Malicious URL Detection using Machine Learning

by

Abubakar Siddeeq
B.Sc, University of Engineering and Technology, Taxila, 2018

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Malicious URL Detection using Machine Learning

by

Abubakar Siddeeq
B.Sc, University of Engineering and Technology, Taxila, 2018

Supervisory Committee

Dr. T. Aaron Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai Sima, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Malicious URL detection is important for cyber security experts and security agencies. With the drastic increase in internet usage, the distribution of such malware is a serious issue. Due to the wide variety of this malware, detection even with antivirus software is difficult. More than 12.8 million malicious URL websites are currently running. In this thesis, several machine learning classifiers along with ensemble methods are used to formulate a framework to detect this malware. Principal component analysis, k-fold cross validation, and hyperparameter tuning are used to improve performance. A dataset from Kaggle is used for classification. Accuracy, precision, recall, and f-score are used as metrics to determine the model performance. Moreover, model behavior with a majority of one label in the dataset is also examined as is typical in the real world.

Contents

List of Tables

List of Figures

GLOSSARY

URL	Uniform Resource Locator
PDF	Portable Document Format
XGBoost	Extreme Gradient Boosting
AdaBoost	Adaptive Boosting
ROC	Receiver Operator Characteristic
SVM	Support Vector Machine
API	Application Programming Interface
KNN	K-nearest Neighbor
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
FQDN	Fully Qualified Domain Name
IP	Internet Protocol
DNS	Domain Name System
SLD	Second Level Domain
TLD	Top Level Domain
RAM	Random Access Memory
SIEM	Security Information and Event Management
PCA	Principal Component Analysis
WEKA	Waikato Environment for Knowledge Analysis
SMOTE	Synthetic Minority Oversampling Technique

ACKNOWLEDGEMENTS

I am grateful to Allah who is the most merciful and my parents for their continuous support, love, prayers and motivation. I wish to express my sincere gratitude to my supervisor, Dr. T. Aaron Gulliver whose guidance, expertise, flexibility and encouragement contributed greatly to my graduate studies. His expertise and knowledge were essential for successful completion of my thesis. I would also like to thank Syed Abdul Aleem for his help during my thesis. I am also thankful to the University of Victoria for financial support and Compute Canada for providing computing resources that enabled me to generate the results in this thesis.

DEDICATION

To my parents, for their continuous support, love and prayers.

To my supervisor, Dr. T Aaron Gulliver, for his constant support, guidance and flexibility.

Chapter 1

Introduction

Anyone can now access the internet and browse websites according to their needs, so internet security has become essential. A Uniform Resource Locator (URL) is a link that forwards users to a website. Fraudulent and/or malicious URLs are generated to scam internet users. Once such a URL is clicked on, malware or a virus will be auto-installed on the device which can compromise the integrity of an organization or network [1]. Cybercriminals can get access to a system or network through such links. With the increase in the variety of malware, it is becoming more difficult for antivirus software to protect against malware. Moreover, with the increase in the availability of automated tools on the web, it has become easier for users to develop malware without advanced programming skills.

One of the most common means of spreading URLs is through email. Due to this, many internet users have become victims. Therefore, steps should be taken to block such emails so it is essential to detect these URLs as soon as possible. Blacklists are used to detect the majority of these URLs. Once a malicious URL is detected by security specialists, it is blacklisted by security tools like anti-malware and antivirus. This approach limits future attempts from the respective source. Although blacklisting is a common approach to preventing future attacks, it is impossible to compile a list of all such URLs.

Among the diverse types of malware, phishing is the most common. Phishing is a cybersecurity attack in which a malicious URL is used to acquire control of websites and user accounts. Hackers employ phishing to access sensitive information such as login credentials and credit card information [2]. Phishing is done by sending

a malicious URL via email. The victim receives a message from a known contact, person, entity, or organization. This message may contain malicious links or software that targets their computer and infects it. For example, a malicious link could direct the user to a website that looks similar to a popular website. The victim can also be tricked into revealing personal information like credit card information, login and password details, or other sensitive information. Phishing attacks are generally easy as most victims have little knowledge of web applications and computer technologies. A phishing email usually contains a Portable Document Format (PDF) or Word document as a malicious attachment that can infect a computer and install malware.

1.1 Related Work

Even though the use of machine learning for detecting malware has not been widely considered, there has been work in this field to detect malicious URLs. However, it is challenging and complex to classify malware using machine learning as the output generated from such classifiers has to be presented in an understandable form. Several studies have been conducted to determine and compare the performance of different approaches. A survey of machine learning approaches for malware detection was given in [3].

In [4], it was shown that the performance of boosted algorithms like Extreme Gradient Boosting (XGBoost) and Adaptive Boosting (AdaBoost) have better Receiver Operator Characteristic (ROC) curves than Support Vector Machine (SVM) and Naive Bayes. In [6], a modified version of the Random Forest classifier was used. In addition, information gain was used with this classifier for better feature representation. As a result, the accuracy obtained was 97.00. In [7], Application Programming Interface (API) functions were used to represent features of the dataset. A combination of classifiers was used with SVM giving the best results when used with the normalized polykernel with precision 97.60. In [9], the URL domain length has been used as the sole feature to detect malicious URLs and phishing attacks. In some instances, a red flag keyword has been used to blacklist phishing attacks. Another study used lexical analysis with defining features to detect such URLs [10]. However, about 9% of the malicious and benign instances were missed out of the more than 2 million URLs that were used as data. In [5], SVM and the random forest classifier were applied on an unbalanced dataset of 400,000 safe and 70,000 malicious URLs.

The accuracy obtained was 96.28% for random forest and 91.07% for SVM.

1.2 Research Objectives

Selection of the machine learning approach and dataset features for malware classification is a complex task. Moreover, previous studies have not provided a universal approach to feature extraction and classification. The research objective of this thesis is to determine which classification approach provides the best results with hyperparameter tuning. Several classifiers are considered, including boosted classifiers, and their performance is evaluated with different dataset sizes. In many datasets, the number of entries for one class is more than for the other classes. Therefore, the effect of an unbalanced dataset on classifier performance will be examined.

1.3 Research Questions

- **Question 1:** What is the performance of the random forest, decision tree, logistic regress, K-nearest Neighbor (KNN), kernel SVC, XGboost, and Adaboost classifiers with hyperparameter tuning?
- **Question 2:** How does the size of the dataset affect the performance of machine learning classifiers in terms of accuracy, precision, recall, f-score, and execution time?
- **Question 3:** To what extent does an unbalanced dataset affect the performance of machine learning classifiers in terms of accuracy, precision, recall, f-score, and execution time?

The remainder of this thesis is organized as follows. Chapter 2 presents a description of the Kaggle dataset and types of URLs. In Chapter 3, the approach used for feature selection and the features selected are described in detail. Moreover, the classification models used in this research are discussed along with the evaluation methods. Chapter 4 presents a detailed evaluation of the decision tree, random forest, kernel SVC, KNN, XGBoost, AdaBoost, and logistic regression classifiers and

ensemble methods in terms of accuracy, precison, recall, f-score and execution time. Finally Chapter 5 provides some conclusions and suggestions for future research.

Chapter 2

Background

2.1 Lexical Structure of a URL

The entries in the Kaggle dataset are URLs. Figure 2.1 presents the lexical structure of a URL. A URL starts with a protocol name such as Hyper Text Transfer Protocol (HTTP) or Hyper Text Transfer Protocol Secure (HTTPS). The Fully Qualified Domain Name (FQDN) is the complete domain name of the server hosting the website, which later translates into an Internet Protocol (IP) address using Domain Name System (DNS) servers. The domain name consists of a Second Level Domain (SLD) which is suffixed with the Top Level Domain (TLD) to which it is registered. The domain is registered with a domain registrar and is unique across the internet [11].

Figure 2.1: The lexical structure of a URL.

The hostname consists of subdomain and domain names. An attacker can easily modify the subdomain name and associate it with any value. The URL may also contain a path to a file which can be used to point to a malicious file. This is why cybersecurity experts struggle to provide a feasible solution to mitigate URL phishing attacks. An attacker can register a domain only once, and if this domain is identified as fraudulent users can be prevented from visiting it [11]. Figure 2.2 presents the

components of a URL. These are described below.

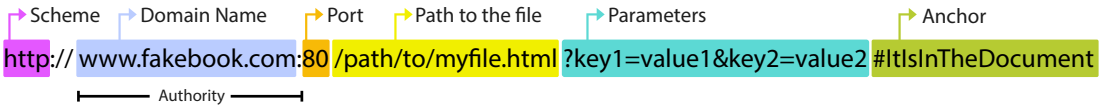

Figure 2.2: URL components.

2.1.1 Scheme

The exchange or transfer of data across a network is governed by the protocol. Figure 2.3 presents the URL scheme which defines the browser protocol to access the resources. The scheme is the initial part of the URL. Typically, web pages use two kinds of protocols, HTTPS and HTTP. HTTP is the unsecured version while HTTPS is the secured version. A web browser can use these protocols to access web pages. A scheme known as Malito is used to access mail clients [11].

Figure 2.3: URL scheme.

2.1.2 Authority

Figure 2.4 presents the URL authority which contains the domain name of the webpage. This section is partitioned from the scheme by //. When a port number is present, a colon is used to separate the domain from the port number. A domain name identifies which website to request. IP addresses can also be used to access web pages. The port number is not required when using the standard HTTP and HTTPS ports [11].

Figure 2.4: URL authority.

2.1.3 Path to Resource

Figure 2.5 presents the URL path to resource which contains the path to the requested resource of the web server. This is the physical location of the resource or the abstraction handle [11].

Figure 2.5: URL path to the resource.

2.1.4 Parameters

Figure 2.6 presents the URL parameters which guide the web browser to do requested actions while providing resources. These values are separated by & [11].

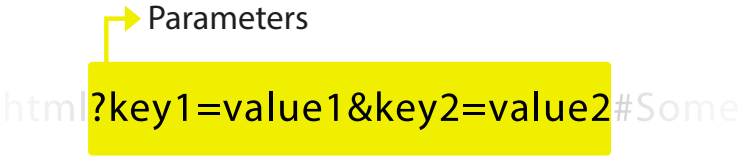

Figure 2.6: URL parameters.

2.1.5 Anchor

Figure 2.7 presents the URL anchor which specifies the part of the document to show. This is also used to indicate an exact point in a video or audio file. Anchors are also known as bookmarks [11].

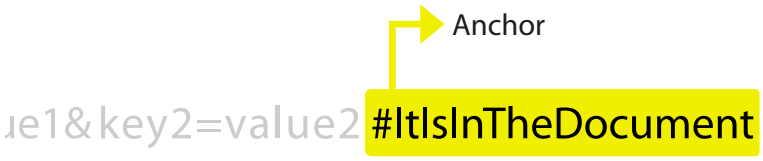

Figure 2.7: URL anchor.

2.2 Types of Malicious URLs

2.2.1 Spam URLs

In a spam attack, an attacker sends emails through illegal and/or unauthorized applications. These emails are sent in bulk and often contain promotional offers. Typically when a user accesses a spam email malware is installed on the device. Attackers use URLs to promote services and products. Emails, social media platforms, or text messages can be used to spread spam URLs. In addition, social media keep track of user activities and this data can be used by attackers. Therefore, social media and social networks are ideal for attacks. An attacker becomes part of a user's social network to increase the likelihood of their accessing the link [12].

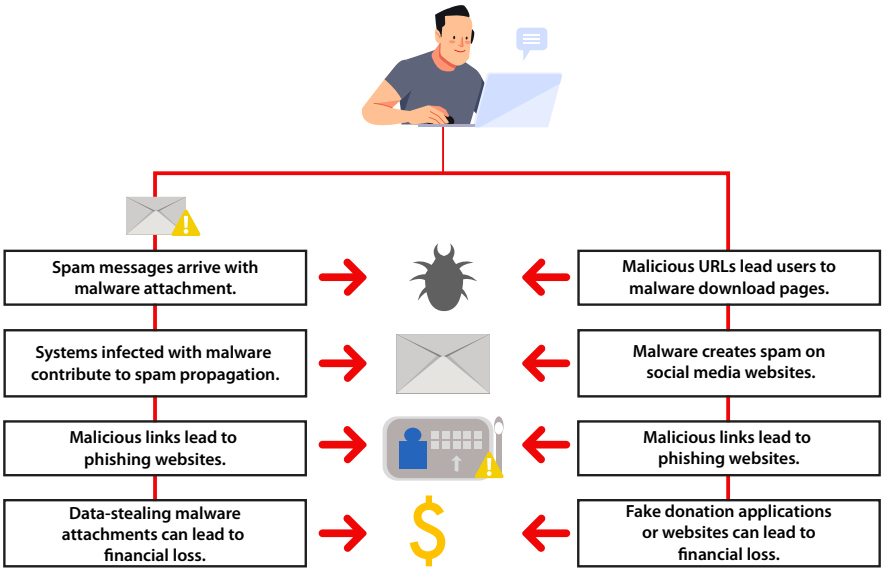

Figure 2.8: The spam attack process on a user device [12].

Figure 2.8 [12] presents the spam attack process on a user device via email and URLs. An email or URL is sent with the attached malware. After malware installation, the system is used for spam link propagation. These links redirect users to phishing websites like donation websites which can lead to financial loss.

1. **Distribution of Spam URLs:** To avoid blacklisting from social media platforms, attackers refrain from sending malware URLs in bulk but instead send them in low volumes infrequently. Experienced attackers also use engaging content to camouflage malicious URLs. This increases the probability of a user

clicking on the link. Once a system is compromised with malware, an attacker can use it to send malicious URLs to other systems, making detection difficult. Compromised or hacked accounts obtained on the black market are used by attackers to send links to their friends or colleagues. In this approach, the owner of the account is unaware of the malware distribution through their account [12].

2. **Masking of Spam URLs:** A feature of URLs is longevity as they are rarely removed by service providers or malware tools. URL shortening and open redirection of URLs are used by attackers to avoid having their static URLs blocked or flagged by authorities [12].

3. **Open Redirecting of Spam URLs:** Web servers can redirect a user to URLs based on the parameters in a web address. This technique is commonly used by attackers. Attackers refrain from using administered web servers to organize spam attacks. Instead, illegal service providers from the black market or cloud web servers are used to limit the cost. Figure 2.9 presents a URL redirection attack in which a landing server is used to launch spam attacks [12].

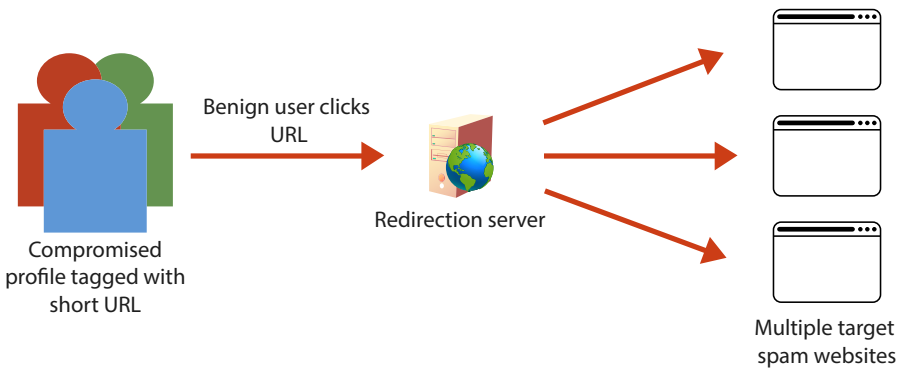

Figure 2.9: A URL redirection attack in which the redirection server is used to launch spam attacks.

4. **URL Shortening:** Google URL shortener, Tiny URL, and Bitly are commonly used to reduce the size of URLs. However, attackers can use them to mask URLs. Figure 2.10 presents the process of URL shortening.

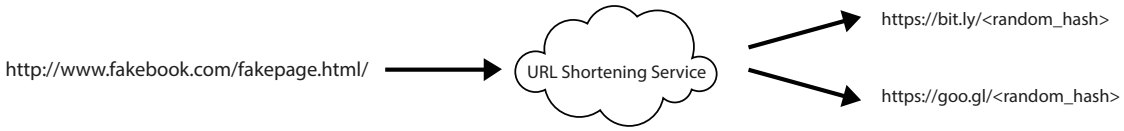

Figure 2.10: Spam URL shortening to mask malicious links.

2.2.2 Malware URLs

Accessing unknown advertisements, attached files, or malicious URLs can install malware on a system [13]. Figure 2.11 presents the different types of malware. These are explained below.

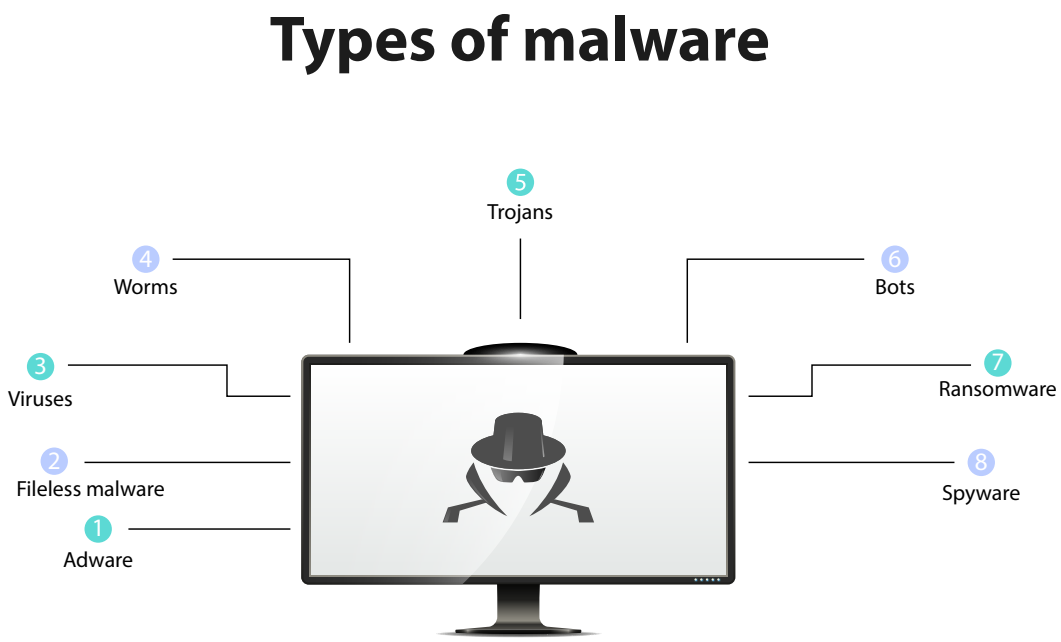

Figure 2.11: The types of malware.

1. **Adware:** Adware is unwanted advertisements injected into a system. These advertisements frequently pop up when the system is running and can be frustrating as they hinder work and consume Random Access Memory (RAM). Furthermore, they may be linked to malicious URLs that can infect a system when clicked on.

2. **Fileless Malware:** Fileless malware works by using an executable program to infect a system. This malware does not affect the system directly, making its

detection difficult. For example, when OceanLotus Group was infected, it took six months to detect the malware. This malware can be avoided by ensuring that only the administrator has the right to access the program.

3. **Viruses:** Viruses directly affect the system and can be spread to other systems in the network. They can be detected using antivirus software and security applications.

4. **Worms:** Worms attack known exploits of a system but cannot spread to other systems. This can be prevented by keeping the operating system updated with the latest patches.

5. **Trojans:** Trojans are developed using social engineering. They look like legitimate software and perform the described functionality while infecting the system. This can be prevented by not using software from an unknown service provider.

6. **Bots:** Bots can perform predefined automated tasks. In addition, they can distribute malware to other devices resulting in bot formation and bot compromised networks. Bots can be used to perform malicious activities on a large scale. Bots can be avoided by using a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) program [13].

7. **Ransomware:** Ransomware is used to gain access and encrypt the data. Ransomware is used to blackmail users in return for cash [13].

8. **Spyware:** Spyware is used by an attacker to keep track of user activities. In addition, it can be used to monitor user input to obtain passwords and private data [13].

2.3 Phishing Attack URLs

Phishing is a type of social engineering used by attackers to access data like social security numbers and credit card information. This occurs when an attacker poses as a trusted source to entice the user to click on a URL, link, or advertisement. For instance, an attacker can generate a malicious URL that resembles Facebook, like fakebook.com, which may be clicked on due to a lack of attention [14]. This can infect the system by installing malicious software [15]. The types of phishing attacks are given below.

1. **Deceptive Phishing:** In a deceptive phishing attack, credit card details, usernames, or passwords are stolen by posing as a known organization or entity. Then, an email containing malicious URLs or infected attachments is sent in large volumes which increases the chance of users clicking on it.

2. **Spear Phishing:** In a spear phishing attack, user personal information, such as their name, the names of their friends, or the organization they work for, is obtained through social engineering. Then, this is used in an email to convince the potential victim to click on a link. This attack has a high success rate but is difficult to develop.

3. **Whale Phishing:** Whale phishing targets top-level business executives and management to collect information. Thus, these attacks can cause significant damage to company finances, market value, and/or reputation.

Chapter 3

Machine Learning

The use of data to train an algorithm to make predictions, similar to humans, is called machine learning. It is a branch of artificial intelligence and is an essential component of the expanding field of data science. Several approaches to machine learning are explained below.

3.1 Machine Learning Methods

3.1.1 Supervised Machine Learning

This approach uses labelled data to train an algorithm to make predictions. Cross validation is used to prevent overfitting, underfitting, and anomalies in prediction. Some supervised learning algorithms are linear regression, logistic regression, decision tree, and random forest [16].

3.1.2 Unsupervised Machine Learning

This approach uses unlabelled data to train an algorithm to make predictions. As a result, discovering hidden data patterns and grouping are done without human interference. Differences and similarities in data are determined for data analysis, customer segmentation, and image or pattern recognition. Some unsupervised learning algorithms are K-means clustering and neural networks [16].

3.2 Machine Learning Models

The goal of machine learning is to perform tasks such as classification, regression, or clustering by training a model based on an algorithm. In this thesis, several machine learning models are considered. Figure 3.1 shows the steps in the machine learning process.

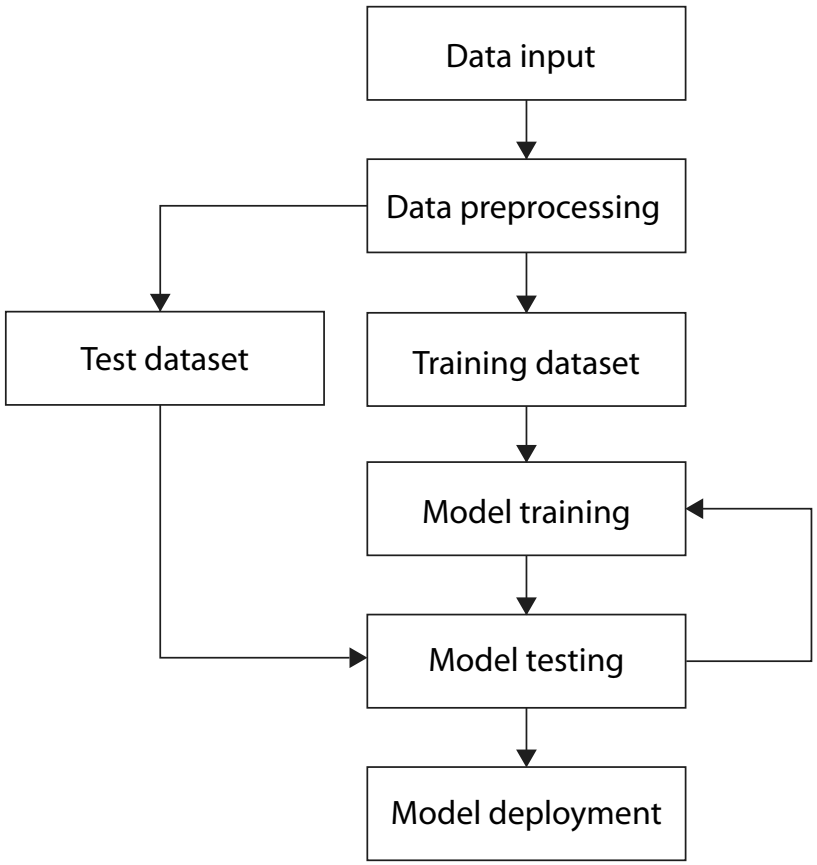

Figure 3.1: The machine learning process.

3.2.1 Dataset

Selecting a suitable dataset is important in machine learning. A machine learning model is trained using this data so its future behavior and outputs are dependant on it. Many public and private sector organizations share their data which can contain information about healthcare, finances, real estate, and weather as well as phishing emails and malicious URLs. The data is typically stored in .csv or .xlsx formats [17]. The URL dataset from the Kaggle website [18], was used for this research. It contains

35000 malicious and non-malicious URLs. King Phisher, ZPhisher, and Python were used to capture these URLs. This dataset contains two classes of URLs, malicious and non malicious.

3.2.2 Data Preprocessing

A classifier cannot be trained with URLs. Therefore, they must be converted into a suitable format. The data preprocessing steps are as follows.

1. **Uncompressing and Formatting:** The dataset obtained from Kaggle was in compressed form. First, it was uncompressed and then Excel was used to get the data in .csv form which is readable by Python.

2. **White Spaces Removal:** The str.stip( ) function was used to remove white spaces.

3. **Labeling:** The dataset entries were labeled as malicious and non-malicious instead of 0 and 1, respectively.

4. **Cleaning:** Duplicate entries were removed using a Phyton script.

3.2.3 Feature Selection

The dataset has ten features whose values are shown in Figure 3.2. This shows the values of the features for each entry in the dataset. The feature url-len has the highest value followed by digit-count.

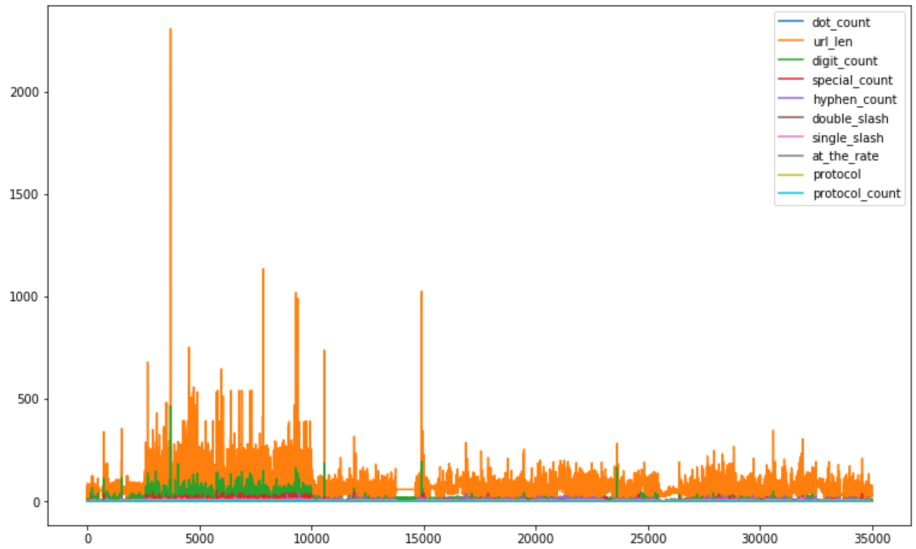

Figure 3.2: Malicious and non-malicious URL dataset feature values.

1. **Data Normalization:** Feature data normalization is applied to avoid the influence of a single feature on model performance just because it has larger values. Min-max scaling was used for data normalization which rescales the feature values from their original range to [0,1] and is given by

$$z = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

where x_{max} is the maximum feature value while x_{min} is the corresponding minimum value.

2. **Dimensionality Reduction:** Principal Component Analysis (PCA) is used for dimensionality reduction. It is commonly used on large datasets to convert a large collection of variables into a smaller set while keeping the majority of information from the original dataset. This is done by determining the relationship between features using the correlation matrix. The variances of the principal components, also known as eigenvalues, are calculated using the eigendecomposition matrix. Principal components with the most significant eigenvalues are kept, while ones with small or negligible eigenvalues are removed. From the 10 features given in Table 3.1, the top 7 were selected based on their eigenvalues.

Feature Name	Feature Description	Eigenvalue
dot-count	The number of dots present in the URL	1.667
url-len	The length of the URL	1.843
digit-count	The number of digits in the URL	2.895
special-count	The number of special digits in the URL	2.468
hyphen-count	The number of hyphens in the URL	1.184
double-slash	The number of double slashes in the URL	0.943
single-slash	The number of single slashes in the URL	0.786
at-the-rate	The number of @ signs in the URL	0.124
protocol	The length of the protocol name	0.016
protocol-count	The number of protocols used in the URL	0.026

Table 3.1: The features with their eigenvalues.

3.3 Machine Learning Models

The performance of a classifier depends on the nature of the problem and the dataset used. In this research, four classifiers and two ensemble methods are used. The models were implemented using the Python language. The reason for using a classifier in Python instead of the Waikato Environment for Knowledge Analysis (WEKA) software is that it allows more liberty in setting the parameters. WEKA is an open source software package that has machine learning algorithms for data processing and provides output visualization. The machine learning models considered here are described below.

3.3.1 Decision Tree

Decision tree is a model which uses trees for classification. Each leaf node in the tree represents a class label while internal nodes represent class attributes. Information gain is used for attribute selection and ordering and is given by

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum \left(\frac{|S_v|}{|S|} \right) \times \text{Entropy}(S_v)$$

where S is the dataset before splitting, A is the number of subsets generated by splitting, and S_v is subset v after splitting.

3.3.2 Random Forest

Random forest is a form of decision tree used to solve regression and classification problems. It uses a group of decision trees to provide predictions by calculating their average value. The importance of feature i is given by

$$RFfi_i = \left(\frac{\sum_j normfi_{ij}}{\sum_{j \in allfeatures, k \in alltrees} normfi_{jk}} \right)$$

where $normfi_{ij}$ is the normalized feature importance for feature i in tree j and $normfi_{jk}$ is the normalized feature importance for feature j in tree k.

3.3.3 SVM

Support Vector Machine (SVM) is used for regression and classification. In this algorithm, the features are plotted in n dimensional space according to their values. The hyperplane is given by

$$y = w + b$$

where w is a vector normal to the hyperplane and b is the offset.

3.3.4 K-nearest Neighbor

K-nearest Neighbor (KNN) is also known as the lazy learning method. It provides classification and prediction using the Euclidean distance to group similar points which is given by

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

where x and y are two points in n dimensional space.

3.3.5 Ensemble Methods

Ensemble methods combine different machine learning models for better results. The ensemble methods used in this thesis are described below.

3.3.6 XGBoost

XGBoost is a gradient boosting technique. It uses gradient boosted trees where each tree is dependant on the residuals of prior trees. The prediction is given by

$$F_2(x) = \sum h_1(x) + h_2(x)$$

where h_1 is the first order derivative and h_2 is the second order derivative.

3.3.7 AdaBoost

AdaBoost is an adaptive boosting technique. It uses boosting to reduce the variance and bias in supervised learning. The output is given by

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

where α_t is the weight assigned to the classifier and h_t is the single split decision tree known as the decision tree stump.

3.4 Model Training

During training, a model processes the dataset several times. Validation accuracy is used as the stopping criteria to prevent the model from overfitting. Training stops when the validation accuracy stops improving. This also reduces execution time and resource utilization.

3.4.1 k-fold Cross Validation

In k-fold cross validation, the dataset is split into test and training sets. k-fold cross validation is used to avoid overfitting and underfitting where k is the number of data

partitions. In this research, $k = 5$. Figure 3.3 shows that the data is split into 5 equal parts. In the first fold, the first part of the data is taken as the test set and the remaining as the training set. Similarly in the second fold the second part of the data, in the third fold the third part of the data, in the fourth fold the fourth part of the data, and in the fifth fold the fifth part of the data is taken as the test set and the remaining parts as the training set. The average of the 5 outputs is taken as the final result.

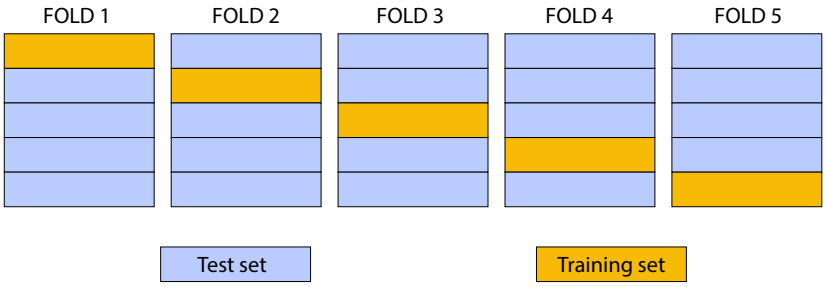

Figure 3.3: 5-fold cross validation of a dataset.

3.4.2 Hyperparameter Tuning

For hyperparameter tuning, GridSearch is used. In GridSearch, a classification model is evaluated using parameter values in the range [1-100] to find the best values. Hyperparameters with the most significant effect on the model are kept, while ones with small or negligible effect are ignored to reduce the execution time. Table 3.2 gives the selected hyperparameters for each classifier. Table 3.3 gives the hyperparameters that were not selected because of their minimal effect on classifier performance.

Classifier	Hyperparameters
Decision Tree	max-depth, min-samples-split, min-samples-leaf, criterion
Random Forest	n-estimators, max-depth, min-samples-split, criterion
AdaBoost	n-estimators, base-estimator, learning-rate, random-state
XGBoost	n-estimators, max-depth, max-features, min-samples-split
K-nearest Neighbor	n-neighbors, weights, leaf-size, algorithm
Kernel SVC	C, kernel, random-state, cache-size
Logistic Regression	C, penalty, random-state, tol

Table 3.2: The classifier hyperparameters that were selected.

Classifier	Hyperparameters
Decision Tree	splitter, min-weight-fraction-leaf, max-features, random-state, max-leaf-nodes, class-weight
Random Forest	min-samples-leaf, min-weight-fraction-leaf, max-features, max-leaf-nodes, random-state
AdaBoost	algorithm
XGBoost	loss, subsample, learning-rate, criterion, min-samples -leaf, max-leaf-nodes
K-nearest Neighbor	p, metric, metric-params, n-jobs
Kernel SVC	degree, gamma, class-weigh, tol, cache-size
Logistic Regression	class-weight, max-iter, n-jobs, fit-intercept

Table 3.3: The classifier hyperparameters that were not selected.

1. **Decision Tree:** In decision tree, max-depth is the maximum depth of the tree, min-samples-split is the minimum number of samples to split an internal node, min-samples-leaf is the minimum number of samples required to split a node, and criterion is the function to measure the node split.

2. **Random Forest:** In random forest, n-estimators is the number of trees, max-depth is the maximum depth of a tree, min-samples-split is the minimum number of samples required to split an internal node, and criterion is the function to measure the node split.

3. **AdaBoost:** In AdaBoost, n-estimators is the maximum number of estimators where boosting is stopped, base-estimator is the estimator from which the boosted ensemble is built, learning-rate is the classifier weight for boosting, and random-state is a random value given to each base estimator.

4. **XGBoost:** In XGBoost, n-estimators is the number of boosting stages, max-depth is the maximum depth of the estimator, max-features is the number of features for each split, and min-samples-split is the minimum number of samples required to split a tree.

5. **K-Nearest Neighbor:** In K-nearest neighbor, n-neighbors is the number of

neighbors, weights is the weight function, leaf-size is the size of a leaf, and algorithm is the algorithm to compute the nearest neighbor.

6. **Kernel SVC:** In kernel SVC, C is the parameter for regularization, kernel is the kernel type used, random-state is a random number used for shuffling the data, and cache-size is the size of the kernel cache.

7. **Logistic Regression:** In logistic regression, C is the parameter for regularization, penalty is the type of penalty, random-state is a random number used for shuffling the data, and tol is the stopping criteria tolerance.

3.5 Model Evaluation

Evaluation metrics are needed to determine the performance of the models. The evaluation metrics used in this research are given below.

3.5.1 Confusion Matrix

An $N \times N$ table is used to obtain the precision, recall, and accuracy scores using True Negative, False Positive, True Positive, or False Negative which are described below [19].

1. **False Positive (FP):** The model prediction is correct while the actual value is false. False positive is also known as a Type I error.

2. **True Negative (TN):** The model prediction is false while the actual value is false.

3. **True Positive (TP):** The model prediction is correct while the actual value is true.

4. **False Negative (FN):** The model prediction is false while the actual value is true. False negative is also known as a Type II error.

3.5.2 Precision

Precision is the ratio of the number of true positive to the sum of the number of true positive and false positive and is given by

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

3.5.3 Accuracy

Accuracy is the fraction of correct predictions for both malicious and non-malcious URLs and is given by

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

3.5.4 Recall

Recall is the ratio of the number of true positive to the sum of the number of true positive and false negative and is given by

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

3.5.5 f-score

f-score is the harmonic mean of recall and precision and is given by

$$f\text{-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

3.5.6 Execution Time

Execution time is the time taken by the classifier or ensemble method for both training and testing.

Chapter 4

Results

This section presents the execution time, recall, precision, f-score, and accuracy of the classification and ensemble methods. Four experiments are performed with different dataset sizes and proportions of malicious and non-malicious URLs. Random sampling is used to obtain datasets of different sizes. Hyperparameter tuning is employed using a GridSearch with a [1-100] grid. The values for accuracy, precision, recall, and f-score are given as percentages.

4.1 Case 1

In this experiment, classification is conducted using a dataset of 35000 entries with hyperparameter tuning. Synthetic Minority Oversampling Technique (SMOTE) is used to balance the class distribution by randomly replicating minority class instances. The results for balanced and unbalanced datasets are given below.

4.1.1 Classification with the Unbalanced Dataset of 35000 Entries

For the classification of the unbalanced dataset, a dataset with an 80:20 split is used so 80% are malicious URLs and 20% are non-malicious URLs. Figure 4.1 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from

decision tree was 99.74 with max-depth = 14. Figure 4.2 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.86 with max-depth = 18. Figure 4.3 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 99.63 with n-estimators = 86. Figure 4.4 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.75 with max-depth = 15. Figure 4.5 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params, and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 97.63 with n-neighbors = 8. Figure 4.6 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 99.59 with $C = 99$. Figure 4.7 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.60 with $C = 28$.

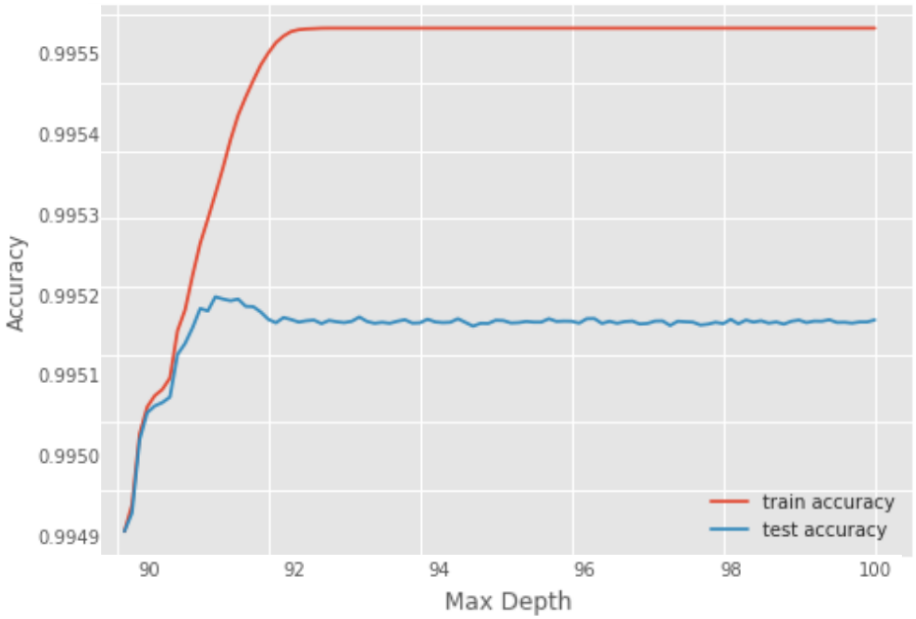

Figure 4.1: Performance of the decision tree classifier with the unbalanced dataset of 35000 entries.

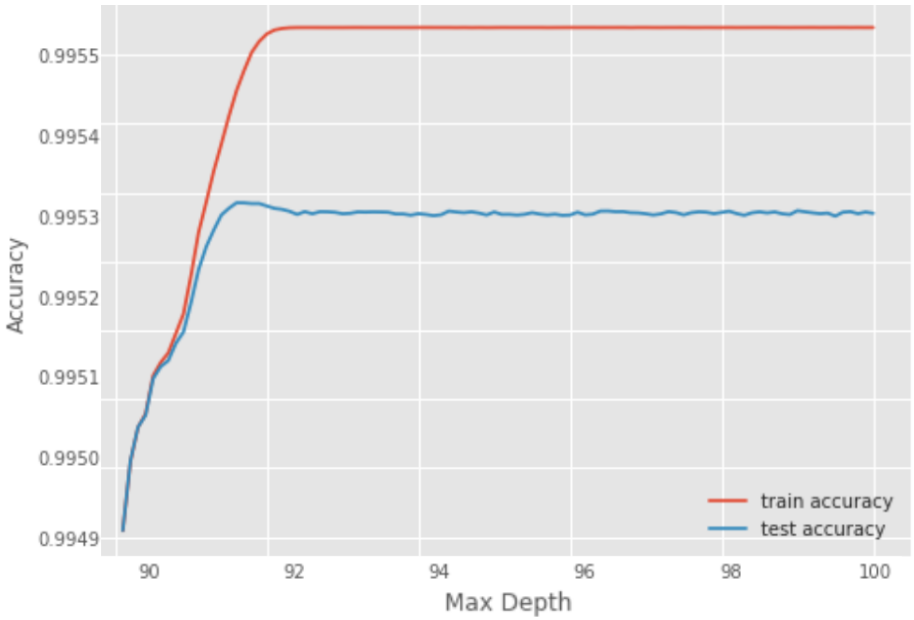

Figure 4.2: Performance of the random forest classifier with the unbalanced dataset of 35000 entries.

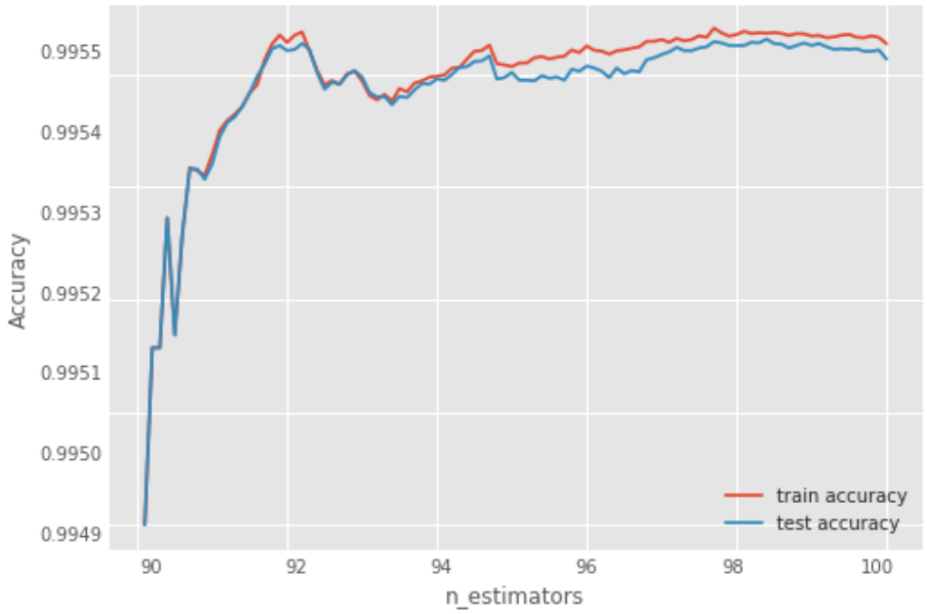

Figure 4.3: Performance of the AdaBoost classifier with the unbalanced dataset of 35000 entries.

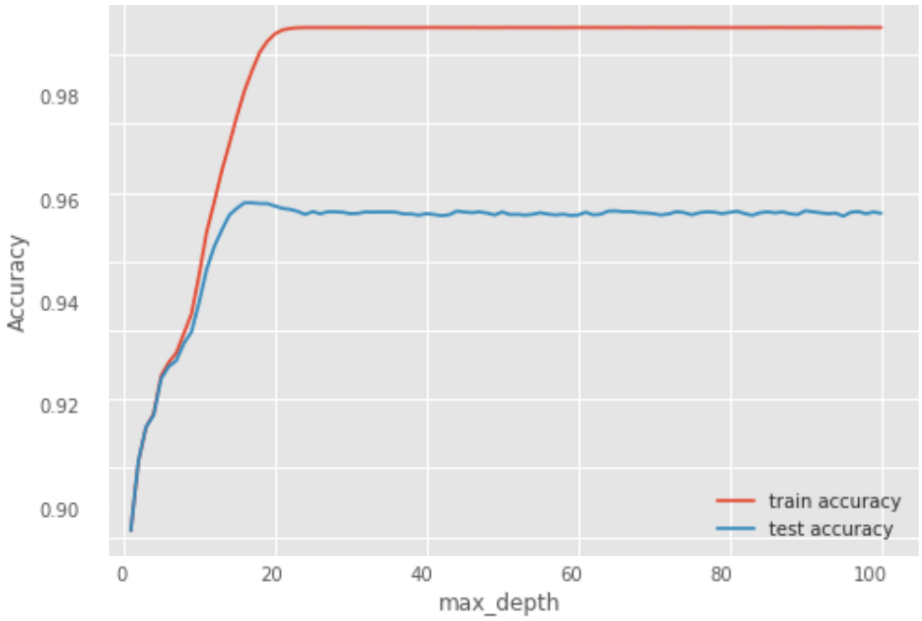

Figure 4.4: Performance of the XGBoost classifier with the unbalanced dataset of 35000 entries.

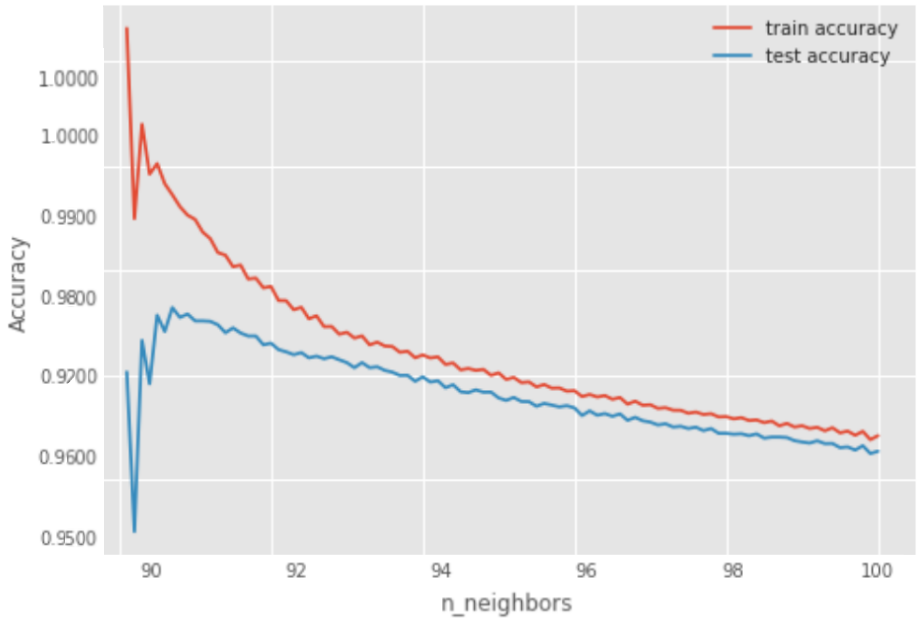

Figure 4.5: Performance of the K-nearest neighbor classifier with the unbalanced dataset of 35000 entries.

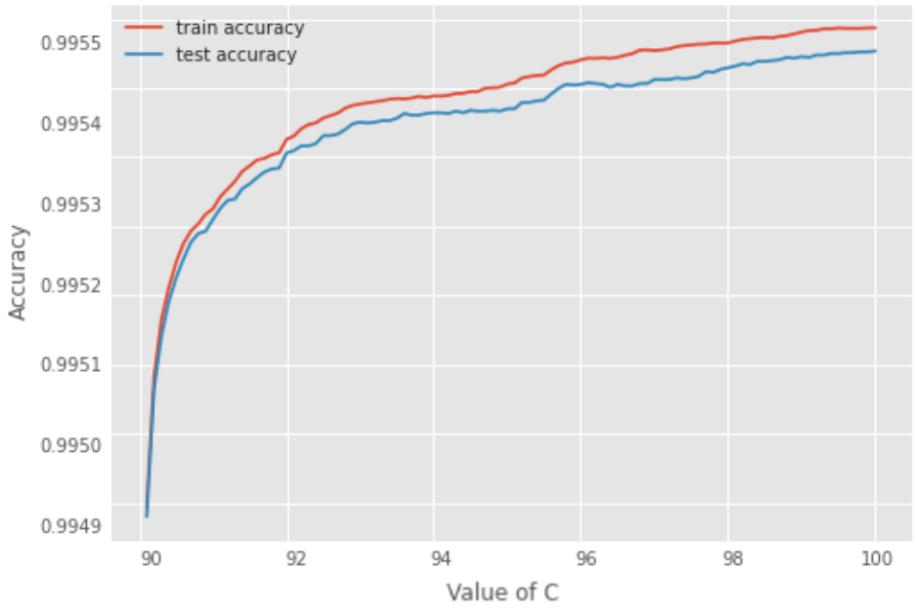

Figure 4.6: Performance of the kernel SVC classifier with the unbalanced dataset of 35000 entries.

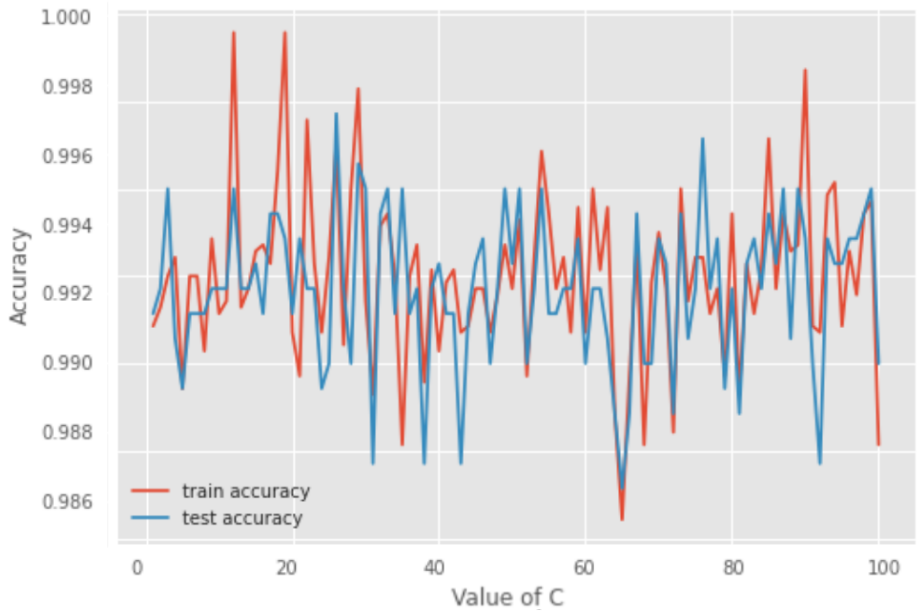

Figure 4.7: Performance of the logistic regression classifier with the unbalanced dataset of 35000 entries.

Table 4.1 shows the testing results after hyperparameter tuning. Among the seven methods, random forest has the highest accuracy, precision, recall, and f-score at 99.86, 99.62, 99.26, 99.46, followed by XGBoost at 99.75, 99.51, 99.17, 99.35, decision tree at 99.74, 99.47, 98.94, 99.29, logistic regression at 99.68, 99.08, 98.98, 98.96, AdaBoost at 99.63, 99.30, 99.06, 99.13, kernel SVC at 99.59, 98.95, 98.58, 98.75, K-nearest neighbor at 97.63, 97.30, 98.06, 97.13. In terms of execution time, decision tree had the lowest at 36.14 s followed by logistic regression at 146.6 s, AdaBoost at 676.1 s, random forest at 1558 s, XGBoost at 2080 s, K-nearest neighbor at 8338 s, and kernel SVC at 59311 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.74	99.47	98.94	99.29	36.14
Random Forest	99.86	99.62	99.26	99.46	1558
AdaBoost	99.63	99.30	99.06	99.13	676.1
XGBoost	99.75	99.51	99.17	99.35	2080
K-nearest Neighbor	97.63	97.30	98.06	97.13	8338
Kernel SVC	99.59	98.95	98.58	98.75	59311
Logistic Regression	99.68	99.08	98.98	98.96	146.6

Table 4.1: Classifier performance with the unbalanced dataset of 35000 entries.

4.1.2 Classification with the Balanced Dataset of 35000 Entries

For the classification of the balanced dataset, a dataset with a 50:50 split is used so 50% are malicious URLs and 50% are non-malicious URLs. Figure 4.8 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 99.58 with the max-depth = 13. Figure 4.9 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.58 with max-depth = 16. Figure 4.10 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 99.58 with n-estimators = 98. Figure 4.11 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.68 with max-depth = 16. Figure 4.12 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 96.78 with n-neighbors = 3. Figure 4.13 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0 and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 99.37 with $C = 100$. Figure 4.14 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty,

random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.48 with $C = 66$.

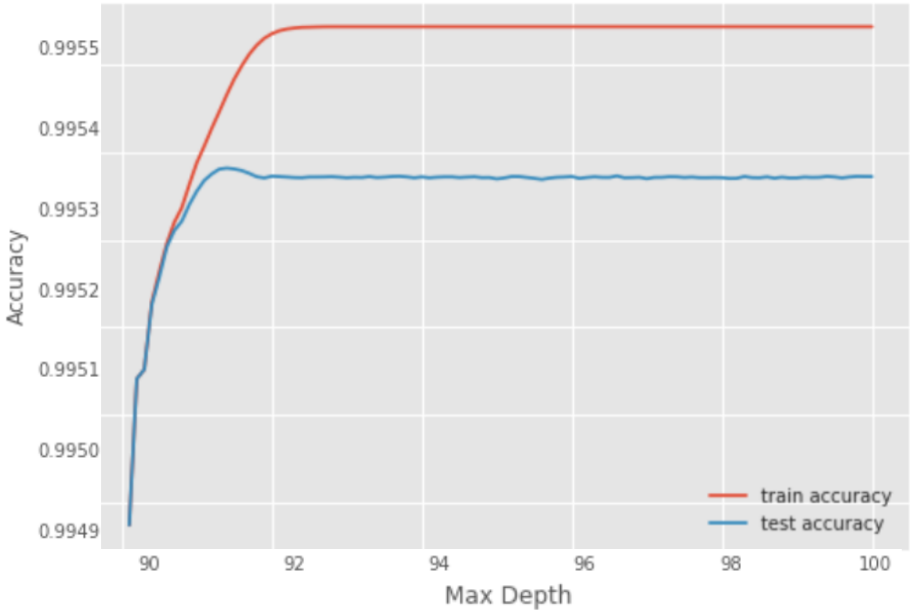

Figure 4.8: Performance of the decision tree classifier with the balanced dataset of 35000 entries.

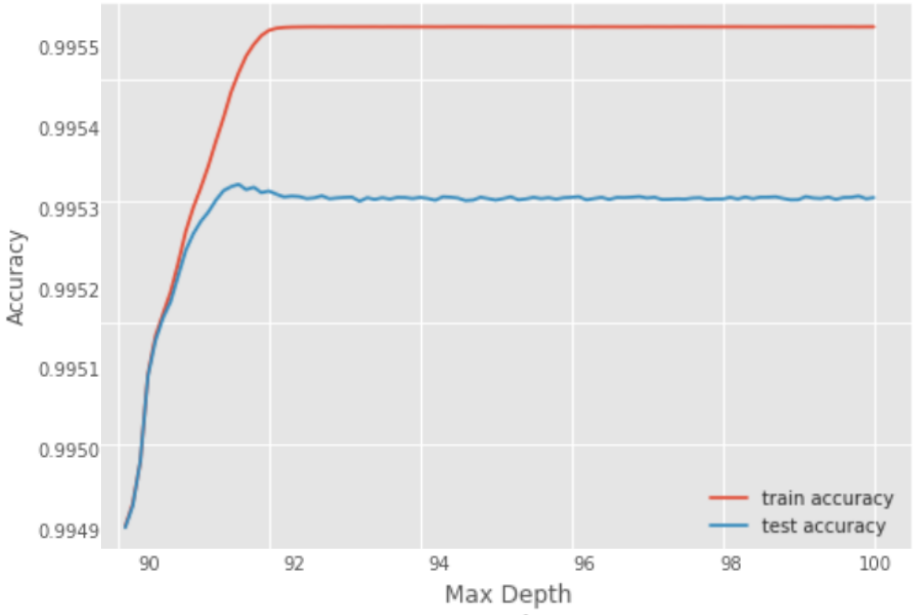

Figure 4.9: Performance of the random forest classifier with the balanced dataset of 35000 entries.

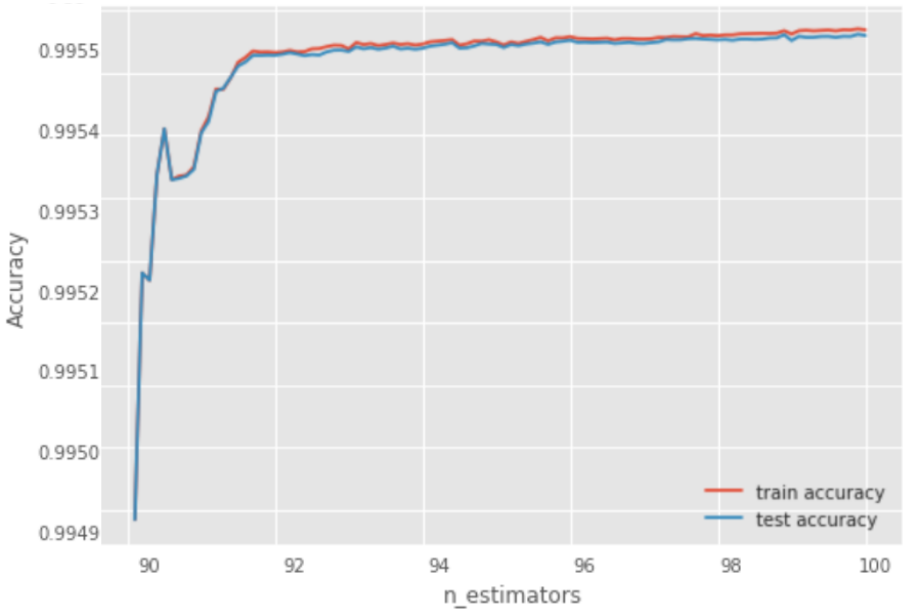

Figure 4.10: Performance of the AdaBoost classifier with the balanced dataset of 35000 entries.

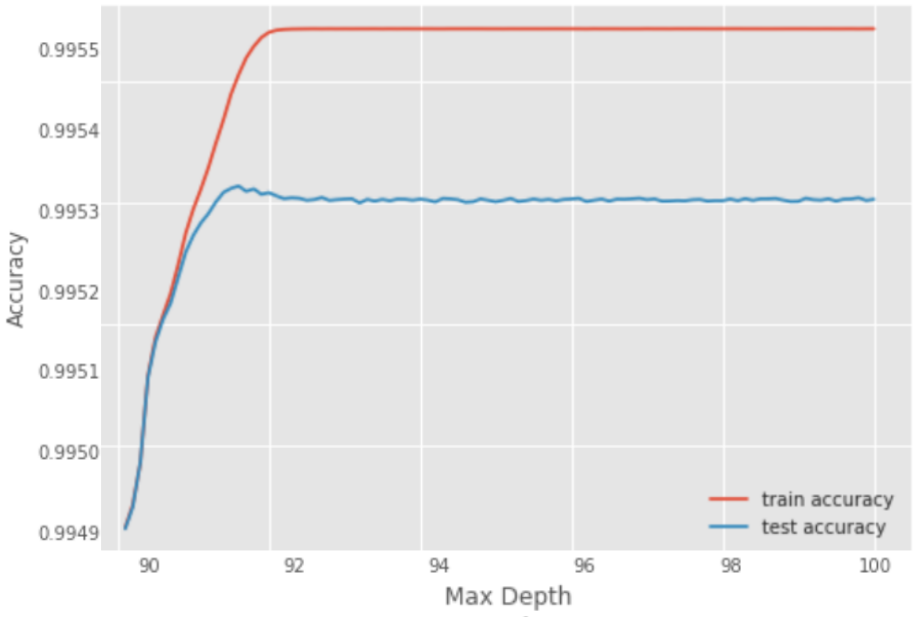

Figure 4.11: Performance of the XGBoost classifier with the balanced dataset of 35000 entries.

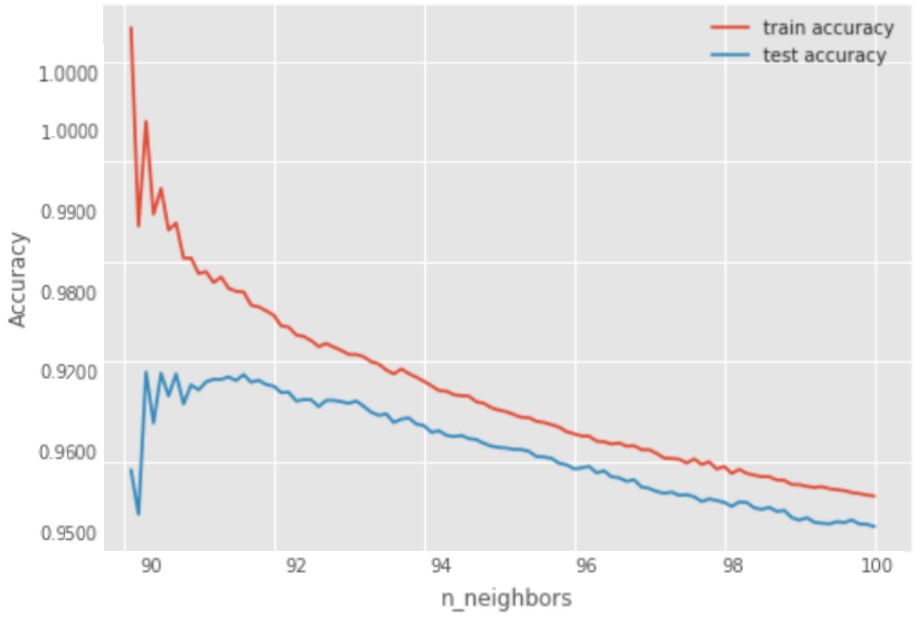

Figure 4.12: Performance of the K-nearest neighbor classifier with the balanced dataset of 35000 entries.

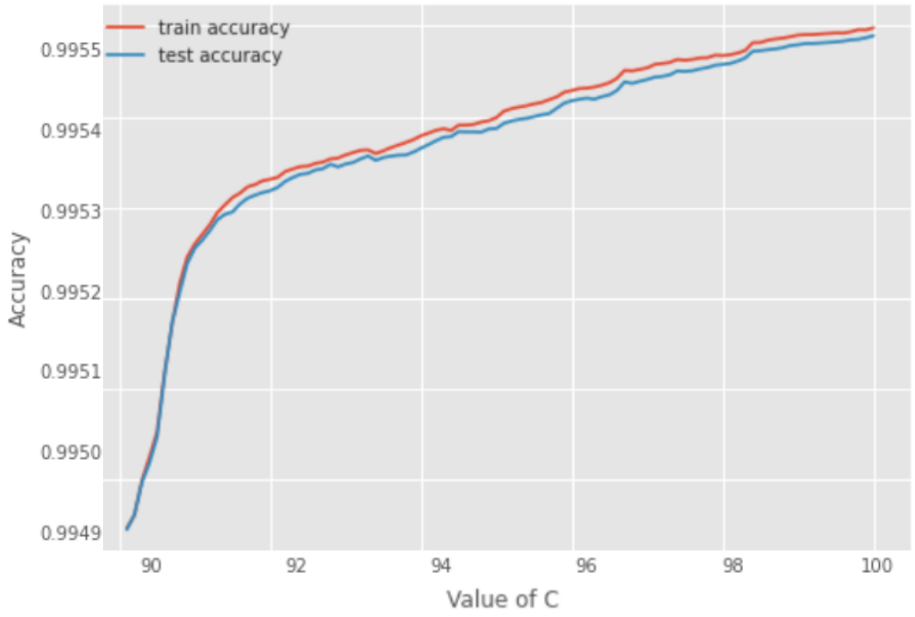

Figure 4.13: Performance of the kernel SVC classifier with the balanced dataset of 35000 entries.

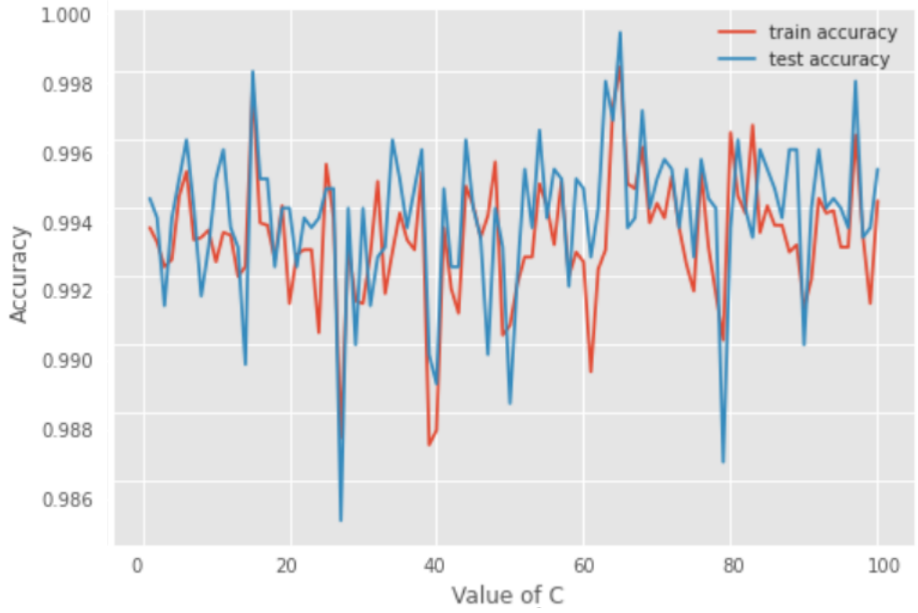

Figure 4.14: Performance of the logistic regression classifier with the balanced dataset of 35000 entries.

Table 4.2 shows the testing results after hyperparameter tuning. Among the seven methods, XGBoost has the highest accuracy, precision, recall, and f-score at 99.68, 99.83, 99.36, 99.66, followed by random forest at 99.58, 99.83, 99.35, 99.53, decision tree at 99.58, 99.81, 99.15, 99.51, AdaBoost at 99.58, 99.82, 99.25, 99.55, logistic regression at 99.48, 99.80, 99.04, 99.40, kernel SVC at 99.37, 99.70, 99.03, 99.33, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37. In terms of execution time, decision tree had the lowest at 23.55 s followed by logistic regression at 101.9 s, AdaBoost at 285.1 s, random forest at 928.3 s, K-nearest neighbor at 1035 s, XGBoost at 1559 s, and kernel SVC at 33445 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.58	99.81	99.15	99.51	23.55
Random Forest	99.58	99.83	99.35	99.53	928.3
AdaBoost	99.58	99.82	99.25	99.55	285.1
XGBoost	99.68	99.83	99.36	99.66	1559
K-nearest Neighbor	96.78	97.33	96.42	97.37	1035
Kernel SVC	99.37	99.70	99.03	99.33	33445
Logistic Regression	99.48	99.80	99.04	99.40	101.9

Table 4.2: Classifier performance with the balanced dataset of 35000 entries.

4.2 Case 2

In this experiment, classification is applied on a dataset of 20000 entries with hyperparameter tuning. SMOTE is used to balance the class distribution by randomly replicating minority class instances. The results for balanced and unbalanced datasets are given below.

4.2.1 Classification with the Unbalanced Dataset of 20000 Entries

For the classification of the unbalanced dataset, a dataset with an 80:20 split is used so 80% are malicious URLs and 20% are non-malicious URLs. Hyperparameter tuning was used to improve the performance of the classifiers. Figure 4.15 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 99.62 with max-depth = 15. Figure 4.16 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.75 with max-depth = 18. Figure 4.17 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 99.62 with n-estimators = 92. Figure 4.18 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.75 with max-depth = 8. Figure 4.19 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p,

metric-params, and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 97.03 with n-neighbors = 5. Figure 4.20 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 99.48 with C = 72. Figure 4.21 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.61 with $C = 8$.

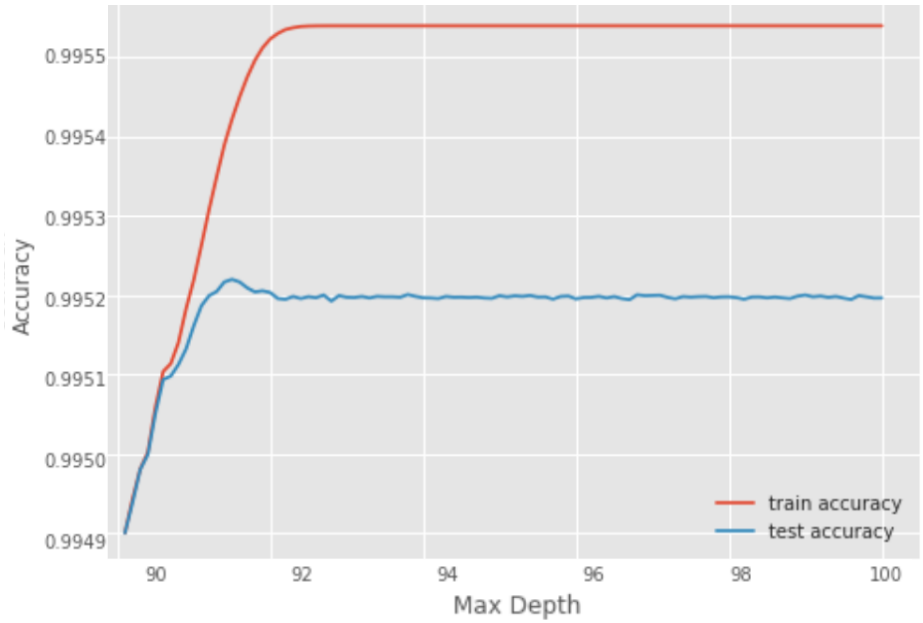

Figure 4.15: Performance of the decision tree classifier with the unbalanced dataset of 20000 entries.

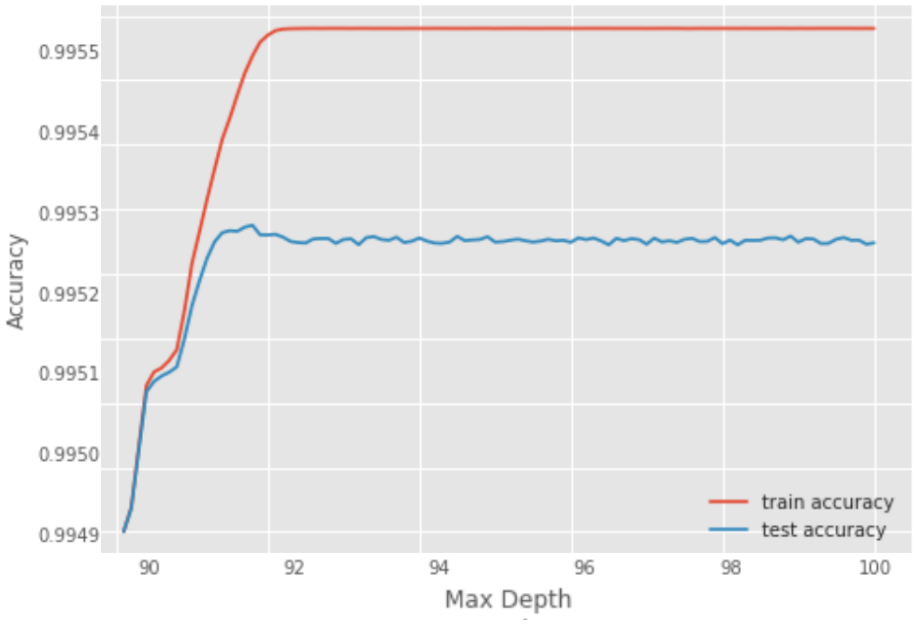

Figure 4.16: Performance of the random forest classifier with the unbalanced dataset of 20000 entries.

Figure 4.17: Performance of the AdaBoost classifier with the unbalanced dataset of 20000 entries.

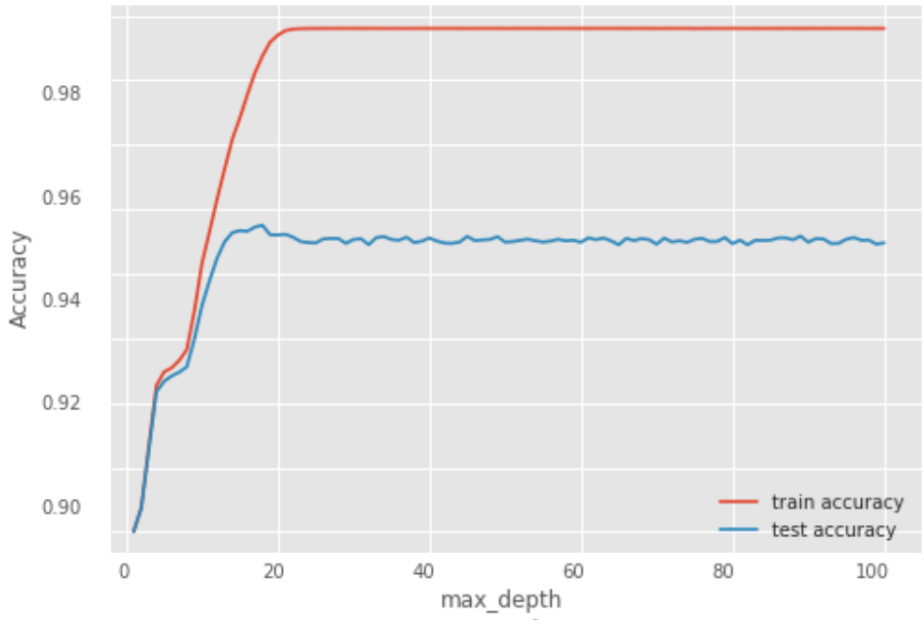

Figure 4.18: Performance of the XGBoost classifier with the unbalanced dataset of 20000 entries.

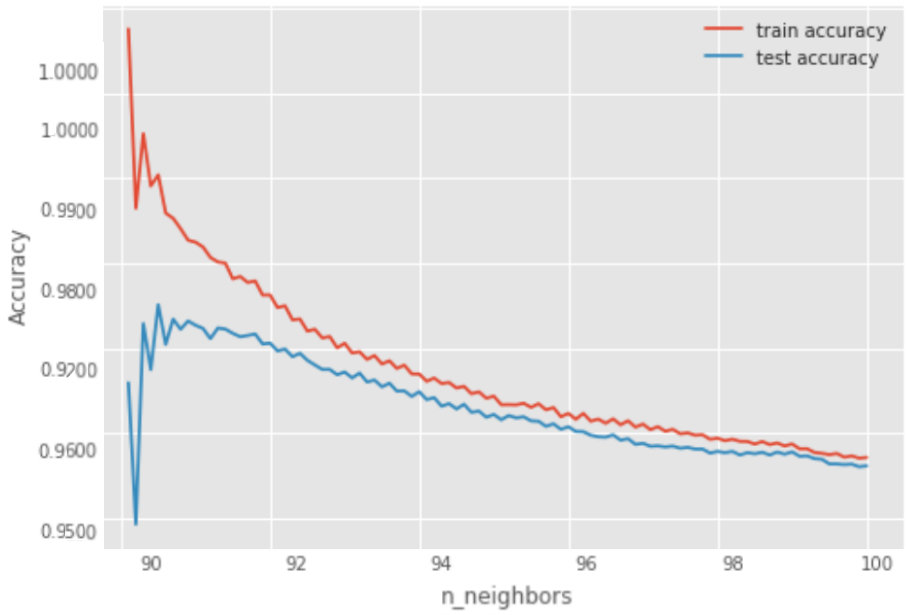

Figure 4.19: Performance of the K-nearest neighbor classifier with the unbalanced dataset of 20000 entries.

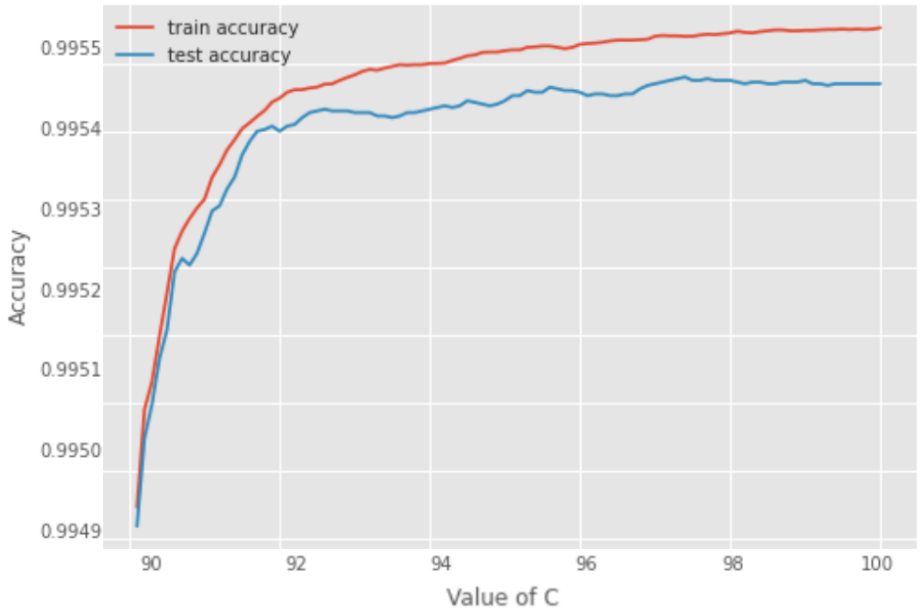

Figure 4.20: Performance of the kernel SVC classifier with the unbalanced dataset of 20000 entries.

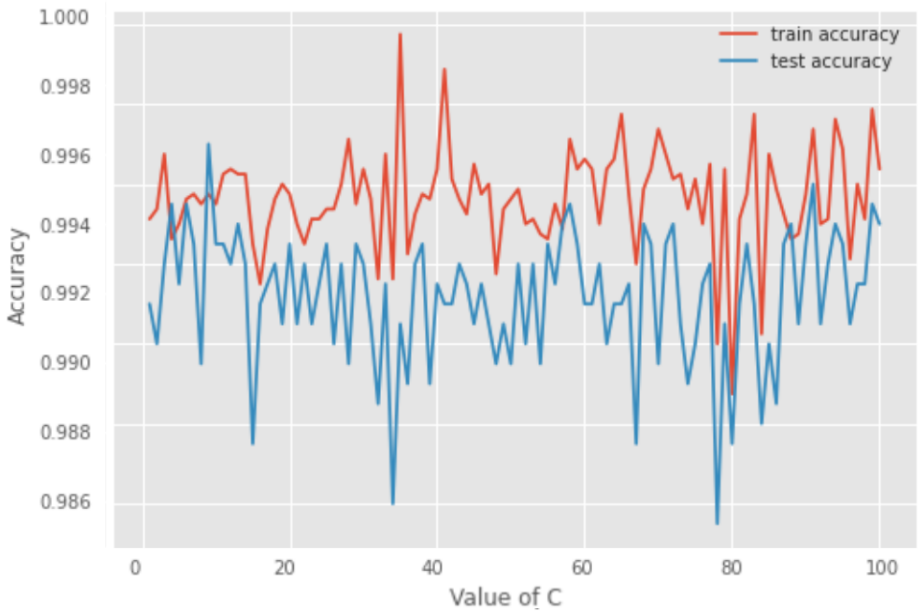

Figure 4.21: Performance of the logistic regression classifier with the unbalanced dataset of 20000 entries.

Table 4.3 shows the testing results after hyperparameter tuning. Among the seven methods, random forest has the highest accuracy, precision, recall, and f-score at

99.75, 99.51, 99.13, 99.31, followed by XGBoost at 99.75, 99.59, 98.92, 99.29, decision tree at 99.62, 99.28, 98.80, 99.08, AdaBoost at 99.62, 99.26, 98.69, 98.96, logistic regression at 99.61, 99.18, 98.80, 99.08, kernel SVC at 99.48, 98.83, 98.35, 98.53, K-nearest neighbor at 97.03, 98.99, 97.06, 98.98. In terms of execution time, decision tree had the lowest at 32.20 s followed by logistic regression at 101.4 s, AdaBoost at 600.5 s, random forest at 1179 s, K-nearest neighbor at 1381 s, XGBoost at 1567 s, and kernel SVC at 26937 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	fScore (%)	Execution Time (s)
Decision Tree	99.62	99.28	98.80	99.08	32.20
Random Forest	99.75	99.51	99.13	99.31	1179
AdaBoost	99.62	99.26	98.69	98.96	600.5
XGBoost	99.75	99.59	98.92	99.29	1567
K-nearest Neighbor	97.03	98.99	97.06	98.98	1381
Kernel SVC	99.48	98.83	98.35	98.53	269.4
Logistic Regression	99.61	99.18	98.80	99.08	101.4

Table 4.3: Classifier performance with the unbalanced dataset of 20000 entries.

4.2.2 Classification with the Balanced Dataset of 20000 Entries

For the classification of the balanced dataset, a dataset with a 50:50 split is used so 50% are malicious URLs and 50% are non-malicious URLs. Hyperparameter tuning was used to improve the performance of the classifiers. Figure 4.22 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 99.46 with max-depth = 14. Figure 4.23 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.58 with max-depth = 18. Figure 4.24 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters

base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 99.46 with n-estimators = 100. Figure 4.25 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.47 with max-depth = 9. Figure 4.26 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params, and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 96.78 with n-neighbors = 10. Figure 4.27 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 99.25 with C = 100. Figure 4.28 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.36 with $C = 86$.

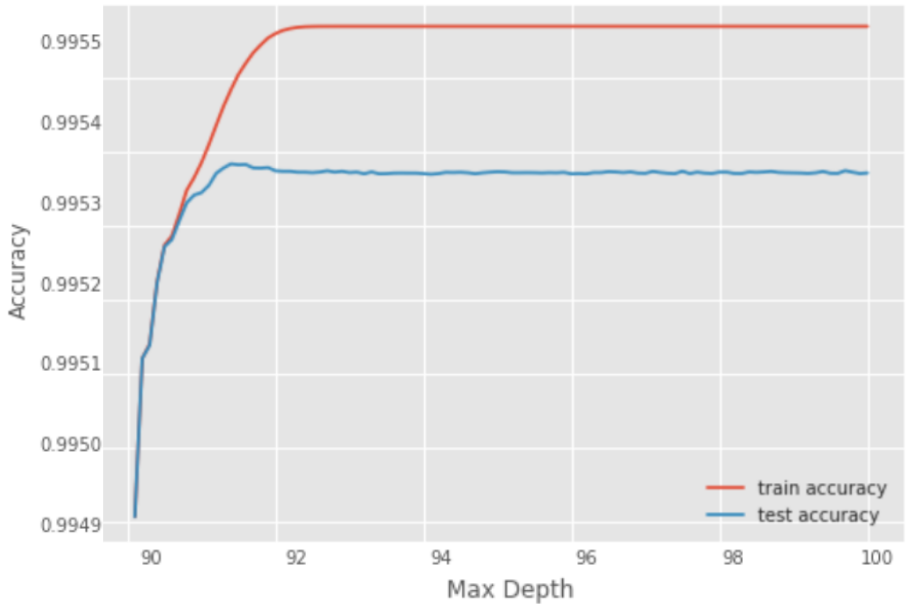

Figure 4.22: Performance of the decision tree classifier with the balanced dataset of 20000 entries.

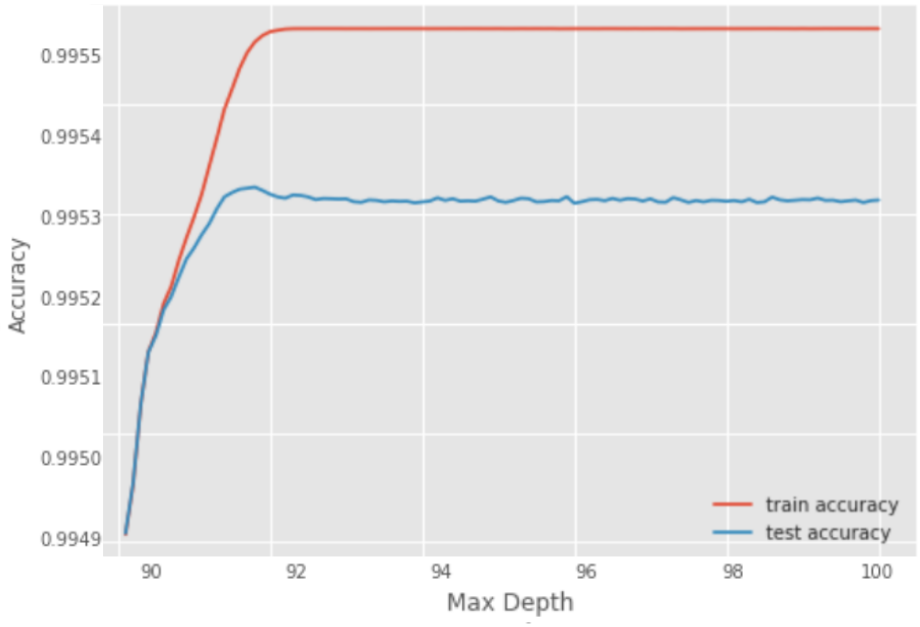

Figure 4.23: Performance of the random forest classifier with the balanced dataset of 20000 entries.

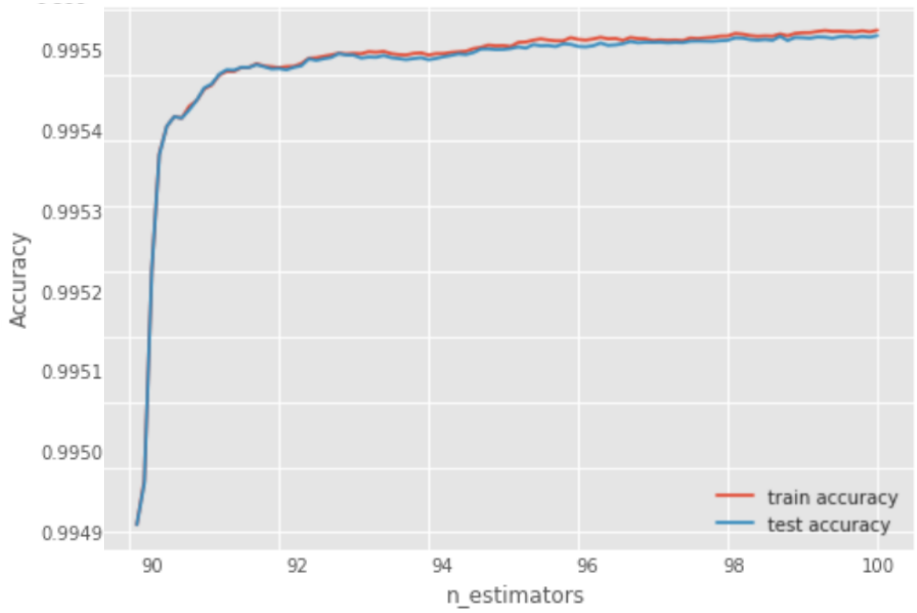

Figure 4.24: Performance of the AdaBoost classifier with the balanced dataset of 20000 entries.

.

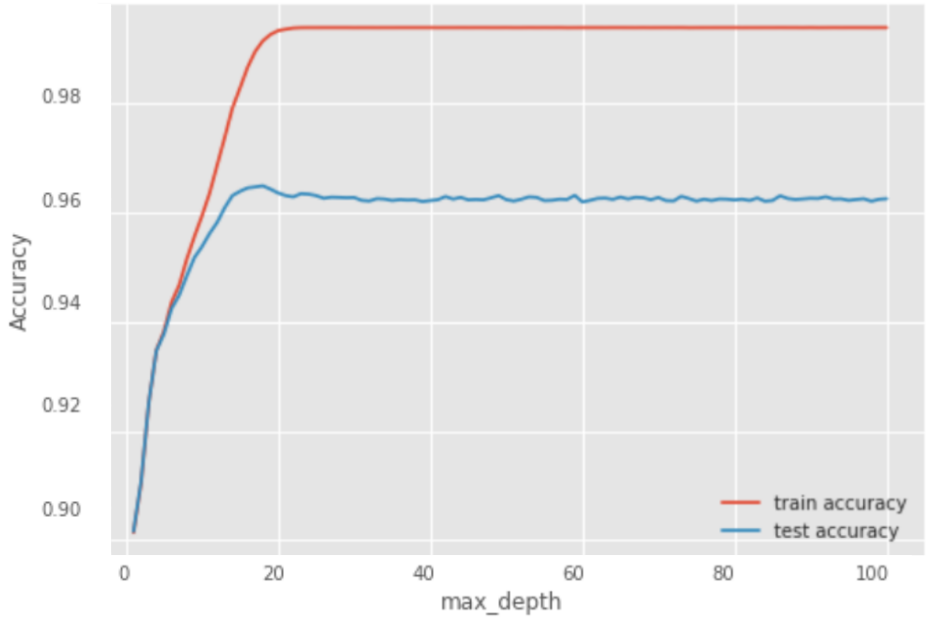

Figure 4.25: Performance of the XGBoost classifier with the balanced dataset of 20000 entries.

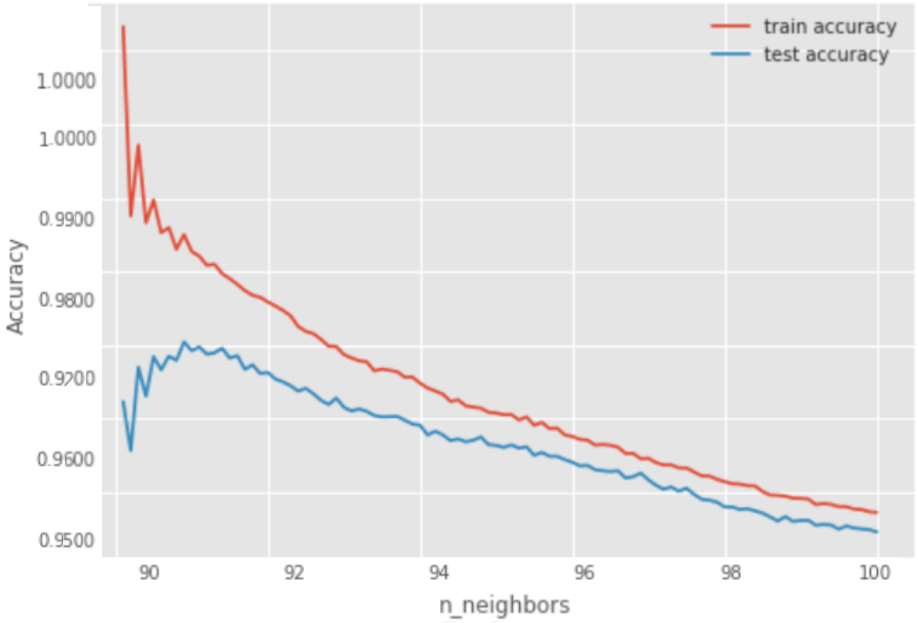

Figure 4.26: Performance of the K-nearest neighbor classifier with the balanced dataset of 20000 entries.

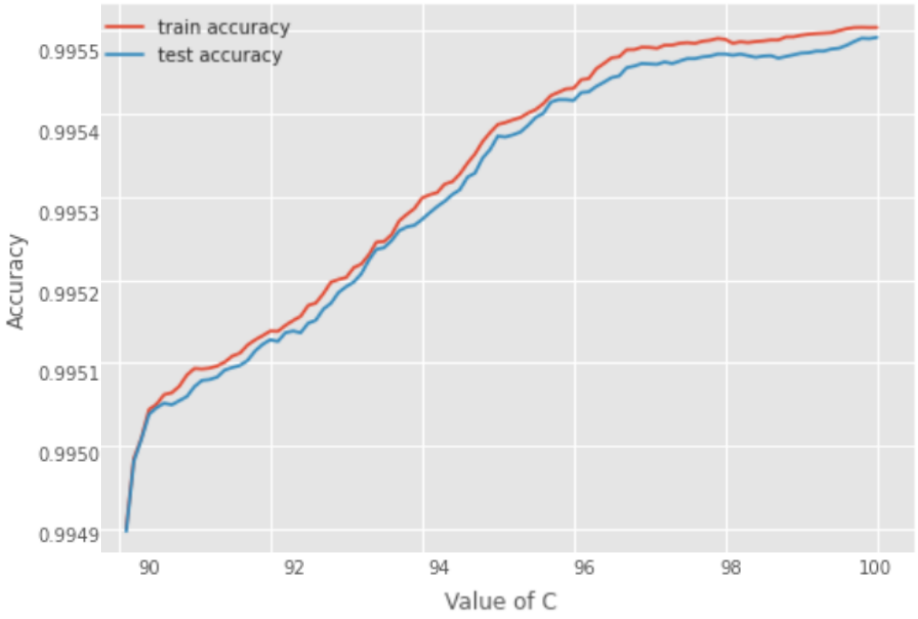

Figure 4.27: Performance of the kernel SVC classifier with the balanced dataset of 20000 entries.

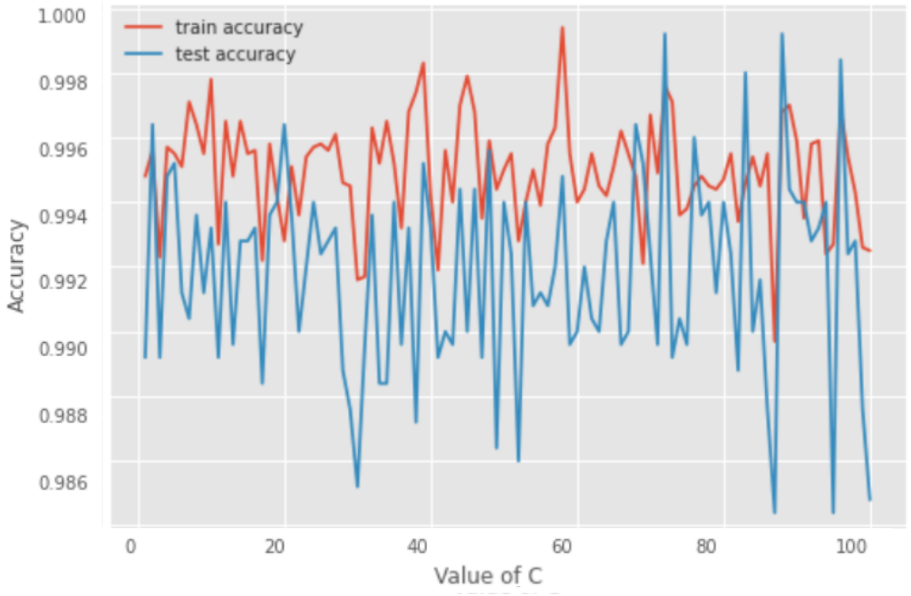

Figure 4.28: Performance of the logistic regression classifier with the balanced dataset of 20000 entries.

Table 4.4 shows the testing results after hyperparameter tuning. Among the seven methods, random forest has the highest accuracy, precision, recall, and f-score at 99.58, 99.82, 99.25, 99.52, followed by XGBoost at 99.47, 99.72, 99.24, 99.45, decision tree at 99.46, 99.61, 99.14, 99.41, AdaBoost at 99.46, 99.61, 99.14, 99.41, logistic regression at 99.36, 99.60, 99.03, 99.30, kernel SVC at 99.25, 99.59, 98.92, 99.29, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37. In terms of execution time, decision tree had the lowest at 16.28 s followed by logistic regression at 85.75 s, AdaBoost at 223.6 s, random forest at 558.1 s, K-nearest neighbor at 1139 s, XGBoost at 1342 s, and kernel SVC at 12694 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.46	99.61	99.14	99.41	16.28
Random Forest	99.58	99.82	99.25	99.52	558.1
AdaBoost	99.46	99.61	99.14	99.41	223.3
XGBoost	99.47	99.72	99.24	99.45	1342
K-nearest Neighbor	96.78	97.33	96.42	97.37	1139
Kernel SVC	99.25	99.59	98.92	99.29	12694
Logistic Regression	99.36	99.60	99.03	99.30	85.75

Table 4.4: Classifier performance with the balanced dataset of 20000 entries.

4.3 Case 3

In this experiment, classification is applied on a dataset of 500 entries with hyperparameter tuning. SMOTE is used to balance the class distribution by randomly replicating minority class instances. The results for balanced and unbalanced datasets are given below.

4.3.1 Classification with the Unbalanced Dataset of 500 Entries

For the classification of the unbalanced dataset, a dataset with an 80:20 split is used so 80% are malicious URLs and 20% are non-malicious URLs. Figure 4.29 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 99.12 with max-depth = 16. Figure 4.30 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features , and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.47 with max-depth = 8. Figure 4.31 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 99.30 with n-estimators as 11. Figure 4.32 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.57 with max-depth = 2. Figure 4.33 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params, and n-jobs were set to 2, none, and none, respec-

tively. The highest accuracy obtained from KNN was 96.78 with n-neighbors = 1. Figure 4.34 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 98.13 with $C = 72$. Figure 4.35 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.27 with $C = 1$.

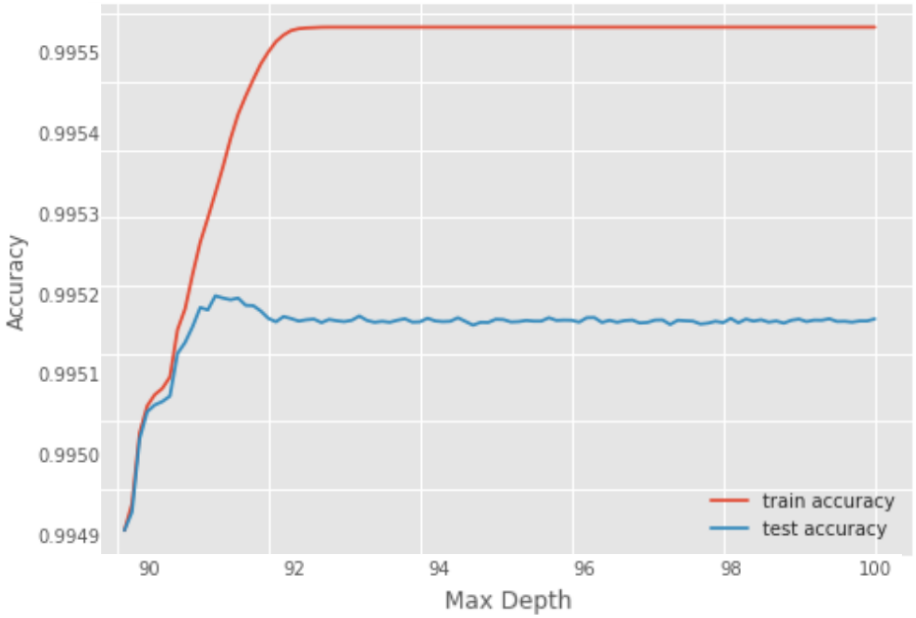

Figure 4.29: Performance of the decision tree classifier with the unbalanced dataset of 500 entries.

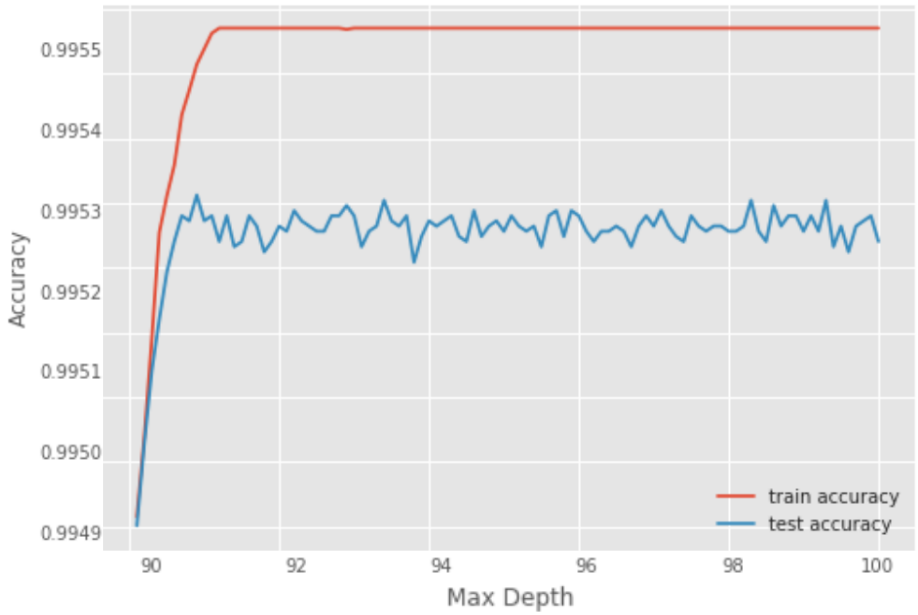

Figure 4.30: Performance of the random forest classifier with the unbalanced dataset of 500 entries.

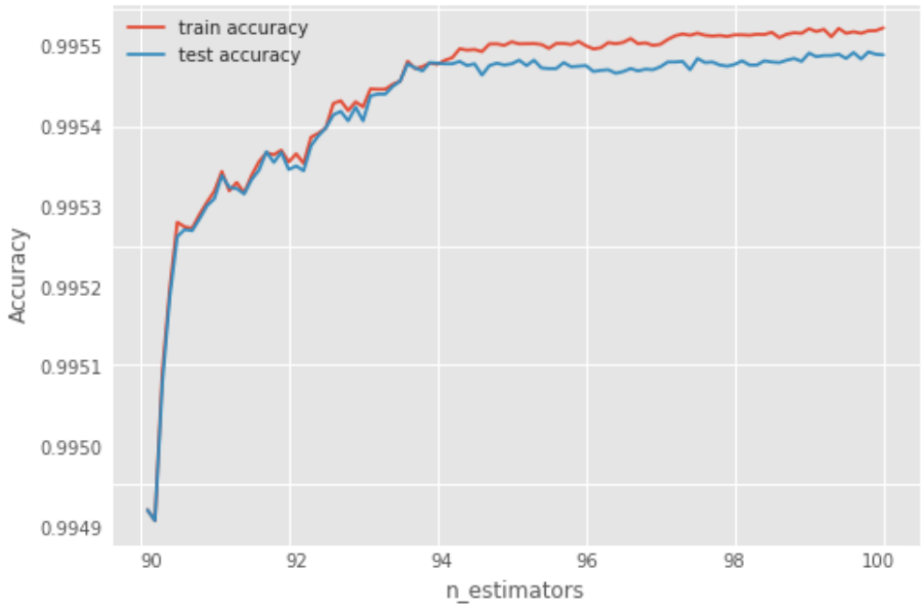

Figure 4.31: Performance of the AdaBoost classifier with the unbalanced dataset of 500 entries.

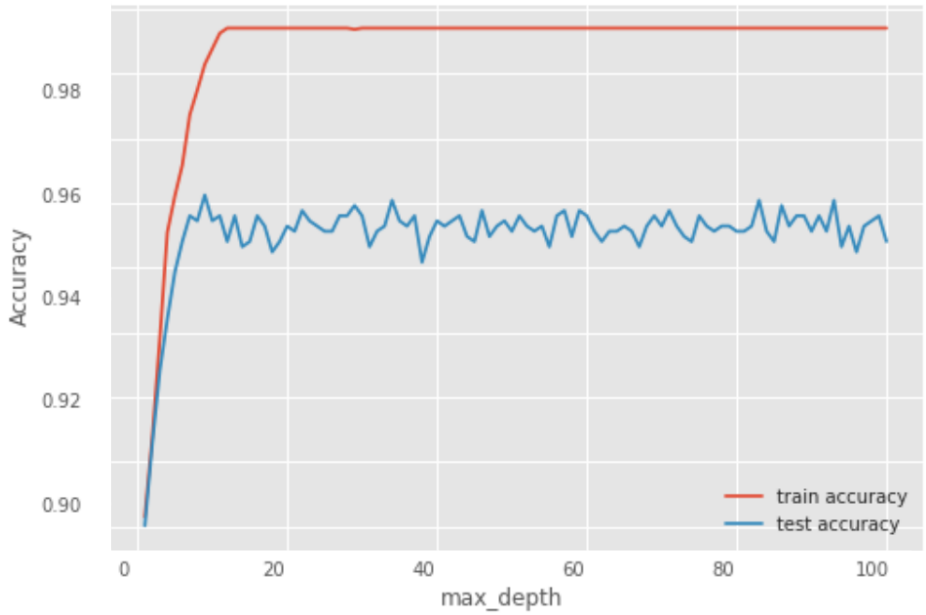

Figure 4.32: Performance of the XGBoost classifier with the unbalanced dataset of 500 entries.

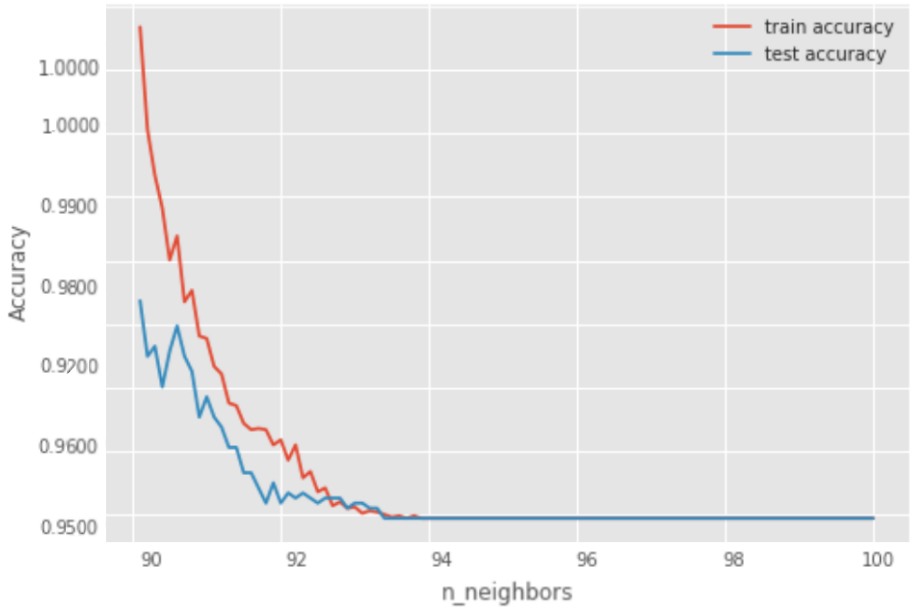

Figure 4.33: Performance of the K-nearest neighbor classifier with the unbalanced dataset of 500 entries.

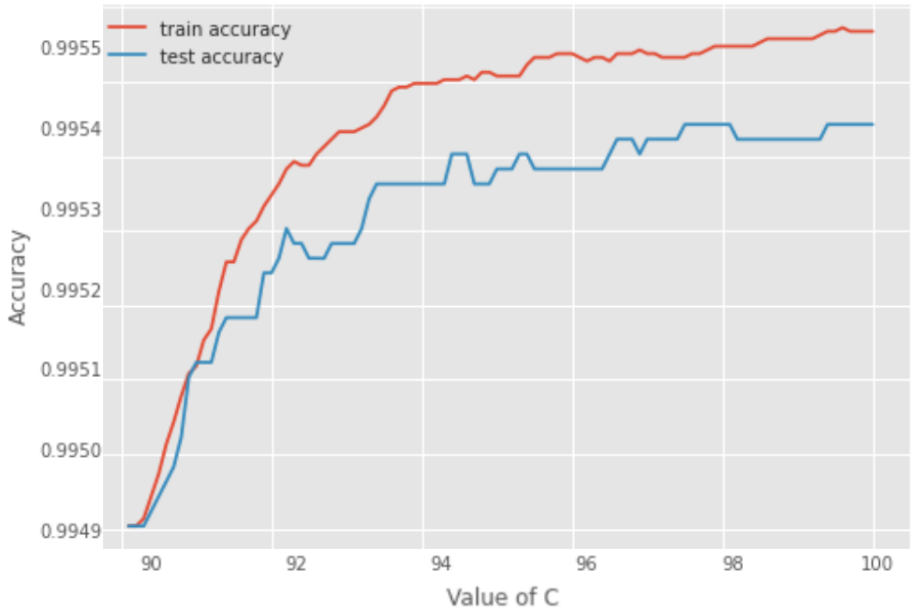

Figure 4.34: Performance of the kernel SVC classifier with the unbalanced dataset of 500 entries.

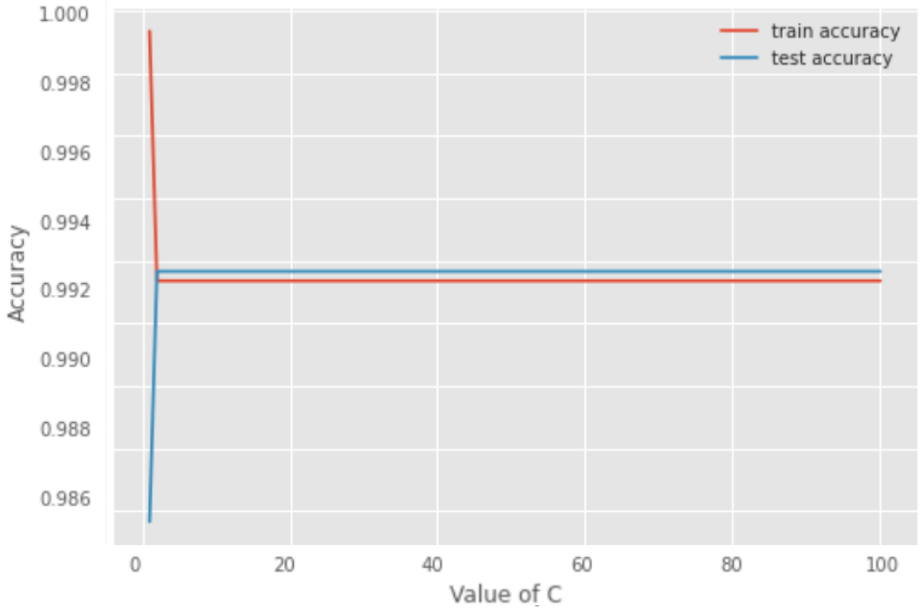

Figure 4.35: Performance of the logistic regression classifier with the unbalanced dataset of 500 entries.

Table 4.5 shows the testing results after hyperparameter tuning. Among the seven methods, XGBoost has the highest accuracy, precision, recall, and f-score at 99.57,

99.70, 98.08, 98.80, followed by random forest at 99.47, 98.70, 98.05, 98.50, AdaBoost at 99.30, 99.01, 98.02, 98.20, logistic regression at 99.27, 98.70, 97.01, 98.10, decision tree at 99.12, 97.20, 98.07, 97.70, kernel SVC at 96.13, 96.30, 93.09, 93.30, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37. In terms of execution time, decision tree had the lowest at 4.210 s followed by kernel SVC at 28.43 s, K-nearest neighbor at 23.70 s, logistic regression at 50.92 s, AdaBoost at 75.24 s, XGBoost at 69.01 s, and random forest at 129.79 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.12	97.20	98.07	97.70	4.210
Random Forest	99.47	98.70	98.05	98.50	129.79
AdaBoost	99.30	99.01	98.02	98.20	75.24
XGBoost	99.57	99.70	98.08	98.80	69.01
K-nearest Neighbor	96.78	97.33	96.42	97.37	23.70
Kernel SVC	96.13	96.30	93.09	93.30	28.43
Logistic Regression	99.27	98.70	97.01	98.10	50.92

Table 4.5: Classifier performance with the unbalanced dataset of 500 entries.

4.3.2 Classification with the Balanced Dataset of 500 Entries

For the classification of the balanced dataset, a dataset with a 50:50 split is used so 50% are malicious URLs and 50% are non-malicious URLs. Figure 4.36 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 99.12 with max-depth = 9. Figure 4.37 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.26 with max-depth = 18. Figure 4.38 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 91.06 with n-estimators as 32. Figure 4.39 shows the performance

of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.13 with max-depth = 8. Figure 4.40 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params, and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 96.78 with n-neighbors = 1. Figure 4.41 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 96.59 with C = 100. Figure 4.42 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.05 with $C = 2$.

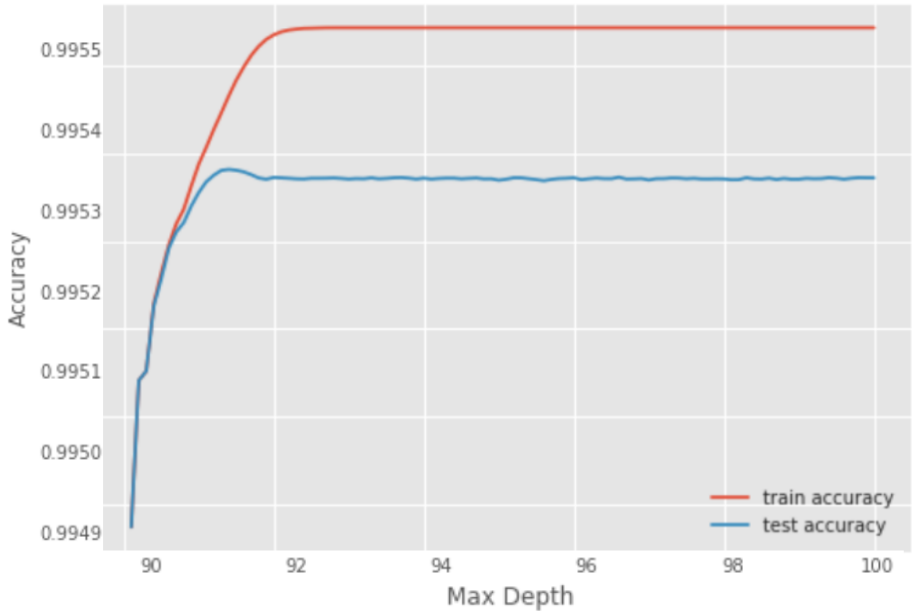

Figure 4.36: Performance of the decision tree classifier with the balanced dataset of 500 entries.

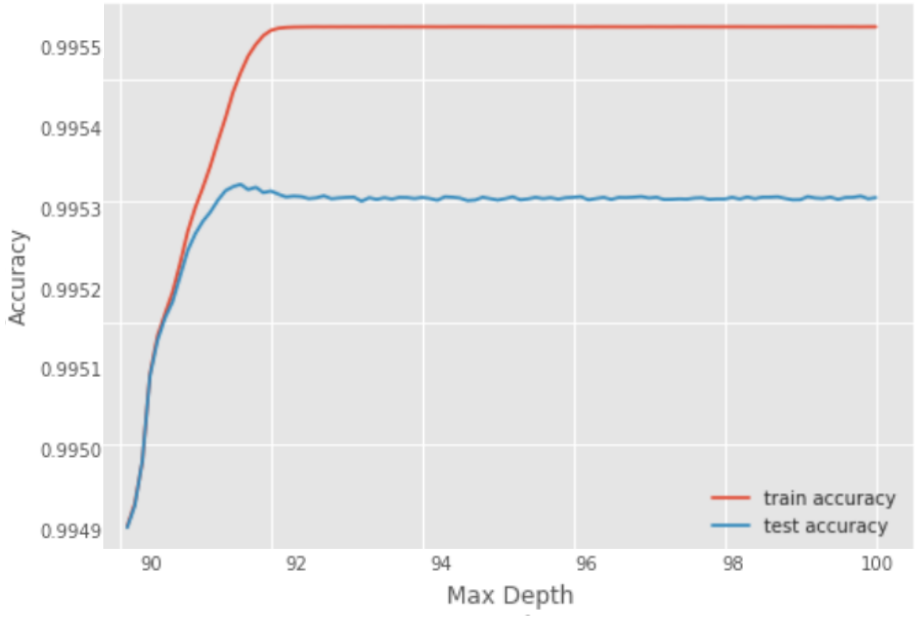

Figure 4.37: Performance of the random forest classifier with the balanced dataset of 500 entries.

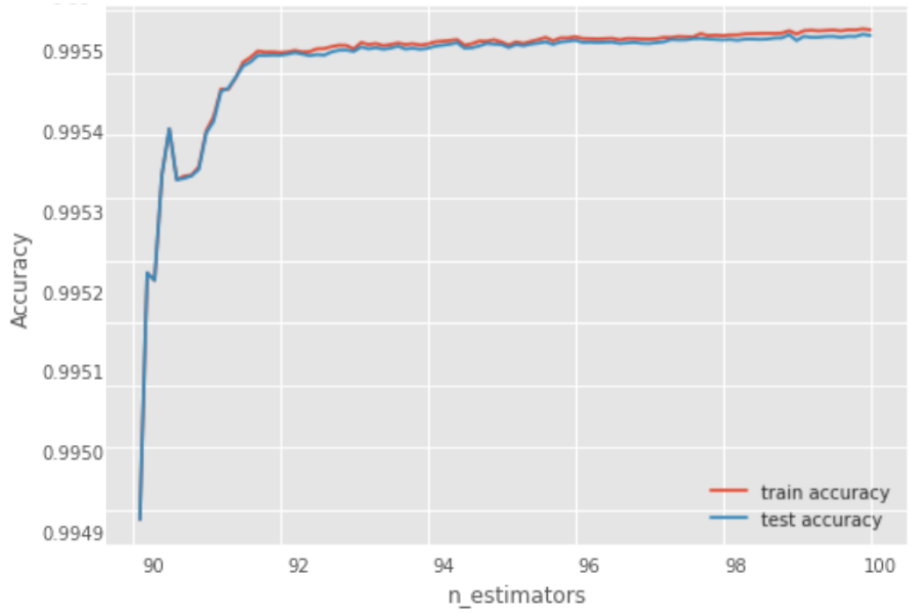

Figure 4.38: Performance of the AdaBoost classifier with the balanced dataset of 500 entries.

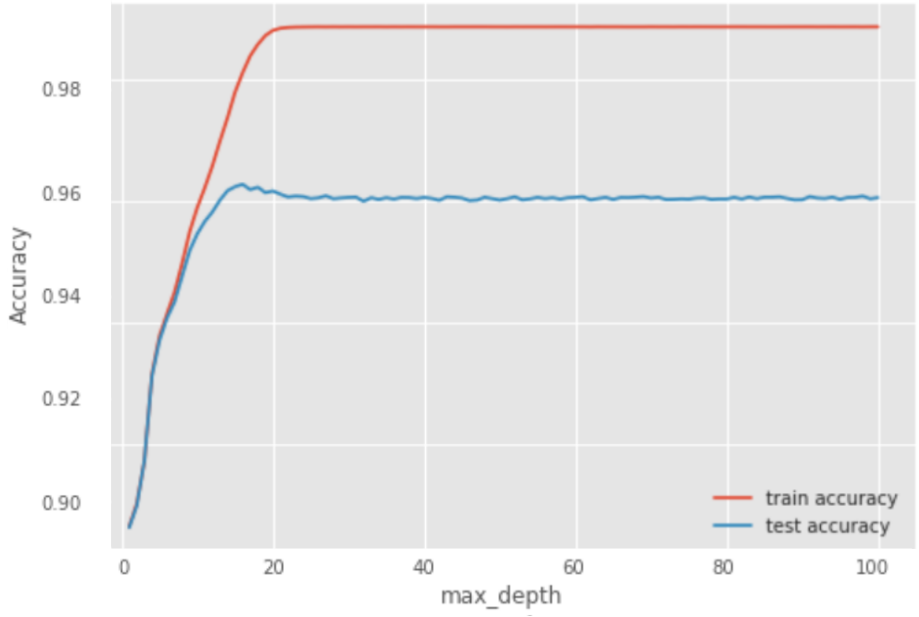

Figure 4.39: Performance of the XGBoost classifier with the balanced dataset of 500 entries.

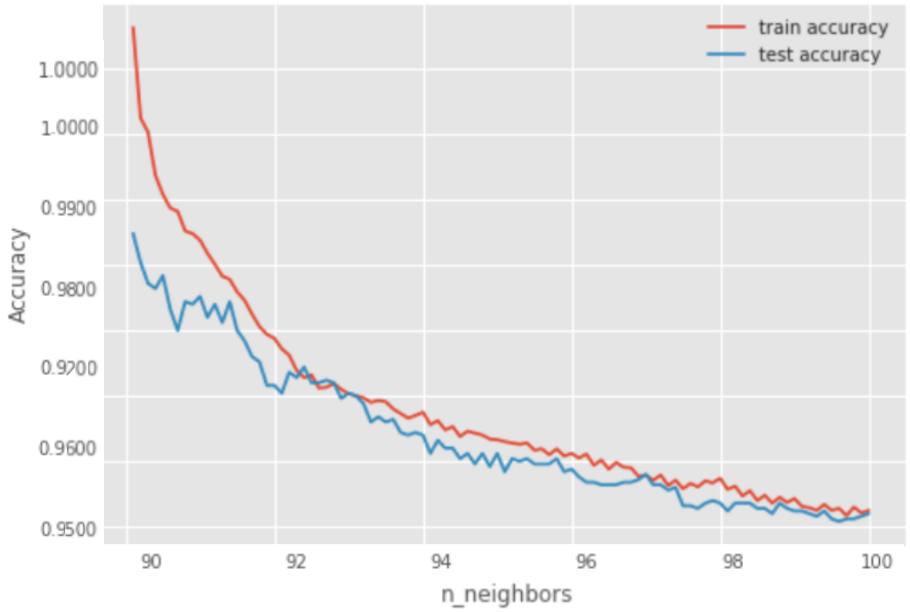

Figure 4.40: Performance of the K-nearest neighbor classifier with the balanced dataset of 500 entries.

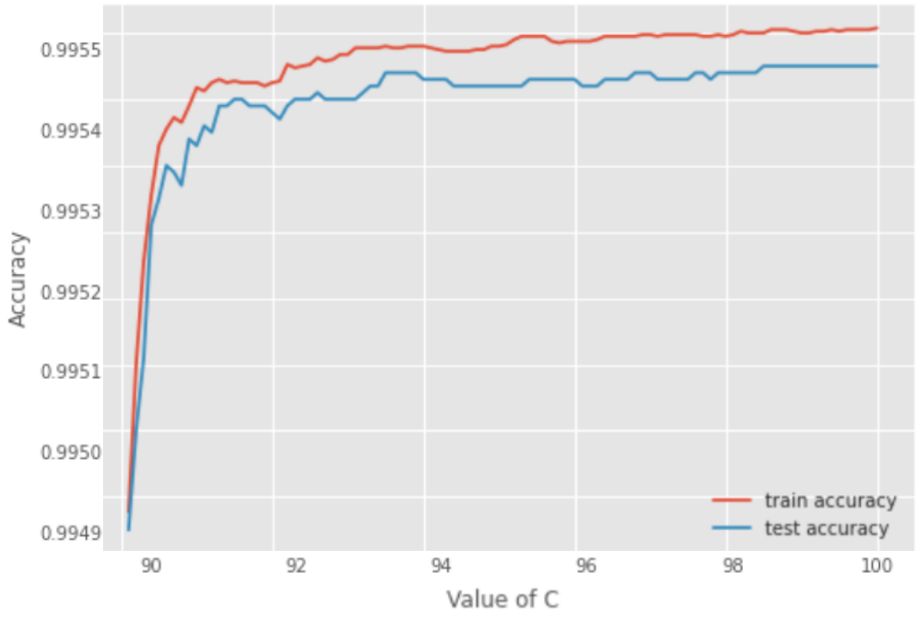

Figure 4.41: Performance of the kernel SVC classifier with the balanced dataset of 500 entries.

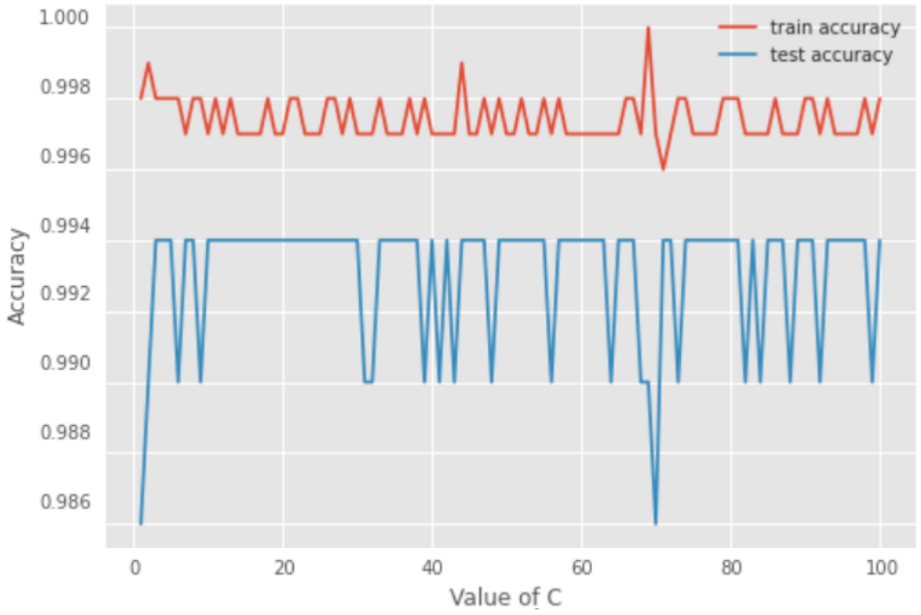

Figure 4.42: Performance of the logistic regression classifier with the balanced dataset of 500 entries.

Table 4.6 shows the testing results after hyperparameter tuning. Among the seven methods, random forest has the highest accuracy, precision, recall, and f-score at

99.26, 99.68, 98.82, 99.28, followed by XGBoost at 99.13, 99.31, 98.81, 99.18, 69.01, decision tree at 91.12, 99.20, 98.07, 98.70, AdaBoost at 99.06, 99.63, 97.34, 98.43, logistic regression at 99.05, 99.57, 97.76, 98.67, kernel SVC at 96.59, 98.90, 94.04, 96.40, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37. In terms of execution time, decision tree had the lowest at 2.210 s followed by kernel SVC at 8.710 s, K-nearest neighbor at 10.16 s, logistic regression at 11.75 s, AdaBoost at 34.59 s, XGBoost at 57.60 s, and random forest at 64.46 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.12	99.20	98.07	98.70	2.210
Random Forest	99.26	99.68	98.82	99.28	64.46
AdaBoost	99.06	99.63	97.34	98.43	34.59
XGBoost	99.13	99.31	98.81	99.18	57.60
K-nearest Neighbor	96.78	97.33	96.42	97.37	10.16
Kernel SVC	96.59	98.90	94.04	96.40	8.710
Logistic Regression	99.05	99.57	97.76	98.67	11.75

Table 4.6: Classifier performance with the balanced dataset of 500 entries.

4.4 Case 4

In this experiment, classification is applied on a dataset of 200 entries with hyperparameter tuning. SMOTE is used to balance the class distribution by randomly replicating minority class instances. The results for balanced and unbalanced datasets are given below.

4.4.1 Classification with the Unbalanced Dataset of 200 Entries

For the classification of the unbalanced dataset, a dataset with an 80:20 split is used so 80% are malicious URLs and 20% are non-malicious URLs. Figure 4.43 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 99.50 with max-depth = 16. Figure 4.44 shows the performance of the random

forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 99.50 with max-depth = 8. Figure 4.45 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 99.50 with n-estimators = 11. Figure 4.46 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.50 with max-depth = 2. Figure 4.47 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params, and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 96.78 with n-neighbors = 1. Figure 4.48 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 96.58 with C = 72. Figure 4.49 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 99.53 with $C = 1$.

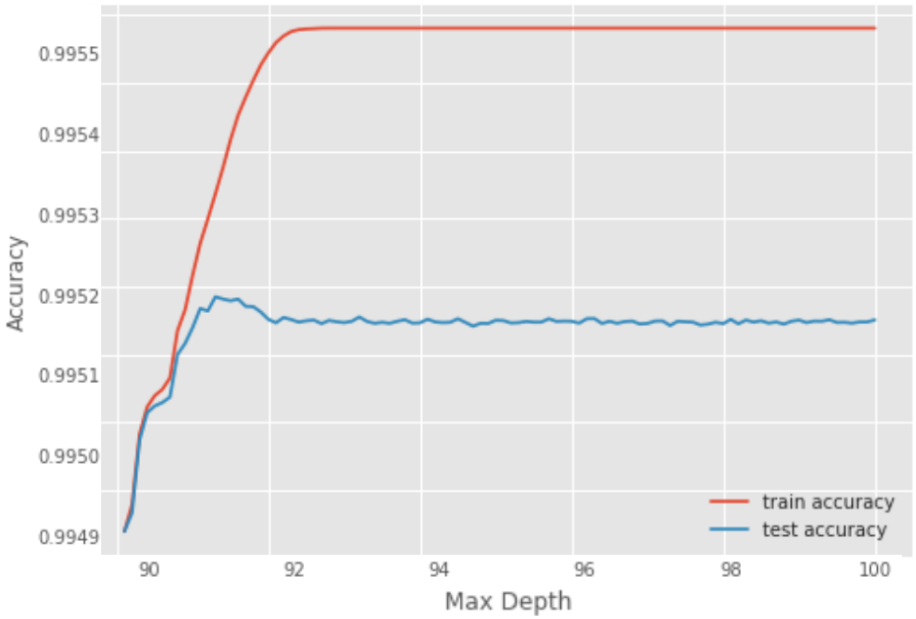

Figure 4.43: Performance of the decision tree classifier with the unbalanced dataset of 200 entries.

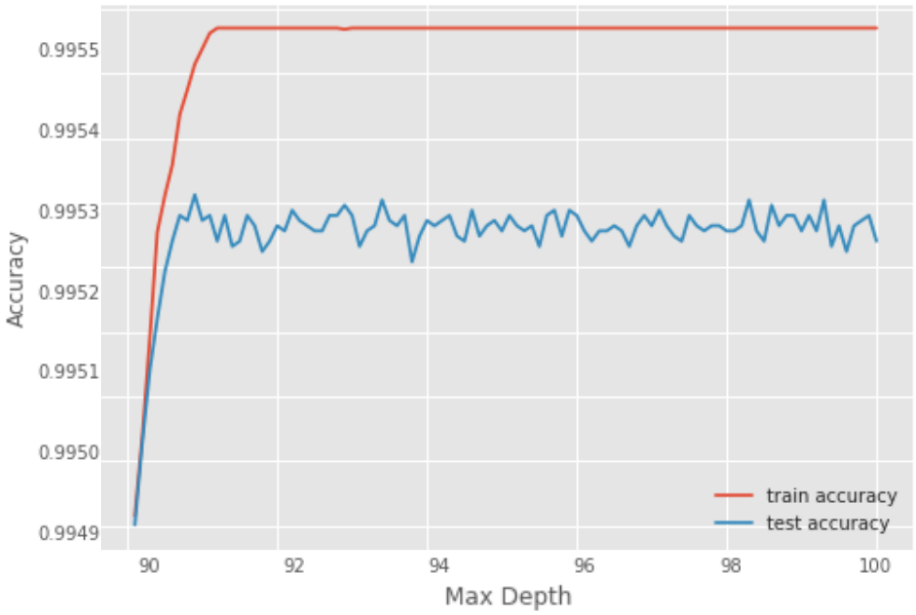

Figure 4.44: Performance of the random forest classifier with the unbalanced dataset of 200 entries.

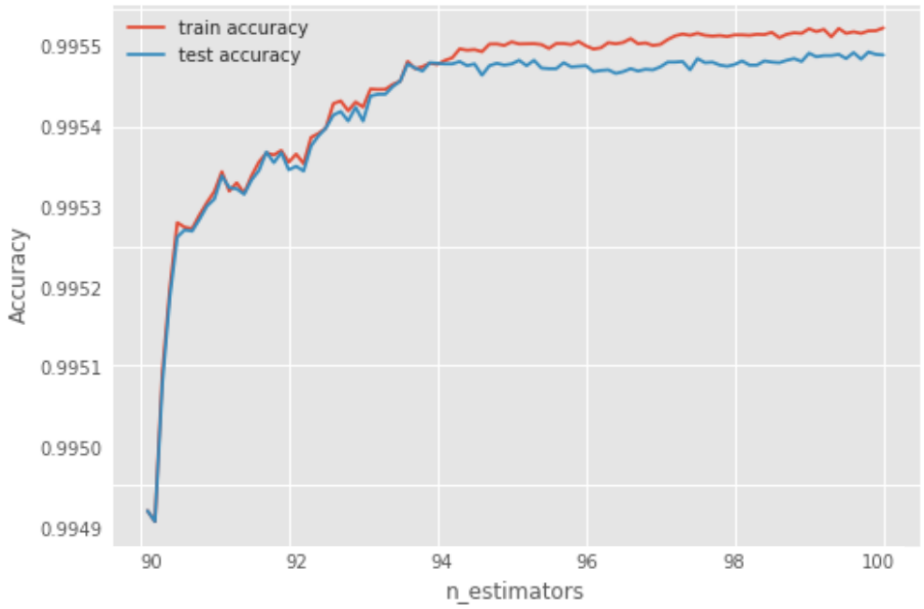

Figure 4.45: Performance of the AdaBoost classifier with the unbalanced dataset of 200 entries.

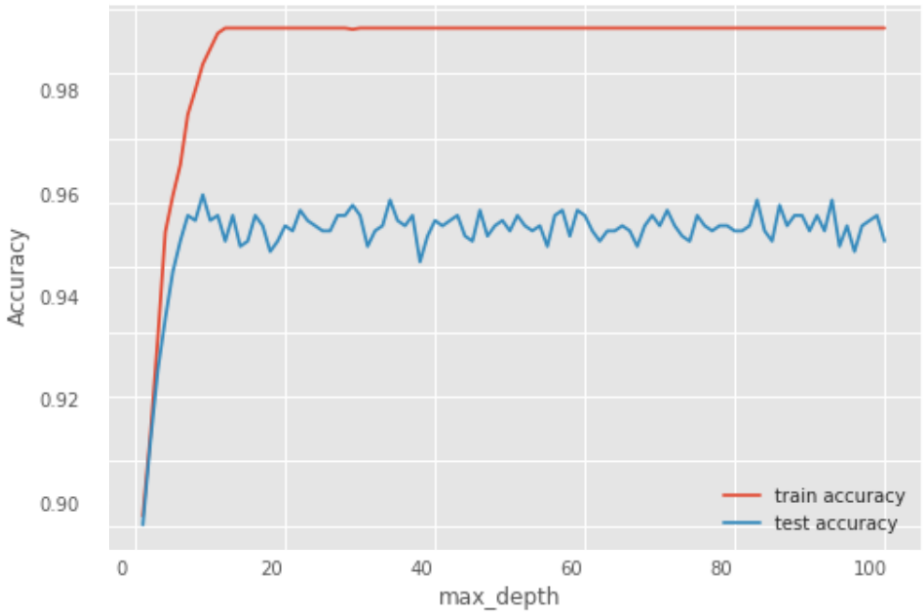

Figure 4.46: Performance of the XGBoost classifier with the unbalanced dataset of 200 entries.

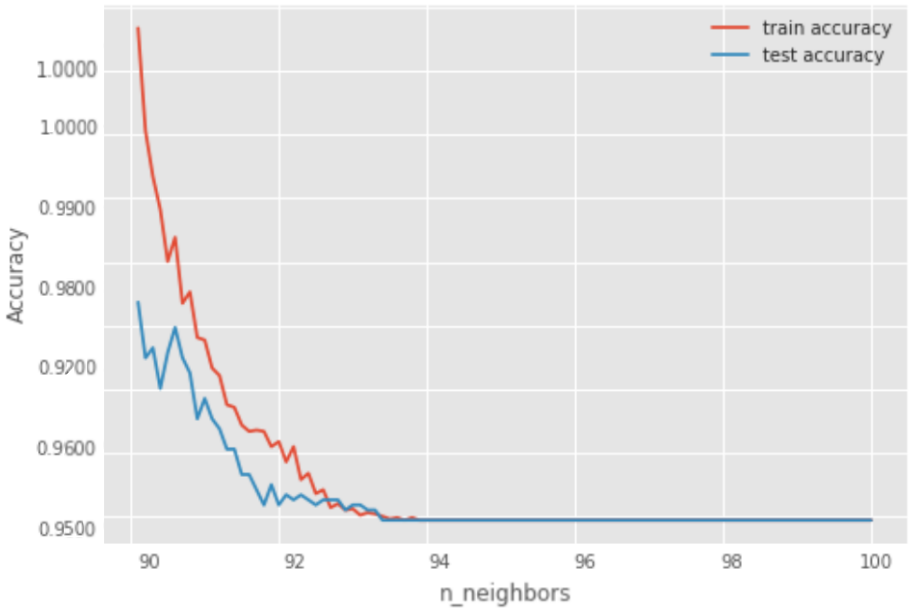

Figure 4.47: Performance of the K-nearest neighbor classifier with the unbalanced dataset of 200 entries.

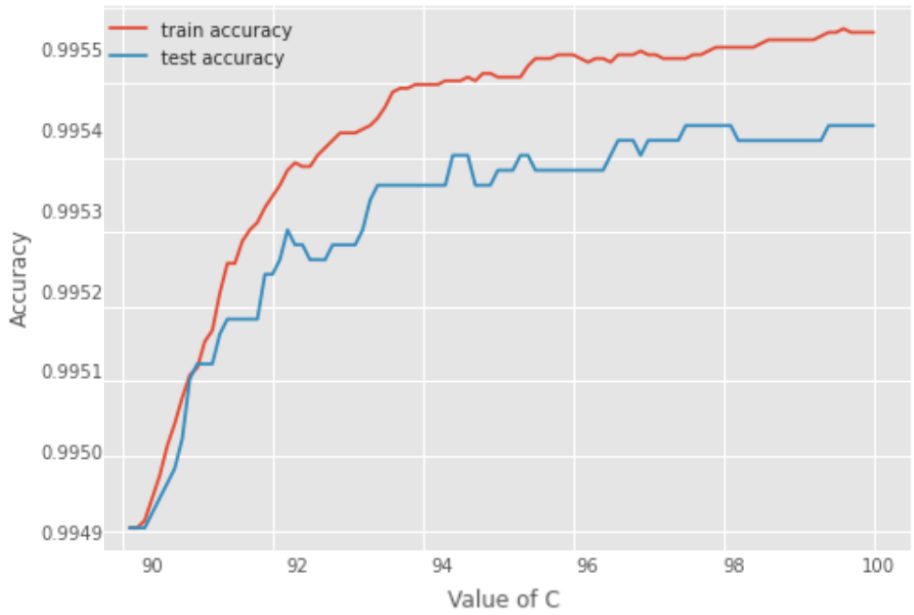

Figure 4.48: Performance of the kernel SVC classifier with the unbalanced dataset of 200 entries.

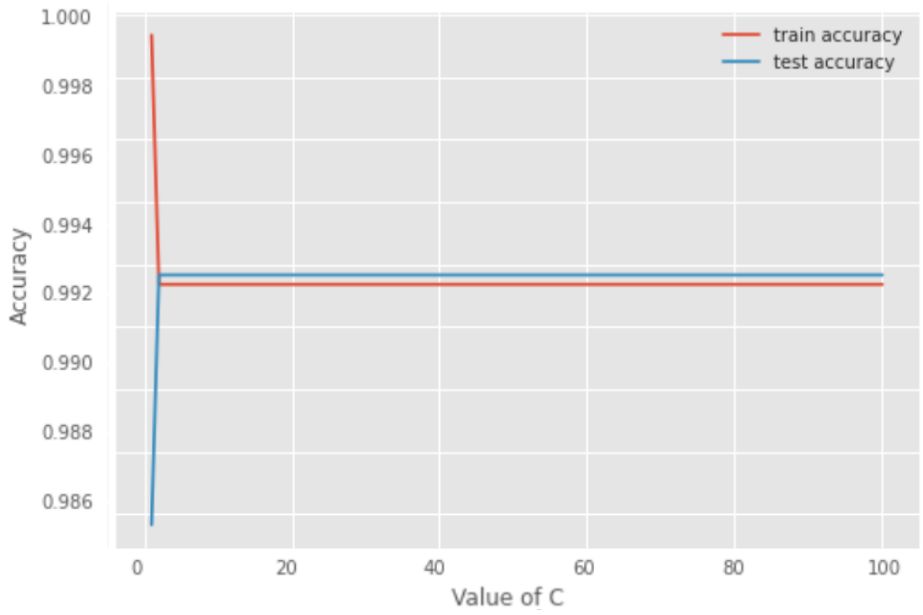

Figure 4.49: Performance of the logistic regression classifier with the unbalanced dataset of 200 entries.

Table 4.7 shows the testing results after hyperparameter tuning. Among the seven methods, random forest has the highest accuracy, precision, recall, and f-score at 99.50, 99.65, 98.37, 98.73, followed by XGBoost at 99.50, 99.56, 97.57, 98.75, decision tree at 99.50, 99.65, 97.57, 98.75, AdaBoost at 99.50, 99.56, 97.57, 98.75, logistic regression at 99.53, 99.38, 95.82, 98.28, kernel SVC at 96.58, 95.88, 93.80, 90.08, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37. In terms of execution time, decision tree had the lowest at 3.210 s followed by kernel SVC at 7.710 s, K-nearest neighbor at 9.16 s, logistic regression at 9.75 s, AdaBoost at 28.59 s, XGBoost at 36.60 s, and random forest at 52.46 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.50	99.65	97.57	98.75	3.210
Random Forest	99.50	99.65	98.37	98.73	52.46
AdaBoost	99.50	99.56	97.57	98.75	28.59
XGBoost	99.50	99.56	97.57	98.75	36.60
K-nearest Neighbor	96.78	97.33	96.42	97.37	9.16
Kernel SVC	96.58	95.88	93.80	90.08	7.710
Logistic Regression	99.53	99.38	95.82	98.28	9.75

Table 4.7: Classifier performance with the unbalanced dataset of 200 entries.

4.4.2 Classification with the Balanced Dataset of 200 Entries

For the classification of the balanced dataset, a dataset with a 50:50 split is used so 50% are malicious URLs and 50% are non-malicious URLs. Figure 4.50 shows the performance of the decision tree classifier with hyperparameter tuning of max-depth. The tuned parameters min-samples-split, min-samples-leaf, and criterion are 2, 1, and Gini, respectively, where Gini is the impurity of the decision tree. The parameters splitter, min-weight-fraction-leaf, max-features, and random-state were set to best, 0, none, and none, respectively. The highest accuracy obtained from decision tree was 96.27 with max-depth = 9. Figure 4.51 shows the performance of the random forest classifier with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and criterion are 100, 2, and Gini, respectively. The parameters min-impurity-decrease, min-samples-leaf, max-features, and random-state were set to 0, 1, sqrt, and none, respectively. The highest accuracy obtained from random forest was 98.80 with max-depth = 18. Figure 4.52 shows the performance of AdaBoost with hyperparameter tuning of n-estimators. The tuned parameters base-estimator, learning-rate, and random-state are none, 1, and none, respectively. The parameter algorithm was set to SAMME.R. The highest accuracy obtained from AdaBoost was 97.70 with n-estimators as 32. Figure 4.53 shows the performance of XGBoost with hyperparameter tuning of max-depth. The tuned parameters n-estimators, min-samples-split, and max-features are 3, 2, and 0, respectively. The parameters subsample, min-weight-fraction-leaf, min-samples-leaf, and random-state were set to 1, 0, 1, and none, respectively. The highest accuracy obtained from XGBoost was 99.01 with max-depth = 8. Figure 4.54 shows the performance of the KNN classifier with hyperparameter tuning of n-neighbors. The tuned parameters weights, leaf-size, and algorithm are uniform, 30, and auto, respectively. The parameters p, metric-params, and n-jobs were set to 2, none, and none, respectively. The highest accuracy obtained from KNN was 96.78 with n-neighbors = 1. Figure 4.55 shows the performance of the kernel SVC classifier with hyperparameter tuning of C. The tuned parameters kernel, random-state, and cache-size are rbf, 0, and 200, respectively. The parameters degree, gamma, class-weight, and cache-size were set to 3, scale, none, and 200, respectively. The highest accuracy obtained from kernel SVC was 93.53 with C = 100. Figure 4.56 shows the performance of the logistic regression classifier with hyperparameter tuning of C. The tuned parameters penalty, random-state, and tol are $l2$, 0, and 10^{-4}, respectively. The parameters class-weight, max-iter, n-jobs, and

fit-intercept were set to none, 100, none, and true, respectively. The highest accuracy obtained from logistic regression was 95.89 with $C = 2$.

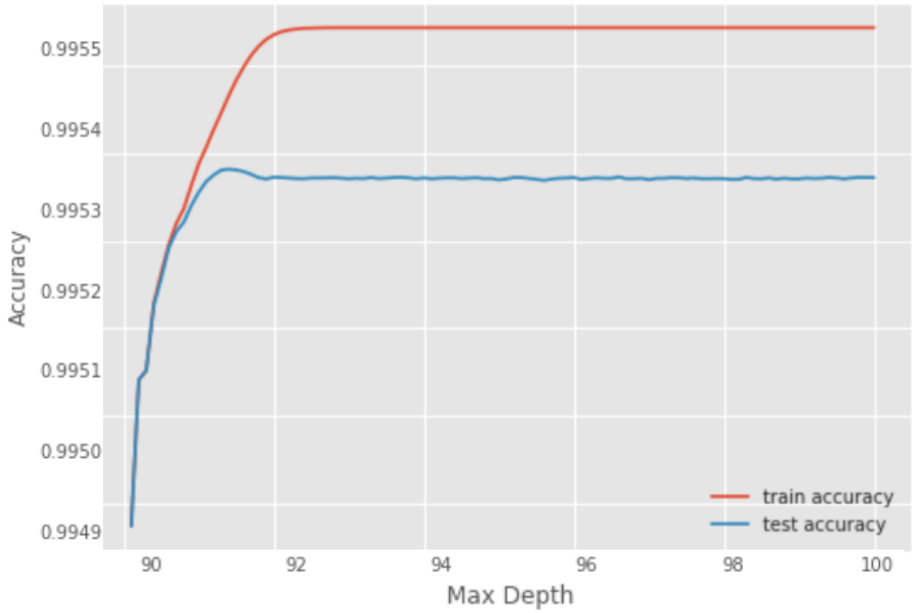

Figure 4.50: Performance of the decision tree classifier with the balanced dataset of 200 entries.

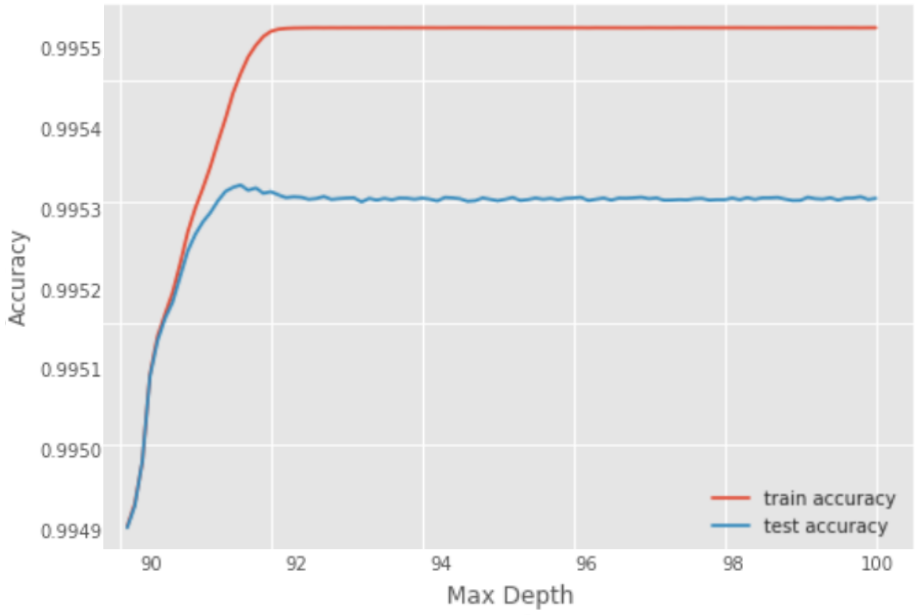

Figure 4.51: Performance of the random forest classifier with the balanced dataset of 200 entries.

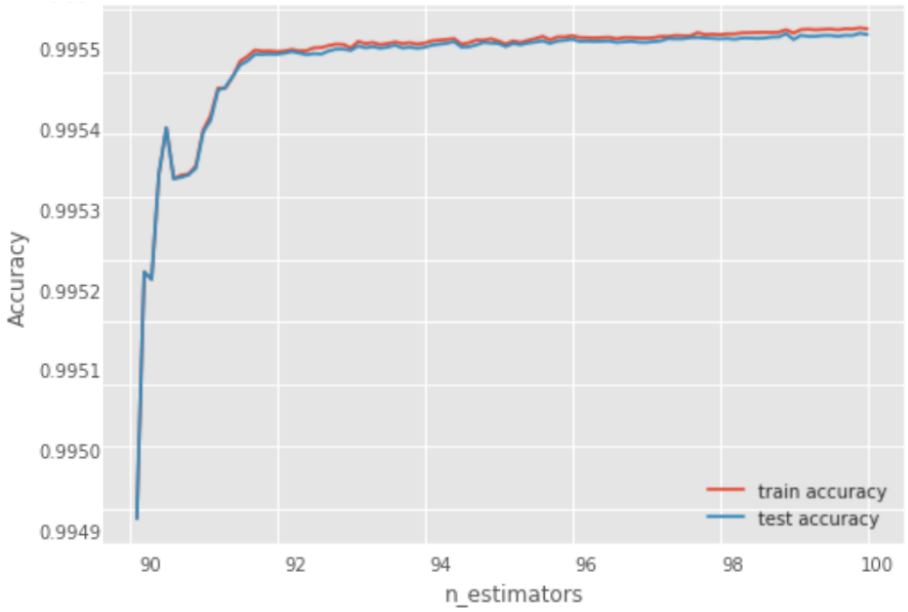

Figure 4.52: Performance of the AdaBoost classifier with the balanced dataset of 200 entries.

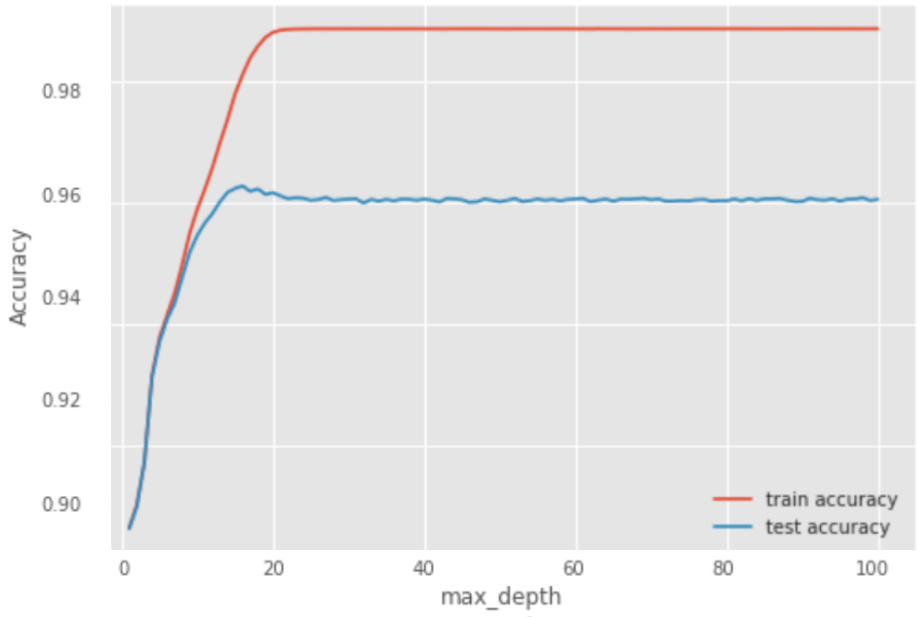

Figure 4.53: Performance of the XGBoost classifier with the balanced dataset of 200 entries.

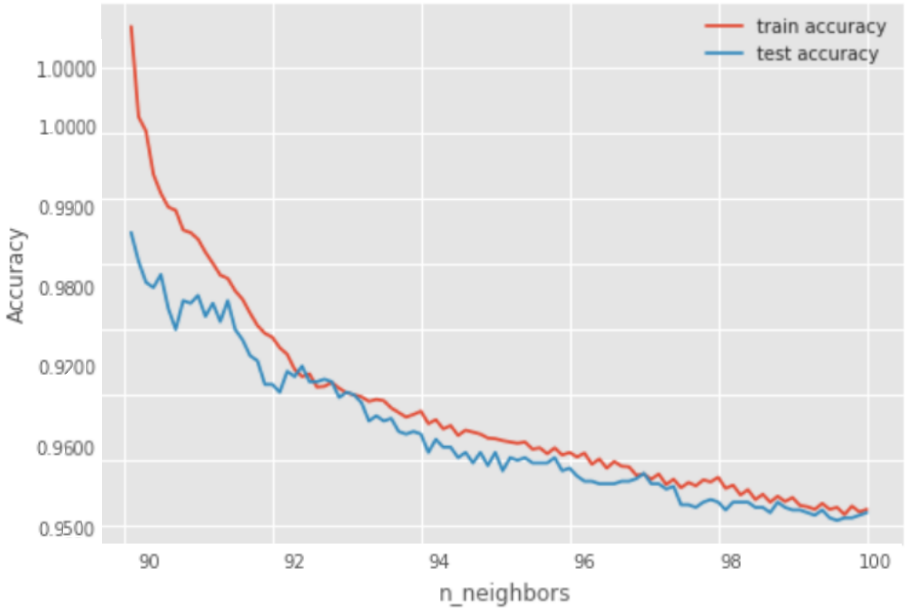

Figure 4.54: Performance of the K-nearest neighbor classifier with the balanced dataset of 200 entries.

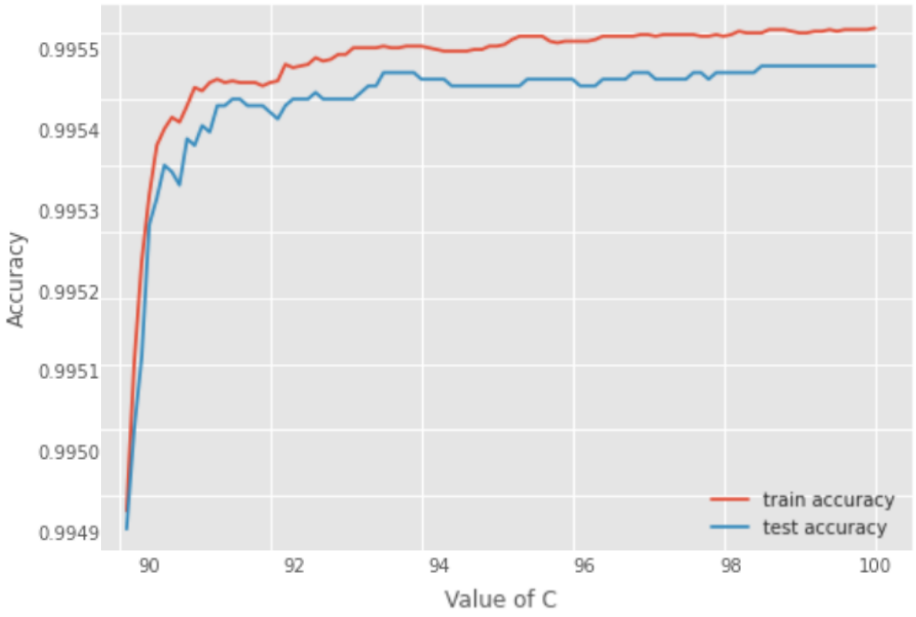

Figure 4.55: Performance of the kernel SVC classifier with the balanced dataset of 200 entries.

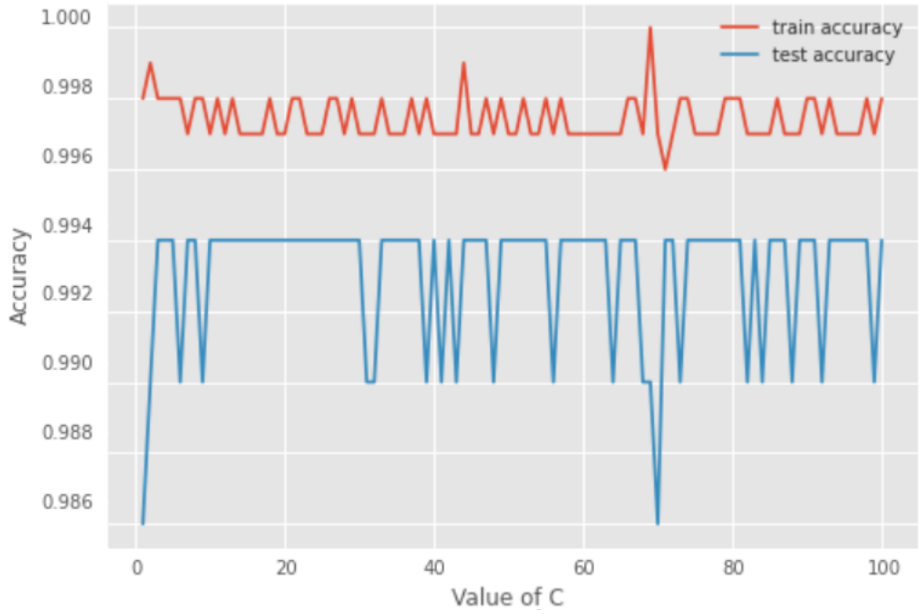

Figure 4.56: Performance of the logistic regression classifier with the balanced dataset of 200 entries.

Table 4.8 shows the testing results after hyperparameter tuning. Among the seven methods, XGBoost has the highest accuracy, precision, recall, and f-score at 99.01, 99.0, 99.01, 99.01, followed by random forest at 98.80, 99.07, 98.70, 99.07, AdaBoost at 98.70, 99.03, 96.35, 97.53, decision tree at 98.27, 96.77, 95.75, 96.57, logistic regression at 97.89, 95.90, 96.09, 95.90, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37, 85.21, kernel SVC at 96.53, 97.37, 91.71, 93.17. In terms of execution time, decision tree had the lowest at 1.210 s followed by kernel SVC at 4.710 s, K-nearest neighbor at 5.160 s, logistic regression at 5.750 s, AdaBoost at 17.59 s, XGBoost at 28.60 s, and random forest at 32.46 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	98.27	96.77	95.75	96.57	1.210
Random Forest	98.80	99.07	98.70	99.07	32.46
AdaBoost	98.70	99.03	96.35	97.53	17.59
XGBoost	99.01	99.01	99.01	99.01	28.60
K-nearest Neighbor	96.78	97.33	96.42	97.37	5.160
Kernel SVC	96.53	97.37	91.71	93.17	4.710
Logistic Regression	97.89	95.90	96.09	95.90	5.750

Table 4.8: Classifier performance with the balanced dataset of 200 entries.

4.5 Case 5

In this experiment, classification is applied on a dataset of 200 entries with hyperparameter tuning. A dataset with an 80:20 split is used so 80% are non-malicious URLs and 20% are malicious URLs. SMOTE is used to balance the class distribution by randomly replicating minority class instances.

4.5.1 Classification with the Unbalanced Dataset of 200 Entries

Table 4.9 shows the testing results after hyperparameter tuning. Among the seven methods, random forest has the highest accuracy, precision, recall, and f-score at 99.61, 99.68, 98.82, 99.28, followed by XGBoost at 99.54, 99.31, 98.81, 99.18, 69.01, decision tree at 99.50, 99.20, 98.07, 98.70, AdaBoost at 99.48, 99.63, 97.34, 98.43, logistic regression at 99.53, 99.57, 97.76, 98.67, kernel SVC at 96.21, 98.90, 94.04, 96.40, K-nearest neighbor at 96.78, 97.33, 96.42, 97.37. In terms of execution time, decision tree had the lowest at 2.210 s followed by kernel SVC at 7.950 s, K-nearest neighbor at 9.24 s, logistic regression at 9.01 s, AdaBoost at 28.79 s, XGBoost at 36.72 s, and random forest at 51.49 s.

Classifier	Accuracy (%)	Precision (%)	Recall (%)	f-score (%)	Execution Time (s)
Decision Tree	99.50	99.20	98.07	98.70	2.210
Random Forest	99.61	99.68	98.82	99.28	51.49
AdaBoost	99.48	99.63	97.34	98.43	28.79
XGBoost	99.54	99.31	98.81	99.18	36.72
K-nearest Neighbor	96.78	97.33	96.42	97.37	9.24
Kernel SVC	96.21	98.90	94.04	96.40	7.950
Logistic Regression	99.53	99.57	97.76	98.67	9.01

Table 4.9: Classifier performance with the unbalanced dataset of 200 entries.

Chapter 5

Conclusion

This research considered the problem of malicious URL detection and the effect of dataset balance on machine learning classifiers. Both machine learning and ensemble methods were considered. A dataset from Kaggle was used for the experiments with k-fold cross validation and hyperparameter tuning. Accuracy, precision, recall, f-score, and execution time were employed as performance evaluation metrics. Datasets of sizes 35000, 20000, 500, and 200 were considered.

The unbalanced dataset with 80:20 split has 80% malicious URLs and 20% non-malicious URLs. Random forest has the highest accuracy with 99.86, 99.75, 99.47, and 99.50 for dataset sizes 35000, 20000, 500, and 200, respectively, followed by XGBoost with 99.75, 99.75, 99.57, and 99.13, and decision tree with 99.74, 99.62, 99.12, and 99.50. This shows that decreasing the size of the dataset does not affect the performance of classification methods with hyperparameter tuning as the difference is negligible. Decision tree had the lowest execution time at 36.14 s, 32.20 s, 4.21 s, and 2.21 s for dataset sizes 35000, 20000, 500, and 200 respectively, followed by logistic regression at 146.56 s, 101.44 s, 50.92 s, and 25.92 s and AdaBoost at 676.05 s, 600.45 s, 75.24 s, and 34.24 s. Random forest which has the highest accuracy took 1158.23 s, 1179.86 s, 129.79 s, and 63.79 s which is considerably higher than decision tree while the difference in accuracy is negligible. Therefore, for unbalanced datasets, decision tree provides the best performance.

The balanced dataset with 50:50 split has 50% malicious URLs and 50% non-malicious URLs. Random forest has the highest accuracy with 99.58, 99.58, 99.26, and 98.80 for dataset sizes 35000, 20000, 500, and 200, respectively, followed by XG-

Boost with 99.68, 99.47, 99.13, and 99.01, and decision tree with 99.58, 99.46, 99.12, and 98.27. This shows that decreasing the size of the dataset does not affect the performance of classification methods with hyperparameter tuning as the difference is negligible. Decision tree had the lowest execution time at 23.55 s, 16.28 s, 2.21 s, and 3.21 s for dataset sizes 35000, 20000, 500, and 200, respectively, followed by logistic regression at 101.95 s, 85.75 s, 11.75 s, and 9.75 s and AdaBoost at 285.03 s, 223.25 s, 34.59 s, and 28.59 s. Random forest which has the highest accuracy took 928.39 s, 558.11 s, 64.46 s, and 52.46 s which is considerably higher than decision tree while the difference in accuracy is negligible. Therefore, for balanced datasets, decision tree provides the best performance.

Research has shown that classification methods have similar accuracy but different execution times when using unbalanced and balanced datasets. Random forest has the highest accuracy with 99.86 and 99.58 at 1558 s and 928.3 s for unbalanced and balanced dataset of 35000 entries, respectively, followed by XGBoost with 99.75 and 99.68 at 2080 s and 1559 s, decision tree with 99.74 and 9.58 at 36.14 s and 23.55 s, logistic regression with 99.68 and 99.48 at 146.6 s and 101.9 s, AdaBoost with 99.63 and 99.58 at 676.1 s and 285.1 s, kernel SVC with 99.59 and 99.37 at 59311 s and 33445 s, and K-nearest neighbor with 97.63 and 96.78 at 8338 s and 1035 s. The difference in execution time between balanced and unbalanced datasets is because of SMOTE which is used to balance the class distribution by randomly replicating minority class instances. In this research, the accuracy with random forest was increased from 96.28 to 99.86 for random forest compared to [5]. The dataset used in [5] was binary data obtained from multiple sources.

5.1 Future Work

In this thesis, different dataset sizes were used to determine the performance of supervised classification algorithms. Unsupervised algorithms can be considered in the future as well as other supervised algorithms. Different training strategies can also be examined.

Bibliography

[1] Gatefy. What is a Malicious URL and How to Block It? https://gatefy.com/blog/what-malicious-url. 18, 3, 2021.

[2] Nadeem, U.B. Nine Different Types of Cybersecurity Attacks You Need to Watch Out For. https://www.elmens.com/tech/9-different-types-of-cybersecurity-attacks-you-need-to-watch-out-for. 11, 3, 2020.

[3] Chan, P.K. and Lippmann, R.P. Machine Learning for Computer Security: Journal of Machine Learning Research. Vol. 7, pp. 2669-2672, 2006.

[4] Kolter, J.Z. and Maloof, M.A. Learning to Detect and Classify Malicious Executables in the Wild: Journal of Machine Learning Research. Vol. 7, pp. 2721-2714, 2006.

[5] Xuan, C.D. and Nguyen, H.D. Malicious URL Detection based on Machine Learning: International Journal of Advanced Computer Science and Applications, Vol. 11, No. 1, 2020

[6] Singhal, P, and Raul, N. Malware Detection Module Using Machine Learning Algorithms to Assist in Centralised Security in Enterprise Networks: International Journal of Network Security and its Application. Vol. 4, No. 1, pp. 61-67, 31, 1, 2012.

[7] Alazab, M., Venkatraman, S., Watters, P., and Alazab, M. Zero-day Malware Detection Based on Supervised Learning Algorithms of API Call Signatures: Proceedings of the Australasian Data Mining Conference. Vol. 121, pp. 171-182, Ballarat, VIC, Australia, 1, 12, 2011.

[8] Chouchane, M.R., Walenstein, A. and Lakhotia, A. Using Markov Chains to Filter Machine-morphed Variants of Malicious Programs: Proceedings of the In-

ternational Conference on Malicious and Unwanted Software. pp. 77-84, Fairfax, VA, USA, 2, 12, 2008.

[9] McGrath, D.K., and Gupta, M. Behind Phishing: An Examination of Phisher Modi Operandi: Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats. San Francisco, CA, USA, 15, 4, 2008

[10] Ma, J., Saul, L.K., Savage, S., and Voelker, G.M. Identifying Suspicious URLs: An Application of Large-Scale Online Learning: Proceedings of the International Conference on Machine Learning. pp. 681-668, Montreal, QC, Canada, 14, 6, 2009.

[11] Pyram, J. 9 Parts of a URL You Should Know. 7, 10, 2020. https://medium.com/@joseph.pyram/9-parts-of-a-url-that-you-should-know-89fea8e11713.

[12] Chandran, S. How Spammers Conduct Mass URL Spam Attacks. https://www.datavisor.com/blog/attack-technique-how-attackers-use-link-shorteners-to-spread-url-spam.

[13] Arctic Wolf. 8 Most Common Types of Malware Attacks. 20, 8, 2018. https://arcticwolf.com/resources/blog/8-types-of-malware. 21, 10, 2021.

[14] Rijnetu, I. Security Alert: Don't Click This Link Spreading on Facebook Messenger. 25, 8, 2017. https://heimdalsecurity.com/blog/security-alert-facebook-messenger-adware.

[15] Fruhlinger, J. What is Phishing? Examples, Types, and Techniques. 12, 4, 2022. https://www.csoonline.com/article/2117843/what-is-phishing-examples-types-and-techniques.html.

[16] IBM Cloud Education. What is Machine Learning? 15, 7, 2020. https://www.ibm.com/cloud/learn/machine-learning.

[17] E. Alpaydin. Introduction to Machine Learning, 4th Edition. Cambridge, MA, USA, MIT Press, 2020.

[18] Kumar, S. Detect Malicious URL Using ML. 24, 6, 2019. https://www.kaggle.com/code/siddharthkumar25/detect-malicious-url-using-ml/data.

[19] Singh, S. Deconfusing the Confusion Matrix. 15, 6, 2020. https://medium.com/analytics-vidhya/deconfusing-the-confusion-matrix-81dba9679981.