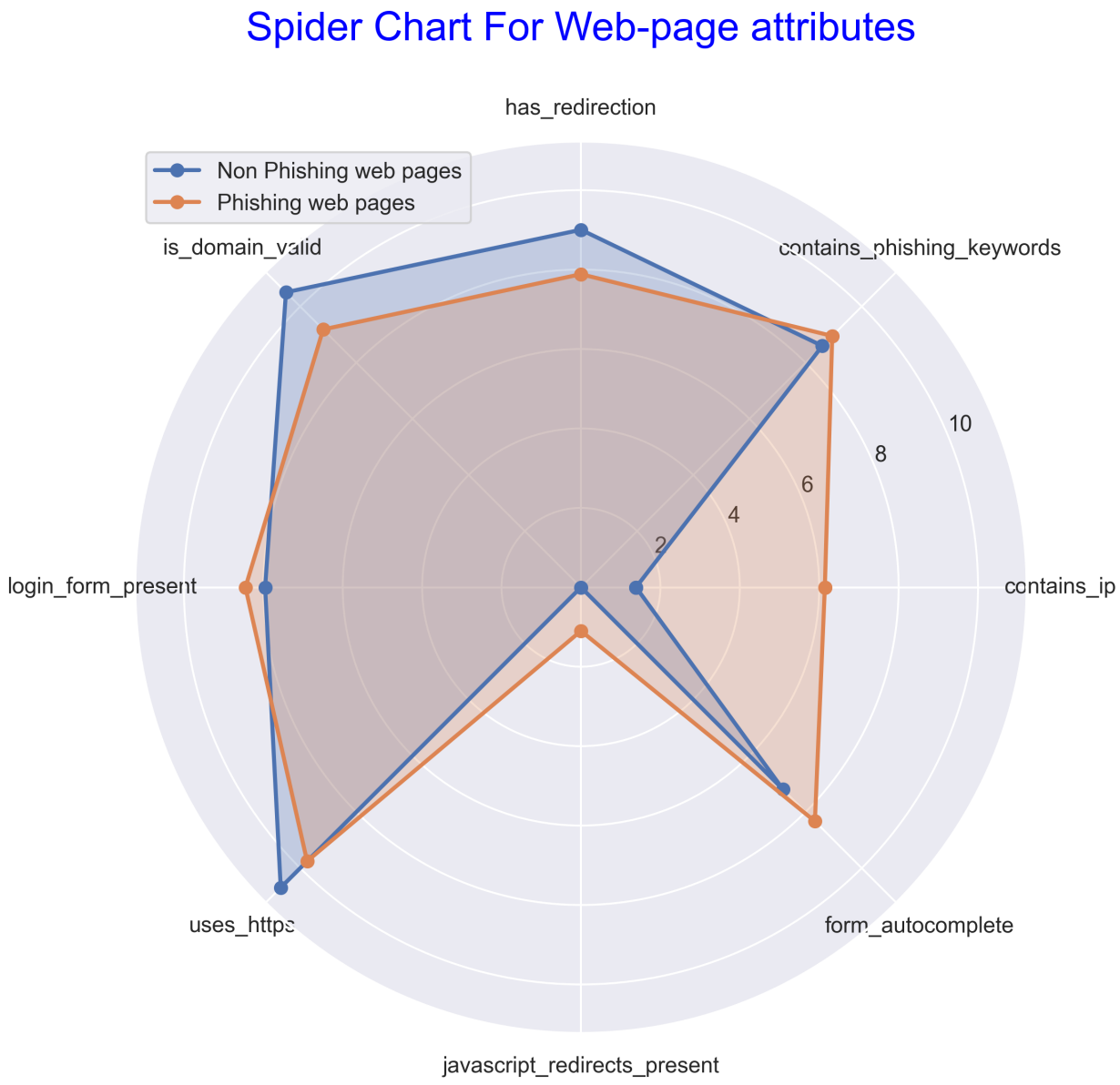
Spider chart for the different web-page attributes.



Note. It is clear from the chart that legitimate pages use HTTPS and they have valid domain names for the URL. Some attributes are not visible, Figures 50 shows the same chart at log level.

Figure 50

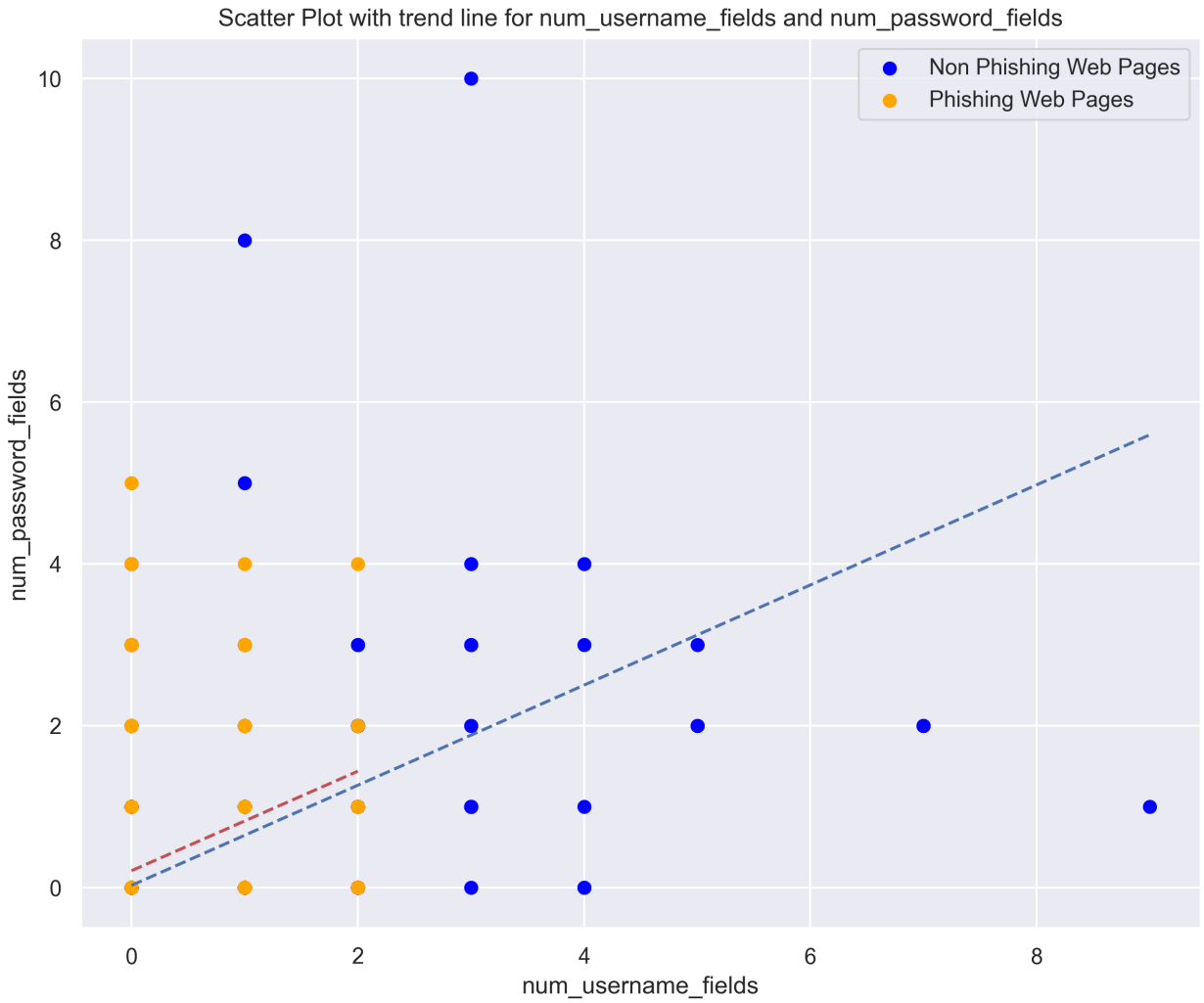
Spider chart for the different web-page attributes using a log scale.



Note. This chart shows more information like phishing URLs user IP addresses, java script redirects, etc.

Figure 51

Scatter Plot for the number of username and password fields in a web page.



Note. The username and password fields follow the same trendline.

Figure 52

Form Action on a web-page vs Labels Distribution Analysis.

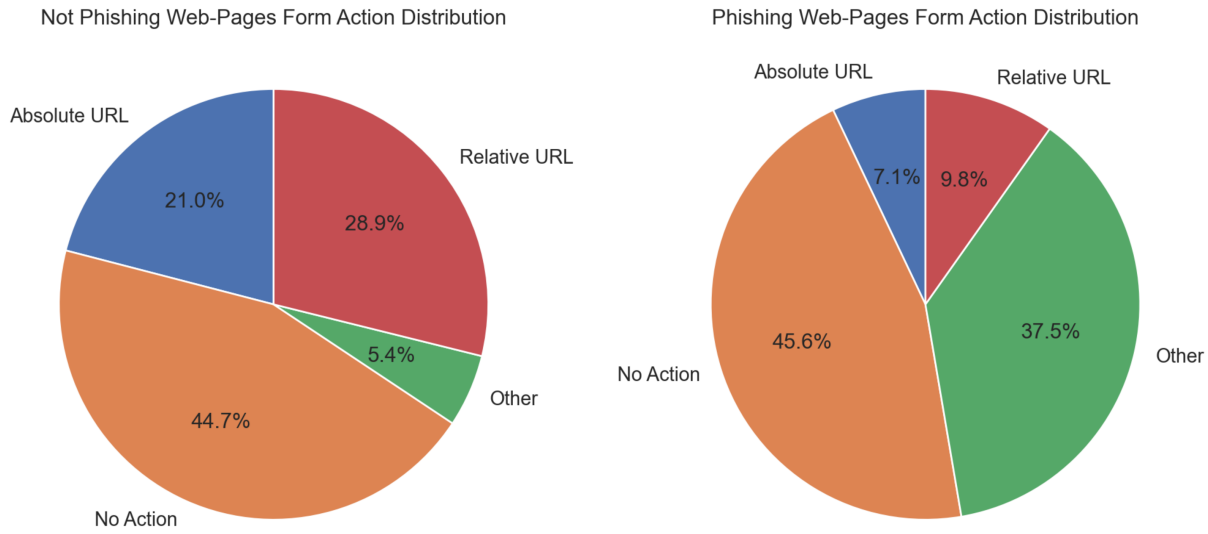
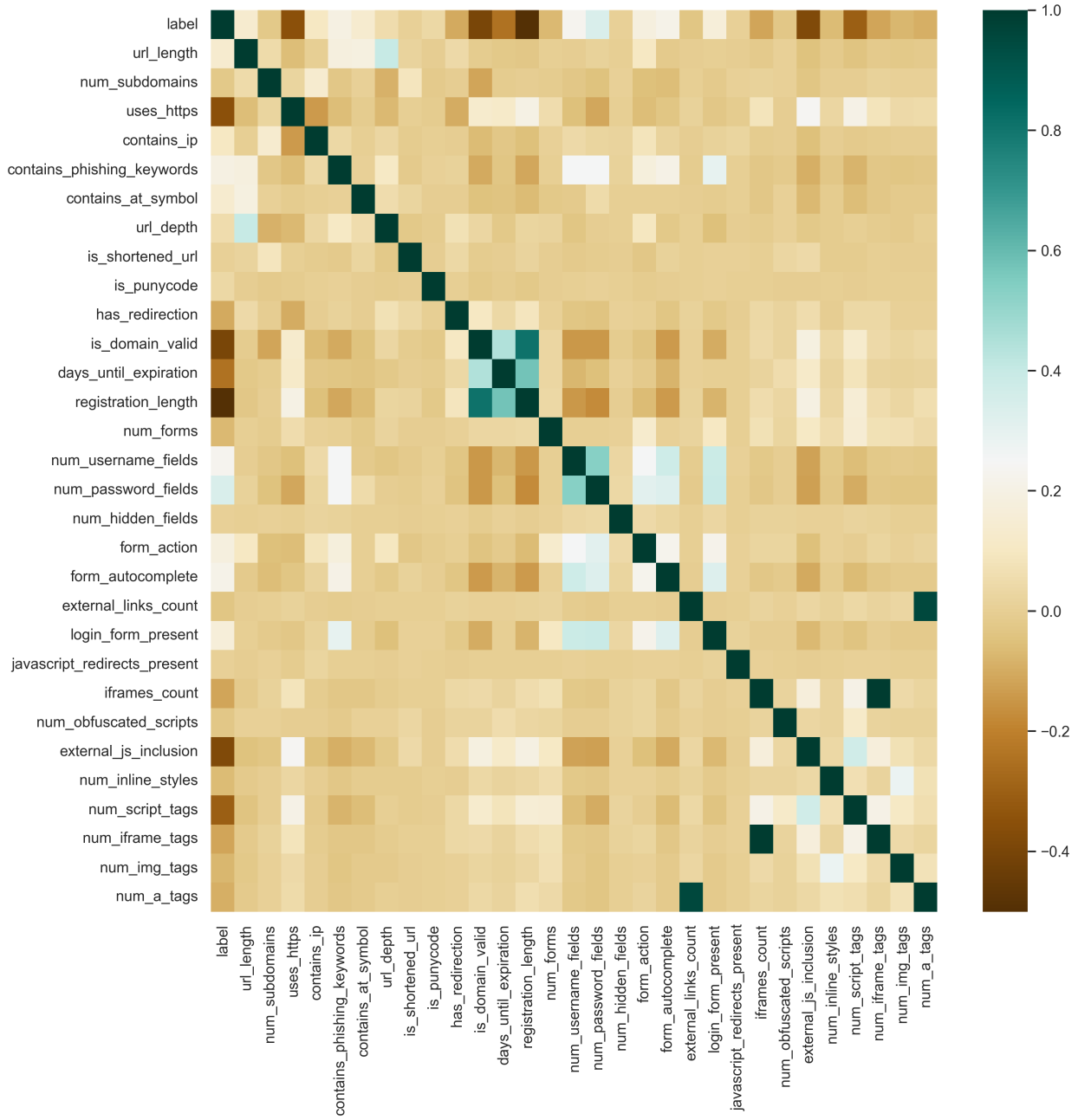


Figure 53

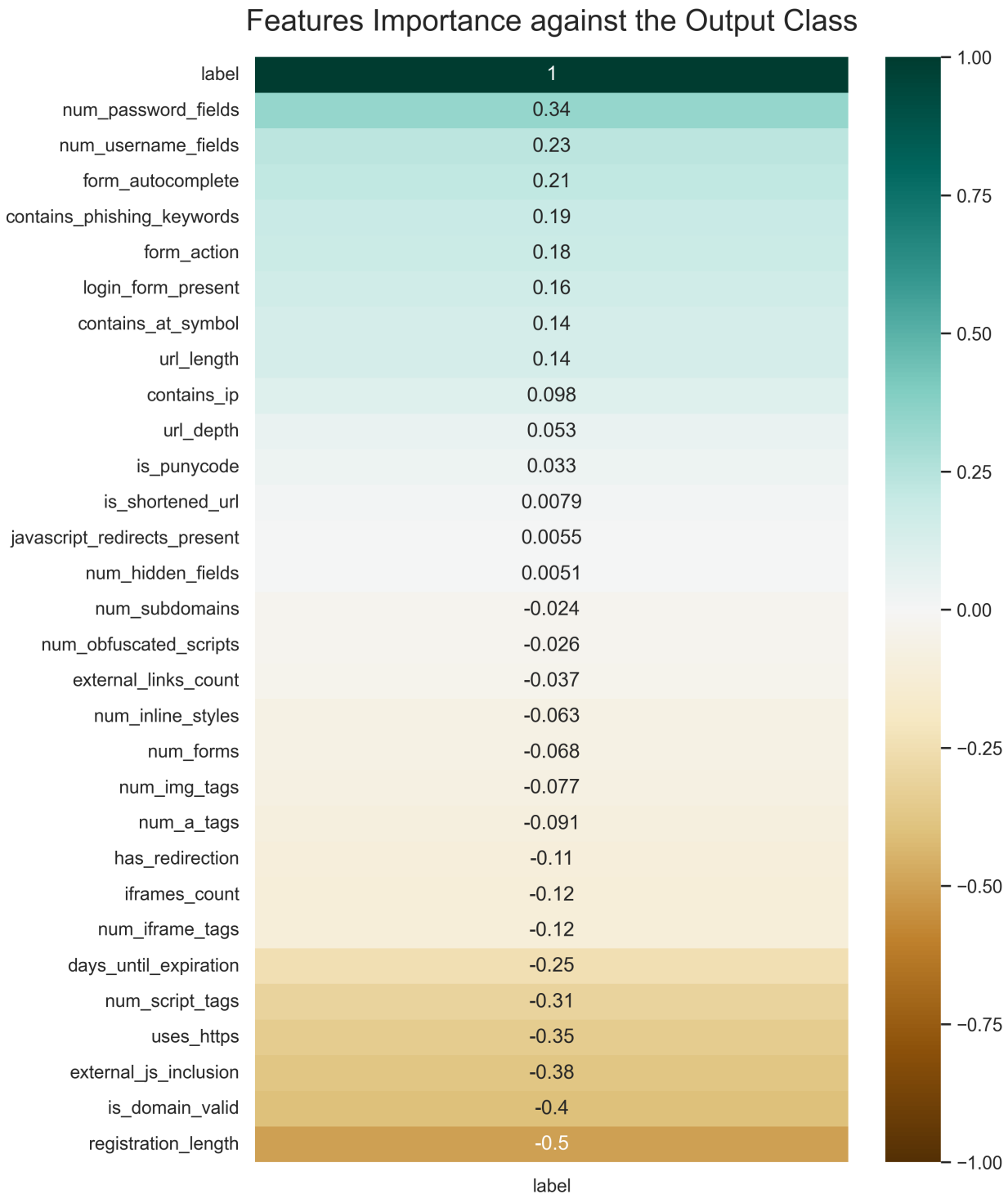
Feature Correlation Analysis Heatmap.



Note. It shows if two features are correlated in the feature set. This analysis helps to remove redundant features that offer no extra information like feature num_a_tags and num_external_links have very high correlation.

Figure 54

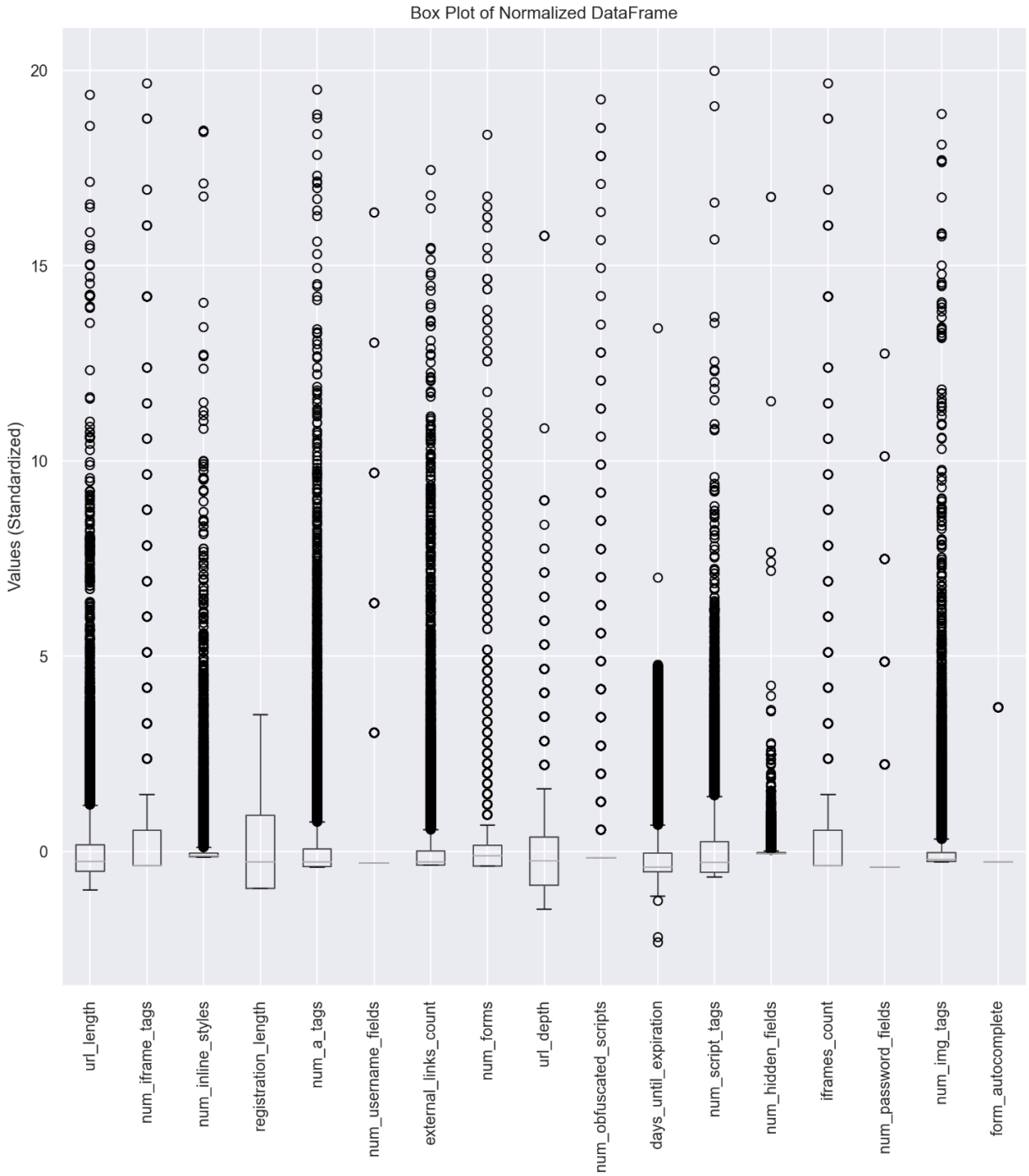
Feature Importance Analysis.



Note. This analysis measures the importance of each feature against the output label.

Figure 55

Normalize Feature Box Plot



Note. This analysis shows the box plots of the features after normalization.

Figure 56

Principal Component Analysis

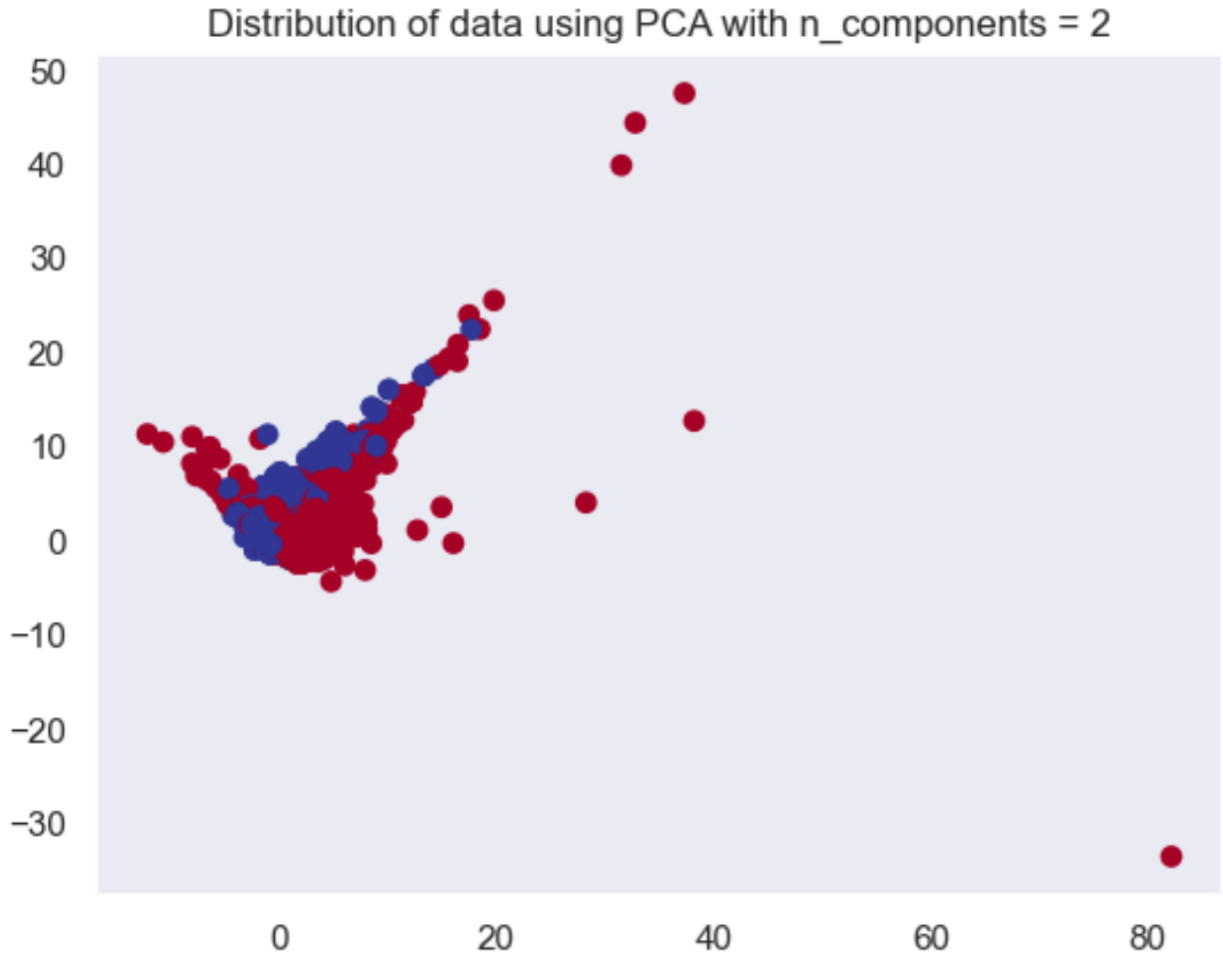
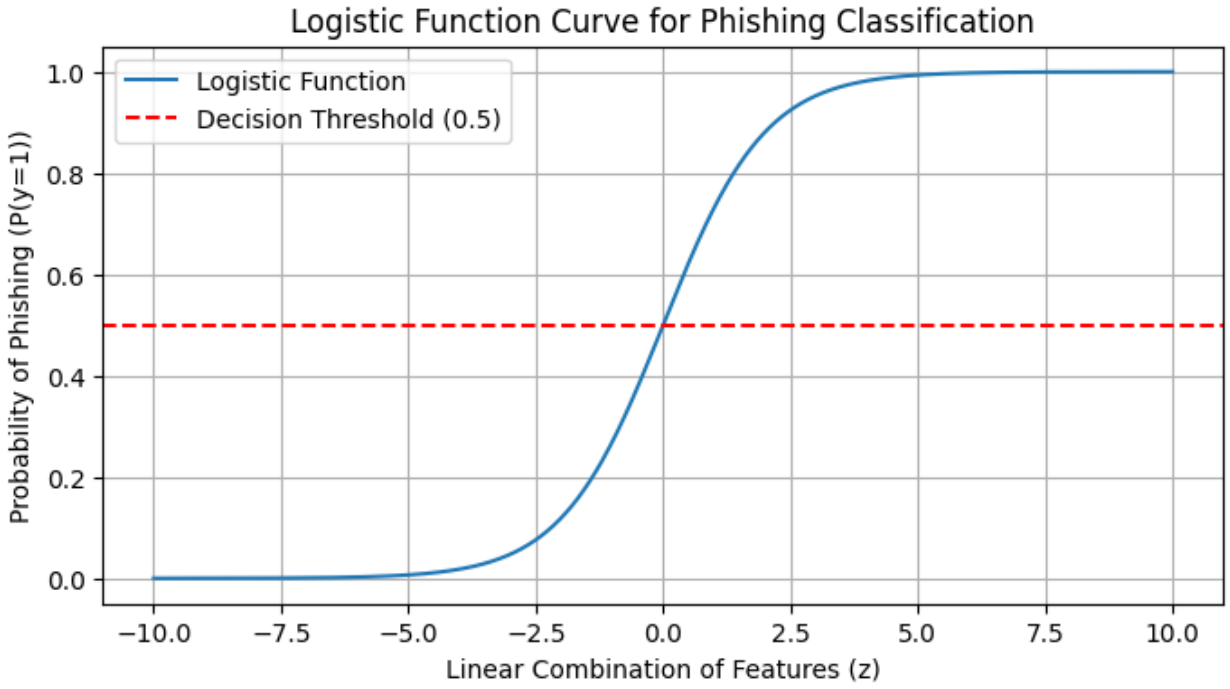


Figure 57

Logistic Function Curve for Phishing Classification



[!ht]

Figure 58

Support Vector Machines

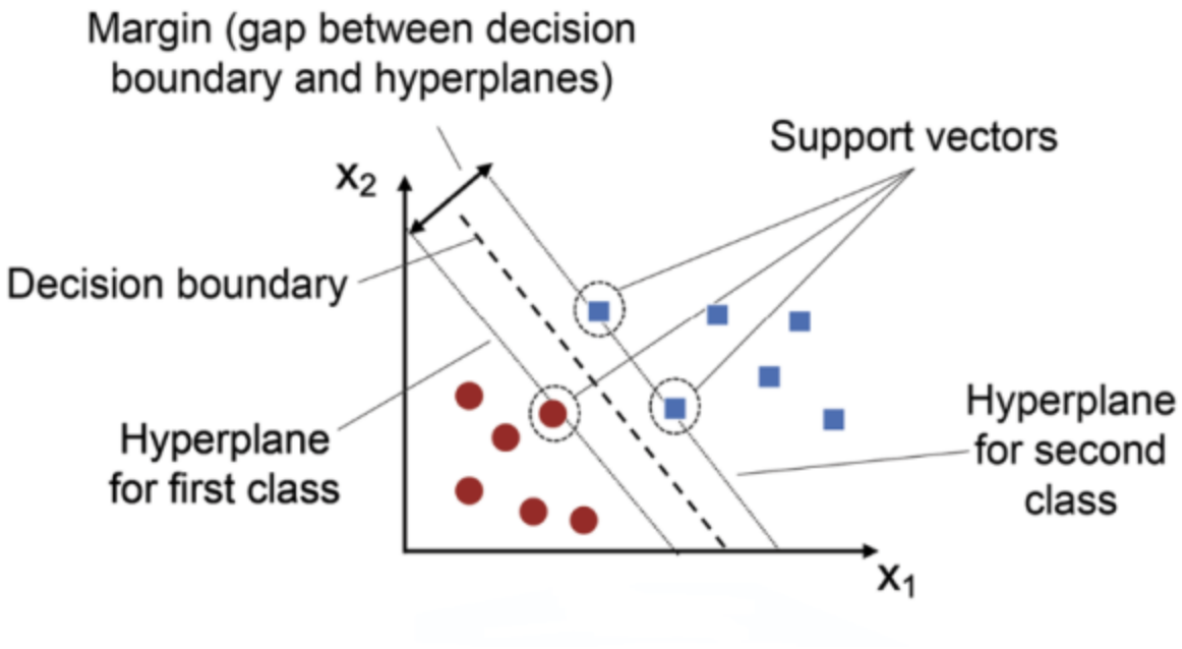


Figure 59

SVM training algorithm (Pedersen & Schoeberl, 2006).

Algorithm 1 Training an SVM

Require: $\boldsymbol{X}$ and $\boldsymbol{y}$ loaded with training labeled data, $\boldsymbol{\alpha} \Leftarrow 0$ or $\boldsymbol{\alpha} \Leftarrow$ partially trained SVM

1: $C \Leftarrow$ some value (10 for example)
2: **repeat**
3: **for all** $\{\boldsymbol{x_i}, y_i\}, \{\boldsymbol{x_j}, y_j\}$ **do**
4: Optimize α_i and α_j
5: **end for**
6: **until** no changes in $\boldsymbol{\alpha}$ or other resource constraint criteria met

Ensure: Retain only the support vectors ($\boldsymbol{\alpha_i} > 0$)

[htbp]

Figure 60

The CART Algorithm

Classification and Regression Tree

1. Start at the root node.
2. For each ordered variable X,
   convert it to an unordered variable X' by grouping its values in the node into a small number of intervals
   if X is unordered, then set X' = X.
3. Perform a chi-squared test of independence of each X' variable versus Y on the data in the node and compute its significance probability.
4. Choose the variable X∗ associated with the X' that has the smallest significance probability.
5. Find the split set {X∗ ∈ S∗} that minimizes the sum of Gini indexes and use it to split the node into two child nodes.
6. If a stopping criterion is reached, exit.
   Otherwise, apply steps 2–5 to each child node.
7. Prune the tree with the CART method.

Figure 61

Random Forest Algorithm (Pedersen & Schoeberl, 2006).

Algorithm 1 Random Forest

```
for i ← 1 to B do
    Draw a bootstrap sample of size N from the training data;
    while node size != minimum node size do
        randomly select a subset of m predictor variables from total p;
        for j ← 1 to m do
            if jth predictor optimizes splitting criterion then
                split internal node into two child nodes;
                break;
            end
        end
    end
end
return the ensemble tree of all B subtrees generated in the outer for loop;
```

Figure 62

Overall System Diagram.

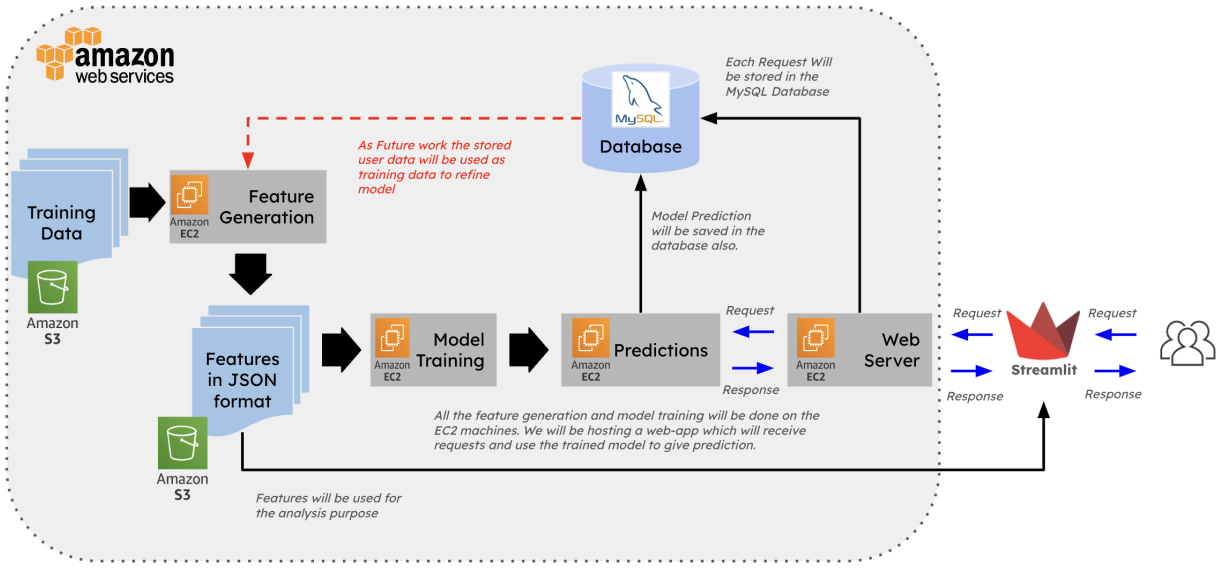


Figure 63

Logistic Regression Process Architecture

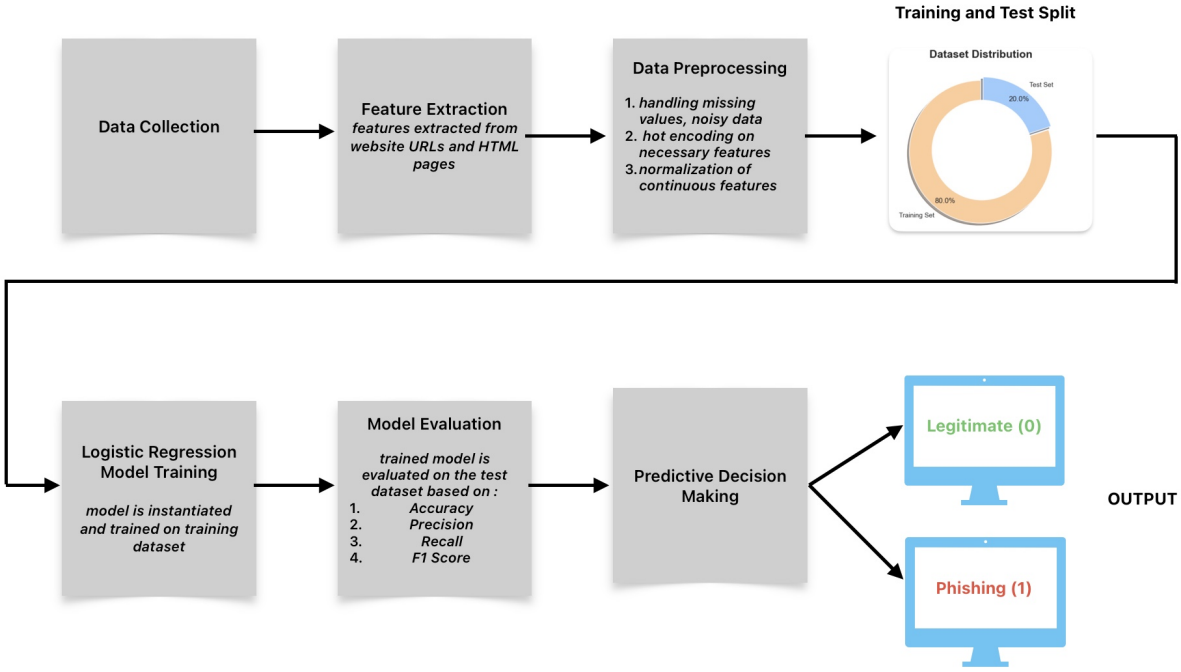
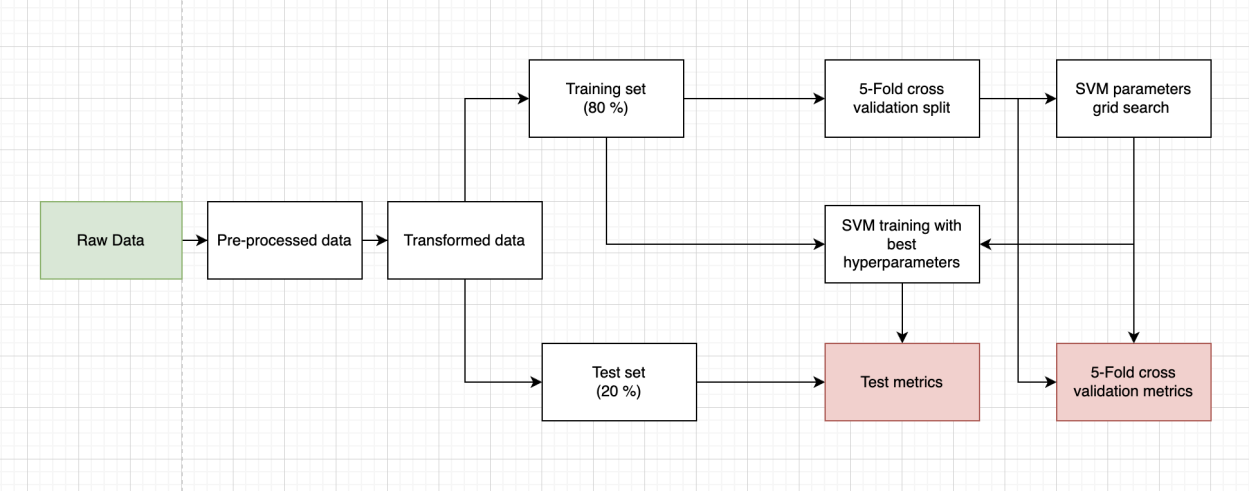


Figure 64

Model Architecture and Data Flow for SVM.



Note. The green node shows the starting point while the red nodes show the end points.

Figure 65

Data Flow and Model Architecture of Decision Tree Model

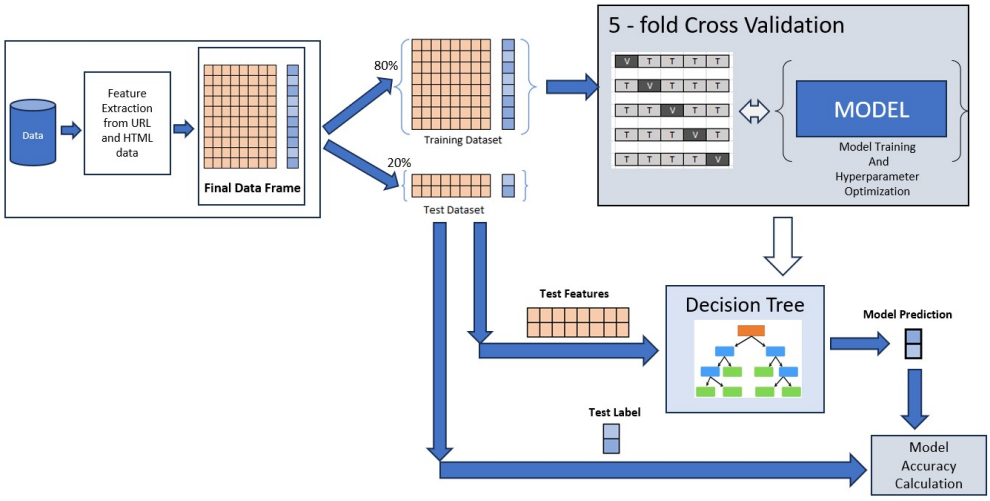
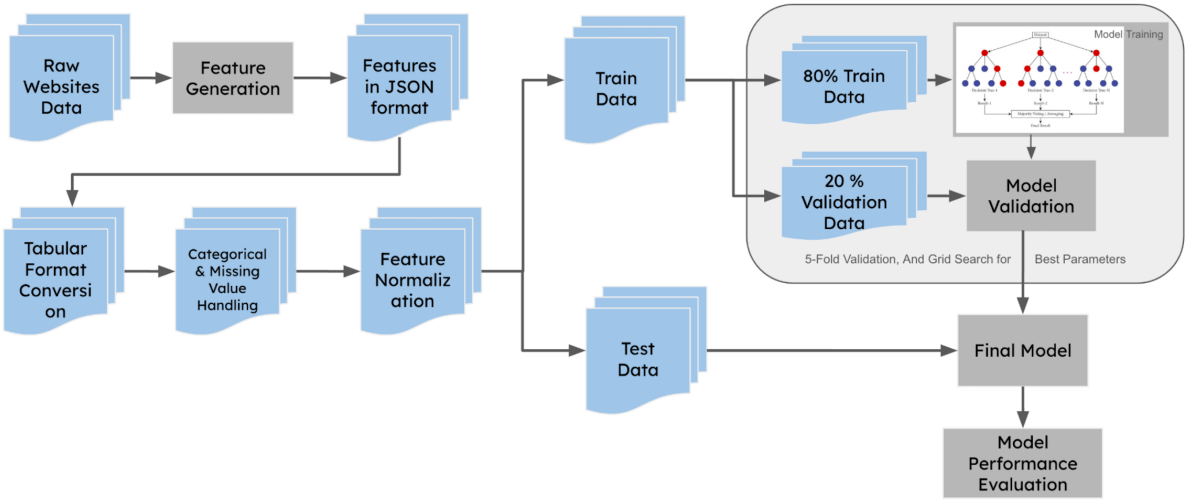


Figure 66

Data flow diagram for Random Forest model



Data Flow from Raw Data to Model Evaluation

Figure 67

Random Forest Flowchart, take from Rodriguez-Galiano et al., 2016

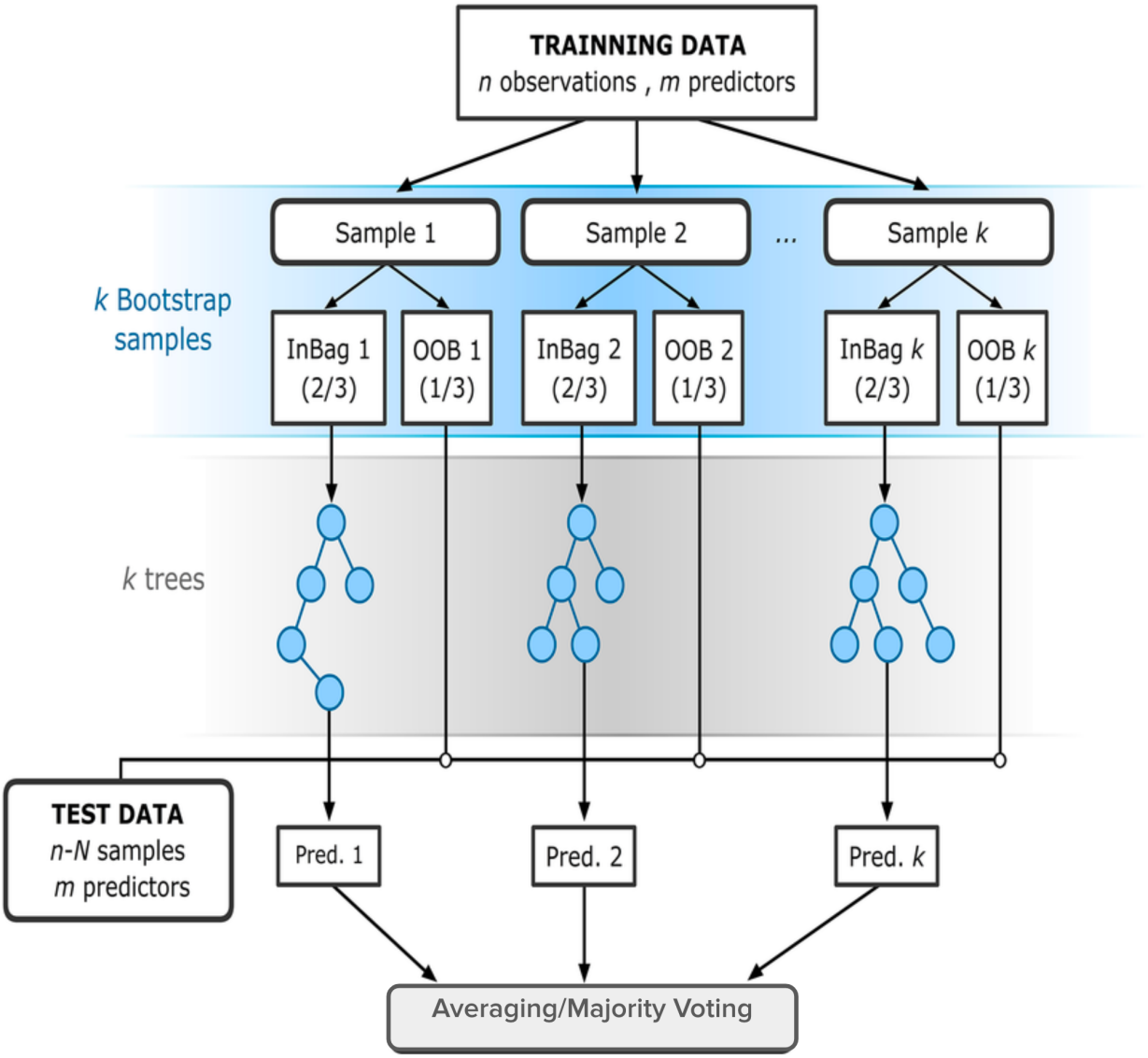


Figure 68

Top 3 levels of one of the Decision Tree (First Estimator in the Random Forest

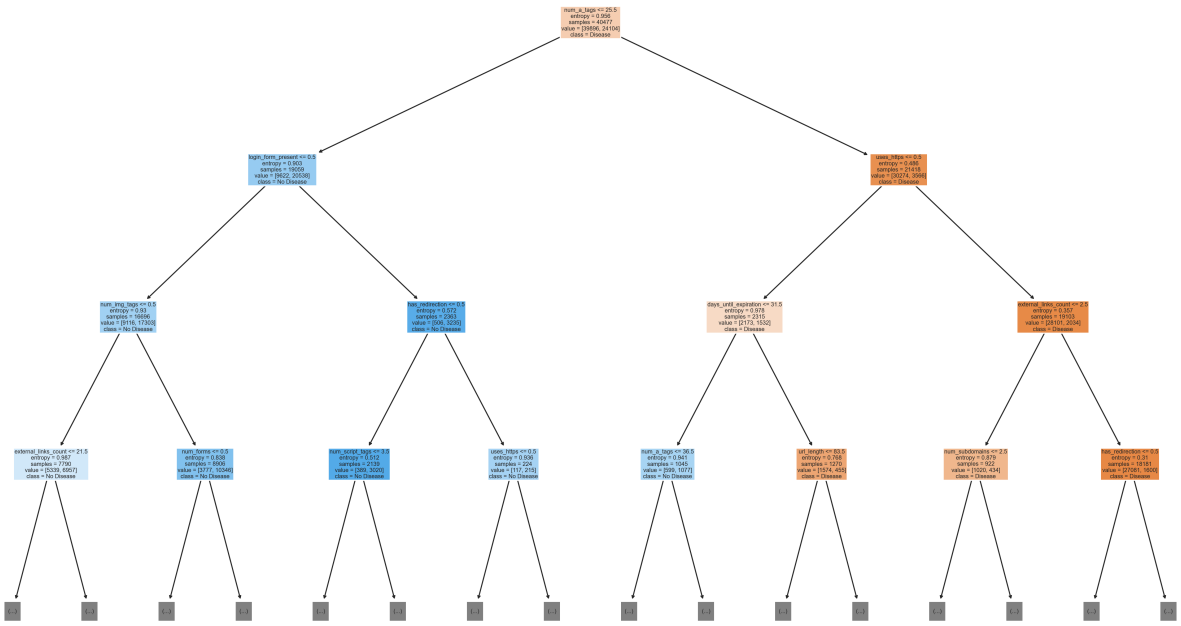


Figure 69

Feature Importance calculated for each feature in the Random Forest Model

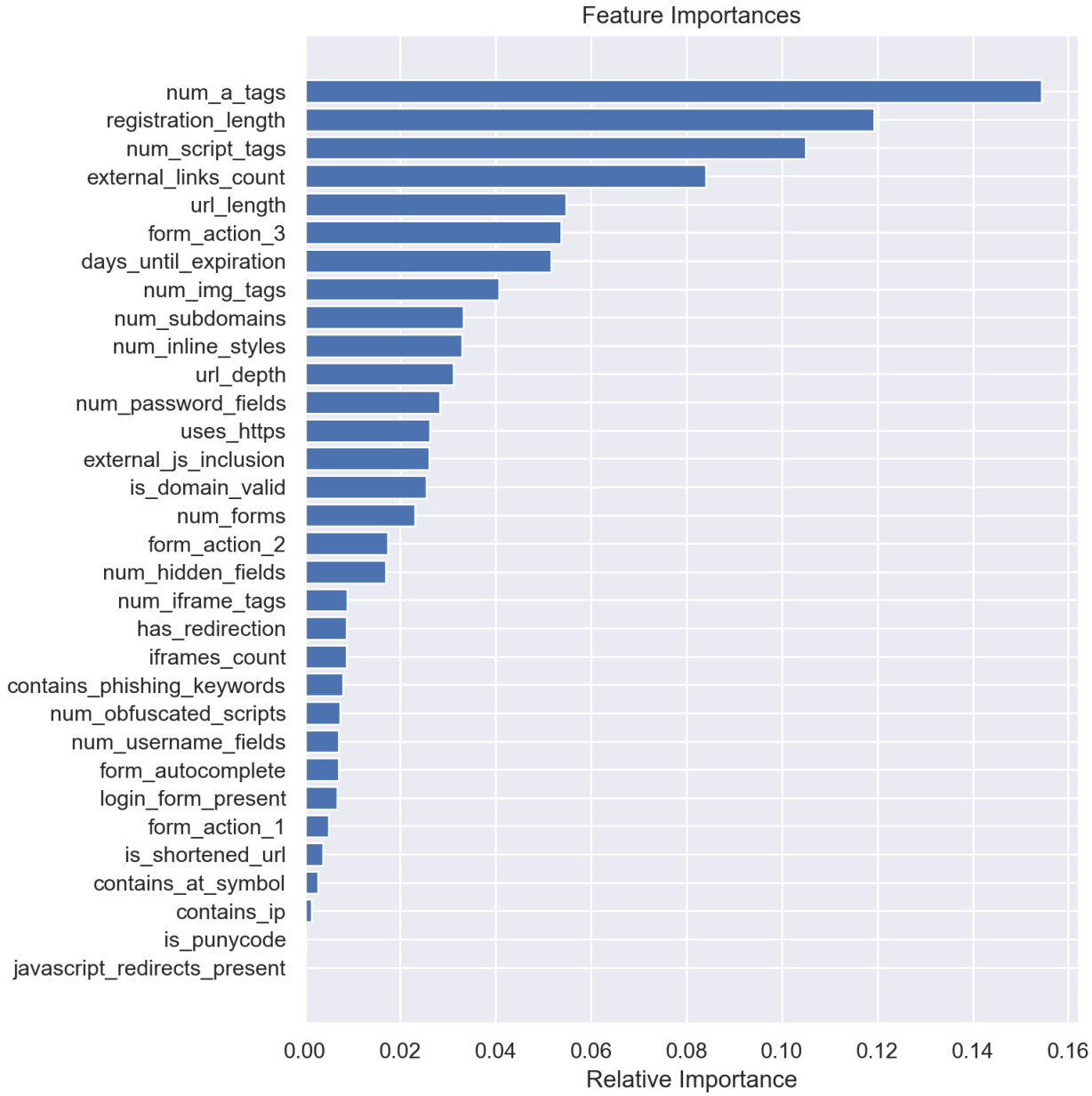


Figure 70

XG Boost Model, tree creation (Guo et al., 2020)

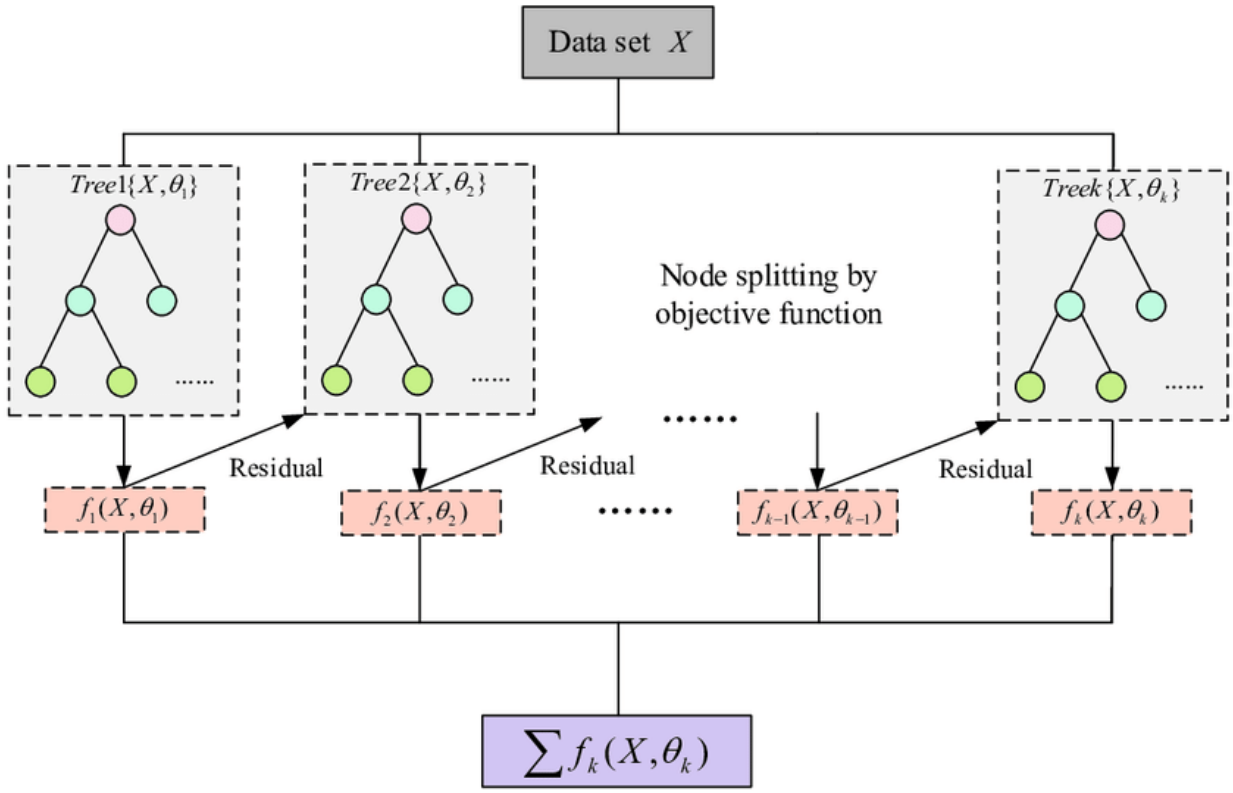


Figure 71

Confusion matrix.

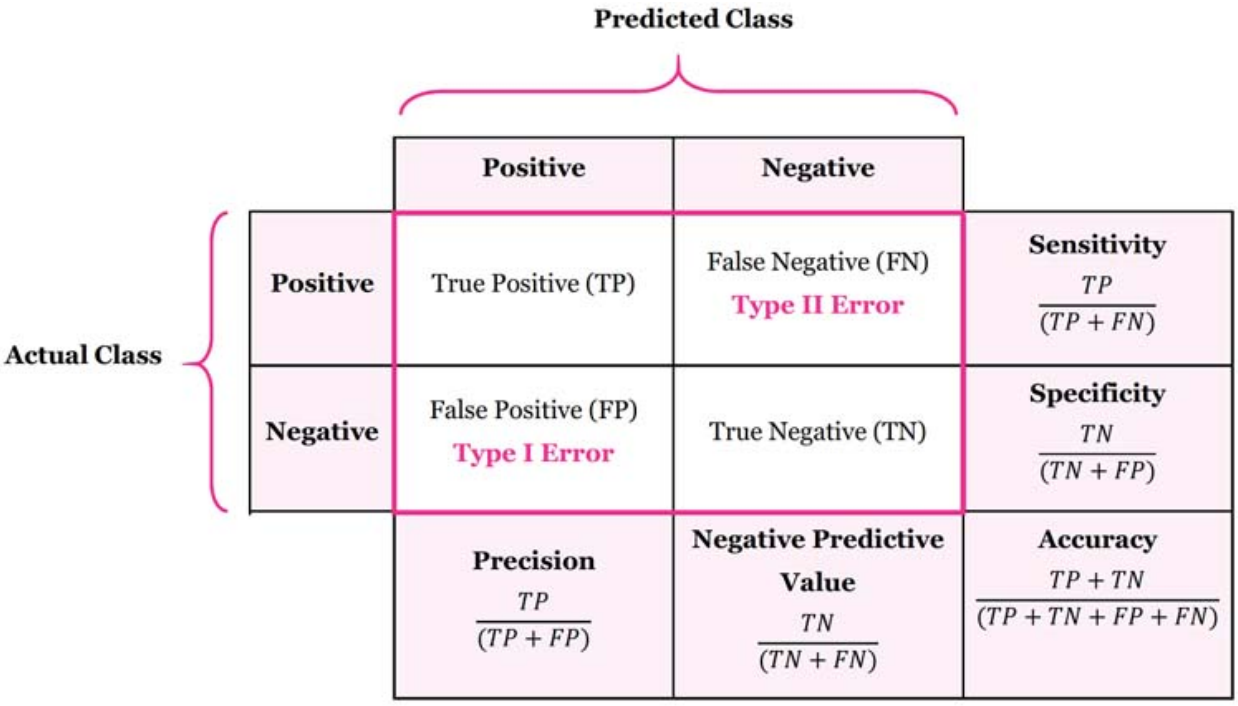
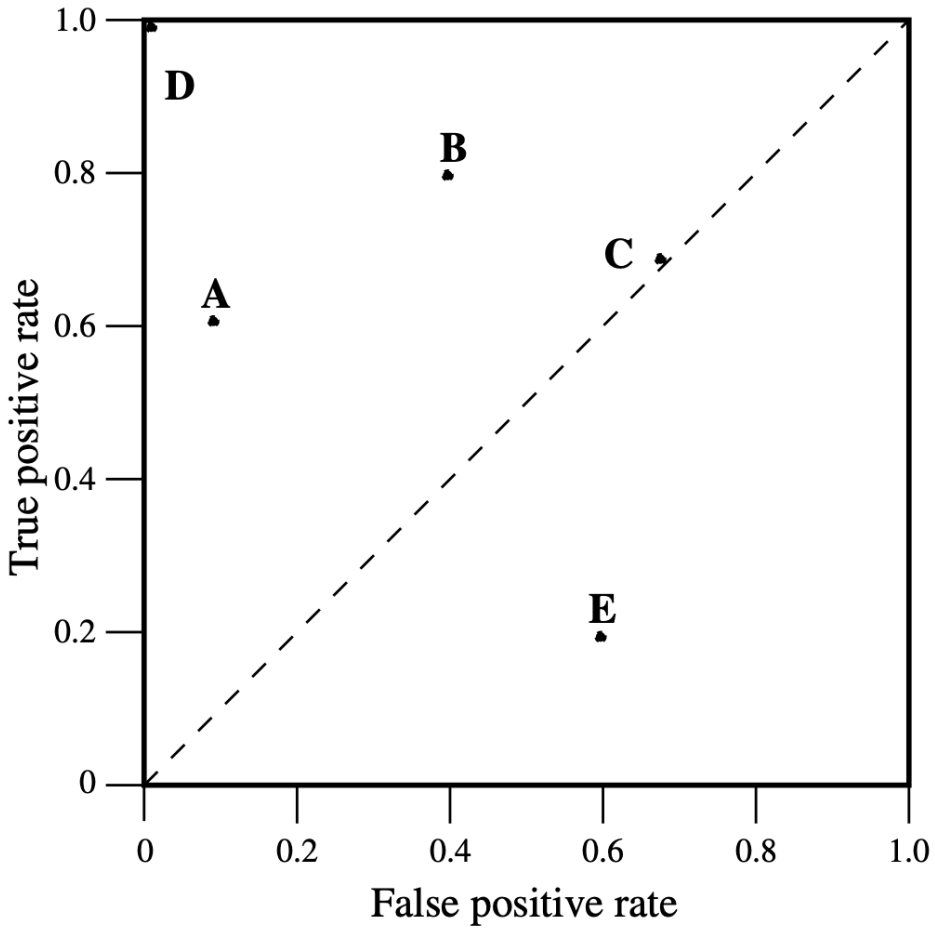


Figure 72

ROC curve showing five classifiers (Fawcett, 2006).



[!ht]

Figure 73

5-fold cross-validation.

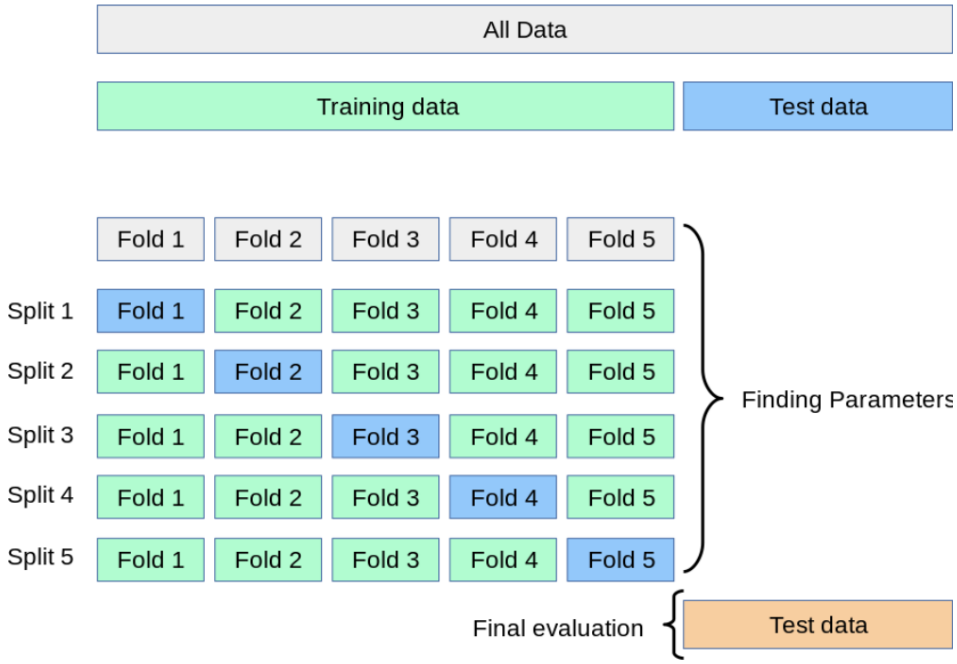
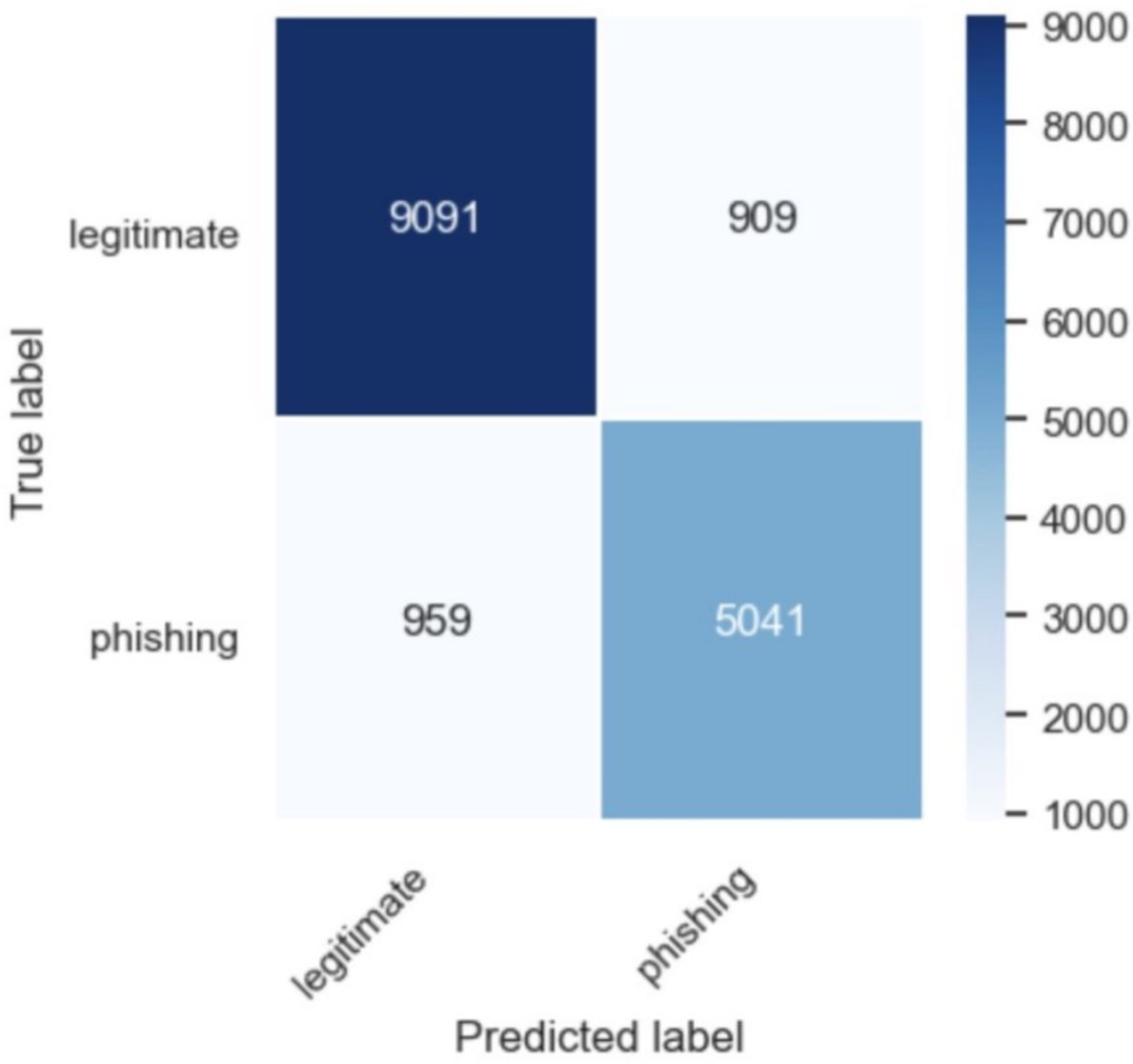


Figure 74

Confusion Metrics for Logistic Regression Model



	precision	recall	f1-score	support
legitimate	0.9046	0.9091	0.9068	10000
phishing	0.8472	0.8402	0.8437	6000
accuracy			0.8833	16000
macro avg	0.8759	0.8746	0.8753	16000
weighted avg	0.8831	0.8832	0.8832	16000

Accuracy: 0.8832
Micro Precision: 0.8832

Figure 75

AUC under ROC for Logistic Regression.

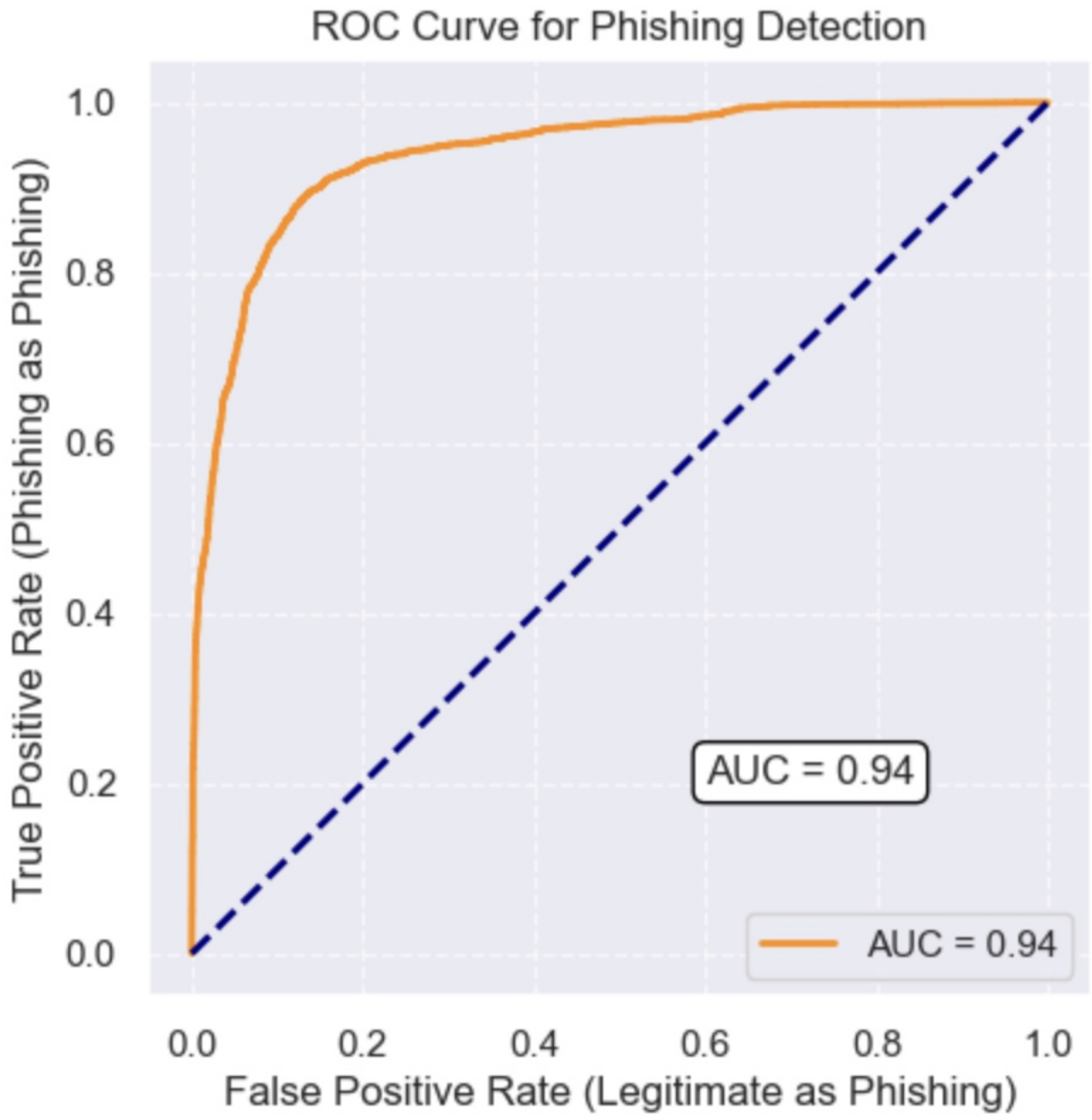
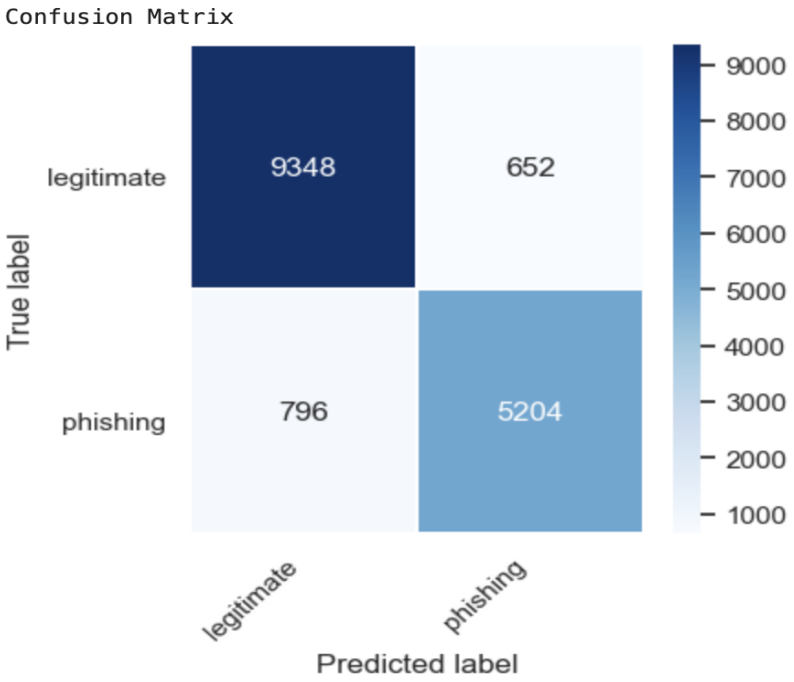


Figure 76

SVM baseline performance metrics with default parameters on the test set.



	precision	recall	f1-score	support
legitimate	0.9215	0.9348	0.9281	10000
phishing	0.8887	0.8673	0.8779	6000
accuracy			0.9095	16000
macro avg	0.9051	0.9011	0.9030	16000
weighted avg	0.9092	0.9095	0.9093	16000

Accuracy: 0.9095
Micro Precision: 0.9095
Micro Recall/TPR: 0.9095
Micro f1-score: 0.9095

Figure 77

ROC curve for the baseline SVM model.

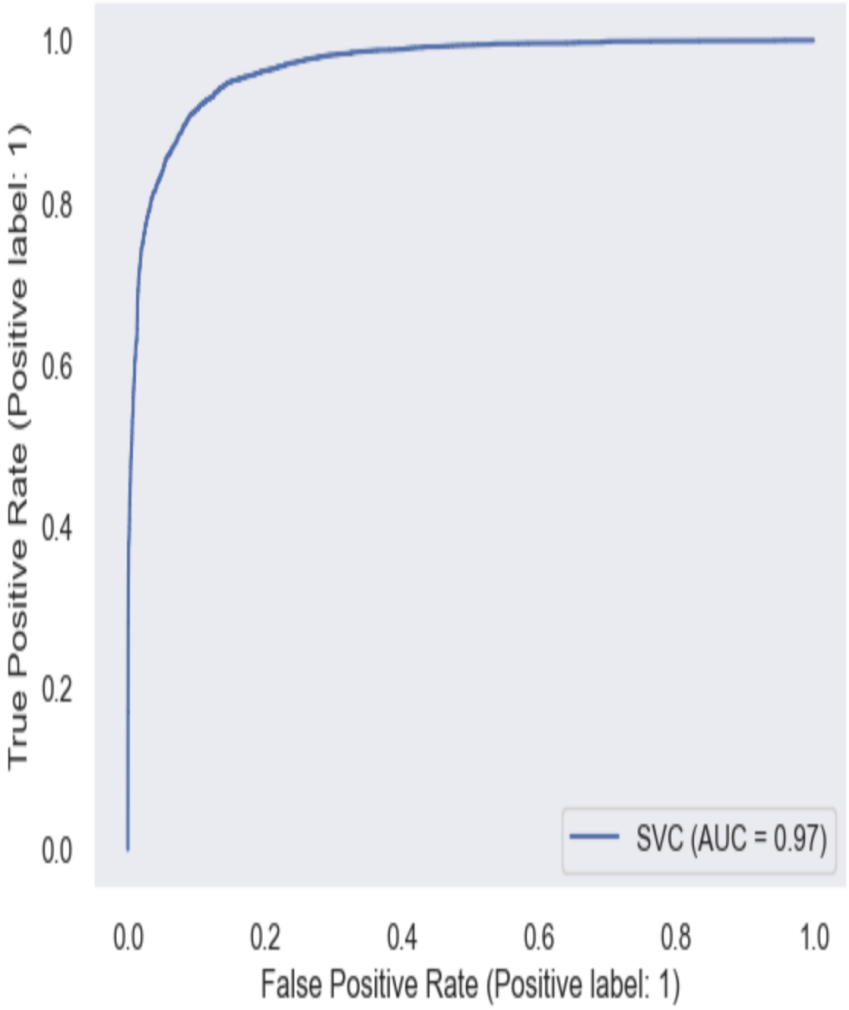
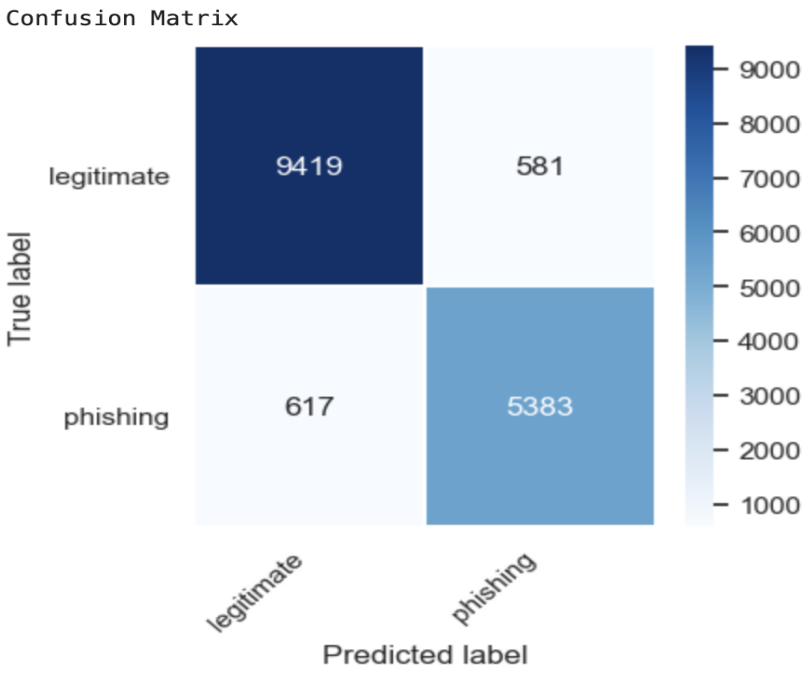


Figure 78

SVM performance metric with tuned hyperparameters on the test set.



	precision	recall	f1-score	support
legitimate	0.9385	0.9419	0.9402	10000
phishing	0.9026	0.8972	0.8999	6000
accuracy			0.9251	16000
macro avg	0.9206	0.9195	0.9200	16000
weighted avg	0.9250	0.9251	0.9251	16000

Accuracy: 0.9251
Micro Precision: 0.9251
Micro Recall/TPR: 0.9251
Micro f1-score: 0.9251

Figure 79

ROC curve for the tuned SVM model.

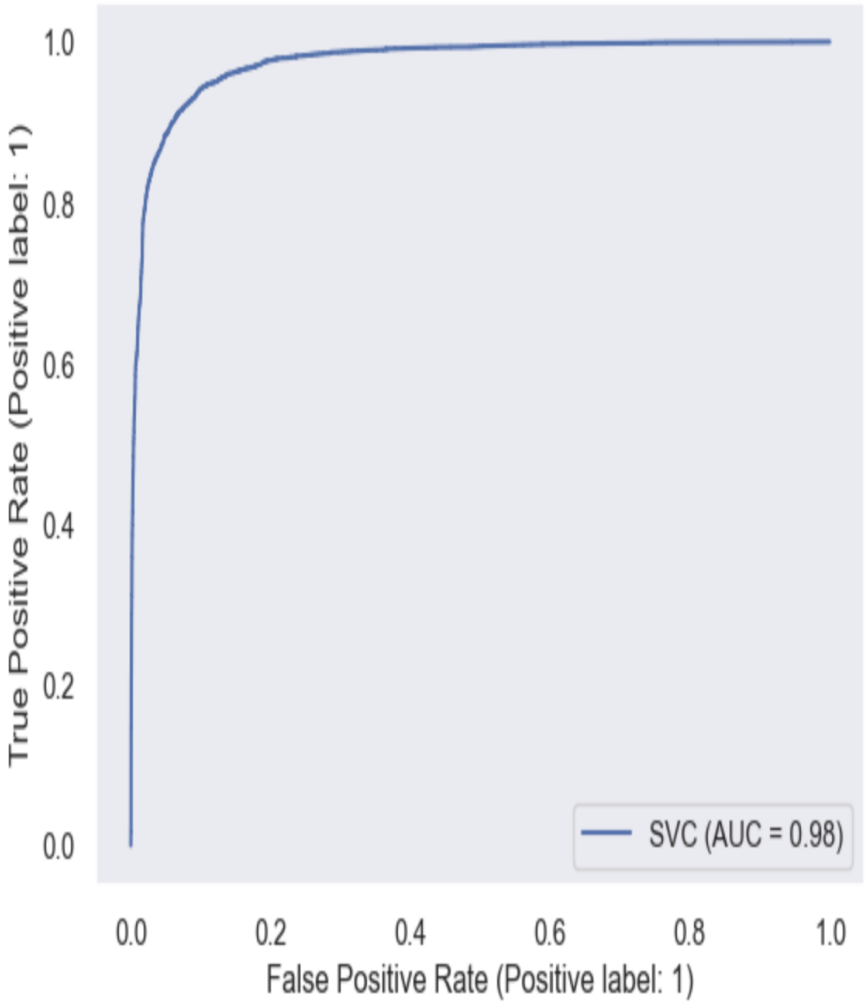
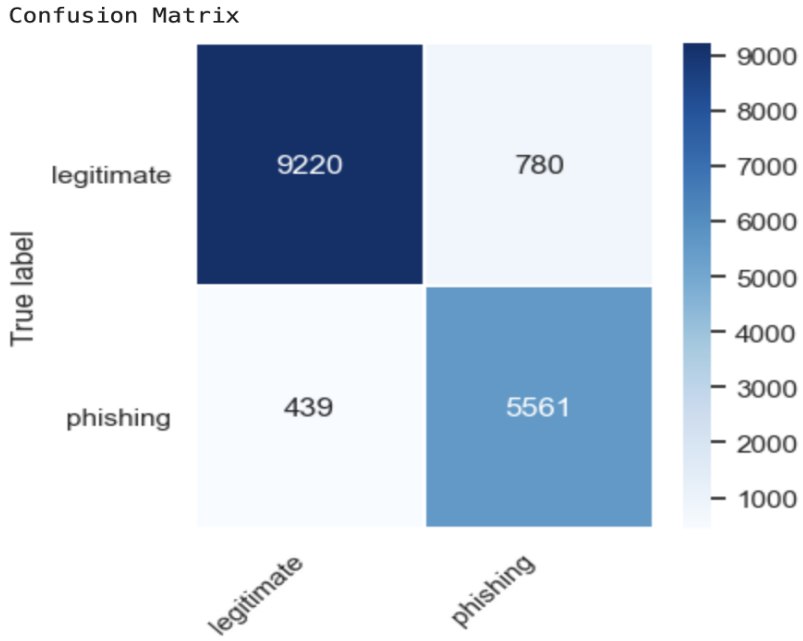


Figure 80

SVM performance metrics using tuned hyperparameters and SMOTE oversampling regularization on the test set.



	precision	recall	f1-score	support
legitimate	0.9546	0.9220	0.9380	10000
phishing	0.8770	0.9268	0.9012	6000
accuracy			0.9238	16000
macro avg	0.9158	0.9244	0.9196	16000
weighted avg	0.9255	0.9238	0.9242	16000

Accuracy: 0.9238
Micro Precision: 0.9238
Micro Recall/TPR: 0.9238
Micro f1-score: 0.9238

Figure 81

ROC curve for the SVM model with SMOTE.

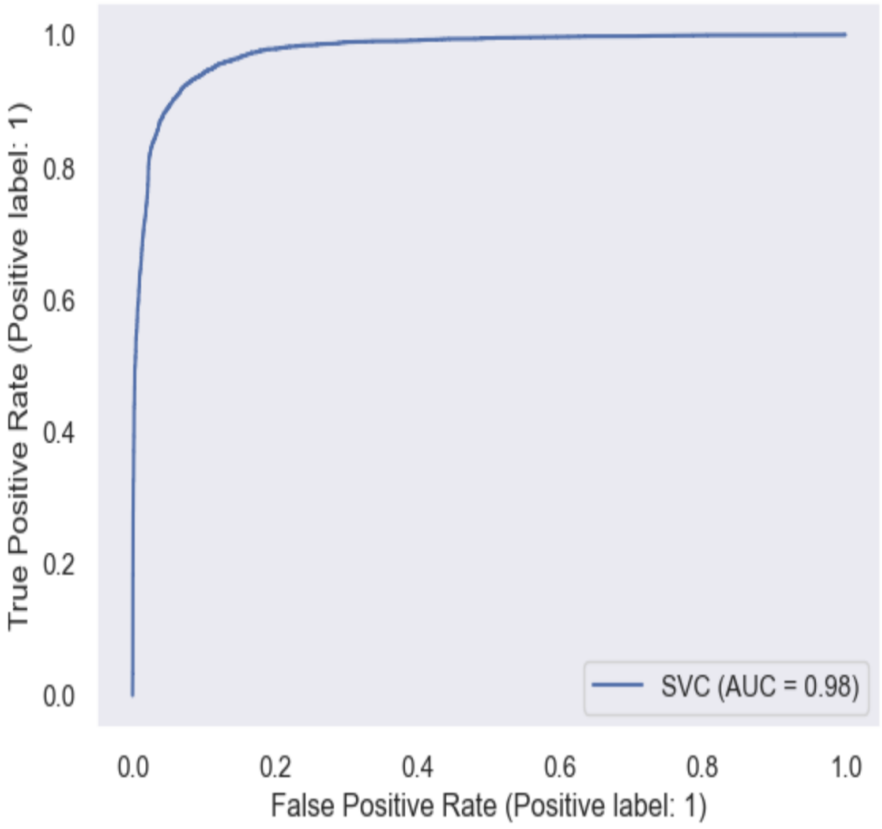


Figure 82

Effect of dimensionality reduction of the performance of SVM.

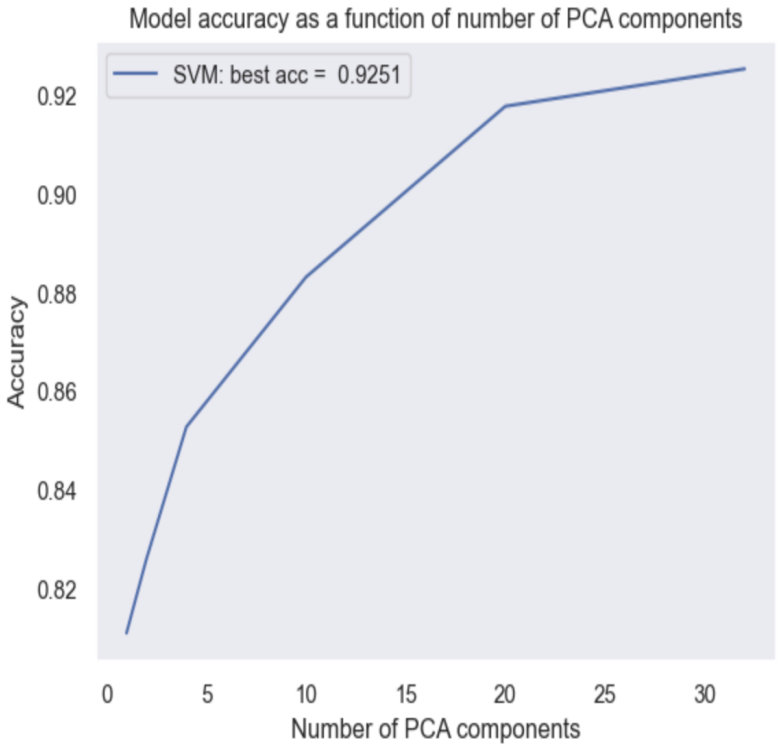
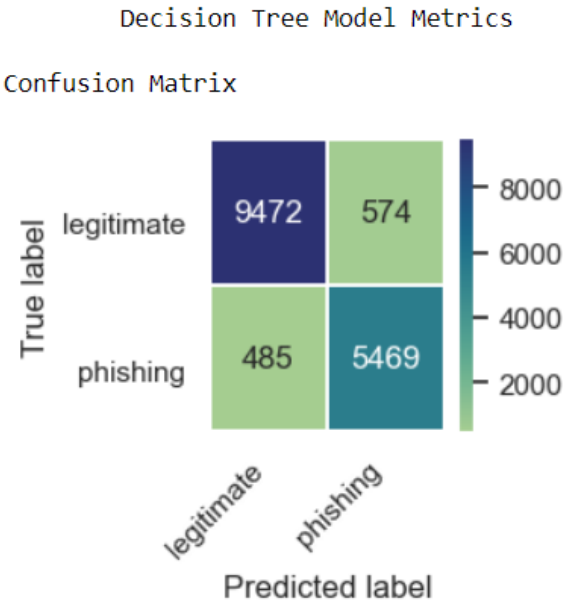


Figure 83

Confusion Matrix - Baseline Decision Tree



	precision	recall	f1-score	support
legitimate	0.9513	0.9429	0.9471	10046
phishing	0.9050	0.9185	0.9117	5954
accuracy			0.9338	16000
macro avg	0.9282	0.9307	0.9294	16000
weighted avg	0.9341	0.9338	0.9339	16000

Accuracy: 0.9338
Micro Precision: 0.9338
Micro Recall/TPR: 0.9338
Micro f1-score: 0.9338

Figure 84

ROC Curve for the Baseline Decision Tree

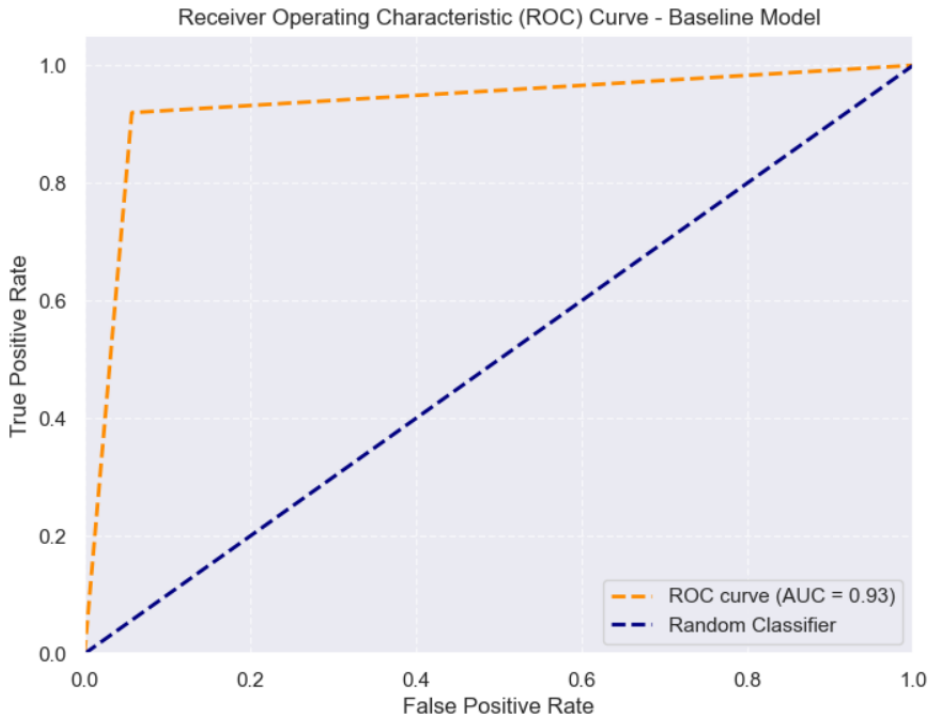
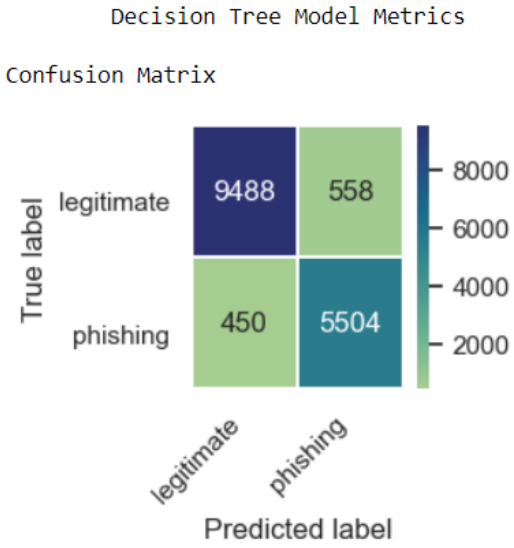


Figure 85

Confusion Matrix for Tuned Decision Tree



	precision	recall	f1-score	support
legitimate	0.9547	0.9445	0.9496	10046
phishing	0.9080	0.9244	0.9161	5954
accuracy			0.9370	16000
macro avg	0.9313	0.9344	0.9328	16000
weighted avg	0.9373	0.9370	0.9371	16000

Accuracy: 0.9370
Micro Precision: 0.9370
Micro Recall/TPR: 0.9370
Micro f1-score: 0.9370

Figure 86

ROC Curve for Tuned Decision Tree

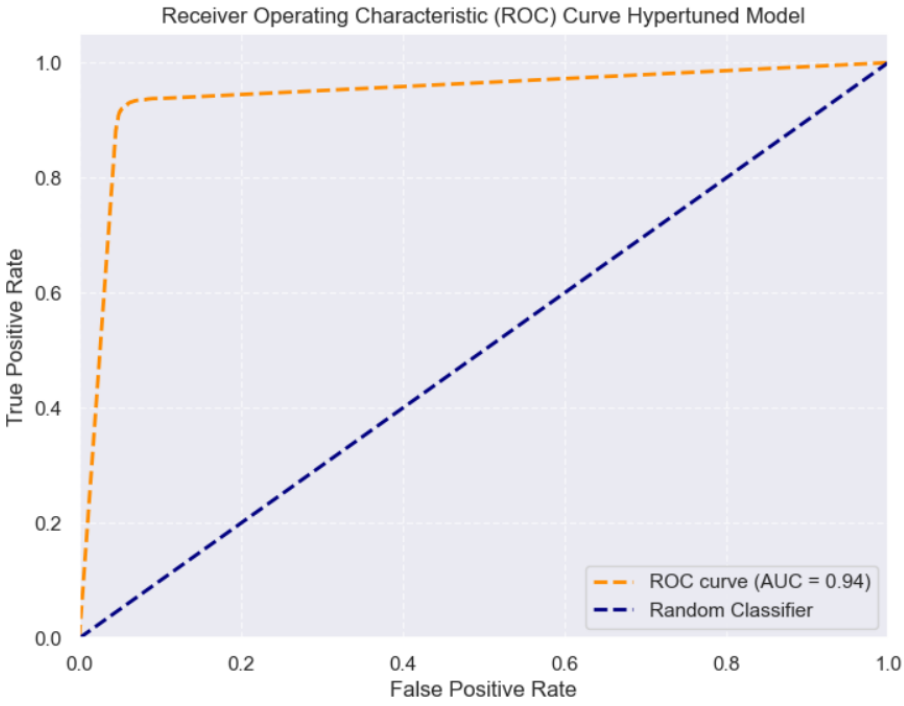
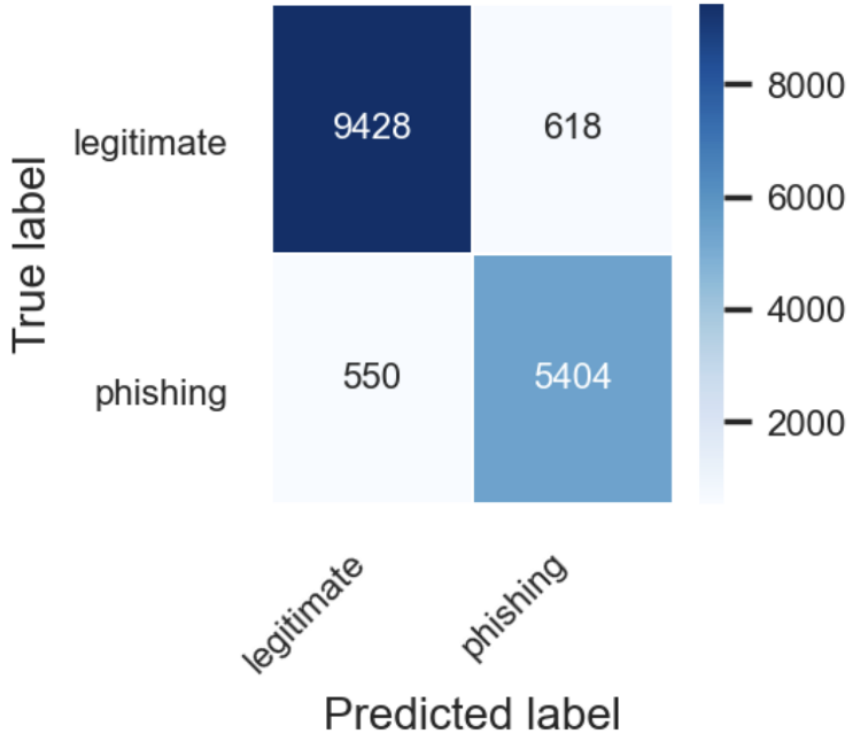


Figure 87

Confusion Metrics for Baseline Random Forest

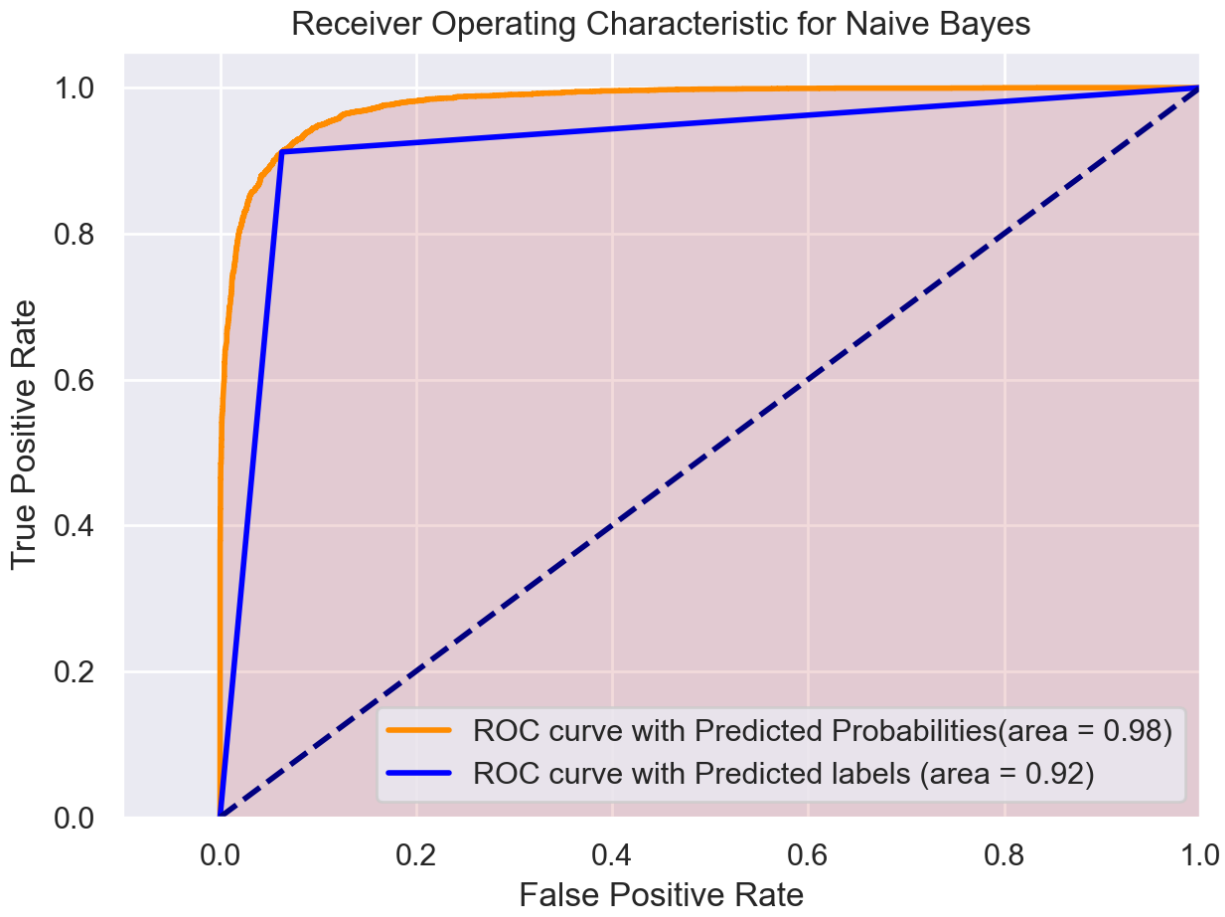


	precision	recall	f1-score	support
legitimate	0.9449	0.9385	0.9417	10046
phishing	0.8974	0.9076	0.9025	5954
accuracy			0.9270	16000
macro avg	0.9211	0.9231	0.9221	16000
weighted avg	0.9272	0.9270	0.9271	16000

Accuracy: 0.9270
Micro Precision: 0.9270
Micro Recall/TPR: 0.9270
Micro f1-score: 0.9270

Figure 88

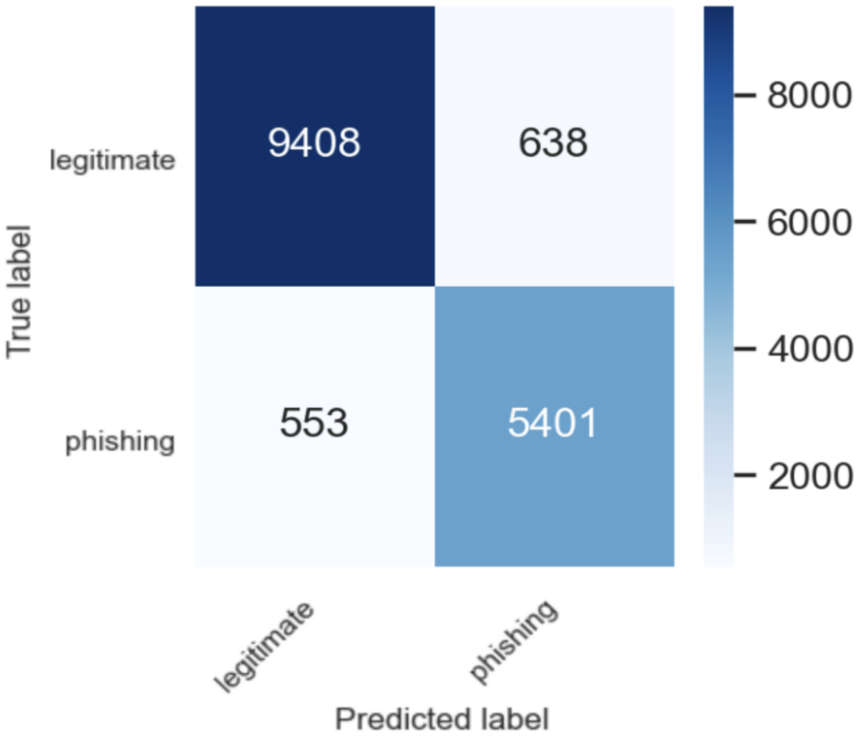
AUC under ROC for Random Forest Model baseline version.



Note. Show sharp turn at the left top corner resulting in higher AUC.

Figure 89

Confusion Metrics for Tuned Random Forest

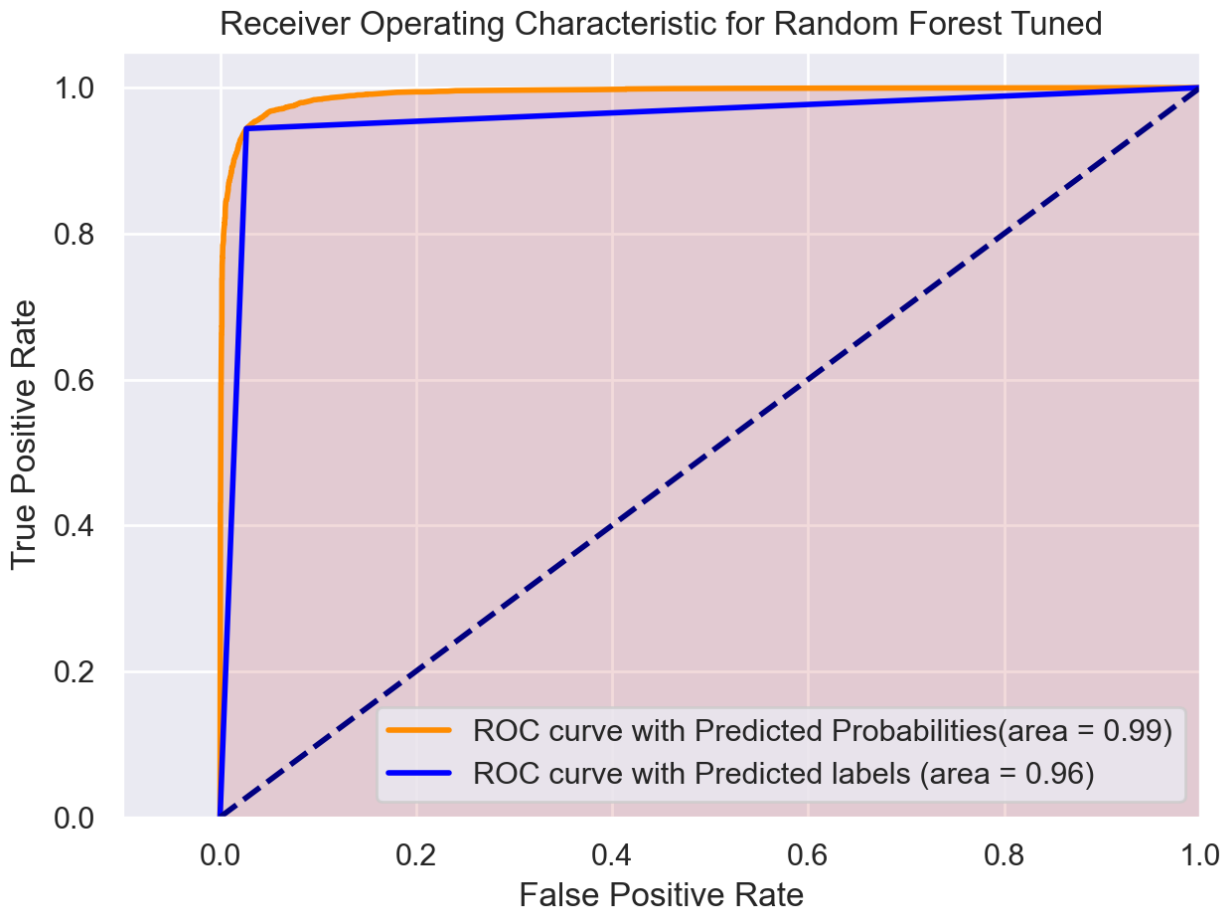


	precision	recall	f1-score	support
legitimate	0.9642	0.9726	0.9684	10000
phishing	0.9537	0.9398	0.9467	6000
accuracy			0.9603	16000
macro avg	0.9589	0.9562	0.9575	16000
weighted avg	0.9603	0.9603	0.9603	16000

Accuracy: 0.9603
Micro Precision: 0.9603
Micro Recall/TPR: 0.9603
Micro f1-score: 0.9603

Figure 90

AUC under ROC for Random Forest Model for tuned version.



Note. Show sharp turn at the left top corner resulting in higher AUC.

Figure 91

Precision vs Recall Curve for Random Forest Models (Baseline and Tuned)

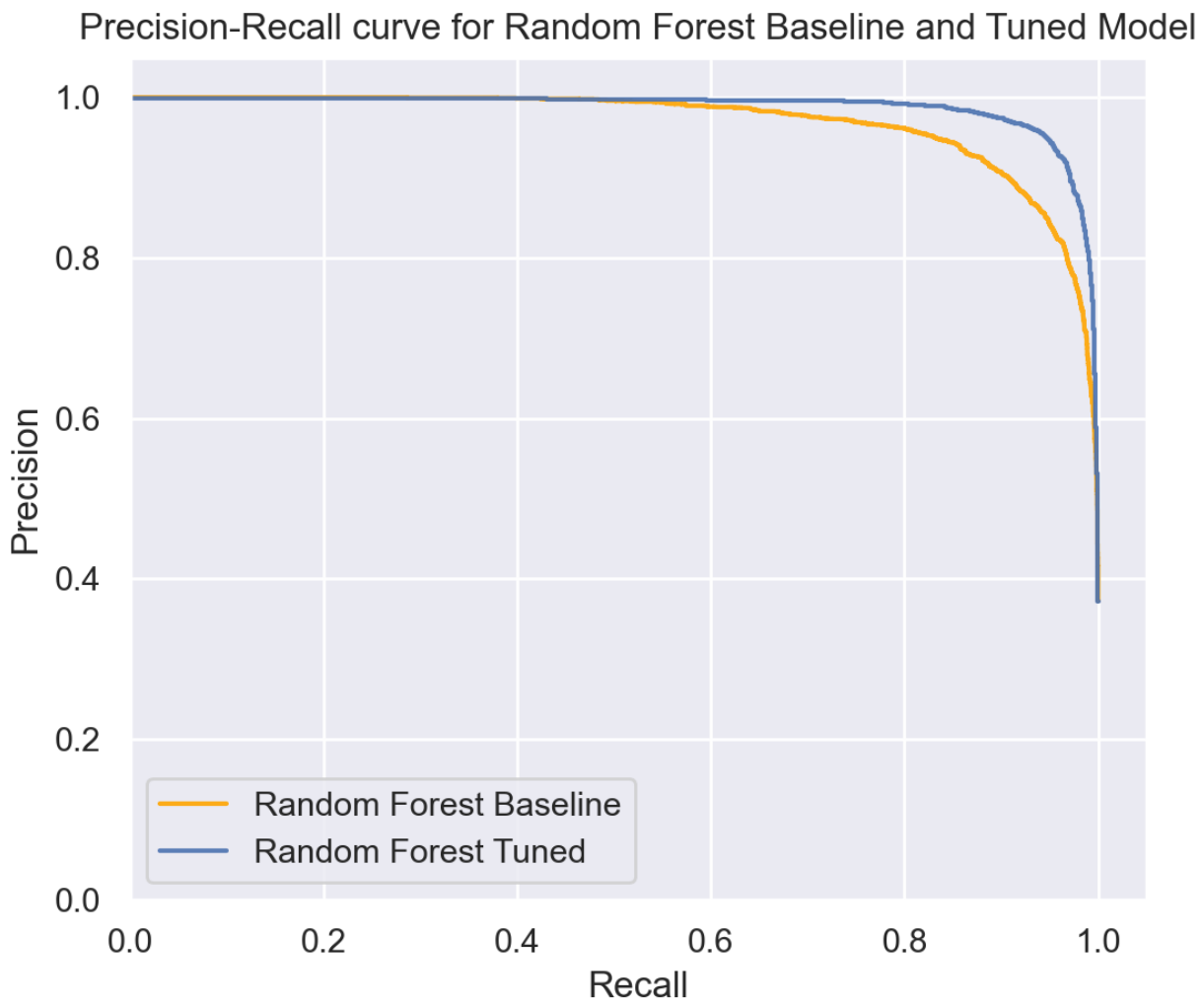
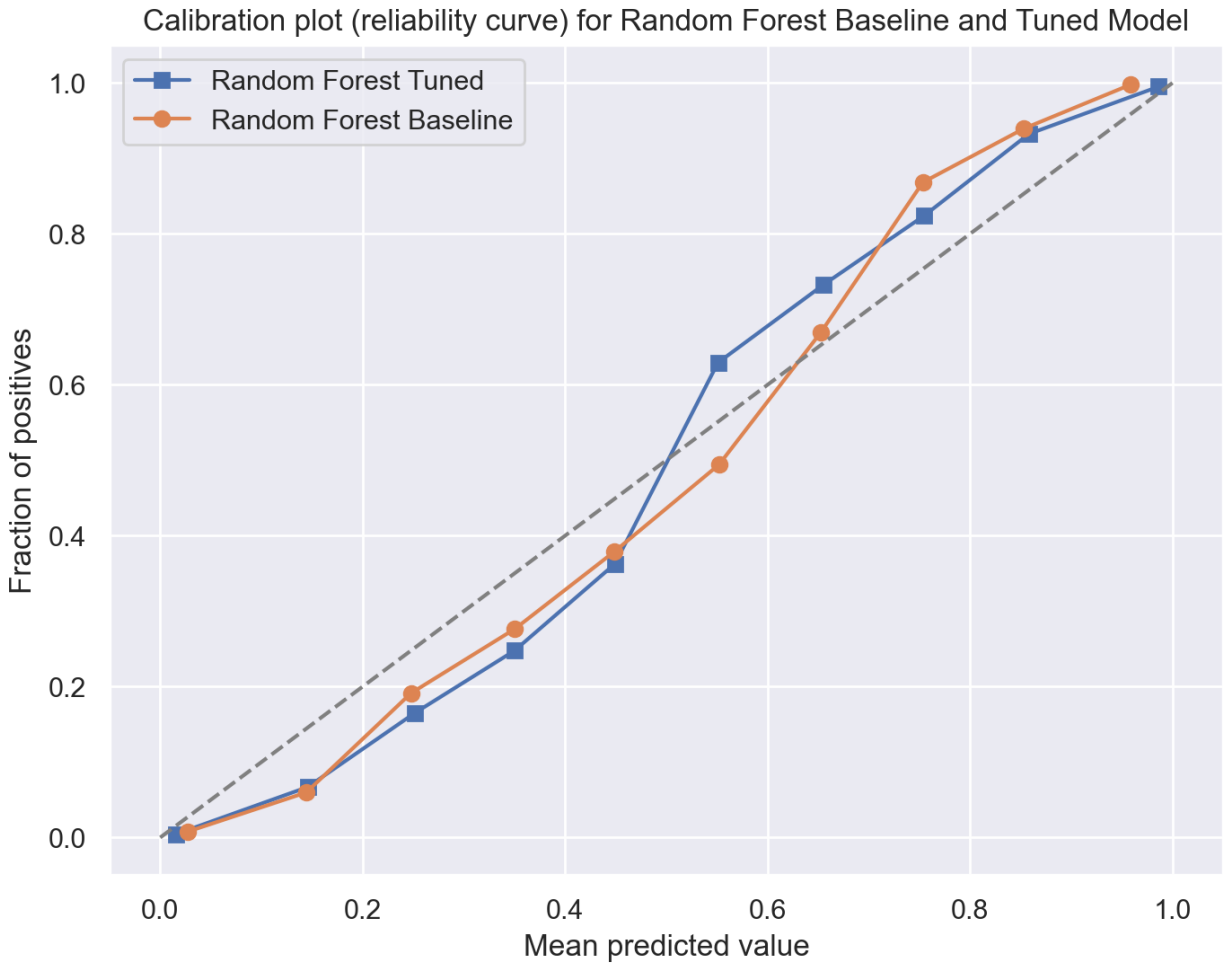


Figure 92

Calibration for Random Forest Models (Baseline and Tuned)



Appendix A

Code for Database Creation

Figure A1

Database Creation Code (Part 1)

```
import os
import sqlite3
import sqlparse

data_basepath = './n96ncsr5g4-1'
table_path = os.path.join(data_basepath, 'table.sql')
index_path = os.path.join(data_basepath, 'index.sql')
db_path = './phishing_index_db'

if os.path.exists(db_path) is True:
    os.remove(db_path)

conn = sqlite3.connect(db_path)
cursor = conn.cursor()
with open(table_path, 'r') as file:
    sql_script = file.read()
cursor.executescript(sql_script)

with open(index_path, 'r') as file:
    sql_script = file.read()

parsed_script = sqlparse.split(sql_script)
insert_statements = [stmt for stmt in parsed_script if 'INSERT' in stmt.upper()]

for i, insert_statement in enumerate(insert_statements):
    cursor.executescript(insert_statement)

conn.commit()
conn.close()
print(f'{db_path} created')
```

Appendix B

Code for URL Feature Extraction

Figure B1

URL Feature Extraction Code (Part 1)

```python
import os
import sqlite3
import re
from urllib.parse import urlparse
import requests
import idna
from datetime import datetime
import whois
import json
from tqdm import tqdm
from multiprocessing import Pool

def extract_whois_features(domain):
    feature = {}
    try:
        whois_info = whois.whois(domain)
        if whois_info.expiration_date is not None and whois_info.creation_date is not None:
            feature['is_domain_valid'] = True
            expiration_date = (
                whois_info.expiration_date[0] if isinstance(whois_info.expiration_date, list) else whois_info.expiration_date
            )
            creation_date = (
                whois_info.creation_date[0] if isinstance(whois_info.creation_date, list) else whois_info.creation_date
            )

            if isinstance(creation_date, str):
                if 'before' in creation_date:
                    # If the date is given as 'before Aug-1996', set it to the start of that month
                    date_components = creation_date.split('-')
                    year = int(date_components[1])
                    month = datetime.strptime(date_components[0].split()[1], '%b').month
                    creation_date = datetime(year, month, 1)
                else:
                    # If it's a specific date, parse it
                    creation_date = datetime.strptime(creation_date, '%Y-%m-%d')

            feature['days_until_expiration'] = (expiration_date - datetime.now()).days
            feature['registration_length'] = (expiration_date - creation_date).days
        else:
            feature['is_domain_valid'] = False
            feature['days_until_expiration'] = -1
            feature['registration_length'] = -1

    except Exception as e:
        feature['is_domain_valid'] = False
        feature['days_until_expiration'] = -1
        feature['registration_length'] = -1
    return feature
```

Figure B2

URL Feature Extraction Code (Part 2)

```
49
50 def is_punycode(domain):
51     try:
52         decoded_domain = idna.decode(domain)
53         return decoded_domain != domain
54     except idna.IDNAError:
55         # This exception occurs if the domain is not valid Punycode
56         return False
57
58 def compute_url_depth(parsed_url):
59     path_components = [component for component in parsed_url.path.split('/') if component is not None]
60     # Compute the depth (number of path components)
61     return len(path_components)
62
63 def has_phishing_keywords(url):
64     phishing_keywords = ['login', 'password', 'bank', 'secure', 'account']
65     return any(keyword in url.lower() for keyword in phishing_keywords)
66
67 def is_shortened_url(url):
68     # Source:
69     # https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/URL%20Feature%20Extraction.ipynb
70     shortening_services = r"bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\.ly|t\.co|tinyurl|tr\.im|is\.gd|cli\.gs|" \
71                           r"yfrog\.com|migre\.me|ff\.im|tiny\.cc|url4\.eu|twit\.ac|su\.pr|twurl\.nl|snipurl\.com|" \
72                           r"short\.to|BudURL\.com|ping\.fm|post\.ly|Just\.as|bkite\.com|snipr\.com|fic\.kr|loopt\.us|" \
73                           r"doiop\.com|short\.ie|kl\.am|wp\.me|rubyurl\.com|om\.ly|to\.ly|bit\.do|t\.co|lnkd\.in|db\.tt|" \
74                           r"qr\.ae|adf\.ly|goo\.gl|bitly\.com|cur\.lv|tinyurl\.com|ow\.ly|bit\.ly|ity\.im|q\.gs|is\.gd|" \
75                           r"po\.st|bc\.vc|twitthis\.com|u\.to|j\.mp|buzurl\.com|cutt\.us|u\.bb|yourls\.org|x\.co|" \
76                           r"prettylinkpro\.com|scrnch\.me|filoops\.info|vzturl\.com|qr\.net|1url\.com|tweez\.me|v\.gd|" \
77                           r"tr\.im|link\.zip\.net"
78
79     search = re.search(shortening_services, url)
80     return search is not None
81
82 def has_redirection(url):
83     try:
84         response = requests.head(url, allow_redirects=True, timeout=5)
85         final_url = response.url
86         if final_url != url:
87             return True
88         else:
89             return False
90     except Exception as e:
91         return False
92
```

Figure B3

URL Feature Extraction Code (Part 3)

```

def extract_url_feature(row):
    row_id, url, html, label, timestamp = row
    parsed_url = urlparse(url)

    feature = {}
    feature['label'] = label

    feature['url_length'] = len(url)
    feature['num_subdomains'] = len(parsed_url.netloc.split('.'))
    feature['uses_https'] = parsed_url.scheme == 'https'
    feature['contains_ip'] = bool(re.search(r'\d+\.\d+\.\d+\.\d+', parsed_url.netloc))
    feature['contains_phishing_keywords'] = has_phishing_keywords(url)
    feature['contains_at_symbol'] = '@' in url
    feature['url_depth'] = compute_url_depth(parsed_url)
    feature['is_shortened_url'] = is_shortened_url(url)
    feature['is_punycode'] = is_punycode(parsed_url.netloc)
    feature['has_redirection'] = has_redirection(url)

    feature.update(extract_whois_features(parsed_url.netloc))

    return row_id, feature

if __name__ == "__main__":
    data_basepath = './n96ncsr5g4-1'
    db_path = './phishing_index_db'
    url_features_path = './url_features.json'

    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM `index`;')
    rows = cursor.fetchall()

    with Pool(16) as p:
        outputs = list(tqdm(p.imap(extract_url_feature, rows), total=len(rows)))

    features = dict(outputs)

    with open(url_features_path, 'w') as json_file:
        json.dump(features, json_file, indent=2)
    print(f"Features have been written to {url_features_path}")

    conn.close()
```

Appendix C

Code for HTML Feature Extraction

Figure C1

HTML Feature Extraction Code (Part 1)

```python
import os
import sqlite3
import re
import json
from tqdm import tqdm
from multiprocessing import Pool
from bs4 import BeautifulSoup


def extract_html_feature(row):
    row_id, url, html, label, timestamp = row

    feature = {}
    feature['label'] = label

    with open(html, 'r', encoding='utf-8') as file:
        html_content = file.read()

    soup = BeautifulSoup(html_content, 'html.parser')

    forms = soup.find_all('form')
    feature['num_forms'] = len(forms)

    for form in forms:
        # Check for username and password fields
        username_fields = form.find_all('input', {'type': 'text', 'name': re.compile(r'user(?:name)?|login', re.IGNORECASE)})
        password_fields = form.find_all('input', {'type': 'password', 'name': re.compile(r'pass(?:word)?', re.IGNORECASE)})

        feature['num_username_fields'] = len(username_fields)
        feature['num_password_fields'] = len(password_fields)

        # Check for hidden fields
        hidden_fields = form.find_all('input', {'type': 'hidden'})
        feature['num_hidden_fields'] = len(hidden_fields)

        action_attribute = form.get('action', '')
        #  Convert form action to numerical feature
        if not action_attribute:
            form_action_numeric = 0  # No action
        elif action_attribute.startswith(('http://', 'https://')):
            form_action_numeric = 1  # Absolute URL
        elif action_attribute.startswith('/'):
            form_action_numeric = 2  # Relative URL
        else:
            form_action_numeric = 3  # Other (may include javascript, mailto, etc.)
        feature['form_action'] = form_action_numeric

        # Check for autocomplete attribute
        autocomplete_attribute = 'autocomplete' in form.attrs
        feature['form_autocomplete'] = 1 if autocomplete_attribute else 0

    if len(forms) == 0:
        feature['num_username_fields'] = 0
        feature['num_password_fields'] = 0
        feature['num_hidden_fields'] = 0
        feature['form_action'] = 0
        feature['form_autocomplete'] = 0


    external_links = soup.find_all('a', href=re.compile(r'^https?://'))
    feature['external_links_count'] = len(external_links)

    login_forms = soup.find_all('form', {'action': re.compile(r'login|signin|authenticate', re.IGNORECASE)})
    feature['login_form_present'] = True if login_forms else False

    javascript_redirects = soup.find_all('script', {'src': re.compile(r'window\.location')})
    feature['javascript_redirects_present'] = True if javascript_redirects else False

    iframes = soup.find_all('iframe')
    feature['iframes_count'] = len(iframes)

    scripts = soup.find_all('script')
    feature['num_obfuscated_scripts'] = sum(
        'eval(' in script.get_text() or 'document.write(' in script.get_text() for script in scripts
    )
    feature['external_js_inclusion'] = bool(soup.find('script', {'src': re.compile(r'^https?://')}))

    inline_styles = soup.find_all(style=True)
    feature['num_inline_styles'] = len(inline_styles)
    feature['num_script_tags'] = len(soup.find_all('script'))
    feature['num_iframe_tags'] = len(soup.find_all('iframe'))
    feature['num_img_tags'] = len(soup.find_all('img'))
    feature['num_a_tags'] = len(soup.find_all('a'))

    return row_id, feature
```

Figure C2

HTML Feature Extraction Code (Part 2)

```

def get_html_paths(directory):
    html_paths = {}
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith('.html'):
                assert file not in html_paths
                html_paths[file] = os.path.join(root, file)
    return html_paths

if __name__ == "__main__":
    data_basepath = './n96ncsr5g4-1'
    db_path = './phishing_index_db'
    html_features_path = './html_features.json'

    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM `index`;')
    rows = cursor.fetchall()

    html_paths = get_html_paths(data_basepath)

    for i in range(len(rows)):
        row_i = rows[i]
        row_id, url, html, label, timestamp = row_i
        full_html_path = html_paths[html]

        # Replace with the full path
        rows[i] = (row_id, url, full_html_path, label, timestamp)

    with Pool(16) as p:
        outputs = list(tqdm(p.imap(extract_html_feature, rows), total=len(rows)))

    features = dict(outputs)

    with open(html_features_path, 'w') as json_file:
        json.dump(features, json_file, indent=2)
    print(f"Features have been written to {html_features_path}")

    conn.close()
```

Appendix D

Data Analysis Code

Figure D1

Data Analysis Code (Part 1). This is the code that generates a bar chart.

```
def plot_grouped_items(dataFrame, columnName, limit=20, bucketSize=1,
                       title='', x_label='', y_label='Count',
                       labels=['Label 0', 'Label 1']):

    columnNameCat = f'{columnName}_category'
    tempDf = dataFrame[[columnName, 'label']]

    # Define the bins for the column with a bucketSize
    bins = [i for i in range(0, limit, bucketSize)] +  [float('inf')]

    # Create labels for each bin
    if bucketSize == 1:
        labels = [f'{str(i)}' for i in range(0, limit-bucketSize, bucketSize)] +  [f'{limit-bucketSize}+']
    else:
        labels = [f'{str(i)}-{str(i+bucketSize)}' for i in range(0, limit-bucketSize, bucketSize)] +  [f'{limit-bucketSize}+']

    # Create a new column 'num_img_tags_category' with the categorized values
    tempDf[columnNameCat] = pd.cut(tempDf[columnName], bins=bins, labels=labels, right=False)

    # Group by num_img_tags and label and count the occurrences
    grouped_data = tempDf.groupby([columnNameCat, 'label']).size().unstack(fill_value=0)


    # Plot stacked bar chart
    ax = grouped_data.plot(kind='bar', stacked=True, figsize=(12, 6), grid=True)

    # Add labels and legend
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.legend(title='Label', labels=['Not Phishing', 'Phishing website'])
    ax.set_title(title)

    if len(labels) > 22:
        ax.set_xticks(np.arange(0, 200, 10))

    ax.set_xticklabels(ax.get_xticklabels(), rotation=0)
    plt.figure(figsize=(10, 8))

    fig1 = plt.gcf()
    plt.show()
    plt.draw()
    fig1.savefig(f'plots/{columnName}_lables_distribution.png', dpi=300)

```

Figure D2

Data Analysis Code (Part 2)

```
plot_grouped_items(df, 'url_length',
                   limit = 200,
                   bucketSize = 1,
                   x_label = "Length of the URL on a web-page",
                   y_label = "Number of web-pages",
                   title='Number of external links on a web-page vs Phishing or Not Phishing websize distribution',
                   labels = [])


plot_grouped_items(df, 'external_links_count',
                   limit = 20,
                   bucketSize = 1,
                   x_label = "Number of external links on a web-page",
                   y_label = "Number of web-pages",
                   title='Number of external links on a web-page vs Phishing or Not Phishing websize distribution',
                   labels = [])

plot_grouped_items(df, 'num_img_tags',
                   x_label = "Number of Images on a web-page",
                   y_label = "Number of web-pages",
                   title='Number of Images on a web-page vs Phishing or Not Phishing websize distribution',
                   labels = [])

plot_grouped_items(df, 'iframes_count',
                   limit = 5,
                   x_label = "Number of iframes on a web-page",
                   y_label = "Number of web-pages",
                   title='Number of Iframes on a web-page vs Phishing or Not Phishing websize distribution',
                   labels = [])
```

Figure D3

Data Analysis Code (Part 3). This code generates a scatter plot between two features and adds a trend line.

```
def scatter_plot_with_trendline(df, x, y, is_log=False):

    plt.figure(figsize=(10, 8))

    # Separate the data into two classes
    class_0 = df[df['label'] == 0].sort_values(by=x)
    class_1 = df[df['label'] == 1].sort_values(by=x)


    # Plot points for class 0
    plt.scatter(class_0[x], class_0[y], color='blue', label='Non Phishing Web Pages')

    # Plot points for class 1
    plt.scatter(class_1[x], class_1[y], color='orange', label='Phishing Web Pages')

    # Fit a line to the data for class 0 and plot it
    z = np.polyfit(class_0[x], class_0[y], 1)
    p = np.poly1d(z)
    plt.plot(class_0[x], p(class_0[x]), "b--")

    # Fit a line to the data for class 1 and plot it
    z = np.polyfit(class_1[x], class_1[y], 1)
    p = np.poly1d(z)
    plt.plot(class_1[x], p(class_1[x]), "r--")

    if is_log:
        plt.xscale('log', basex=2)
        plt.yscale('log', basey=2)

    plt.xlabel(x)
    plt.ylabel(y)
    plt.title(f'Scatter Plot with trend line for {x} and {y}')
    plt.legend()
    plt.grid(True)
    fig1 = plt.gcf()
    plt.show()
    plt.draw()
    fig1.savefig(f'plots/scatter_plot_with_trendline_{x}_{y}.png', dpi=300, bbox_inches='tight')

scatter_plot_with_trendline(df, 'url_length', 'num_subdomains')

scatter_plot_with_trendline(df, 'num_username_fields', 'num_password_fields')
```

Figure D4

Data Analysis Code (Part 4). This code creates a spider chart between a given set of attributes.

```
import math

df = pd.DataFrame.from_dict(features, orient='index')


categories = ["contains_ip", "contains_phishing_keywords",
              "has_redirection", "is_domain_valid",  "login_form_present",
             "uses_https", "javascript_redirects_present", "form_autocomplete"]


values_dataset1, values_dataset2 = [], []
for c in categories:
    values_dataset1.append(math.log(df[(df['label'] == 1) & (df[c] == True) ].shape[0]))
    values_dataset2.append(math.log(df[(df['label'] == 0) & (df[c] == True) ].shape[0]))

# Number of categories
num_categories = len(categories)

# Calculate angle for each category
angles = np.linspace(0, 2 * np.pi, num_categories, endpoint=False).tolist()

# Repeat the first value to close the circle
values_dataset1 += values_dataset1[:1]
values_dataset2 += values_dataset2[:1]
angles += angles[:1]

plt.figure(figsize=(8, 8))

# Create a spider chart for Dataset 2
plt.polar(angles, values_dataset2, marker='o', linestyle='solid', linewidth=2, label='Non Phishing web pages')
plt.fill(angles, values_dataset2, alpha=0.25)

# Create a spider chart for Dataset 1
plt.polar(angles, values_dataset1, marker='o', linestyle='solid', linewidth=2, label='Phishing web pages')
plt.fill(angles, values_dataset1, alpha=0.25)


# Set labels for each category
plt.xticks(angles[:-1], categories)

# Set the title of the plot
plt.title('Spider Chart For Web-page attributes', size=20, color='blue', y=1.1)

# Display legend
plt.legend(loc='upper left')
plt.savefig('plots/spider_chart_log.png', dpi=300, bbox_inches='tight')

# Show the plot
plt.show()
```

Figure D5

Data Analysis Code (Part 5). This code creates the box plot for features.

```
from sklearn.preprocessing import StandardScaler

cont_feats = {
    "url_length",
    "url_depth",
    "days_until_expiration",
    "registration_length",
    "num_forms",
    "num_username_fields",
    "num_password_fields",
    "num_hidden_fields",
    "form_autocomplete",
    "external_links_count",
    "iframes_count",
    "num_obfuscated_scripts",
    "num_inline_styles",
    "num_script_tags",
    "num_iframe_tags",
    "num_img_tags",
    "num_a_tags",
  }

# Set the resolution for inline plotting
%config InlineBackend.figure_format = 'retina'
plt.figure(figsize=(14, 8))

df = df[list(cont_feats)].sample(64000)

# Normalize the DataFrame using StandardScaler
scaler = StandardScaler()
normalized_df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

# Create a box plot for the normalized DataFrame
plt.figure(figsize=(12, 12))
# plt.yscale('log')

filtered_df = normalized_df[normalized_df < 20]

ax = filtered_df.boxplot()
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.title('Box Plot of Normalized DataFrame')
plt.ylabel('Normalize Feature Values')
plt.show()

fig1 = plt.gcf()
plt.show()
plt.draw()
fig1.savefig(f'plots/standardize_feature_box_plot.png', dpi=300, bbox_inches='tight')
```

Figure D6

Data Analysis Code (Part 6). This code creates the form action pie chart.

```
import pandas as pd
import seaborn as sns

df = pd.DataFrame.from_dict(features, orient='index')
df = df[["form_action", "label"]]

import pandas as pd
import matplotlib.pyplot as plt

# Count the occurrences of each label and form_action combination
df['form_action'] = df['form_action'].replace(
    {0: 'No Action', 1: 'Absolute URL', 2: 'Relative URL', 3: 'Other'})


label_counts = df.groupby(['label', 'form_action']).size().unstack(fill_value=0)

# Create two pie charts, one for each label
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Pie chart for Label 0
axes[0].pie(label_counts.loc[0], labels=label_counts.columns, autopct='%1.1f%%', startangle=90)
axes[0].set_title('Not Phishing Web-Pages Form Action Distribution')

# Pie chart for Label 1
axes[1].pie(label_counts.loc[1], labels=label_counts.columns, autopct='%1.1f%%', startangle=90)
axes[1].set_title('Phishing Web-Pages Form Action Distribution')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
plt.savefig('plots/form_action_dist.png', dpi=300, bbox_inches='tight')
```

Figure D7

Data Analysis Code (Part 7). This code creates the violin chart for 2 attributes and for 2 output classes

```
def create_violin_cahrt(df, x, y):
    df = df[[x, y, "label"]]

    plt.figure(figsize=(10,6))
    plt.grid(True)

    # Draw a nested violinplot and split the violins for easier comparison
    sns.violinplot(data=df, x=x, y=y, hue="label", split=True)
    plt.savefig(f'plots/violin_plot_{x}_{y}.png', dpi=300, bbox_inches='tight')


df = load_features()
create_violin_cahrt(df, "num_subdomains", "url_length")
df['form_action'] = df['form_action'].replace({0: 'No Action', 1: 'Absolute URL', 2: 'Relative URL', 3: 'Other'})
create_violin_cahrt(df, "form_action", "url_length")
```

Figure D8

Data Analysis Code (Part 8)

```
def plot_pca(X_pca, y, n_components):
    X_list = [X_pca[:, i] for i in range(n_components)]

    if len(X_list) == 3:
        ax = plt.axes(projection='3d')
    else:
        ax = plt.axes()
    ax.scatter(*X_list, c=y, cmap='RdYlBu')
    ax.set_title(f'Distribution of data using PCA with {n_components = }')
    plt.show()
    plt.close()
```

Executed at 2023.11.19 23:40:40 in 2ms

```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X_train_norm)
X_train_pca = pca.transform(X_train_norm)
X_test_pca = pca.transform(X_test_norm)

plot_pca(X_train_pca, y_train, n_components=2)
```

Executed at 2023.11.19 23:40:42 in 1s 314ms

Figure D9

Data Analysis Code (Part 9)

```
plt.figure(figsize=(12, 12))
df = load_features()
heatmap = sns.heatmap(df.corr(), cmap='BrBG')
plt.savefig('plots/correlation.png', dpi=300, bbox_inches='tight')
heatmap.set_title('Correlation Heatmap');
```

```
df = load_features()
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(df.corr()[['label']].sort_values(by='label', ascending=False), vmin=-1, vmax=1, annot=True, cmap='BrBG')
heatmap.set_title('Features Importance against the Output Class', fontdict={'fontsize':18}, pad=16);
plt.savefig('plots/feature_importance.png', dpi=300, bbox_inches='tight')

```

Appendix E

Model Code

Figure E1

Data Loading Model Training

In [2]:
```
data_basepath = './n96ncsr5g4-1'
url_features_path = './url_features.json'
html_features_path = './html_features.json'
```

In [3]:
```
with open(url_features_path) as f:
    url_features = json.load(f)

with open(html_features_path) as f:
    html_features = json.load(f)
```

In [4]:
```
features = {}
for k in url_features.keys():
    url_feat = url_features[k]
    html_feat = html_features[k]

    url_feat.update(html_feat)
    features[k] = url_feat
```

In [5]:
```
df = pd.DataFrame.from_dict(features, orient='index')
df.head()
```

Figure E2

Data Split For Model Training

In [9]:
```
from sklearn.model_selection import train_test_split

X = df.drop('label', axis=1).to_numpy().astype(float)
y = df['label'].to_numpy()

test_size = 0.2
seed = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=test_size, random_state=seed)
print(f'Number of training samples: {X_train.shape}')
print(f'Number of test samples: {X_test.shape}')
```

```
Number of training samples: (64000, 32)
Number of test samples: (16000, 32)
```

Figure E3

Data Normalization for Model Training

In [13]:

```
cont_feats = {
    "url_length",
    "url_depth",
    "days_until_expiration",
    "registration_length",
    "num_forms",
    "num_username_fields",
    "num_password_fields",
    "num_hidden_fields",
    "form_autocomplete",
    "external_links_count",
    "iframes_count",
    "num_obfuscated_scripts",
    "num_inline_styles",
    "num_script_tags",
    "num_iframe_tags",
    "num_img_tags",
    "num_a_tags",
  }
columns = df.columns[1:]
cont_ids = [i for i, c in enumerate(columns) if c in cont_feats]


mean = X_train[:, cont_ids].mean(axis=0, keepdims=True)
std = X_train[:, cont_ids].std(axis=0, keepdims=True)
X_train_norm = X_train.copy()
X_test_norm = X_test.copy()
X_train_norm[:, cont_ids] = (X_train[:, cont_ids] - mean) / std
X_test_norm[:, cont_ids] = (X_test[:, cont_ids] - mean) / std
```

Figure E4

Logistic Regression Model Training (Part 1)

In [19]:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.model_selection import GridSearchCV

parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3, 3, 7),
    'solver'  : ['saga','lbfgs'],
}

grid_search = GridSearchCV(
    LogisticRegression(max_iter=1000), parameters, scoring=make_scorer(accuracy_score), cv=5, n_jobs=2, return_tr
)
grid_search.fit(X_train_norm, y_train)
print(f'Best training accuracy with 5-fold cross validation: {grid_search.best_score_}')
print(f'Best parameters: \n{grid_search.best_params_}')
```

Figure E5

Logistic Regression Model Training (Part 2)

```
Best training accuracy with 5-fold cross validation: 0.8823593750000001
Best parameters:
{'C': 1000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
```

Fit with best parameters

[20]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000, penalty='l2', C=1000, solver='lbfgs')
log_reg.fit(X_train_norm, y_train)
```

[20]:
```
LogisticRegression(C=1000, max_iter=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[21]:
```
print("        Logistic Regression Model Metrics       \n")
compute_model_metrics(log_reg, X_test_norm, y_test)
```

Figure E6

SVM Model Training

In [ ]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.model_selection import GridSearchCV
import numpy as np

# Grid search.
parameters = [
        {"kernel": ["linear"], "C": [10000]},
        {"kernel": ["poly"], "degree": [2, 3, 4], "C": np.logspace(-1, 3, 4)},
        {"kernel": ["rbf"], "gamma": [0.1, 0.01, 0.001], "C": np.logspace(-1, 3, 4),}
]


grid_search = GridSearchCV(
    SVC(), parameters, scoring=make_scorer(accuracy_score), cv=5, n_jobs=16, return_train_score=True
)
grid_search.fit(X_train_norm, y_train)
print(f'Best training accuracy with 5-fold cross validation: {grid_search.best_score_}')
print(f'Best parameters: \n{grid_search.best_params_}')
```

Fit with the best parameters

In [19]:

```
from sklearn.svm import SVC

svm_clf = SVC(C=1000, gamma=0.01, kernel='rbf')
svm_clf.fit(X_train_norm, y_train)
```

Out[19]:

```
SVC(C=1000, gamma=0.01)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:

```
print("        SVM Model Metrics       \n")
compute_model_metrics(svm_clf, X_test_norm, y_test)
```

Figure E7

Decision Tree Model Training

In [ ]:

```
# Define the parameter grid for GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.model_selection import GridSearchCV

dt_parameters = {
    "criterion": ["gini", "entropy"],
    "max_depth": [None, 10, 20, 30, 40, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "ccp_alpha": [0.0, 0.001, 0.01, 0.1, 1.0]
}

dt_simple = DecisionTreeClassifier(random_state=123)

# Create a scorer for GridSearchCV
scorer = make_scorer(accuracy_score)

# Perform grid search with cross-validation on the subset
dt_grid_search = GridSearchCV(
    dt_simple,
    dt_parameters,
    scoring=scorer,
    cv=5,
    n_jobs=5,
    return_train_score=True,
)

dt_grid_search.fit(X_train_norm, y_train)

# Display the best parameters and corresponding accuracy on the subset
print(f'Best parameters : {dt_grid_search.best_params_}')
print(f'Best training accuracy with cross-validation : {dt_grid_search.best_score_}')
```

Fit with the best model

Pruned Decision Tree - Cost-complexity pruning

In [21]:

```
import time
from sklearn.tree import DecisionTreeClassifier

start_time_pruned = time.time()

# Creating a new decision tree with the best parameters, including ccp_alpha for pruning
dt_pruned = DecisionTreeClassifier(
    random_state=123,
    ccp_alpha=0.0,
    criterion='entropy',
    max_depth= 20,
    min_samples_leaf=1,
    min_samples_split= 2
)

# Fitting the pruned decision tree on the full dataset
dt_pruned.fit(X_train_norm, y_train)

end_time_pruned = time.time()
time_taken_pruned = end_time_pruned - start_time_pruned
```

In [22]:

```
print("        Decision Tree Model Metrics       \n")
compute_model_metrics(dt_pruned, X_test_norm, y_test)
```

Figure E8

Baseline Random Forest Algorithm

```
from sklearn.metrics import classification_report

def train_random_forest(X_train, y_train):
    # Create a base model
    rf = RandomForestClassifier()

    grid_search = GridSearchCV(estimator=rf,
                               scoring={'accuracy': make_scorer(accuracy_score),
                                        'precision': make_scorer(precision_score, average='macro'),
                                        'recall': make_scorer(recall_score, average='macro'),
                                        'f1': make_scorer(f1_score, average='macro')},
                               cv = 2, n_jobs = -1, verbose = 2, refit='accuracy')

    # Fit the grid search to the data
    grid_search.fit(X_train, y_train)
    return grid_search.best_estimator_

model_b = train_random_forest(X_train, y_train)
print(model_b)

y_score_b = model_b.predict_proba(X_test)
y_pred_b = model_b.predict(X_test)
compute_model_metrics(model_b, y_test, y_pred_b)

calculate_metrics(y_test, y_pred_b)

plot_confusion_matrix(y_test, y_pred_b)
print(classification_report(y_test, y_pred_b, labels=[1, 2, 3]))
plt.figure(figsize=(5,5))

plot_roc_curve(y_test,  y_score_b[0::, -1], y_pred_b, "Naive Bayes")
plot_precision_recall_curve(y_test, y_score_b[0::, -1])

calibration_plot(y_test, y_score_b[0::, -1])

```

Figure E9

Tuned Random Forest Algorithm with Grid Search

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score

def train_random_forest(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'criterion': ['gini', 'entropy'],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10, 20],
        'min_samples_leaf': [1, 2, 4, 10],
        'bootstrap': [True, False]
    }

    # Create a base model
    rf = RandomForestClassifier()

    # Manually create the splits in CV in order to be able to fix a random_state (GridSearchCV doesn't have that argument)
    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                               scoring={'accuracy': make_scorer(accuracy_score),
                                        'precision': make_scorer(precision_score, average='macro'),
                                        'recall': make_scorer(recall_score, average='macro'),
                                        'f1': make_scorer(f1_score, average='macro')},
                               cv = 2, n_jobs = -1, verbose = 2, refit='accuracy')

    # Fit the grid search to the data
    grid_search.fit(X_train, y_train)
    return grid_search

grid_search = train_random_forest(X_train, y_train)

model = grid_search.best_estimator_
print(model)

y_score = model.predict_proba(X_test)
y_pred = model.predict(X_test)

calculate_metrics(y_test, y_pred)

plot_confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))

plot_roc_curve(y_test,  y_score[0::, -1], y_pred, "Random Forest Tuned")

# Call the function to plot Precision-Recall curve
plot_precision_recall_curve(y_test, y_score[0::, -1])

calibration_plot(y_test, y_score[0::, -1])
```

Figure E10

PCA For all the models

In [19]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

models = {
    'SVM': SVC(C=1000, gamma=0.01, kernel='rbf'),
    'logistic_regression': LogisticRegression(
        penalty="l2",
        C=1e4,
        max_iter=1000
    ),
    'decision_tree': DecisionTreeClassifier(
        random_state=123,
        ccp_alpha=0.0,
        criterion='entropy',
        max_depth= 20,
        min_samples_leaf=1,
        min_samples_split= 2
    ),
    'random_forest': RandomForestClassifier(
        max_depth=None,
        n_estimators=200,
        max_features=3,
        min_samples_leaf=1,
        min_samples_split=3,
    )
}
```

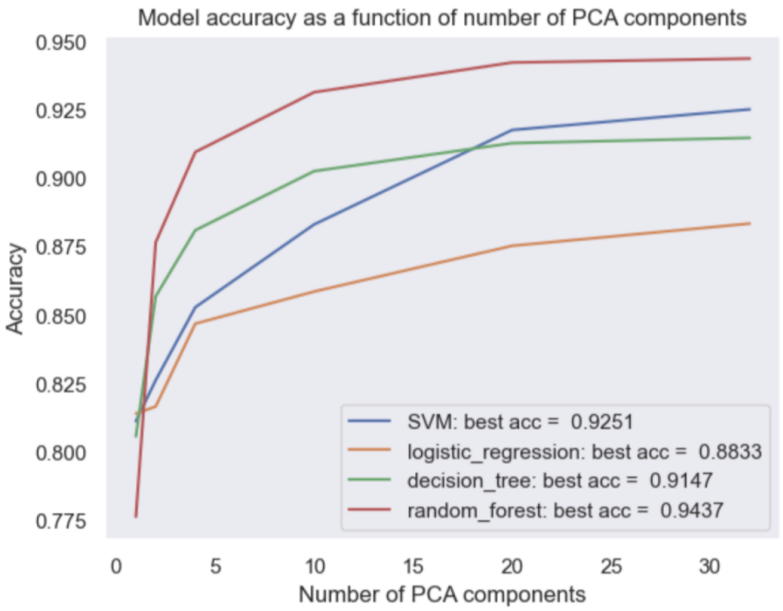
In [23]:
```
from sklearn.preprocessing import StandardScaler

def test_with_pca_components(model, X_train_norm, y_train, X_test_norm, y_test, components_list):
    pca = PCA()
    pca.fit(X_train_norm)
    X_train_norm_pca = pca.transform(X_train_norm)
    X_test_norm_pca = pca.transform(X_test_norm)

    reports = []
    for n in components_list:
        model.fit(X_train_norm_pca[:, :n], y_train)
        y_test_pred = model.predict(X_test_norm_pca[:, :n])
        report = metrics.classification_report(
            y_test, y_test_pred, target_names=['legitimate', 'phishing'], digits=4, output_dict=True
        )
        reports.append(report)
        print(f'Testing for {n = }, acc: {report["accuracy"]}')

    return reports
```

In [24]:
```
components = [1, 2, 4, 10, 20, X_train_norm.shape[1]]

reports = {}
for name, model in models.items():
    print(f'Running {name}')
    reports[name] = test_with_pca_components(model, X_train_norm, y_train, X_test_norm, y_test, components)
```

Figure E11

PCA vs Model Accuracy

```
plt.figure()
for name, rep in reports.items():
    acc = [r['accuracy'] for r in rep]
    max_acc = max(acc)
    plt.plot(components, acc, label=f'{name}: best acc = {max_acc: .4f}')

plt.xlabel('Number of PCA components')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model accuracy as a function of number of PCA components')
plt.show()
```




```
for metric_name in ['precision', 'recall', 'f1-score']:
    plt.figure()
    for name, rep in reports.items():
        metric = [r['phishing'][metric_name] for r in rep]
        best_metric = max(metric)
        plt.plot(components, metric, label=f'{name}: best {metric_name} = {best_metric: .4f}')

    plt.xlabel('Number of PCA components')
    plt.ylabel(metric_name)
    plt.legend()
    plt.title(f'Model {metric_name} as a function of number of PCA components')
    plt.show()
```

Figure E12

SMOTE

In [16]:
```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train_norm, y_train)
```

Figure E13

Model Evaluation Metrics Statistics

:
```
from sklearn import metrics

def compute_model_metrics(model, X, y_true):
    y_pred = model.predict(X)

    print("Confusion Matrix")
    confusion_matrix = metrics.confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(4, 4))
    ax = sns.heatmap(confusion_matrix, cmap="Blues",  annot=True, fmt="d", linewidths=.1)
    xticks = [i + 0.5 for i in range(2)]
    ax.set_xticks(
        xticks, ['legitimate', 'phishing'], rotation=45, ha='right'
    )
    ax.set_yticks(
        xticks, ['legitimate', 'phishing'], rotation=0, va='top'
    )
    ax.set_ylabel("True label")
    ax.set_xlabel("Predicted label")
    plt.show()

    report = metrics.classification_report(
        y_true, y_pred, target_names=['legitimate', 'phishing'], digits=4
    )
    print(report)
    test_acc = metrics.accuracy_score(y_true, y_pred)
    print(f'Accuracy: {test_acc:.4f}')

    test_precision = metrics.precision_score(y_true, y_pred, average='micro')
    print(f'Micro Precision: {test_precision:.4f}')

    test_recall = metrics.recall_score(y_true, y_pred, average='micro')
    print(f'Micro Recall/TPR: {test_recall:.4f}')

    test_f1 = metrics.f1_score(y_true, y_pred, average='micro')
    print(f'Micro f1-score: {test_f1:.4f}')
```

Figure E14

Model Evaluation Metrics (Part 1)

```
import seaborn as sns
import scikitplot as skplt
from sklearn.metrics import auc
from sklearn.metrics import (roc_auc_score,
                             confusion_matrix, roc_curve, precision_recall_curve)


from sklearn.calibration import calibration_curve
import matplotlib.pyplot as plt

def plot_feature_importance(model, feature_names):
    importances = model.feature_importances_
    indices = np.argsort(importances)
    plt.figure(figsize=(5,7))
    plt.title('Feature Importances')
    plt.barh(range(len(indices)), importances[indices], color='b', align='center')
    plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
    plt.xlabel('Relative Importance')
    plt.grid()
    fig1 = plt.gcf()
    plt.show()
    plt.draw()
    fig1.savefig('feature_importance.png', dpi=300)

def plot_cumulative_gain_chart(X, y):
    plt.figure(figsize=(4,5))
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.3, random_state=42)
    clf = RandomForestClassifier()
    clf.fit(X_train, y_train)
    y_probas = clf.predict_proba(X_test)
    skplt.metrics.plot_cumulative_gain(y_test, y_probas)
    plt.show()


def plot_tree_diagram(model):
    # show only a small portion of random forest tree
    from sklearn.tree import plot_tree
    plt.figure(figsize=(20,12))
    # increase the boxes size
    plt.rcParams['figure.figsize'] = [50, 30]
    # increase the font size
    # plt.rcParams.update({'font.size': 20})
    plot_tree(rf_best.estimators_[1], feature_names = list(df.iloc[:, 1:].columns),class_names=['Disease', "No Disease"],filled=True, max_depth=3);
    plt.savefig('sample_tree.png', dpi=300, bbox_inches='tight')

#def plot_feature_importance(model, feature_names):
def get_raw_scores_for_output_class(X_test, model):
    predicted_classes = model.predict(X_test)
    probabilities = model.predict_proba(X_test)
    output_class_scores = [probabilities[i][predicted_class] for i, predicted_class in enumerate(predicted_classes)]
    return output_class_scores

```

Figure E15

Model Evaluation Metrics (Part 2)

```python
def plot_Lift_curve(y_val, y_pred, step=0.1):
    # https://howtolearnmachinelearning.com/articles/the-lift-curve-in-machine-learning/

    #Define an auxiliar dataframe to plot the curve
    aux_lift = pd.DataFrame()
    #Create a real and predicted column for our new DataFrame and assign values
    aux_lift['real'] = y_val
    aux_lift['predicted'] = y_pred
    #Order the values for the predicted probability column:
    aux_lift.sort_values('predicted',ascending=False,inplace=True)

    #Create the values that will go into the X axis of our plot
    x_val = np.arange(step,1+step,step)
    #Calculate the ratio of ones in our data
    ratio_ones = aux_lift['real'].sum() / len(aux_lift)
    print(ratio_ones)
    #Create an empty vector with the values that will go on the Y axis our our plot
    y_v = []

    #Calculate for each x value its correspondent y value
    for x in x_val:
        num_data = int(np.ceil(x*len(aux_lift))) #The ceil function returns the closest integer bigger than our number
        data_here = aux_lift.iloc[:num_data,:]   # ie. np.ceil(1.4) = 2
        ratio_ones_here = data_here['real'].sum()/len(data_here)
        y_v.append(ratio_ones_here / ratio_ones)

   #Plot the figure
    fig, axis = plt.subplots()
    fig.figsize = (4,4)
    axis.plot(x_val, y_v, 'g-', linewidth = 3, markersize = 5)
    axis.plot(x_val, np.ones(len(x_val)), 'k-')
    axis.set_xlabel('Proportion of sample')
    axis.set_ylabel('Lift')
    plt.grid()
    plt.title('Lift Curve')
    plt.show()



def calculate_metrics(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred)
    # compute the specificity
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    specificity = tn / (tn+fp)

    # compute the Mathew's correlation coefficient
    mcc = (tp*tn - fp*fn) / np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))


    print(f'Accuracy: {accuracy}')
    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
    print(f'F1 Score: {f1}')
    print(f'ROC AUC: {auc}')
    print(f'Specificity: {specificity}')
    print(f'MCC: {mcc}')
```

Figure E16

Model Evaluation Metrics (Part 3)

```
def plot_confusion_matrix(y_true, y_pred):
    # dont round off the values in the heat map plot, show exact value

    plt.figure(figsize=(3,3))
    cf_matrix = confusion_matrix(y_true, y_pred)
    sns.heatmap(cf_matrix, annot=True, cmap='Blues', fmt='.0f')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()



def plot_precision_recall_curve(y_true, y_scores):
    plt.figure(figsize=(6,5))
    precision, recall, _ = precision_recall_curve(y_true, y_scores)
    plt.step(recall, precision, color='b', alpha=0.9, where='post')
    plt.grid()
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.05])
    plt.title('Precision-Recall curve')
    plt.show()


def calibration_plot(y_test, y_scores):
    # # Predict probabilities
    # probs = clf.predict_proba(X_test)[:, 1]

    # Compute the calibration curve
    fraction_of_positives, mean_predicted_value = calibration_curve(y_test, y_scores, n_bins=10)

    # Plot the calibration curve
    plt.figure(figsize=(8, 6))
    plt.plot(mean_predicted_value, fraction_of_positives, 's-')
    plt.plot([0, 1], [0, 1], '--', color='gray')
    plt.grid()
    plt.ylabel('Fraction of positives')
    plt.xlabel('Mean predicted value')
    plt.title('Calibration plot (reliability curve)')
    plt.show()


# Call the function to plot ROC curve
def plot_roc_curve(y_true, y_scores, y_pred, model_name = "Random Forest"):
    plt.figure(figsize=(7,5))
    fpr, tpr, thresholds = roc_curve(y_true, y_scores)

    fpr2, tpr2, thresholds2 = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr2, tpr2)
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve with Predicted Probabilities(area = %0.2f)' % auc(fpr, tpr))
    plt.plot(fpr2, tpr2, color='blue', lw=2, label='ROC curve with Predicted labels (area = %0.2f)' % auc(fpr2, tpr2))
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.fill_between(fpr, tpr, step='post', alpha=0.2, color='r')

    plt.xlim([-0.1, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic for '+ model_name)
    plt.grid()
    plt.legend(loc="lower right")
    fig1 = plt.gcf()
    plt.show()
    plt.draw()
    fig1.savefig('roc_auc.png', dpi=200)
```

Figure E17

Model Evaluation Metrics (Part 4)

```
def compute_model_metrics(model, y_true, y_pred):
    SMALL_SIZE = 8
    MEDIUM_SIZE = 10
    BIGGER_SIZE = 12
    plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
    plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
    plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
    plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
    plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize


    from sklearn import metrics
    print("Confusion Matrix")
    confusion_matrix = metrics.confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(2, 2))
    ax = sns.heatmap(confusion_matrix, cmap="Blues",  annot=True, fmt="d", linewidths=.1)
    xticks = [i + 0.5 for i in range(2)]
    ax.set_xticks(
        xticks, ['legitimate', 'phishing'], rotation=45, ha='right'
    )
    ax.set_yticks(
        xticks, ['legitimate', 'phishing'], rotation=0, va='top'
    )
    ax.set_ylabel("True label")
    ax.set_xlabel("Predicted label")
    plt.show()

    report = metrics.classification_report(
        y_true, y_pred, target_names=['legitimate', 'phishing'], digits=4
    )
    print(report)
    test_acc = metrics.accuracy_score(y_true, y_pred)
    print(f'Accuracy: {test_acc:.4f}')

    test_precision = metrics.precision_score(y_true, y_pred, average='micro')
    print(f'Micro Precision: {test_precision:.4f}')

    test_recall = metrics.recall_score(y_true, y_pred, average='micro')
    print(f'Micro Recall/TPR: {test_recall:.4f}')

    test_f1 = metrics.f1_score(y_true, y_pred, average='micro')
    print(f'Micro f1-score: {test_f1:.4f}')
```

Figure E18

Precision Recall and Calibration Plots

```
4
5  def plot_precision_recall_curve_merge(y_true, y_scores, y_scores_b):
6      plt.figure(figsize=(6,5))
7      precision, recall, _ = precision_recall_curve(y_true, y_scores)
8      precision_b, recall_b, _ = precision_recall_curve(y_true, y_scores_b)
9      # add legend for each line
10     plt.step(recall_b, precision_b, color='orange', alpha=0.9, where='post', label = "Random Forest Baseline")
11     plt.step(recall, precision, color='b', alpha=0.9, where='post', label = "Random Forest Tuned")
12     plt.grid()
13     plt.xlabel('Recall')
14     plt.ylabel('Precision')
15     plt.ylim([0.0, 1.05])
16     plt.xlim([0.0, 1.05])
17     plt.title('Precision-Recall curve for Random Forest Baseline and Tuned Model')
18     plt.legend()
19     plt.legend(loc='lower left')
20     plt.show()
21
22
23 def calibration_plot_merge(y_test, y_scores, y_scores_b):
24     # Compute the calibration curve
25     fraction_of_positives, mean_predicted_value = calibration_curve(y_test, y_scores, n_bins=10)
26     fraction_of_positives_b, mean_predicted_value_b = calibration_curve(y_test, y_scores_b, n_bins=10)
27
28     # Plot the calibration curve
29     plt.figure(figsize=(8, 6))
30     plt.plot(mean_predicted_value, fraction_of_positives, 's-', label = "Random Forest Tuned")
31     plt.plot(mean_predicted_value_b, fraction_of_positives_b, 'o-', label = "Random Forest Baseline")
32     plt.plot([0, 1], [0, 1], '--', color='gray')
33     plt.grid()
34     plt.ylabel('Fraction of positives')
35     plt.xlabel('Mean predicted value')
36     plt.title('Calibration plot (reliability curve) for Random Forest Baseline and Tuned Model')
37     plt.legend()
38     plt.legend(loc='upper left')
39     plt.show()
40
41
42 plot_precision_recall_curve_merge(y_test, y_score[0::, -1], y_score_b[0::, -1])
43 calibration_plot_merge(y_test, y_score[0::, -1], y_score_b[0::, -1])
```