MITIGATING TRAFFIC FINGERPRINTING IN TOR: ANALYZING MODIFIED TTRAFFIC'S IMPACT ON ONION SERVICES

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

MASTER OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

by

SINDHU GAYATHRI TUNUGUNTLA (20MIS7007)
SANJAY GADDE (20MIS7004)
GNANITHA JETTI (20MIS7064)

Under the Guidance of

Prof.Asish Kumar Dalai

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

AUGUST 2024

CERTIFICATE

This is to certify that the Capstone Project work titled "**MITIGATING TRAFFIC FINGERPRINTING IN TOR: ANALYZING MODIFIED TRAFFIC'S IMPACT ON ONION SERVICES**" that is being submitted by **Sindhu Gayathri Tunuguntla (20MIS7007), Sanjay Gadde (20MIS7004), Gnanitha Jetti (20MIS7064)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of Bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have they been submitted to any other Institute or University for the award of any degree or diploma and the same is certified.

Prof.Asish Kumar Dalai
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR **DEAN**
Integrated M. Tech SE School of Computer Science and Engineering

ACKNOWLEDGEMENT

It is my pleasure to express with a deep sense of gratitude to **Prof. Asish Kumar Dalai**, Assistant Director Website for his constant guidance, continual encouragement, and understanding; More than all, he taught me patience in my endeavor throughout the course of the project work. The success and outcome of this project required a lot of guidance and assistance, and we are privileged to have this all through our project's completion. Sir took a keen interest in my project work and guided me until the completion of our project work by providing all the necessary information for developing a good system.

I would like to express my gratitude to **G. Viswanathan** (Chancellor), **Dr. S. V. Kota Reddy** (Vice Chancellor), and **Dr. Saroj Kumar Panigrahy** (Associate - DEAN School of Computer Science and Engineering) for providing with an environment to work in and for his inspiration during the course's tenure.

In jubilant mood I express ingeniously my whole-hearted thanks to all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. Last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati — Sindhu Gayathri Tunuguntla (20MIS7007)

Date: 30-08-2024. — **Name of the Student**

ABSTRACT

Classifying network traffic is important for traffic shaping and monitoring. In the last two decades, with the emergence of privacy concerns, the importance of privacy-preserving technologies has risen. The Tor network, which provides anonymity to its users and supports anonymous services known as Onion Services, is a trendy way to achieve online anonymity. However, this anonymity (especially with Onion Services) is frequently misused, encouraging governments and law enforcement agencies to de-anonymize them. Therefore, in this paper, we try to identify the classifiable of Onion Service traffic, focusing on three main contributions. First, we try to identify Onion Service traffic from other Tor traffic. The techniques we have used can identify Onion Service traffic with >99% accuracy. However, there are several modifications that can be made to the Tor traffic to obfuscate its information leakage. In our second contribution, we evaluate how our techniques perform when such modifications have been made to the Tor traffic. Our experimental results show that these conditions make the Onion Service traffic less distinguishable (in some cases, the accuracy drops by more than 15%.) In our final contribution, we identify the most influential feature combinations for our classification problem and evaluate their impact.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

Tor is an anonymity network that hides the identity of its users by routing the traffic through multiple intermediary nodes. Tor also supports the provision of anonymous services known as Onion Services (also known as hidden services) with *.onion* as the top-level domain name. Tor's ability to act as a censorship circumvention tool has encouraged security experts, network defenders, and law enforcement agencies to identify Tor traffic from other encrypted and non-encrypted traffic. For example, I tried to classify Tor traffic from non-Tor Traffic, tried to classify the application types in Tor traffic, and tried to classify Tor traffic from other anonymity network traffic such as I2P traffic and Web-mix Traffic. However, in this work, we intend to explore the distinguishability of Onion Service traffic from standard Tor traffic using traffic analysis. We formulate three research questions to act as a foundation for our work. First, we try to answer the question A standard Tor circuit that is created to visit a web service on the Internet via Tor consists of three Tor nodes. An Onion Service circuit, which is the only way to access an Onion Service, consists of six Tor nodes. As the traffic in both these circuits (standard Tor and Onion Service) is encrypted, we assume that we can use the information leaked from the metadata (e.g., direction, timestamps, packet size) to identify unique patterns that can distinguish them. Onion Services have been used to host illegal websites, and more recently, they have been used as Command and Control (C&C) servers for botnets. Therefore, from the perspective of governments and law enforcement agencies, they want to track and shut down such services and regulate the Onion Service traffic. Even businesses might find it useful to restrict access to such websites to protect their systems from potential bad actors (e.g., hackers) and attacks. As a result, having techniques for identifying Onion Service traffic can be useful for two main reasons; 1. Such techniques can act as a steppingstone for fingerprinting of Onion Services. 2. They can be useful to restrict Onion Service traffic in sensitive and confidential systems. Second, we try to investigate the same problem under different settings. Specifically, we try. There are certain techniques that can be implemented in Tor to change its traffic patterns. Introducing padding, using dummy bursts and delays, and splitting the traffic are a few examples of such techniques. These techniques were developed to obfuscate the information leakage of Tor traffic.

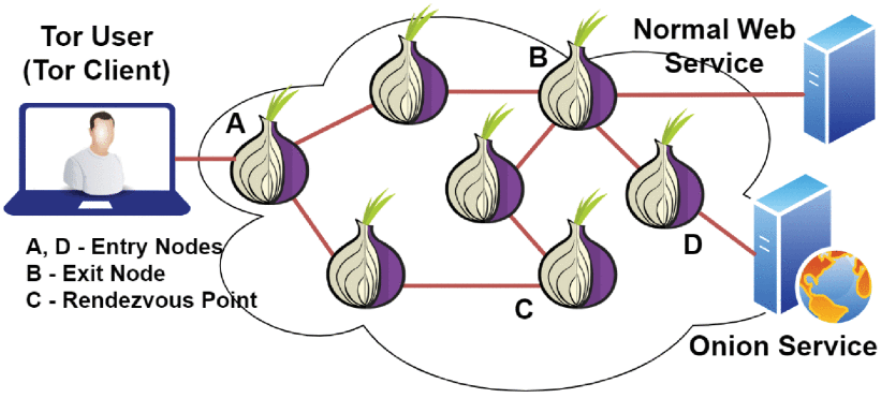


Figure – 1

1.1 OBJECTIVE

Classifying network traffic is important for traffic shaping and monitoring. In the last two decades, with the emergence of privacy concerns, the importance of privacy-preserving technologies has risen. The Tor network, which provides anonymity to its users and supports anonymous services known as Onion Services, is a trendy way to achieve online anonymity. However, this anonymity (especially with Onion Services) is frequently misused, encouraging governments and law enforcement agencies to de-anonymous them. Therefore, in this paper, we try to identify the classifiability of Onion Service traffic, focusing on three main contributions. First, we try to identify Onion Service traffic from other Tor traffic. The techniques we have used can identify Onion Service traffic with >99% accuracy. However, there are several modifications that can be made to the Tor traffic to obfuscate its information leakage. In our second contribution, we evaluate how our techniques perform when such modifications have been made to the Tor traffic. Our experimental results show that these conditions make the Onion Service traffic less distinguishable (in some cases, the accuracy drops by more than 15%.) In our final contribution, we identify the most influential feature combinations for our classification problem and evaluate their impact.

CHAPTER 2
LITERATURE SURVEY

‘‘Tor: The second generation onion router,’’

We present Tor, a circuit-based low-latency anonymous communication service. This second-generation Onion Routing system addresses limitations in the original design by adding perfect forward secrecy, congestion control, directory servers, integrity checking, configurable exit policies, and a practical design for location-hidden services via rendezvous points. Tor works on the real-world Internet, requires no special privileges or kernel modifications, requires little synchronization or coordination between nodes, and provides a reasonable tradeoff between anonymity, usability, and efficiency. We briefly describe our experiences with an international network of more than 30 nodes. We close with a list of open problems in anonymous communication.

‘‘Enhancing Tor’s performance using real-time traffic classification,’’

Tor is a low-latency anonymity-preserving network that enables its users to protect their privacy online. It consists of volunteer-operated routers from around the world that serve hundreds of thousands of users daily. Due to congestion and a low relay-to-client ratio, Tor suffers from performance issues that can potentially discourage its wider adoption, and result in an overall weaker anonymity to all users. We seek to improve the performance of Tor by defining different classes of service for its traffic. We recognize that although most of the Tor traffic is interactive web browsing, a small amount of bulk downloading consumes an unfair amount of Tor's scarce bandwidth. Furthermore, these traffic classes have different time and bandwidth constraints; therefore, they should not be given the same Quality of Service (QoS), which Tor offers them today. We propose and evaluate different Tor, a machine-learning-based approach that classifies Tor's encrypted circuits by application in real time and subsequently assigns distinct classes of service to each application. Our experiments confirm that we can classify circuits we generated on the live Tor network with an extremely high accuracy that exceeds 95%. We show that our real-time classification in combination with QoS can improve the experience of Tor clients, as our simple techniques result in a 75% improvement in responsiveness and an 86% reduction in download times at the median for interactive users.

‘‘Characterization of Tor traffic using time based features,’’

Traffic classification has been the topic of many research efforts, but the quick evolution of Internet services and the pervasive use of encryption makes it an open challenge. Encryption is essential in protecting the privacy of Internet users, a key technology used in the different privacy enhancing tools that have appeared in recent years. Tor is one of the most popular of them, it decouples the sender from the receiver by encrypting the traffic between them and routing it through a distributed network of servers. In this paper, we present a time analysis on Tor traffic flows, captured between the client and the entry node. We define two scenarios, one to detect Tor traffic flows and the other

to detect the application type: Browsing, Chat, Streaming, Mail, P2P or File Transfer. In addition, with this paper we publish the Tor labelled dataset we generated and used to test our classifiers. Traffic classification has been the topic of many research efforts, but the quick evolution of Internet services and the pervasive use of encryption makes it an open challenge. Encryption is essential in protecting the privacy of Internet users, a key technology used in the different privacy enhancing tools that have appeared in recent years. Tor is one of the most popular of them, it decouples the sender from the receiver by encrypting the traffic between them and routing it through a distributed network of servers. In this paper, we present a time analysis on Tor traffic flows, captured between the client and the entry node. We define two scenarios, one to detect Tor traffic flows and the other to detect the application type: Browsing, Chat, Streaming, Mail, P2P or File Transfer. In addition, with this paper we publish the Tor labelled dataset we generated and used to test our classifiers. Traffic classification has been the topic of many research efforts, but the quick evolution of Internet services and the pervasive use of encryption makes it an open challenge. Encryption is essential in protecting the privacy of Internet users, a key technology used in the different privacy enhancing tools that have appeared in recent years. Tor is one of the most popular of them, it decouples the sender from the receiver by encrypting the traffic between them and routing it through a distributed network of servers. In this paper, we present a time analysis on Tor traffic flows, captured between the client and the entry node. We define two scenarios, one to detect Tor traffic flows and the other to detect the application type: Browsing, Chat, Streaming, Mail, P2P or File Transfer. In addition, with this paper we publish the Tor labelled dataset we generated and used to test our classifiers.

‘‘Tor traffic classification from raw packet header using convolutional neural network,’’

As the amount of network traffic is growing exponentially, traffic analysis and classification are playing a significant role for efficient resource allocation and network management. However, with emerging security technologies, this work is becoming more difficult by encrypted communication such as Tor, which is one of the most popular encryption techniques. This paper proposes an approach to classify Tor traffic using hexadecimal raw packet header and convolutional neural network model. Compared with competitive machine learning algorithms, our approach shows remarkable accuracy. To validate this method publicly, we use UNB-CIC Tor network traffic dataset. Based on the experiments, our approach shows 99.3% accuracy for the fractionized Tor/non-Tor traffic classification.

CHAPTER 3
SYSTEM ANALYSIS

3.1
EXISTING SYSTEM

Computer networks consist of many protocols for data transmission and these protocols include HTTP, Voice Protocol, Email Protocol and many more. This protocol is often vulnerable of attacks for data steal and to overcome from this issue TOR protocol was introduced which provide anonymity or security to user's data. Tor, which provides anonymity is called ONION Services. Often Onion services were used by illegal websites to avoid detection and perform malicious activities. Therefore, in proposed paper the author applying machine learning algorithms to classify Tor Services as Onion Services or normal Tor services.

Disadvantages

1. Less accuracy.

PROPOSED SYSTEM

In the proposed work, the author evaluated the performance of machine learning algorithms such as SVM (Support Vector Machines), KNN and Random Forest. Each algorithm's performance is evaluated in terms of accuracy, precision, recall and FSCORE. In proposed work we have trained SVM, KNN and Random Forest on Original TOR dataset, Modified WTFPAD dataset and OS Onion Services dataset. First algorithms get trained on Original TOR data, second time algorithm get trained on WTFPAD dataset and then for third time we combine OS and WTFPAD dataset to classify network as Tor or Onion. Each algorithm is tuned using Hyper parameters.

Advantages

1. High Accuracy.

3.2 EXTENSION CONCEPT

In proposed work author has used traditional machine learning algorithms and have not used any advance machine learning algorithms like XGBOOST or ADABOOST so as extension we have experimented with XGBOOST algorithm which will use multiple estimators for training and then used estimator with best accuracy. Extension ADABOOST can get 100% accuracy for TOR or Onion service classification.

3.3 PROCESS MODEL

SDLC (Umbrella Model):

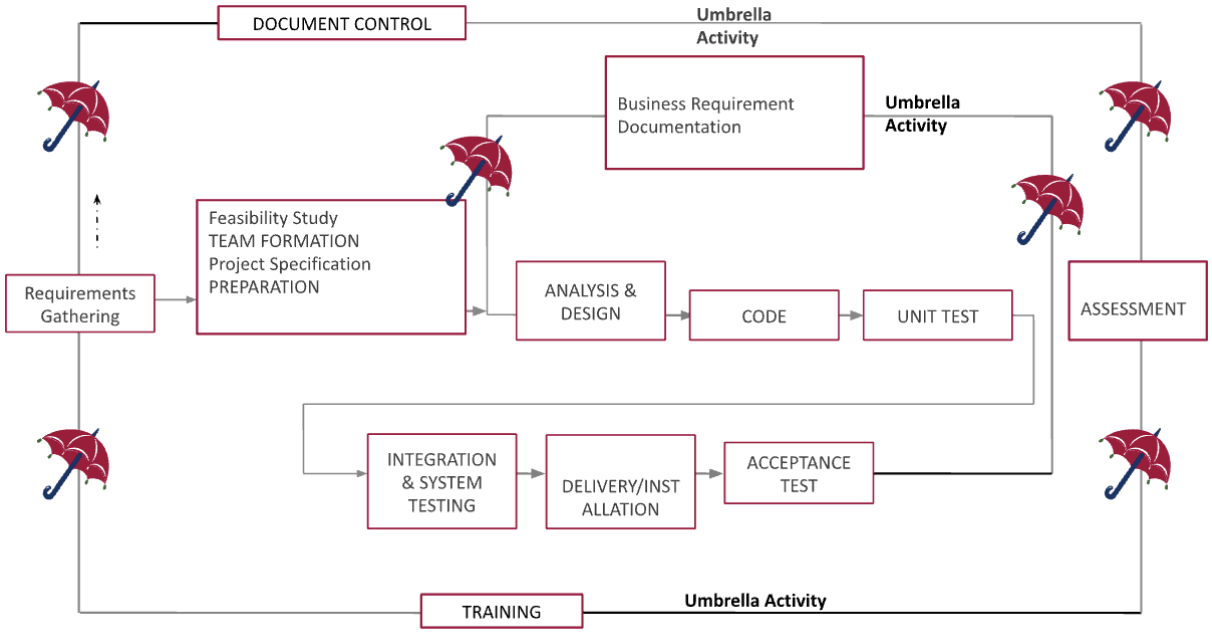


Figure - 2

3.4 SOFTWARE REQUIREMENT SPECIFICATION

3.4.1. Overall Description

A Software Requirements Specification (SRS) – a requirements specification for a software system is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). System requirements specification: A structured collection of information that embodies the requirements of a system. A business analyst, sometimes titled system analyst, is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions.

3.4.2. External Interface Requirements

User Interface

The user interface of this system is a user-friendly python Graphical User Interface.

Hardware Interfaces

The interaction between the user and the console is achieved through python capabilities.

Software Interfaces

The required software is python.

SYSTEM REQUIREMENT:

HARDWARE REQUIREMENTS:

Processor - Intel i3(min)

- Speed - 1.1 GHz
- RAM - 4GB (min)
- Hard Disk - 500 GB

SOFTWARE REQUIREMENTS:

- Operating System - Windows10(min)

Programming Language - Python with Jupiter notebook

CHAPTER 4
SYSTEM DESIGN

UML Diagram:

The Unified Modelling Language allows the software engineer to express an analysis model using the modelling notation governed by a set of syntactic semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspectives. Each view is defined by a set of diagrams, which is as follows.

- **User Model View**
  This view represents the system from the user's perspective. The analysis representation describes a usage scenario from the end-user's perspective.
- **Structural Model view**
  In this model the data and functionality arrived from inside the system. This model view models the static structures.
- **Behavioral Model View**
  It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.
- **Implementation Model View**
  In this the structural and behavioral as parts of the system are represented as they are to be built.
- **Environmental Model View**
  In these the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

Use case Diagram: A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the several types of users of a system and the several ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

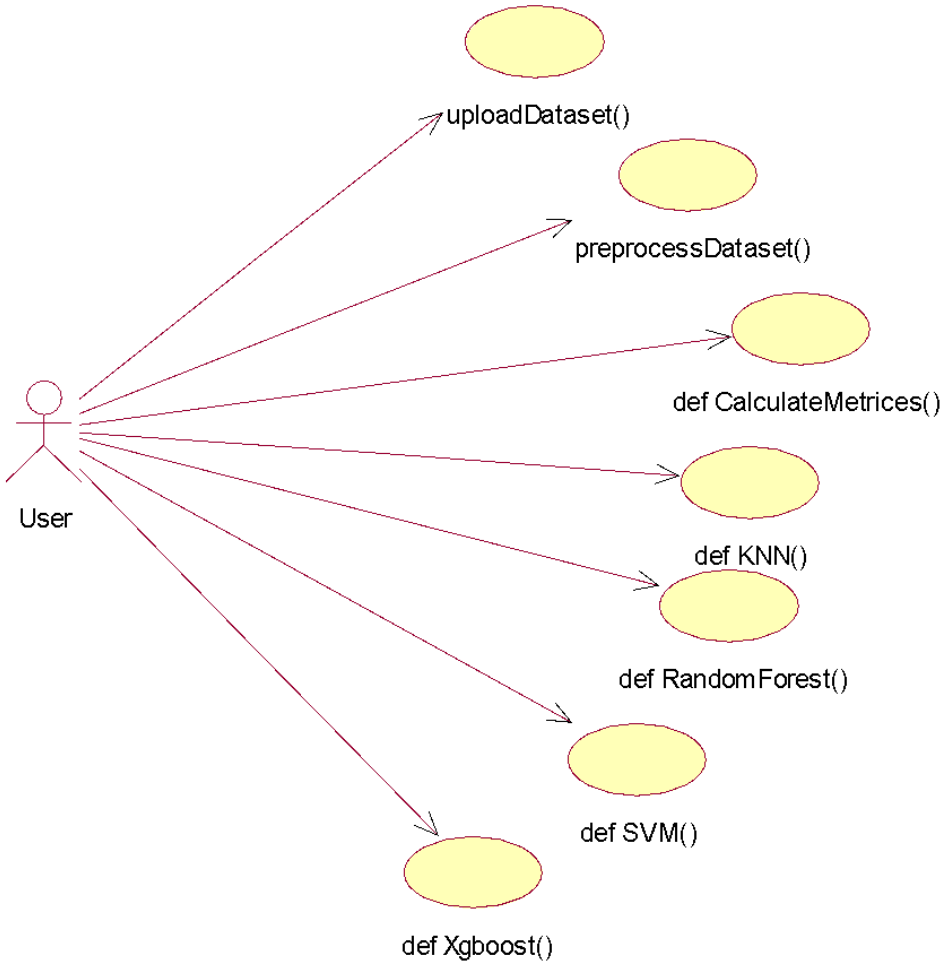


Figure – 3

Class Diagram: The **class diagram** is the main building block of object-oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake.

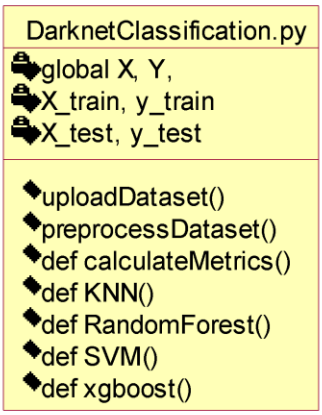


Figure – 4

Sequence diagram: A **sequence diagram** is an interaction diagram showing how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

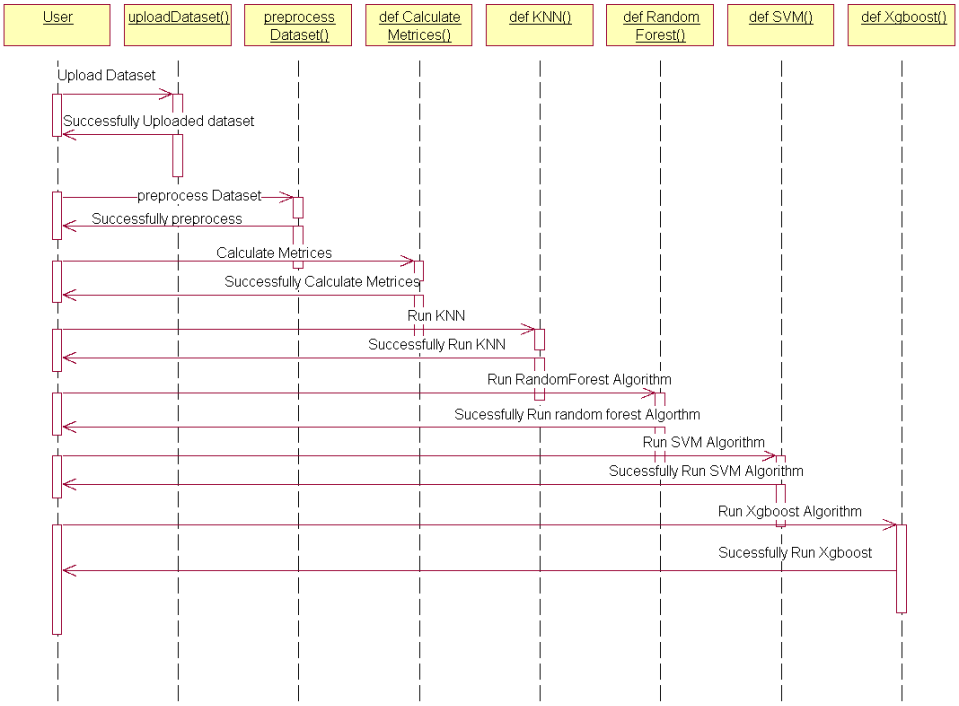


Figure – 5

Collaboration diagram: A **collaboration diagram** describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing a system's static structure and dynamic behavior.

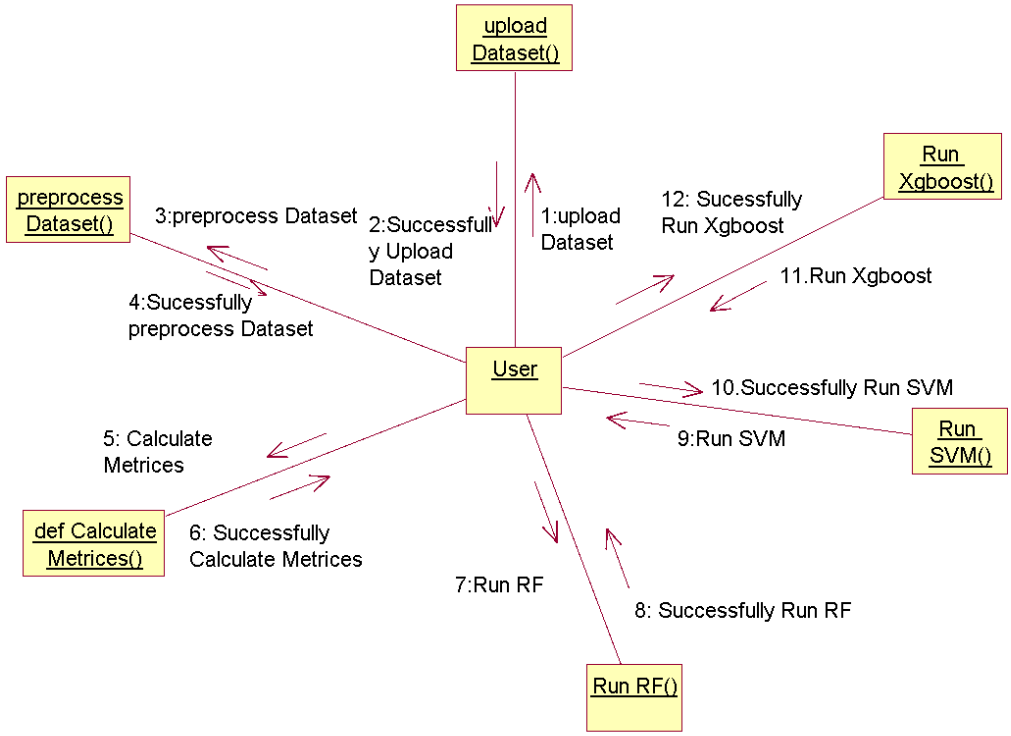


Figure – 6

Component Diagram: In the Unified Modelling Language, a **component diagram** depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems. Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.

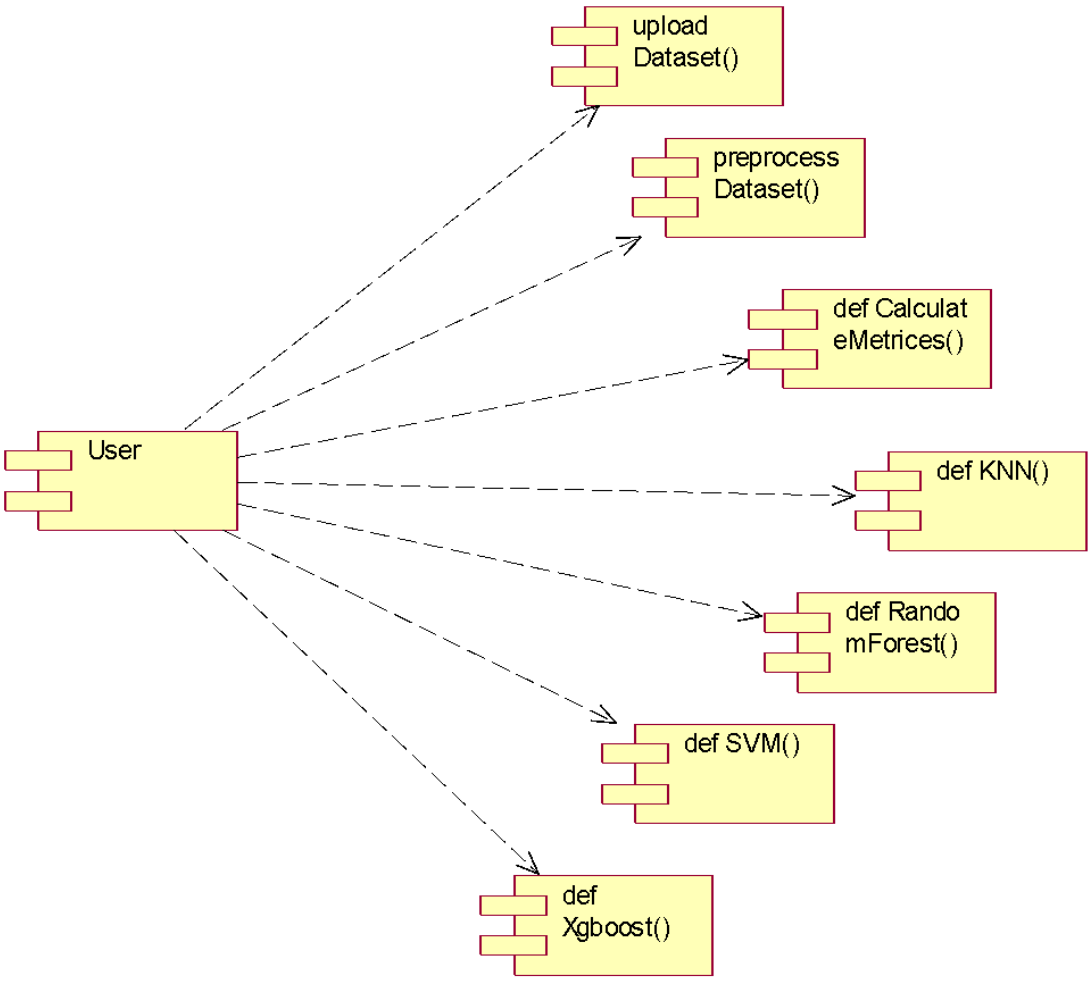


Figure – 7

Deployment Diagram: A **deployment diagram** in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI). The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

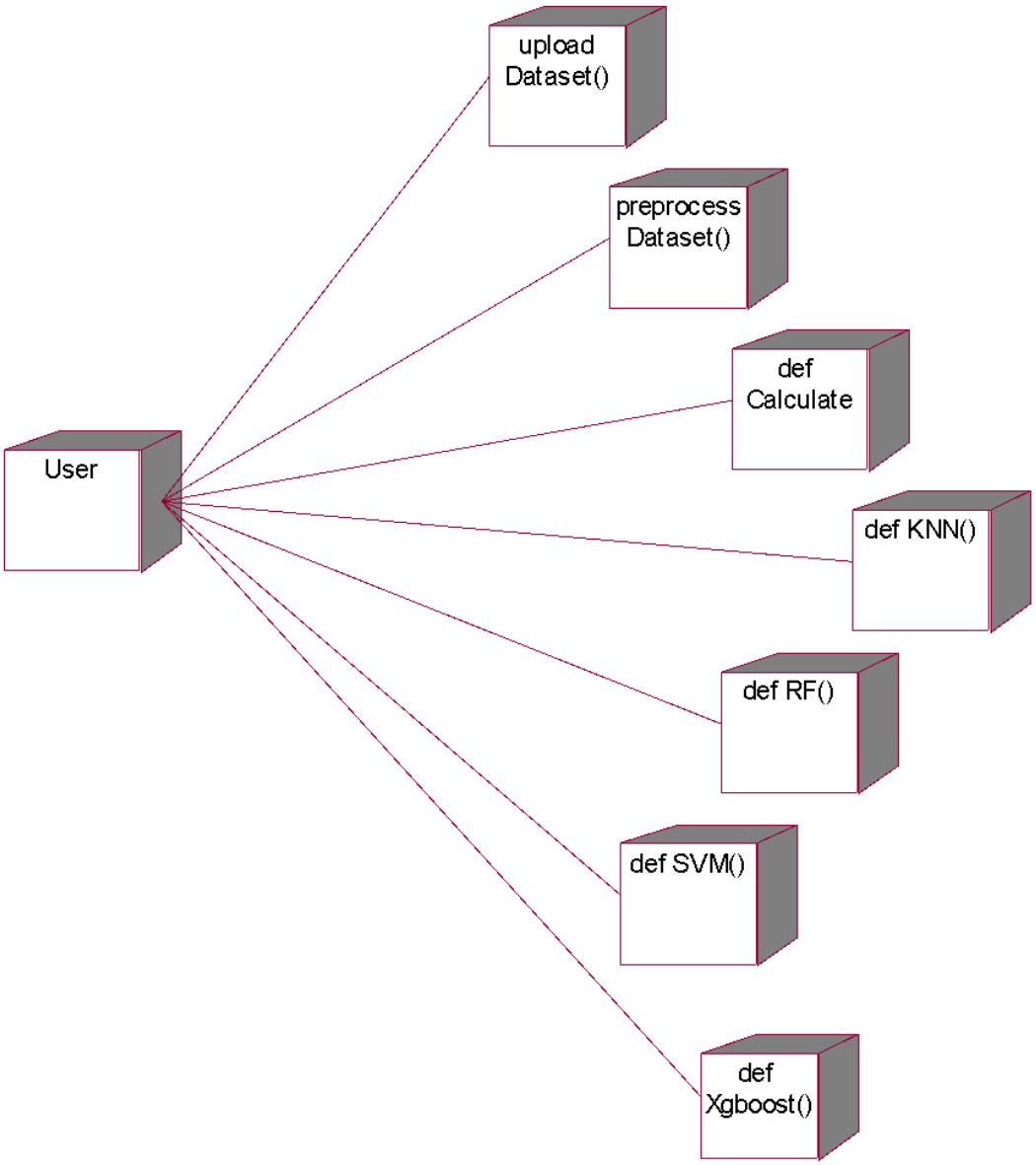

upload
Dataset()
preprocess
Dataset()
def
Calculate
User
def KNN()
def RF()
def SVM()
def
Xgboost()

Figure - 8

CHAPTER 5
IMPLEMENTATION

5.1 Python

* One of the most popular languages is Python. Guido van Rossum released this language in 1991. Python is available on the Mac, Windows, and Raspberry Pi operating systems. The syntax of Python is simple and identical to that of English. When compared to Python, it was seen that the other language requires a few extra lines.

*It is an interpreter-based language because code may be run line by line after it has been written. This implies that rapid prototyping is possible across all platforms. Python is a big language with a free, binary-distributed interpreter standard library.

*It is inferior to maintenance that is conducted and is straightforward to learn. It is an object-oriented, interpreted programming language. It supports several different programming paradigms in addition to object-oriented programming, including functional and procedural programming.

*It supports several different programming paradigms in addition to object-oriented programming, including practical and procedural programming. Python is mighty while maintaining a straightforward syntax. Classes, highly dynamic data types, modules, and exceptions are covered. Python can also be utilized by programmers that require programmable interfaces as an external language.

5.2 Installation

To install Python on your computer, follow these basic steps:

- Step 1: Visit the Python website Go to the official Python website at https://www.python.org/.
- Step 2: Select the operating system Choose the appropriate installer for your operating system. Python supports Windows, macOS, and various Linux distributions. Make sure to select the correct version that matches your operating system.
- Step 3: Check which version of Python is installed; if the 3.7.0 version is not there, uninstall it through the control panel.
- Step 4: Install Python 3.7.0 using Cmd.
- Step 5: Install all libraries that are required to run the project.
- Step 6: Run.

5.3 Python Features

- **Easy:** Because Python is a more accessible and straightforward language, Python programming is easier to learn.

- **Interpreted language:** Python is an interpreted language; therefore, it can be used to examine the code line by line and provide results.
- **Open Source:** Python is a free online programming language since it is open source.
- **Portable:** Python is portable because the same code may be used on several computer standard
- **libraries:** Python offers a sizable library that we may utilize to create applications quickly.
- **GUI:** It stands for GUI (Graphical User Interface)
- **Dynamical typed: Python** is a dynamically typed language; therefore, the type of the value will be determined at runtime.

5.4 Python GUI (Tkinter)

- Python provides a wide range of options for GUI development (Graphical User Interfaces).
- Tkinter, the most widely used GUI technique, is used for all of them.
- The Tk GUI toolkit offered by Python is used with the conventional Python interface.
- Tkinter is the easiest and quickest way to write Python GUI programs.
- Using Tkinter, creating a GUI is simple.
- A part of Python's built-in library is Tkinter. The GUI programs were created.
- Python and Tkinter together give a straightforward and quick way. The Tk GUI toolkit's object-oriented user interface is called Tkinter.

Making a GUI application is easy using Tkinter. Following are the steps:

1) Install the Tkinter module in place.

2) The GUI application Makes the primary window

3) Include one or more of the widgets mentioned above in the GUI application.

4) Set up the main event loop such that it reacts to each user-initiated event.

Although Tkinter is the only GUI framework included in the Python standard library, Python includes a GUI framework. The default library for Python is called Tkinter. Tk is a scripting language often used in designing, testing, and developing GUIs. Tk is a free, open-source widget toolkit that may be used to build GUI applications in a wide range of computer languages.

5.5 Python IDLE

- Python IDLE offers a full-fledged file editor, which gives you the ability to write and execute Python programs from within this program. The built-in file editor also includes several features, like code completion and automatic indentation, that will speed up your coding workflow.
- Guido Van Rossum named Python after the British comedy group Monty Python while the name IDLE was chosen to pay tribute to Eric Idle, who was one of the Monty Python's founding members. IDLE comes bundled with the default implementation of the Python language.

- IDLE executes statements like Python Shell. IDLE is used to create, modify, and execute Python code. IDLE provides a fully featured text editor to write Python scripts and provides features like syntax highlighting, auto-completion, and smart indent.
- IDLE has two modes: interactive and script. We wrote our first program, “Hello, World!” in interactive mode. Interactive mode immediately returns the results of commands you enter the shell. In script mode, you will write a script and then run it.
- The IDE Python IDLE is a good place to start as it helps you become familiar with the way Python works and understand its syntax. This IDE is good to start programming in Python due to its great debugger, but once you are fluent and start developing projects it is necessary to jump to another, more complete IDE.
- Python IDLE (Integrated Development and Learning Environment) is an interactive development environment included with the Python programming language. It provides a convenient way to write, execute, and debug Python code.
- Debugging: IDLE provides basic debugging capabilities to help you find and fix errors in your code. You can set breakpoints, step through code, inspect variables, and track the program's execution.
- Python Help: IDLE provides access to the Python documentation and built-in help. You can access the help menu to find information about Python modules, functions, classes, and more.
- Script Execution: In addition to the interactive shell, IDLE allows you to run Python scripts stored in files. You can write your code in the editor and execute it as a script to see the output or interact with the program.
- Customization: IDLE can be customized to suit your preferences. You can modify settings related to syntax highlighting, indentation, fonts, and more.
- Python IDLE serves as a beginner-friendly development environment and learning tool. It is suitable for writing small scripts, testing code snippets, experimenting with Python features, and learning the language's basics. However, for more advanced development projects, you may consider using other code editors or integrated development environments (IDEs) that provide additional features and better project management capabilities.

5.6 Libraries

In Python, libraries (also referred to as modules or packages) are collections of pre-written code that provide additional functionality and tools to extend the capabilities of the Python language. Libraries contain reusable code that developers can leverage to perform specific tasks without having to write everything from scratch. Python libraries are designed to solve common problems, such as handling data, performing mathematical operations, interacting with databases, working with files, implementing networking protocols, creating graphical user interfaces (GUIs), and much more. They provide ready-to-use functions, classes, and methods that simplify complex operations and save development time.

Libraries in Python offer various advantages:

- Code Reusability
- Efficiency
- Collaboration
- Domain-Specific Functionality
- To use a Python library, you need to install it first.

There are some libraries following:

Pandas: Pandas are a Python computer language library for data analysis and manipulation. It offers a specific operation and data format for handling time series and numerical tables. It differs significantly from the release3-clause of the BSD license. It is a well-liked open source of opinion that is utilized in machine learning and data analysis. Pandas are a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Pandas are a Python library used for working with data sets.

- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" refers to "Panel Data" and "Python Data Analysis" and was created by Wes McKinney in 2008.
- Pandas allow us to Analyse big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets and make them readable and relevant.

Relevant data is very important in data science. Pandas are a Python library for data analysis. Started by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool, pandas have grown into one of the most popular Python libraries. It has an extremely active community of contributors. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself.

NumPy: The NumPy Python library for multi-dimensional, big-scale matrices adds a huge number of high-level mathematical functions. It is possible to modify NumPy by utilizing a Python library. Along with line, algebra, and the Fourier transform operations, it also contains several matrices-related functions. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices, and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

- NumPy is a Python library used for working with arrays.

- It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open-source project, and you can use it freely.
- NumPy stands for Numerical Python.
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

Matplotlib: It is a multi-platform, array-based data visualization framework built to interact with the whole SciPy stack. MATLAB is proposed as an open-source alternative. Matplotlib is a Python extension and a cross-platform toolkit for graphical plotting and visualization. Matplotlib is a popular Python library for creating static, animated, and interactive visualizations. It provides a flexible and comprehensive set of tools for generating plots, charts, histograms, scatter plots, and more. Matplotlib is widely used in various fields, including data analysis, scientific research, and data visualization. Here are some key features and functionalities of the Matplotlib library:

- Plotting Functions
- Customization Options
- Multiple Interfaces
- Integration with NumPy and pandas
- Subplots and Figures
- Saving and Exporting

Scikit-learn: The most stable and practical machine learning library for Python is scikit-learn. Regression, dimensionality reduction, classification, and clustering are just a few of the helpful tools it provides through the Python interface for statistical modeling and machine learning. It is an essential part of the Python machine learning toolbox used by JP Morgan. It is frequently used in various machine learning applications, including classification and predictive analysis. Scikit-learn (also referred to as sklearn) is a widely used open-source machine learning library for Python. It provides a comprehensive set of tools and algorithms for various machine learning tasks, including classification, regression, clustering, dimensionality reduction, model selection, and pre-processing. Here are some key features and functionalities of the Scikit-learn library:

- Easy-to-Use Interface
- Broad Range of Algorithms
- Data Pre-processing and Feature Engineering
- Model Evaluation and Validation

- Integration with NumPy and pandas
- Robust Documentation and Community Support

Keras: Google's Keras is a cutting-edge deep learning API for creating neural networks. It is created in Python and is designed to simplify the development of neural networks. Additionally, it enables the use of various neural networks for computation. Deep learning models are developed and tested using the free and open-source Python software known as Keras. Keras is a high-level deep learning library for Python. It is designed to provide a user-friendly and intuitive interface for building and training deep learning models. Keras acts as a front-end API, allowing developers to define and configure neural networks while leveraging the computational backend engines, such as Tensor Flow or Theano. Here are some key features and functionalities of the Keras library:

- User-Friendly API
- Multi-backend Support
- Wide Range of Neural Network Architectures
- Pre-trained Models and Transfer Learning
- Easy Model Training and Evaluation
- GPU Support

h5py: The h5py Python module offers an interface for the binary HDF5 data format. Thanks to p5py, the top can quickly halt the vast amount of numerical data and alter it using the NumPy library. It employs common syntax for Python, NumPy, and dictionary arrays. h5py is a Python library that provides a simple and efficient interface for working with datasets and files in the Hierarchical Data Format 5 (HDF5) format. HDF5 is a versatile data format commonly used for storing and managing large volumes of numerical data. Here are some key features and functionalities of the h5py library:

- HDF5 File Access
- Dataset Handling
- Group Organization
- Attributes
- Compatibility with NumPy
- Performance

Tensor flow: TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. TensorFlow is an end-to-end open-source platform for machine learning. TensorFlow is a rich system for managing all aspects of a machine learning system; however, this class focuses on using a particular TensorFlow API to develop and train machine learning models. TensorFlow is a popular open-source library for machine learning and deep learning. It provides a comprehensive set of tools,

APIs, and computational resources for building and training various types of machine learning models, especially neural networks. Here are some key features and functionalities of TensorFlow:

- Neural Network Framework
- Computational Graphs
- Automatic Differentiation
- GPU and TPU Support
- Distributed Computing
- Deployment Capabilities

Tkinter: Tkinter is an acronym for "Tk interface". Tk was developed as a GUI extension for the Tcl scripting language by John Ousterhout. The first release was in 1991. Tkinter is the de facto way in Python to create Graphical User interfaces (GUIs) and is included in all standard Python Distributions. In fact, it's the only framework built into the Python standard library. Tkinter is a standard Python library used for creating graphical user interfaces (GUIs). It provides a set of modules and classes that allow you to develop interactive and visually appealing desktop applications. Here are some key features and functionalities of Tkinter:

- Cross-Platform Compatibility
- Simple and Easy-to-Use
- Widgets and Layout Management
- Event-Driven Programming
- Customization and Styling
- Integration with Other Libraries

NLTK: NLTK is a toolkit built for working with NLP in Python. It provides us various text processing libraries with a lot of test datasets. A variety of tasks can be performed using NLTK such as tokenizing, parse tree visualization, etc. NLTK (Natural Language Toolkit) is the go-to API for NLP (Natural Language Processing) with Python. It is a powerful tool to pre-process text data for further analysis, like with ML models. It helps convert text into numbers, which the model can then easily work with. NLTK (Natural Language Toolkit) is a Python library widely used for working with human language data and implementing natural language processing (NLP) tasks. It provides a set of tools, corpora, and resources for tasks such as tokenization, stemming, tagging, parsing, sentiment analysis, and more. Here are some key features and functionalities of NLTK:

- Text Processing
- Part-of-Speech Tagging
- Named Entity Recognition
- Chunking and Parsing
- Sentiment Analysis
- WordNet Integration

Scipy: SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. SciPy is a powerful scientific computing library for Python that provides a wide range of mathematical algorithms and functions. It builds upon NumPy, another fundamental library for numerical computing, and extends its capabilities by adding additional tools for scientific and technical computing tasks. Here are some key features and functionalities of SciPy:

- Numerical Integration:
- Optimization and Root Finding
- Linear Algebra
- Signal and Image Processing
- Statistics

5.7 Algorithms Used

RANDOM FOREST

Machine learning, a fascinating blend of computer science and statistics, has witnessed incredible progress, with one standout algorithm being the **Random Forest**. **Random forests or Random Decision Trees** is a collaborative team of **decision trees** that work together to provide a single output. Originating in 2001 through Leo Breiman, Random Forest has become a cornerstone for machine learning enthusiasts. In this article, we will explore the fundamentals and implementation of **Random Forest Algorithm.** Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating several Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance. In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks) This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results. Random forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.

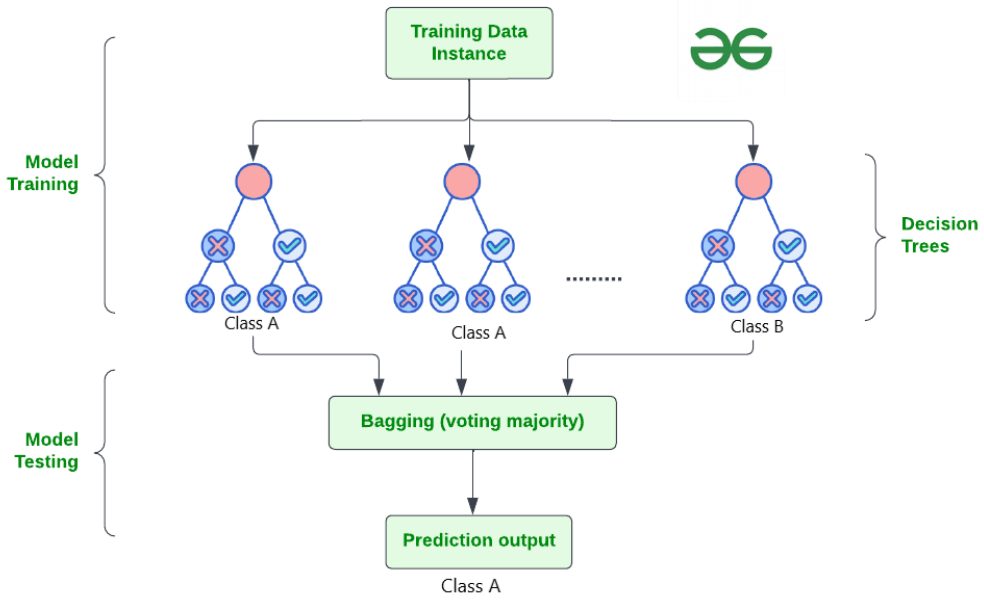


Figure - 9

How Does Random Forest Work?

The random Forest algorithm works in several steps which are discussed below–>

- **Ensemble of Decision Trees:** Random Forest leverages the power of ensemble learning by constructing an army of Decision Trees. These trees are like individual experts, each specializing in a particular aspect of the data. Importantly, they operate independently, minimizing the risk of the model being overly influenced by the nuances of a single tree.
- **Random Feature Selection:** To ensure that each decision tree in the ensemble brings a unique perspective, Random Forest employs random feature selection. During the training of each tree, a random subset of features is chosen. This randomness ensures that each tree focuses on different aspects of the data, fostering a diverse set of predictors within the ensemble.
- **Bootstrap Aggregating or Bagging:** The technique of bagging is a cornerstone of Random Forest's training strategy which involves creating multiple bootstrap samples from the original dataset, allowing instances to be sampled with replacement. This results in different subsets of data for each decision tree, introducing variability in the training process and making the model more robust.
- **Decision Making and Voting:** When it comes to making predictions, each decision tree in the Random Forest casts its vote. For classification tasks, the final prediction is determined by the mode (most frequent prediction) across all the trees. In regression tasks, the average of the individual tree predictions is taken. This internal voting mechanism ensures a balanced and collective decision-making process.

Key Features of Random Forest

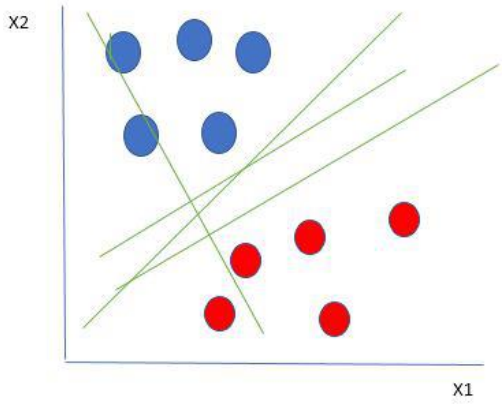
Some of the Key Features of Random Forest are discussed below–>

- **High Predictive Accuracy:** Imagine Random Forest as a team of decision-making wizards. Each wizard (decision tree) looks at a part of the problem, and together, they weave their insights into a powerful prediction tapestry. This teamwork often results in a more accurate model than a single wizard could achieve.
- **Resistance to Overfitting:** Random Forest is like a cool-headed mentor guiding its apprentices (decision trees). Instead of letting each apprentice memorize every detail of their training, it encourages a more well-rounded understanding. This approach helps prevent getting too caught up with the training data which makes the model less prone to overfitting.
- **Large Datasets Handling:** Dealing with a mountain of data? Random Forest tackles it like a seasoned explorer with a team of helpers (decision trees). Each helper takes on a part of the dataset, ensuring that the expedition is not only thorough but also surprisingly quick.
- **Variable Importance Assessment:** Think of Random Forest as a detective at a crime scene, figuring out which clues (features) matter the most. It assesses the importance of each clue in solving the case, helping you focus on the key elements that drive predictions.
- **Built-in Cross-Validation:** Random Forest is like having a personal coach that keeps you in check. As it trains each decision tree, it also sets aside a secret group of cases (out-of-bag) for testing. This built-in validation ensures your model doesn't just ace the training but also performs well on new challenges.
- **Handling Missing Values:** Life is full of uncertainties, just like datasets with missing values. Random Forest is the friend who adapts to the situation, making predictions using the information available. It doesn't get flustered by missing pieces; instead, it focuses on what it can confidently tell us.
- **Parallelization for Speed:** Random Forest is your time-saving buddy. Picture each decision tree as a worker tackling a piece of a puzzle simultaneously. This parallel approach taps into the power of modern tech, making the whole process faster and more efficient for handling large-scale projects.

SUPPORT VECTOR MACHINE (SVM):

Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for various tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection. SVMs are adaptable and efficient in a variety of applications because they can manage high-dimensional data and nonlinear relationships. SVM algorithms are very effective as we try to find the maximum separating hyperplane between the different classes available in the target feature. Support Vector Machine (SVM) is an supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it is best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries to ensure that the

margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three. Let us consider two independent variables x_1, x_2, and one dependent variable which is either a blue circle or a red circle.



Linearly Separable Data points

Figure - 10

From the figure above, there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregate our data points or classify between red and blue circles.

Support Vector Machine Terminology

- **Hyperplane: Hyperplane is the decision boundary used to separate different classes' data points in a feature space.** In the case of linear classifications, it will be a linear equation i.e., wx+b = 0.
- **Support Vectors:** Support vectors are the closest data points to the hyperplane, which plays a critical role in deciding the hyperplane and margin.
- **Margin**: Margin is the distance between the support vector and hyperplane. The main objective of the support vector machine algorithm is to maximize the margin. The wider margin indicates better classification performance.
- **Kernel**: Kernel is the mathematical function, which is used in SVM to map the original input data points into high-dimensional feature spaces, so, that the hyperplane can be easily found out even if the data points are not linearly separable in the original input space. Some of the common kernel functions are linear, polynomial, radial basis function (RBF), and sigmoid.
- **Hard Margin:** The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of various categories without any misclassifications.

- **Soft Margin:** When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft-margin SVM formulation, which softens the strict margin requirement and permits certain misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations.
- **C:** Margin maximization and misclassification fines are balanced by the regularization parameter C in SVM. The penalty for going over the margin or misclassifying data items is decided by it. A stricter penalty is imposed with a greater value of C, which results in a smaller margin and fewer misclassifications.
- **Hinge Loss:** A typical loss function in SVMs is hinge loss. It punishes incorrect classifications or margin violations. The objective function in SVM is frequently formed by combining it with the regularization term.
- **Dual Problem:** A dual Problem of the optimization problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The dual formulation enables the use of kernel tricks and more effective computing.

SVM implementation in Python

Predict if cancer is Benign or malignant. Using historical data about patients diagnosed with cancer enables doctors to differentiate malignant cases and benign ones are given independent attributes.

Steps

- Load the breast cancer dataset from sklearn.datasets
- Separate input features and target variables.
- Buil and train the SVM classifiers using RBF kernel.
- Plot the scatter plot of the input features.
- Plot the decision boundary.
- Plot the decision boundary

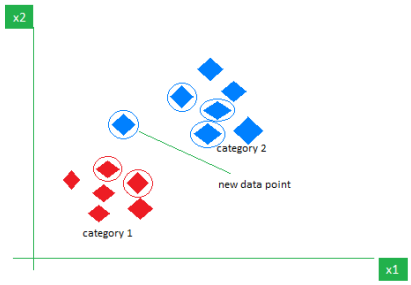
K-NEAREST NEIGHBORS(KNN):

The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method employed to tackle classification and regression problems. Evelyn Fix and Joseph Hodges developed this algorithm in 1951, which Thomas Cover later expanded. The article explores the fundamentals, workings, and implementation of the KNN algorithm.

What is the K-Nearest Neighbors Algorithm?

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features:



KNN Algorithm working visualization

Figure – 11

Now, given another set of data points (also called testing data), allocate these points to a group by analyzing the training set. Note that the unclassified points are marked as 'White'

Why do we need a KNN algorithm?

(KNN) algorithm is a versatile and widely used machine learning algorithm that is primarily used for its simplicity and ease of implementation. It does not require any assumptions about the underlying data distribution. It can also handle both numerical and categorical data, making it a flexible choice for several types of datasets in classification and regression tasks. It is a non-parametric method that makes predictions based on the similarity of data points in each dataset. KNN is less sensitive to outliers compared to other algorithms.

The KNN algorithm works by finding the K nearest neighbors to a given data point based on a distance metric, such as Euclidean distance. The class or value of the data point is then determined by the majority vote or average of the K neighbors. This approach allows the algorithm to adapt to different patterns and make predictions based on the local structure of the data.

Advantages of the KNN Algorithm

- Easy to implement
- Adapts Easily
- Few Hyper parameters

Disadvantages of the KNN Algorithm

- Does not scale
- Curse of Dimensionality
- Prone to Overfitting

XGBOOST:

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for "Extreme Gradient Boosting" and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression. One of the key features of XGBoost is its efficient handling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on large datasets in a reasonable amount of time. XGBoost can be used in a variety of applications, including Kaggle competitions, recommendation systems, and click-through rate prediction, among others. It is also highly customizable and allows for fine-tuning of various model parameters to optimize performance. XgBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington. It is a library written in C++ which optimizes the training for Gradient Boosting.

XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions. In this algorithm, decision trees are created in sequential form. Weights play a key role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Before understanding the XGBoost, we first need to understand the trees especially the decision tree:

Decision Tree: A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions.

Bagging: A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. Each base classifier is trained in parallel with a training set generated by randomly drawing, with replacement, N examples (or data) from the original training dataset, where N is the size of the original training set. The training set for each of the base classifiers is independent of each other. Many of the original data may be repeated in the resulting training set while others may be left out. Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

Boosting: Boosting is an ensemble modelling, a technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued, and models are added until either the complete training data set is predicted correctly, or the maximum number of models are added.

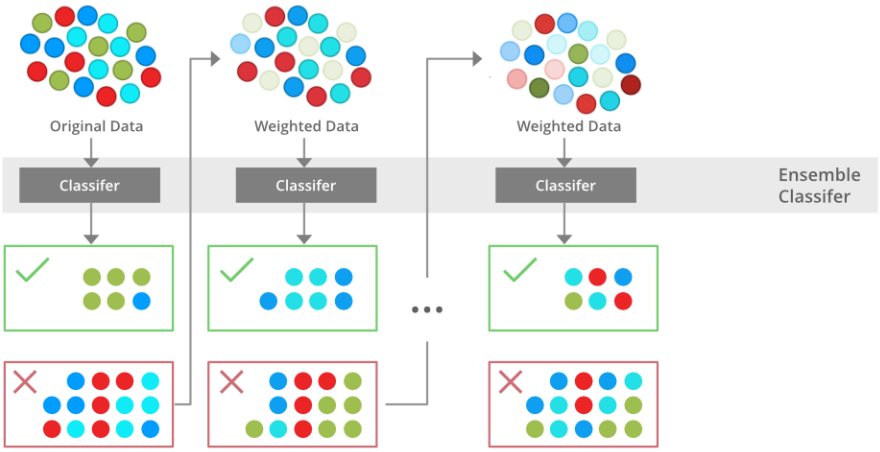


Figure – 12

Gradient Boosting: Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels. There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees).

WTF-PAD (Website Fingerprinting Protection with Adaptive Defense): WTF-PAD is a technique designed to protect against website fingerprinting attacks. Website fingerprinting (WF) attacks enable an adversary to infer the specific websites a user is visiting over an encrypted connection, such as through a VPN or Tor, by analyzing the patterns of the encrypted traffic. WTF-PAD seeks to mitigate this by introducing noise and adaptive padding to traffic patterns, making it harder for an adversary to accurately identify the websites being visited.

- **Adaptive Defense:** WTF-PAD dynamically adjusts the amount and timing of padding based on observed traffic patterns, which helps to mask the unique characteristics of different websites.
- **Low Overhead:** It aims to provide a balance between protection and performance, adding minimal extra data and latency to the traffic.
- **Compatibility:** Designed to work with existing anonymization tools like Tor, enhancing their resistance to WF attacks without significant modifications.

AdaBoost (Adaptive Boosting): AdaBoost is a machine learning algorithm used to improve the performance of other algorithms by combining multiple weak classifiers to form a strong classifier. It is an ensemble learning method, which means it combines the predictions of several base estimators to improve robustness over a single estimator.

- **Weak Classifiers:** It uses simple models (often decision stumps, which are decision trees with a single split) as its base classifiers.
- **Iterative Process:** AdaBoost trains the weak classifiers in a sequential manner. In each iteration, it adjusts the weights of the training instances, increasing the weights of the misclassified instances so that the subsequent classifier focuses more on the hard-to-classify cases.
- **Weighted Voting:** The final model is a weighted sum of the weak classifiers. The weight of each classifier is determined by its accuracy, with more accurate classifiers being given more influence in the final decision.
- **Error Reduction:** By focusing on the mistakes of previous classifiers, AdaBoost effectively reduces the overall error rate of the model.

Defense Dataset: A dataset where some form of traffic obfuscation or defense mechanism is applied. This could include techniques like padding, delaying, or modifying traffic patterns to make it harder for an adversary to perform traffic analysis or fingerprinting. The goal of the defense is to reduce the accuracy of any traffic analysis or fingerprinting attacks. This dataset is used to evaluate how effective the defense mechanisms are in protecting the anonymity of the traffic, such as in Onion Services on the Tor network.

- **Examples of Defenses:**
    - **WTF-PAD:** Adaptive padding to disguise traffic patterns.
    - **Traffic Shaping:** Altering the shape of traffic flows to look uniform.
    - **Dummy Traffic:** Injecting fake traffic to mask real traffic.

No Defense Dataset: A dataset where no defense mechanisms are applied to network traffic. This is the raw or unaltered traffic data, reflecting normal operations without any added obfuscation techniques. The no defense dataset serves as a baseline or control group. It is used to assess the vulnerability of traffic to fingerprinting and other forms of analysis without any protective measures. It helps in comparing how much more identifiable or vulnerable traffic is when no defenses are in place.

Characteristics:

- The traffic patterns are easily distinguishable.

- It is more susceptible to analysis and attacks, such as identifying which websites are being visited or correlating user activity.

For example, if you're working on evaluating the impact of a defense mechanism like WTF-PAD on Tor traffic, you would collect or use two datasets:

1. **No Defense Dataset:** Raw traffic data from Tor without WTF-PAD applied.

2. **Defense Dataset:** Traffic data from Tor with WTF-PAD applied.

CHAPTER 6
TESTING

Implementation and Testing:

Implementation is one of the most important tasks in a project. It is the phase in which one must be cautious because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving a successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

Implementation

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may require extensive user training. The initial parameters of the system should be modified due to programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained on the inkjet or dot matrix printer, available at the user's disposal. The proposed system is quite easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place, ensuring all components of the system property function as a unit. The test data should be chosen such that it passed through all conditions. Testing is the state of implementation aimed at ensuring the system works accurately and efficiently before the operation starts. The following is the description of the testing strategies, which were carried out during the testing period.

System Testing

Testing has become an integral part of any system or project, especially in information technology. The importance of testing is a method of justifying if one is ready to move further, be it to be check if one is capable to withstand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to the user to use the software must be tested to see whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data

are repeated. Thus, the code was exhaustively checked for all possible correct data and the outcomes were also checked.

Module Testing

To locate errors, each module is tested individually. This enables us to detect errors and correct them without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus, all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example, the job classification module is tested separately. This module is tested with different jobs and its approximate execution time, and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works more efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately, and their corresponding results are obtained which reduces the process waiting time.

Integration Testing

After the module testing, the integration testing is applied. When linking the modules there may be a chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus, the mapping of jobs with resources is done correctly by the system.

Acceptance Testing

When that user finds no major problems with its accuracy, the system passes through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

Test Case Id	Test Case Name	Test Case Desc.	Test Steps			Test Case Status	Test Priority
			Step	Expected	Actual		
01	Upload Dataset	Test whether Dataset is uploaded or not into the system	If the Dataset may not upload	We cannot do further operations	Water Quality Dataset uploaded we will do further operations	High	High

02	Preprocess & Normalized Dataset	Test whether the Pre-process & Normalized Dataset Successfully or not	If the Pre-process & Normalized Dataset may not Run Successfully	We cannot do further operations	we will do further operations	High	High
03	Run Calculate Metrices	Test whether Calculate Metrices Algorithm Run Successfully or not	If the Calculate Metrices Algorithm may not Run Successfully	We cannot do further operations	we will do further operations	High	High
04	Train run SVM Algorithm	Test whether SVM Algorithm Run Successfully or not	If the **SVM** Algorithm may not Run Successfully	We cannot do further operations	we will do further operations	High	High
05	Train run KNN Algorithm	Test whether KNN Algorithm Run Successfully or not	If the **KNN** Algorithm may not Run Successfully	We cannot do further operations	we will do further operations	High	High
06	Train Run Random Forest Algorithm	Test whether Random Forest Algorithm Run Successfully or not	If the Random Forest Algorithm may not Run Successfully	We cannot do further operations	we will do further operations	High	High
07	Train Run Xgboost Algorithm	Test whether Xgboost Algorithm Run Successfully or not	If the Xgboost Algorithm may not Run Successfully	We cannot do further operations	we will do further operations	High	High

08	Predict	Test prediction whether successfully or not	If the Predict may not Successfully	We cannot do further operations	we will do further operations	High	High

CHAPTER 7
APPENDIX

```
#import python classes and packages
import pandas as pd
import numpy as np
from skfeature.function.similarity_based import fisher_score #import fischer score features selected algorithm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import mutual_info_classif #import info gain features selected algorithm
from sklearn.feature_selection import SelectKBest
import pickle
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV #grid class for tuning each algorithm
import timeit

#loading no defence dataset
with open("Dataset/X_train_NoDef.pkl", 'rb') as handle:
    no_def_X = np.array(pickle.load(handle, encoding='latin1'))
handle.close()
no_def_X = no_def_X[:,0:300]
no_def_X = pd.DataFrame(no_def_X)
print("No Defence Dataset")
no_def_X
```

```
No Defence Dataset
```

```
      0    1    2    3    4    5    6    7    8    9    ...  290  291  292  \
0    -1.0  1.0 -1.0 -1.0  1.0  1.0  1.0 -1.0 -1.0 -1.0  ... -1.0 -1.0 -1.0
1     1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0  1.0 -1.0 -1.0  ...  1.0  1.0 -1.0
2     1.0 -1.0  1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0  ... -1.0  1.0  1.0
3    -1.0  1.0  1.0  1.0 -1.0  1.0  1.0 -1.0  1.0  1.0  ... -1.0 -1.0 -1.0
4     1.0 -1.0  1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0  ... -1.0 -1.0 -1.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
9495  1.0 -1.0  1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0  ... -1.0 -1.0 -1.0
9496 -1.0 -1.0  1.0  1.0  1.0  1.0 -1.0 -1.0 -1.0 -1.0  ...  1.0  1.0  1.0
9497 -1.0  1.0  1.0 -1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0  ... -1.0 -1.0 -1.0
9498  1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0  1.0  1.0  ... -1.0 -1.0 -1.0
9499 -1.0  1.0  1.0 -1.0  1.0  1.0 -1.0  1.0 -1.0  1.0  ... -1.0 -1.0 -1.0

      293  294  295  296  297  298  299
0    -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
1     1.0  1.0 -1.0 -1.0 -1.0 -1.0 -1.0
2     1.0  1.0  1.0  1.0  1.0  1.0  1.0
3    -1.0 -1.0 -1.0 -1.0  1.0 -1.0 -1.0
4    -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
...   ...  ...  ...  ...  ...  ...  ...
9495 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
9496  1.0 -1.0 -1.0  1.0  1.0  1.0  1.0
9497 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
9498 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
9499 -1.0  1.0 -1.0  1.0 -1.0 -1.0  1.0

[9500 rows x 300 columns]
```

```
#loading no defense labels
with open("Dataset/y_train_NoDef.pkl", 'rb') as handle:
    no_def_Y = np.array(pickle.load(handle, encoding='latin1'))
handle.close()
print("Tor Traffic Class Labels")
print(no_def_Y)
```

```
Tor Traffic Class Labels
[93 78  6 ... 44 25 26]
```

```
#loading WTFPAD dataset
with open("Dataset/WTFPAD.pkl", 'rb') as handle:
    pad_X = np.array(pickle.load(handle, encoding='latin1'))
handle.close()
pad_X = pad_X[:,0:300]
pad_X = pd.DataFrame(pad_X)
```

```
print("WTFPAD Dataset")
pad_X
```

```
WTFPAD Dataset

      0    1    2    3    4    5    6    7    8    9    ...  290  291  292  \
0     1.0  1.0 -1.0  1.0 -1.0  1.0  1.0 -1.0  1.0 -1.0  ... -1.0  1.0  1.0
1     1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  ... -1.0  1.0  1.0
2     1.0  1.0  1.0  1.0  1.0  1.0 -1.0  1.0  1.0 -1.0  ... -1.0 -1.0 -1.0
3     1.0 -1.0  1.0  1.0 -1.0 -1.0  1.0 -1.0  1.0 -1.0  ... -1.0 -1.0 -1.0
4     1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  ...  1.0 -1.0  1.0
...   ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
9495  1.0  1.0  1.0  1.0 -1.0 -1.0  1.0  1.0 -1.0 -1.0  ...  1.0 -1.0  1.0
9496 -1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  ...  1.0  1.0 -1.0
9497 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0  ... -1.0 -1.0 -1.0
9498 -1.0  1.0 -1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0  1.0  ... -1.0  1.0 -1.0
9499  1.0  1.0 -1.0  1.0 -1.0 -1.0  1.0 -1.0 -1.0 -1.0  ...  0.0  0.0  0.0

      293  294  295  296  297  298  299
0     1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
1     1.0 -1.0  1.0  1.0 -1.0 -1.0 -1.0
2    -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
3    -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
4     1.0 -1.0  1.0 -1.0 -1.0  1.0  1.0
...   ...  ...  ...  ...  ...  ...  ...
9495 -1.0  1.0  1.0 -1.0  1.0  1.0 -1.0
9496  1.0 -1.0  1.0 -1.0  1.0  1.0  1.0
9497 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0
9498  1.0 -1.0  1.0 -1.0  1.0 -1.0  1.0
9499  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[9500 rows x 300 columns]
```

```
#loading WTFPAD class labels
with open("Dataset/label_WTFPAD.pkl", 'rb') as handle:
    pad_Y = np.array(pickle.load(handle, encoding='latin1'))
handle.close()
print("WTFPAD Traffic Class Labels")
print(pad_Y)
```

```
WTFPAD Traffic Class Labels
[ 0  0  0 ... 94 94 94]
```

```
#now apply information Gain features selection algorithm to select relevant
features from dataset
selected_features = SelectKBest(mutual_info_classif, k=50)#using info gain
select 50 features
no_def_X = selected_features.fit_transform(no_def_X, no_def_Y)
info_gain = mutual_info_classif(no_def_X, no_def_Y)
info_gain = pd.Series(info_gain)
info_gain.plot(kind='bar', color='teal', figsize=(12,4))
plt.xlabel("Feature Name")
plt.ylabel("Importance")
plt.title("Information Gain Features Selected")
plt.xticks(rotation=90)
plt.show()


#get correlation features selection
corr_matrix = pd.DataFrame(no_def_X).corr()
corr_matrix.plot(kind='bar', figsize=(12,4))
plt.show()


#calculate fisher score for each features
ranks = fisher_score.fisher_score(no_def_X, no_def_Y)
ranks = pd.Series(ranks)
ranks.plot(kind='bar', color='teal', figsize=(12,4))
plt.xlabel("Feature Name")
plt.ylabel("Importance")
plt.title("Fisher Score Features Selected")
plt.xticks(rotation=90)
plt.show()


#dataset shuffling for both No-defence and PAD
pad_X = selected_features.transform(pad_X)
indices = np.arange(no_def_X.shape[0])
np.random.shuffle(indices)
no_def_X = no_def_X[indices]
no_def_Y = no_def_Y[indices]

indices = np.arange(pad_X.shape[0])
np.random.shuffle(indices)
pad_X = pad_X[indices]
```

```
pad_Y = pad_Y[indices]
print("Dataset Preprocessing Completed")
```

```
Dataset Preprocessing Completed
```

```
#split both No-Defence and PAD dataset into train and test
def_X_train, def_X_test, def_y_train, def_y_test =
train_test_split(no_def_X, no_def_Y, test_size=0.2)
pad_X_train, pad_X_test, pad_y_train, pad_y_test = train_test_split(pad_X,
pad_Y, test_size=0.2)
print("Train & Test Dataset Split")
print("80% audio features used to train algorithms :
"+str(def_X_train.shape[0]))
print("20% audio features used to rest algorithms :
"+str(def_X_test.shape[0]))
```

```
Train & Test Dataset Split
80% audio features used to train algorithms : 7600
20% audio features used to rest algorithms : 1900
```

```
#define global variables to save accuracy and other metrics
accuracy = []
precision = []
recall = []
fscore = []
train_time = []
test_time = []
```

```
#function to calculate accuracy and other metrics
def calculateMetrics(algorithm, predict, y_test, training_time,
testing_time):
    a = accuracy_score(y_test,predict)*100
    p = precision_score(y_test, predict,average='macro') * 100
    r = recall_score(y_test, predict,average='macro') * 100
    f = f1_score(y_test, predict,average='macro') * 100
    train_time.append(training_time)
    test_time.append(testing_time)
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    print(algorithm+" Accuracy  :  "+str(a))
    print(algorithm+" Precision : "+str(p))
    print(algorithm+" Recall    : "+str(r))
```

```
    print(algorithm+" FScore    : "+str(f))


#train KNN algorithm on No-Defence Dataset
 #defining KNN tuning parameters
 tuning_param = {'n_neighbors' : [1], 'p' : [1]}
 knn_no_defence = GridSearchCV(KNeighborsClassifier(), tuning_param,
cv=5)#defining svm with tuned parameters
 start = timeit.default_timer()
 knn_no_defence.fit(no_def_X, no_def_Y)#now train KNN
 end = timeit.default_timer()
 predict = knn_no_defence.predict(def_X_test) #perfrom prediction on test
data
 end1 = timeit.default_timer()
 calculateMetrics("Original (No Defence) KNN", predict, def_y_test, (end -
start), (end1 - end))
```

```
Original (No Defence) KNN Accuracy  :  86.21052631578947
 Original (No Defence) KNN Precision : 91.88225876274299
 Original (No Defence) KNN Recall    : 86.29179025583143
 Original (No Defence) KNN FScore    : 86.92316349239
```

```
#train Random Forest algorithm on No-Defence Dataset
 #defining Random Forest tuning parameters
 tuning_param = {'n_estimators' : [90]}
 rf_no_defence = GridSearchCV(RandomForestClassifier(), tuning_param,
cv=5)#defining svm with tuned parameters
 start = timeit.default_timer()
 rf_no_defence.fit(no_def_X, no_def_Y)#now train KNN
 end = timeit.default_timer()
 predict = rf_no_defence.predict(def_X_test) #perfrom prediction on test data
 end1 = timeit.default_timer()
 calculateMetrics("Original (No Defence) Random Forest", predict, def_y_test,
(end - start), (end1 - end))
```

```
Original (No Defence) Random Forest Accuracy  :  86.57894736842105
 Original (No Defence) Random Forest Precision : 92.34651358387856
 Original (No Defence) Random Forest Recall    : 86.59348583925859
 Original (No Defence) Random Forest FScore    : 87.33594643664601
```

```
#train SVM algorithm on No-Defence Dataset
 #defining SVM tuning parameters
 tuning_param = {'C' : [100], 'kernel': ['rbf']}
 svm_no_defence = GridSearchCV(svm.SVC(), tuning_param, cv=5)#defining svm
```

```
with tuned parameters
start = timeit.default_timer()
svm_no_defence.fit(no_def_X, no_def_Y)#now train KNN
end = timeit.default_timer()
predict = svm_no_defence.predict(def_X_test) #perfrom prediction on test
data
end1 = timeit.default_timer()
calculateMetrics("Original (No Defence) SVM", predict, def_y_test, (end -
start), (end1 - end))
```

```
Original (No Defence) SVM Accuracy  :  86.89473684210526
Original (No Defence) SVM Precision : 93.28713050104696
Original (No Defence) SVM Recall    : 86.92904959900186
Original (No Defence) SVM FScore    : 87.69026848647484
```

```
#train KNN algorithm on WTFPAD Dataset
#defining KNN tuning parameters
tuning_param = {'n_neighbors' : [1], 'p' : [1]}
knn_pad = GridSearchCV(KNeighborsClassifier(), tuning_param, cv=5)#defining
svm with tuned parameters
start = timeit.default_timer()
knn_pad.fit(pad_X[0:8000], pad_Y[0:8000])#now train KNN
end = timeit.default_timer()
predict = knn_pad.predict(pad_X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
calculateMetrics("WTFPAD Defence KNN", predict, pad_y_test, (end - start),
(end1 - end))
```

```
WTFPAD Defence KNN Accuracy  :  84.15789473684211
WTFPAD Defence KNN Precision : 84.48796077672786
WTFPAD Defence KNN Recall    : 84.04800618817644
WTFPAD Defence KNN FScore    : 83.82246547032027
```

```
#train Random Forest algorithm on WTFPAD Dataset
#defining Random Forest tuning parameters
tuning_param = {'n_estimators' : [100]}
rf_pad = GridSearchCV(RandomForestClassifier(), tuning_param, cv=5)#defining
svm with tuned parameters
start = timeit.default_timer()
rf_pad.fit(pad_X[0:8000], pad_Y[0:8000])#now train KNN
end = timeit.default_timer()
predict = rf_pad.predict(pad_X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
```

```
calculateMetrics("WTFPAD Defence Random Forest", predict, pad_y_test, (end - start), (end1 - end))
```

```
WTFPAD Defence Random Forest Accuracy  :  84.10526315789474
WTFPAD Defence Random Forest Precision : 84.34746732801047
WTFPAD Defence Random Forest Recall    : 83.9566813849403
WTFPAD Defence Random Forest FScore    : 83.84138636105763
```

```
#train SVM algorithm on WTFPAD Dataset
#defining SVM tuning parameters
tuning_param = {'C' : [100], 'kernel': ['rbf']}
svm_pad = GridSearchCV(svm.SVC(), tuning_param, cv=5)#defining svm with tuned parameters
start = timeit.default_timer()
svm_pad.fit(pad_X, pad_Y)#now train KNN
end = timeit.default_timer()
predict = svm_pad.predict(pad_X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
calculateMetrics("WTFPAD Defence SVM", predict, pad_y_test, (end - start), (end1 - end))
```

```
WTFPAD Defence SVM Accuracy  :  99.42105263157895
WTFPAD Defence SVM Precision : 99.5363010246185
WTFPAD Defence SVM Recall    : 99.37735665514782
WTFPAD Defence SVM FScore    : 99.42021105616358
```

```
pad_data = pad_X[0:482]
os_data = pd.read_csv("Dataset/OS.csv")#read OS onion service dataset
os_data.fillna(0, inplace = True)
os_data = os_data.values
pad_data = pad_data[:,0:6]
os_data = os_data[:,1:7]
X = []
Y = []
for i in range(len(pad_data)):#merge pad and OS data
    X.append(pad_data[i])
    Y.append(0)
for i in range(len(os_data)):
    X.append(os_data[i])
    Y.append(1)
X = np.asarray(X)
Y = np.asarray(Y)
print("OS & WTFPAD Data merging completed")
```

```
OS & WTFPAD Data merging completed
```

```
def calculateMetric(algorithm, predict, y_test, training_time, testing_time):
    labels = ['WPPAD', 'Original']
    a = accuracy_score(y_test,predict)*100
    p = precision_score(y_test, predict,average='macro') * 100
    r = recall_score(y_test, predict,average='macro') * 100
    f = f1_score(y_test, predict,average='macro') * 100
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    train_time.append(training_time)
    test_time.append(testing_time)
    print(algorithm+" Accuracy  :  "+str(a))
    print(algorithm+" Precision : "+str(p))
    print(algorithm+" Recall    : "+str(r))
    print(algorithm+" FScore    : "+str(f))
    conf_matrix = confusion_matrix(y_test, predict)
    plt.figure(figsize =(6, 3))
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels,
annot = True, cmap="viridis" ,fmt ="g");
    ax.set_ylim([0,len(labels)])
    plt.title(algorithm+" Confusion matrix")
    plt.xticks(rotation=90)
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()

indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X = X[indices]
Y = Y[indices]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
#train KNN algorithm on WTFPAD Dataset
#defining KNN tuning parameters
tuning_param = {'n_neighbors' : [4], 'p' : [1]}
knn = GridSearchCV(KNeighborsClassifier(), tuning_param, cv=5)#defining svm
with tuned parameters
start = timeit.default_timer()
knn.fit(X_train, y_train)#now train KNN
end = timeit.default_timer()
predict = knn.predict(X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
```

```
calculateMetric("WTFPAD KNN Top 6 Features", predict, y_test, (end - start),
(end1 - end))
```

```
WTFPAD KNN Top 6 Features Accuracy  :  97.92746113989638
WTFPAD KNN Top 6 Features Precision : 97.9247311827957
WTFPAD KNN Top 6 Features Recall    : 97.9247311827957
WTFPAD KNN Top 6 Features FScore    : 97.9247311827957
```

```
#train Random Forest algorithm on WTFPAD Dataset
#defining Random Forest tuning parameters
tuning_param = {'n_estimators' : [5], 'max_depth': [5]}
rf = GridSearchCV(RandomForestClassifier(), tuning_param, cv=5)#defining svm
with tuned parameters
start = timeit.default_timer()
rf.fit(X_train, y_train)#now train KNN
end = timeit.default_timer()
predict = rf.predict(X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
calculateMetric("WTFPAD Random Forest Top 6 Features", predict, y_test, (end
- start), (end1 - end))
```

```
WTFPAD Random Forest Top 6 Features Accuracy  :  98.96373056994818
WTFPAD Random Forest Top 6 Features Precision : 99.01960784313727
WTFPAD Random Forest Top 6 Features Recall    : 98.9247311827957
WTFPAD Random Forest Top 6 Features FScore    : 98.96147223417994
```

```
#train SVM algorithm on WTFPAD Dataset
#defining SVM tuning parameters
tuning_param = {'C' : [7], 'kernel': ['rbf']}
svm_cls = GridSearchCV(svm.SVC(), tuning_param, cv=5)#defining svm with
tuned parameters
start = timeit.default_timer()
svm_cls.fit(X_train, y_train)#now train KNN
end = timeit.default_timer()
predict = svm_cls.predict(X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
calculateMetric("WTFPAD SVM Top 6 Features", predict, y_test, (end - start),
(end1 - end))
```

```
WTFPAD SVM Top 6 Features Accuracy  :  97.92746113989638
WTFPAD SVM Top 6 Features Precision : 97.93814432989691
```

```
WTFPAD SVM Top 6 Features Recall    : 98.0
WTFPAD SVM Top 6 Features FScore    : 97.92696025778731
```

```
#extension XGBoost training on merge dataset
ab_cls = AdaBoostClassifier(n_estimators=100)
start = timeit.default_timer()
ab_cls.fit(X_train, y_train)#now train KNN
end = timeit.default_timer()
predict = ab_cls.predict(X_test) #perfrom prediction on test data
end1 = timeit.default_timer()
calculateMetric("WTFPAD Extension AdaBoost Top 6 Features", predict, y_test,
(end - start), (end1 - end))
```

```
WTFPAD Extension AdaBoost Top 6 Features Accuracy  :  100.0
WTFPAD Extension AdaBoost Top 6 Features Precision : 100.0
WTFPAD Extension AdaBoost Top 6 Features Recall    : 100.0
WTFPAD Extension AdaBoost Top 6 Features FScore    : 100.0
```

```
#showing all algorithms with scenario A and B performance values
columns = ["Algorithm Name","Precison","Recall","FScore","Accuracy",
"Training Time", "Testing Time"]
values = []
algorithm_names = ["KNN No Defence", "Random Forest No Defence", "SVM No
Defence", "KNN WTFPAD Defence", "Random Forest WTFPAD Defence",
                   "SVM WTFPAD Defence", "KNN WTFPAD Top 6 Features", "Random
Forest WTFPAD Top 6 Features",
                    "SVM WTFPAD Top 6 Features", "Extension AdaBoost Top 6
Fatures"]
for i in range(len(algorithm_names)):

values.append([algorithm_names[i],precision[i],recall[i],fscore[i],accuracy[i
], train_time[i], test_time[i]])
temp = pd.DataFrame(values,columns=columns)
temp
```

```
                        Algorithm Name    Precision      Recall      FScore
\
0                        KNN No Defence   91.882259   86.291790   86.923163
1             Random Forest No Defence   92.346514   86.593486   87.335946
2                        SVM No Defence   93.287131   86.929050   87.690268
3                   KNN WTFPAD Defence   84.487961   84.048006   83.822465
```

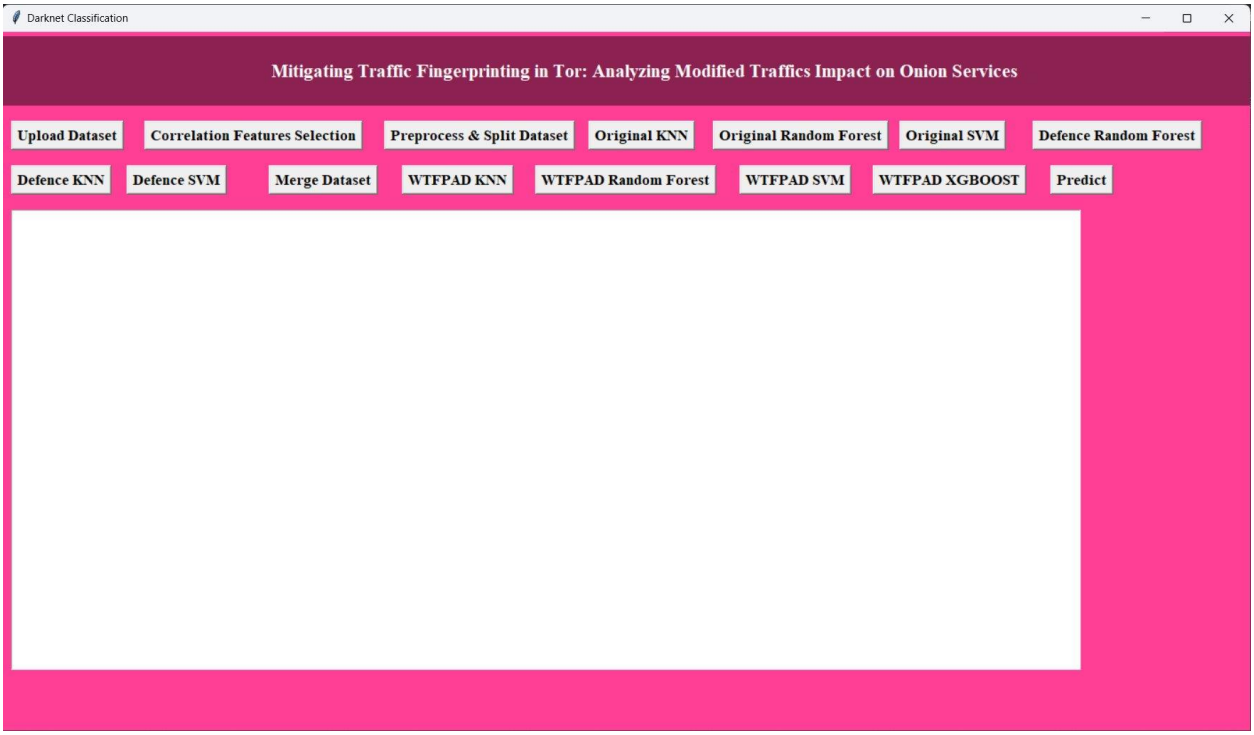
```
4         Random Forest WTFPAD Defence   84.347467   83.956681   83.841386
5                   SVM WTFPAD Defence   99.536301   99.377357   99.420211
6            KNN WTFPAD Top 6 Features   97.924731   97.924731   97.924731
7  Random Forest WTFPAD Top 6 Features   99.019608   98.924731   98.961472
8            SVM WTFPAD Top 6 Features   97.938144   98.000000   97.926960
9     Extension AdaBoost Top 6 Features  100.000000  100.000000  100.000000

     Accuracy  Training Time  Testing Time
0   86.210526       4.227495      0.964663
1   86.578947      13.886483      0.199665
2   86.894737     120.662982      4.770831
3   84.157895       3.081830      0.808509
4   84.105263      14.787587      0.229543
5   99.421053     123.778263      4.807250
6   97.927461       0.074018      0.012011
7   98.963731       0.150688      0.001739
8   97.927461       0.039732      0.001213
9  100.000000       0.504562      0.027284
```

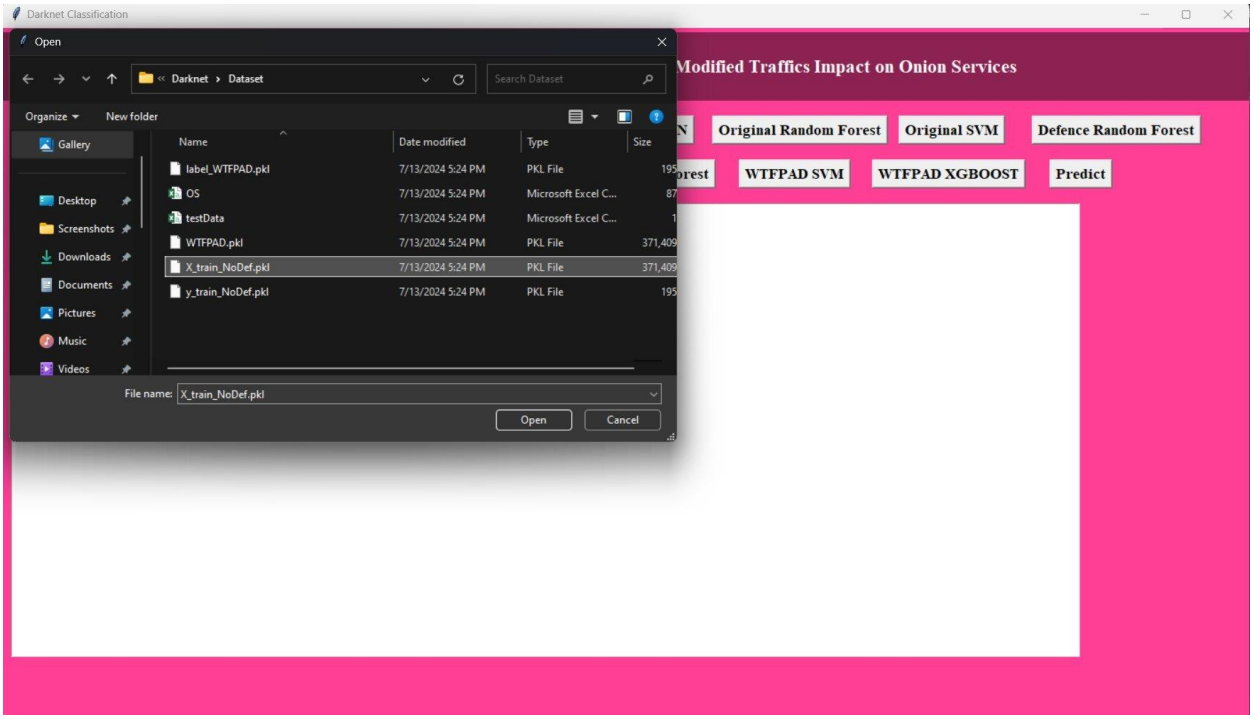
```
#read test data features
testData = pd.read_csv("Dataset/testData.csv")
testData = testData.values
#using extension AdaBoost classify network type
predict = ab_cls.predict(testData)
labels = ['Tor Service', 'Onion Service']
#loop and display predicted values
for i in range(len(testData)):
    print("Test Network Data : "+str(testData[i])+" Predicted As ====>
"+labels[predict[i]]+"\n")
```

```
Test Network Data : [-1. -1. -1. -1. -1. 1.] Predicted As ====> Tor Service

Test Network Data : [0.84823504 0.96453901 0.85238095 0.97142857 0.94285714
0.95652174] Predicted As ====> Onion Service

Test Network Data : [0.90715681 0.97101449 0.9047619  0.95714286 0.95714286
0.96402878] Predicted As ====> Onion Service

Test Network Data : [ 1. -1. -1. -1. -1.  1.] Predicted As ====> Tor Service

Test Network Data : [0.90392597 0.94366197 0.9        0.95714286 0.92857143
0.92857143] Predicted As ====> Onion Service

Test Network Data : [0.9337613  0.98591549 0.95714286 1.         1.
0.98591549] Predicted As ====> Onion Service
```

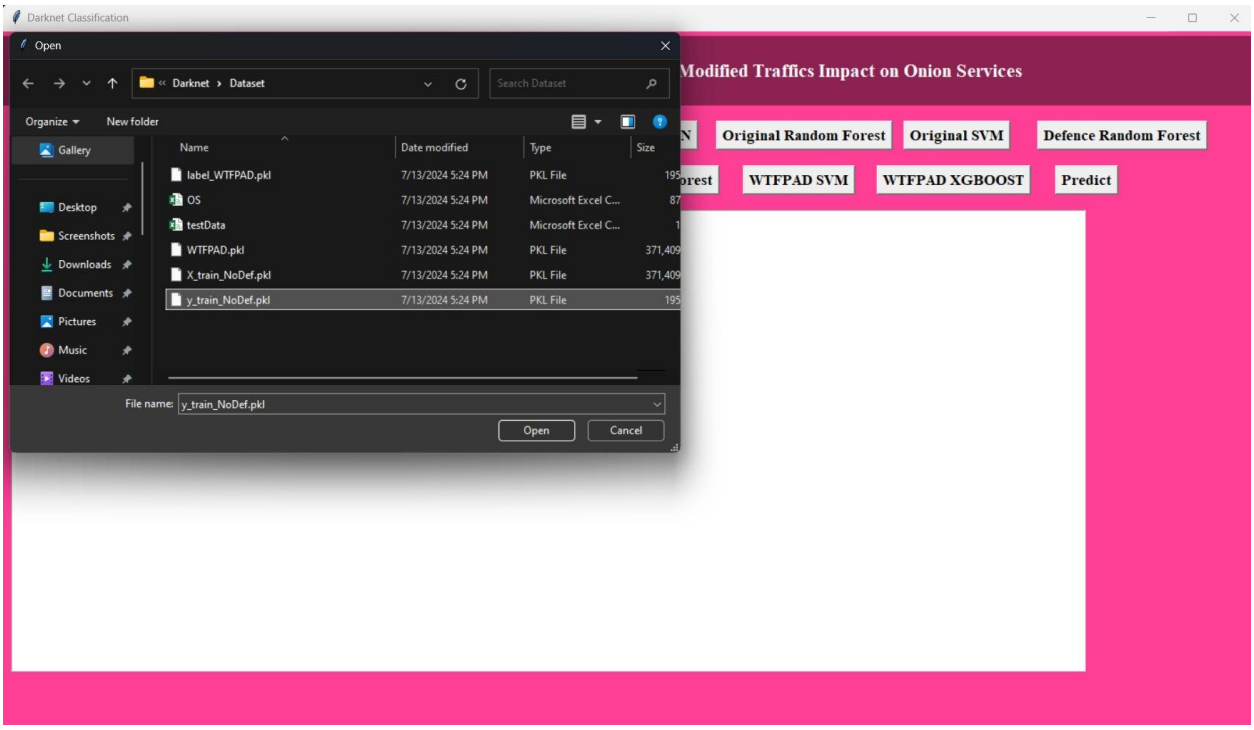
```
Test Network Data : [ 1.  1.  1. -1. -1. -1.] Predicted As ====> Tor Service

Test Network Data : [0.7728585  0.83076923 0.72857143 0.77142857 0.81428571
0.85714286] Predicted As ====> Onion Service

Test Network Data : [-1.  1. -1.  1. -1. -1.] Predicted As ====> Tor Service

Test Network Data : [ 1.  1.  1. -1. -1. -1.] Predicted As ====> Tor Service
```

CHAPTER 8
RESULTS AND DISCUSSION

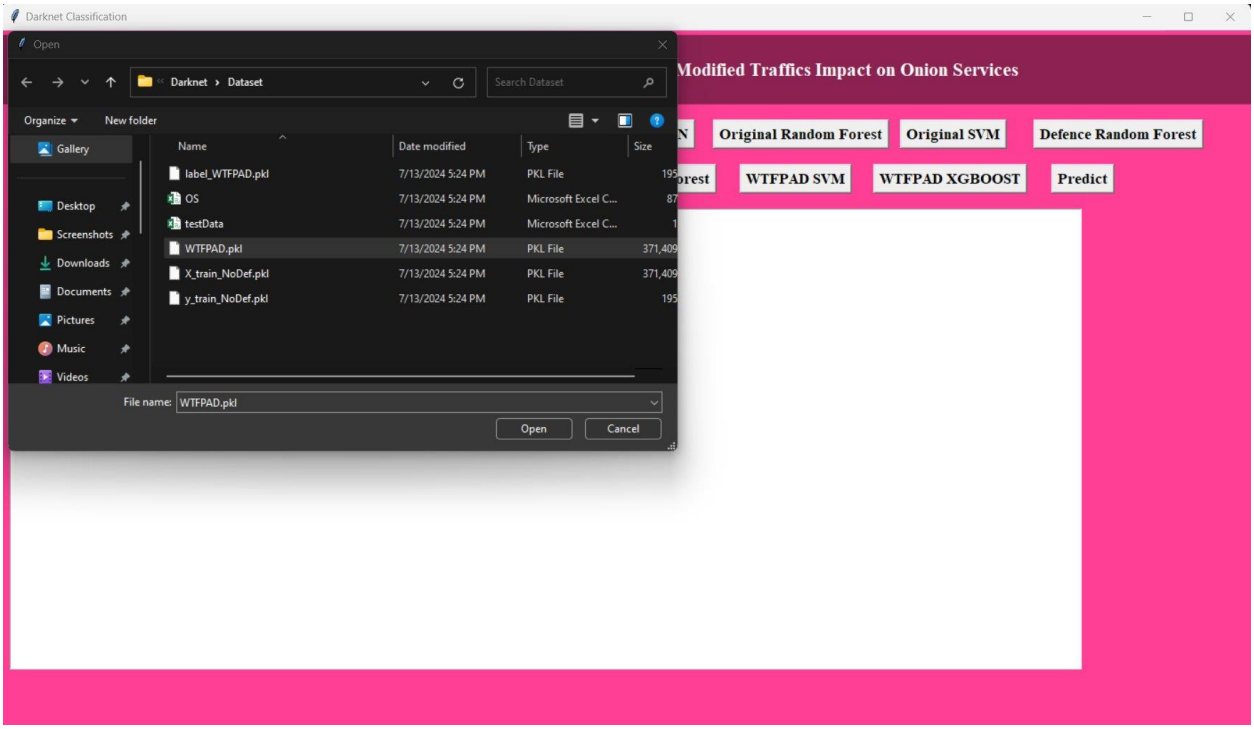




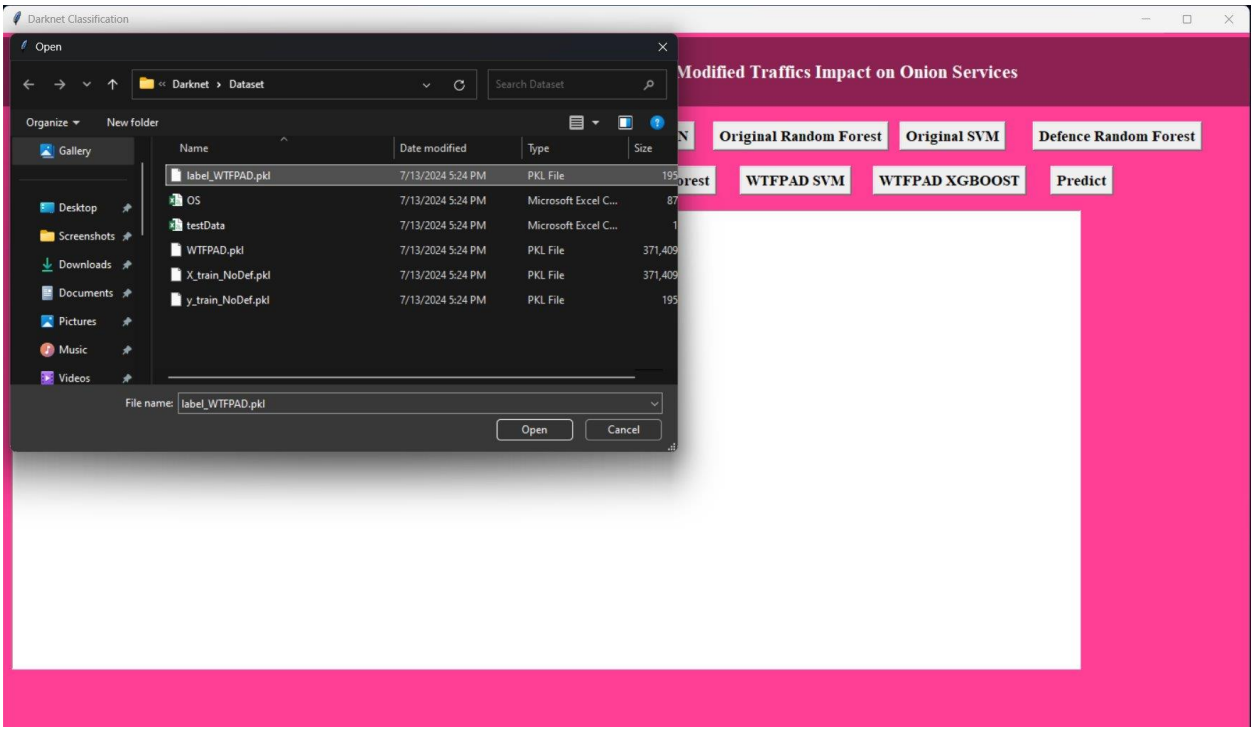
In the above screen loading and displaying NO DEFENCE original TOR database.



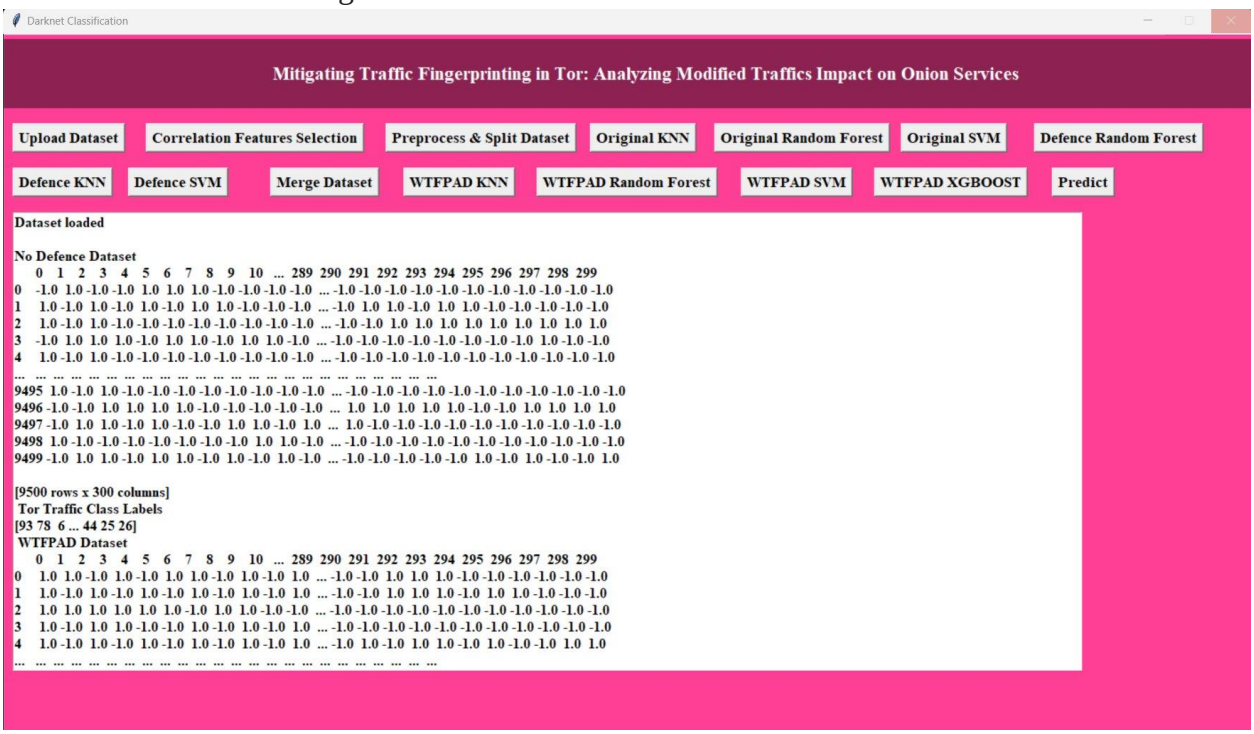
In the above screen loading Original TOR class labels dataset.



In the above screen loading and displaying WTFPAD dataset.

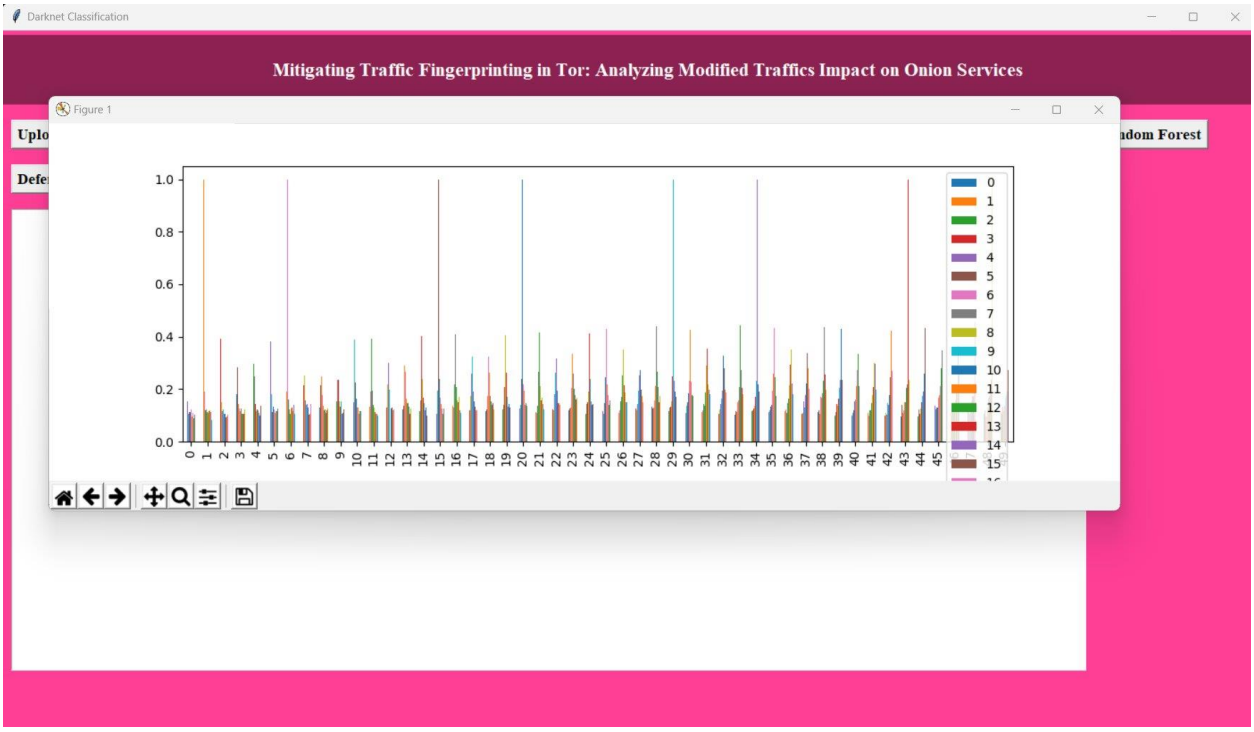


In the above screen loading WTFPAD class labels.

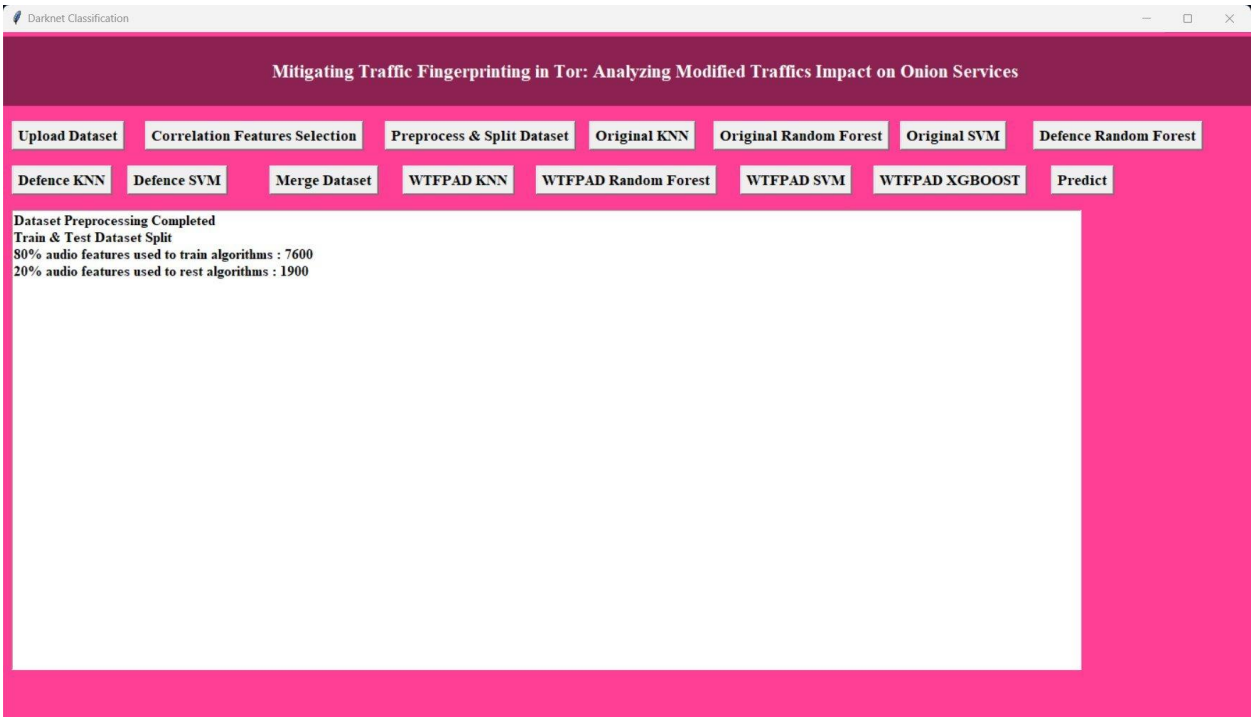


In the above screen loading WTFPAD class labels and then defining code to select relevant 50 features from dataset using Information Gain algorithm and below is the selected features importance graph.

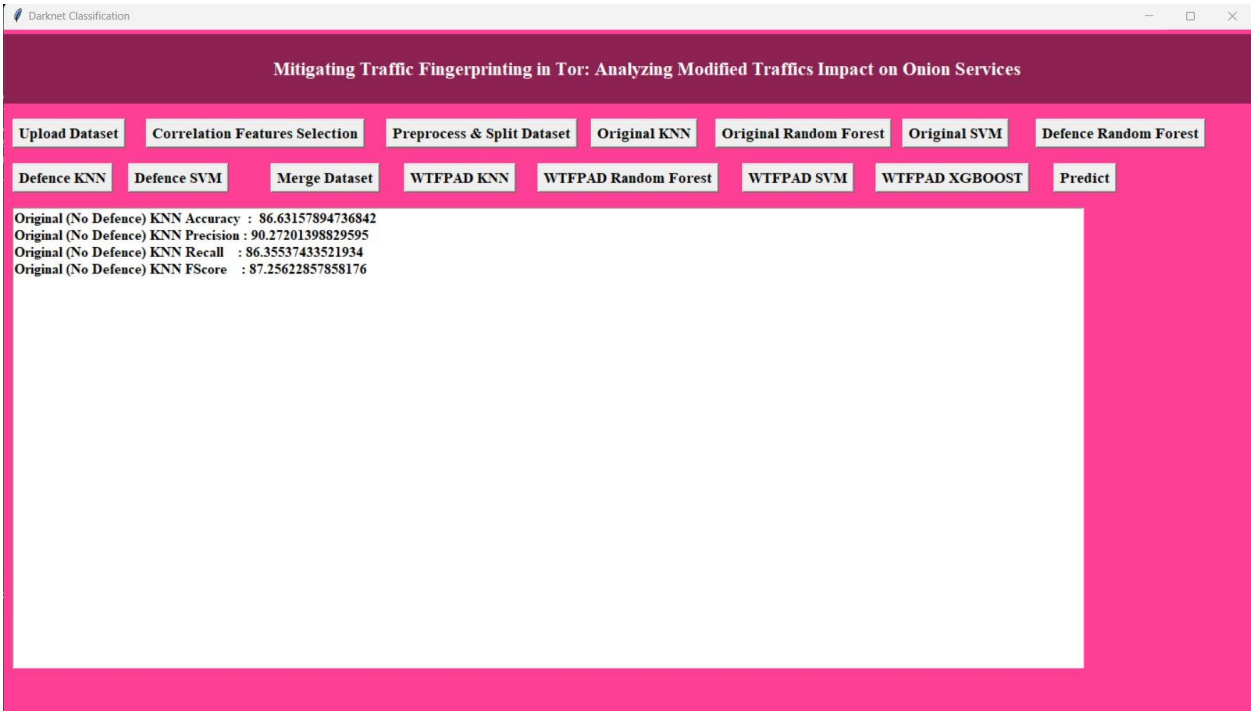
In the above graph x-axis represents features names and y-axis represents importance or relevant score value.



In the above screen calculating features importance using Correlation Coefficient.



In the above screen applying dataset processing like shuffling and splitting dataset into train and test.

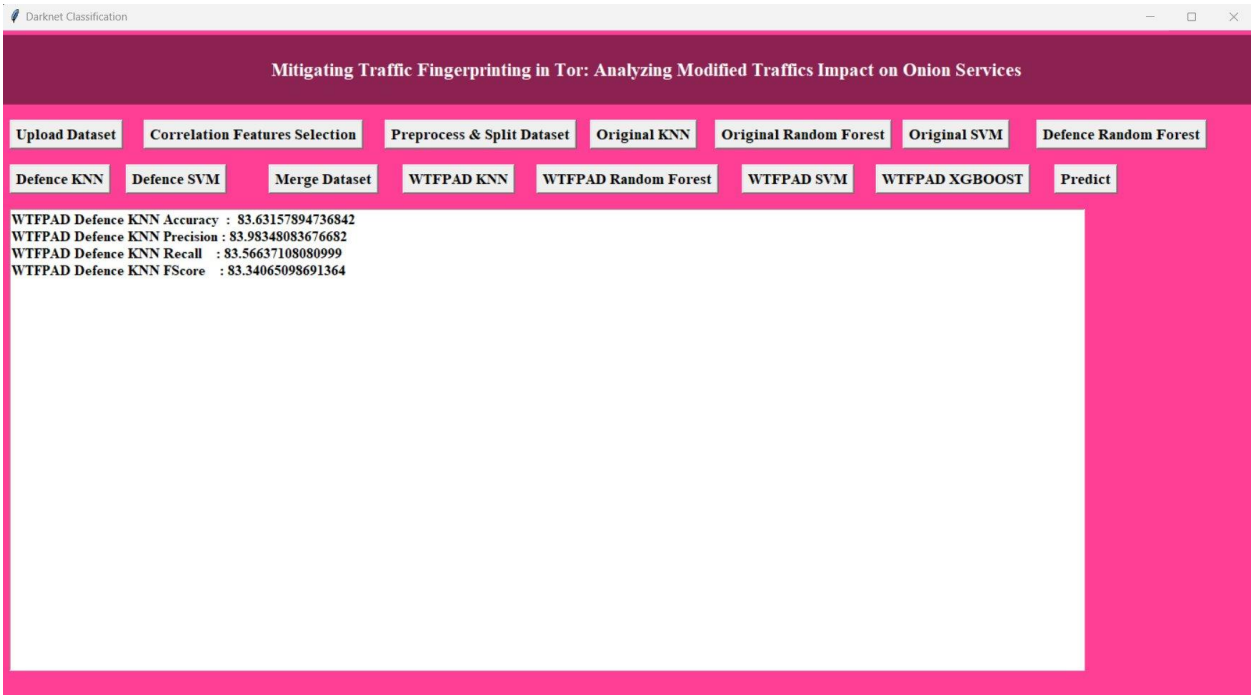


In the above screen training KNN on original No Defence dataset and then KNN got 86% accuracy and in above screen KNN optimize by using hyper parameters.

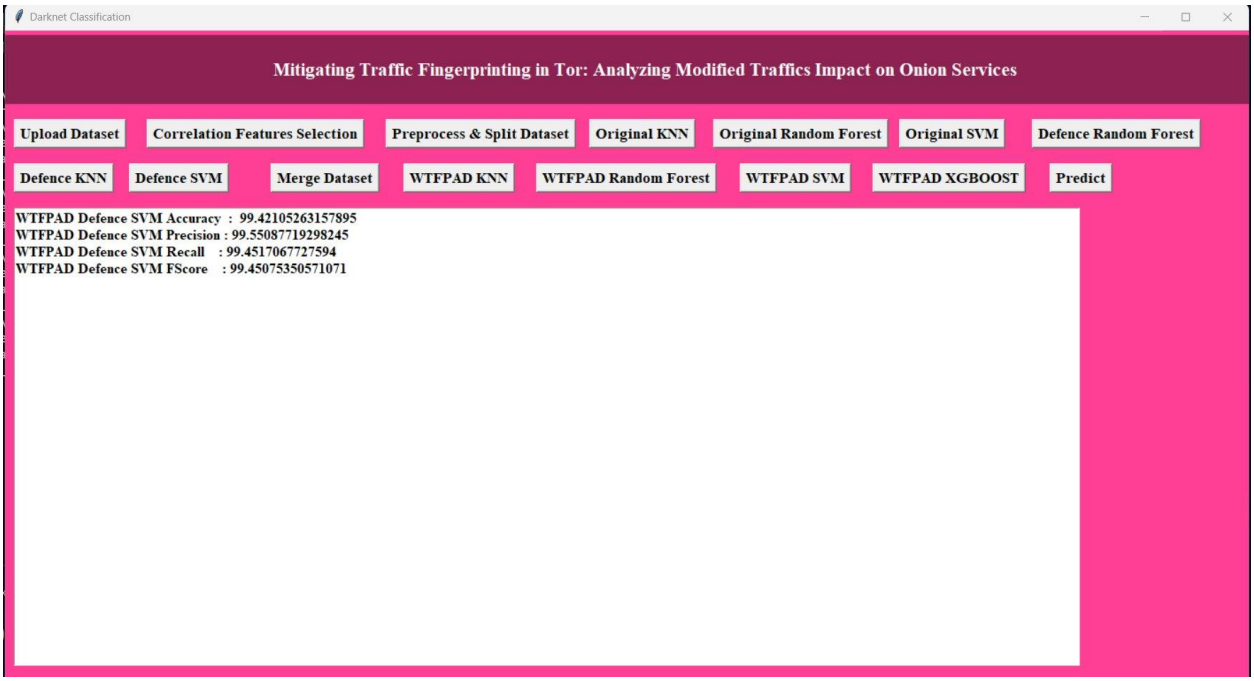
In the above screen Random Forest got 86.7% accuracy on original no defense dataset.

Then SVM 90.99% which is higher than KNN and Random Forest on Original No Defence dataset.

In the above screen Random got 83.2% accuracy.

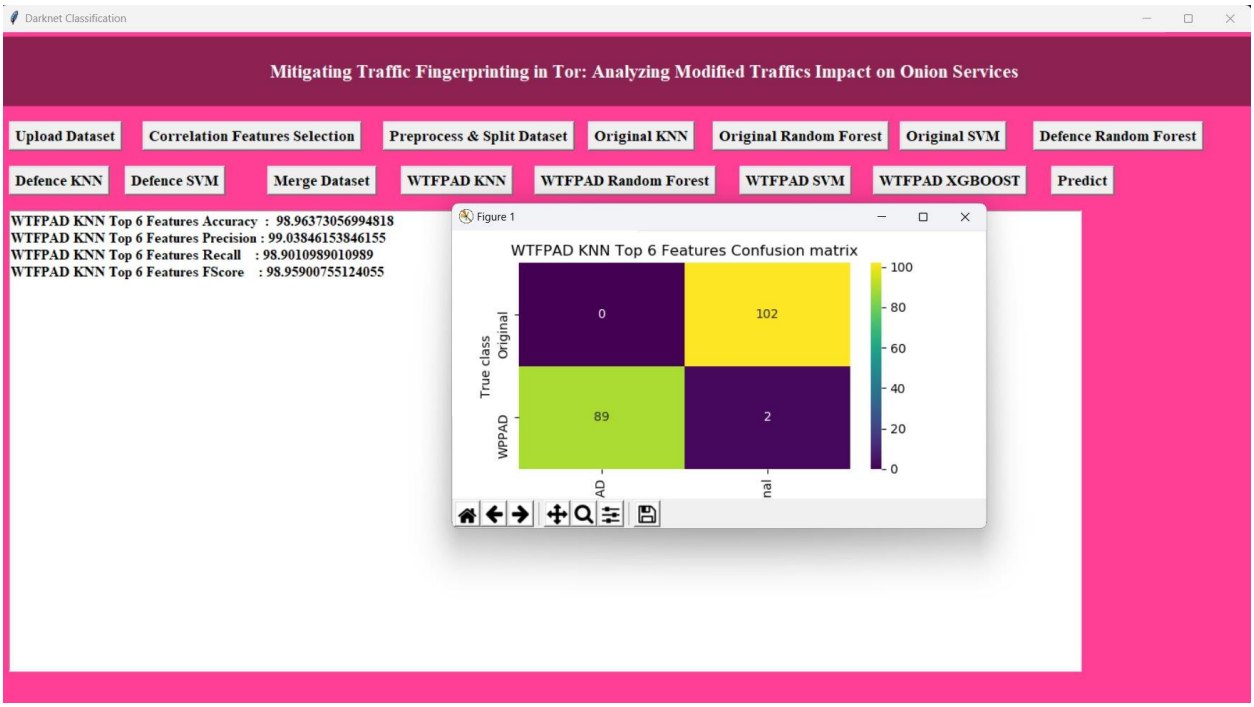


In the above screen KNN got 83.5% accuracy on WTFPAD dataset.

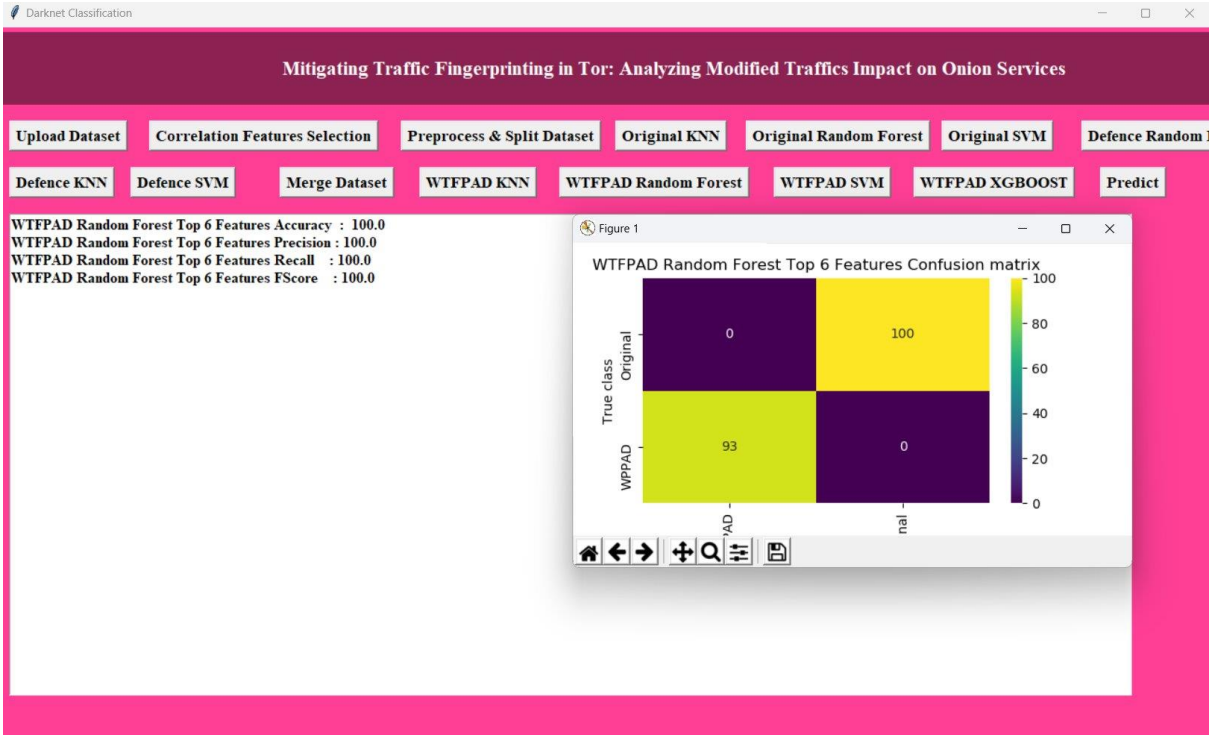


In the above screen SVM got 99% accuracy on WTFPAD dataset which is higher than KNN and Random Forest.

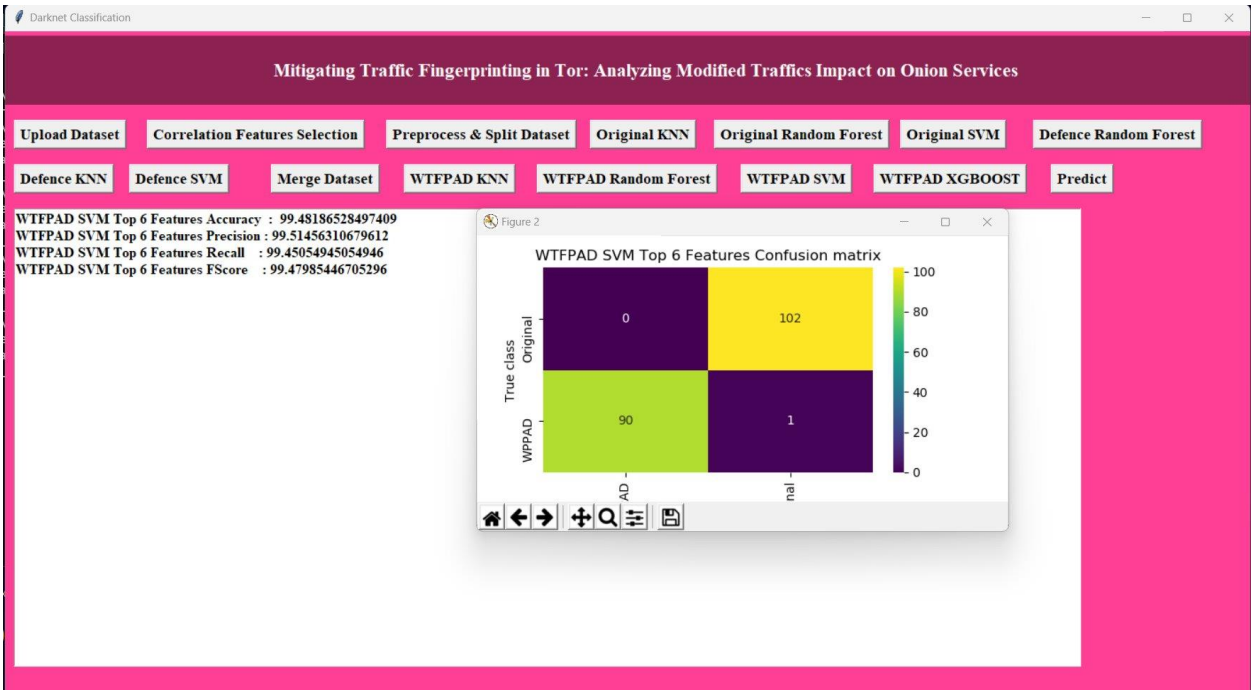
In the above screen merging WTFPAD and OS Onion services dataset for TOR or onion classification.



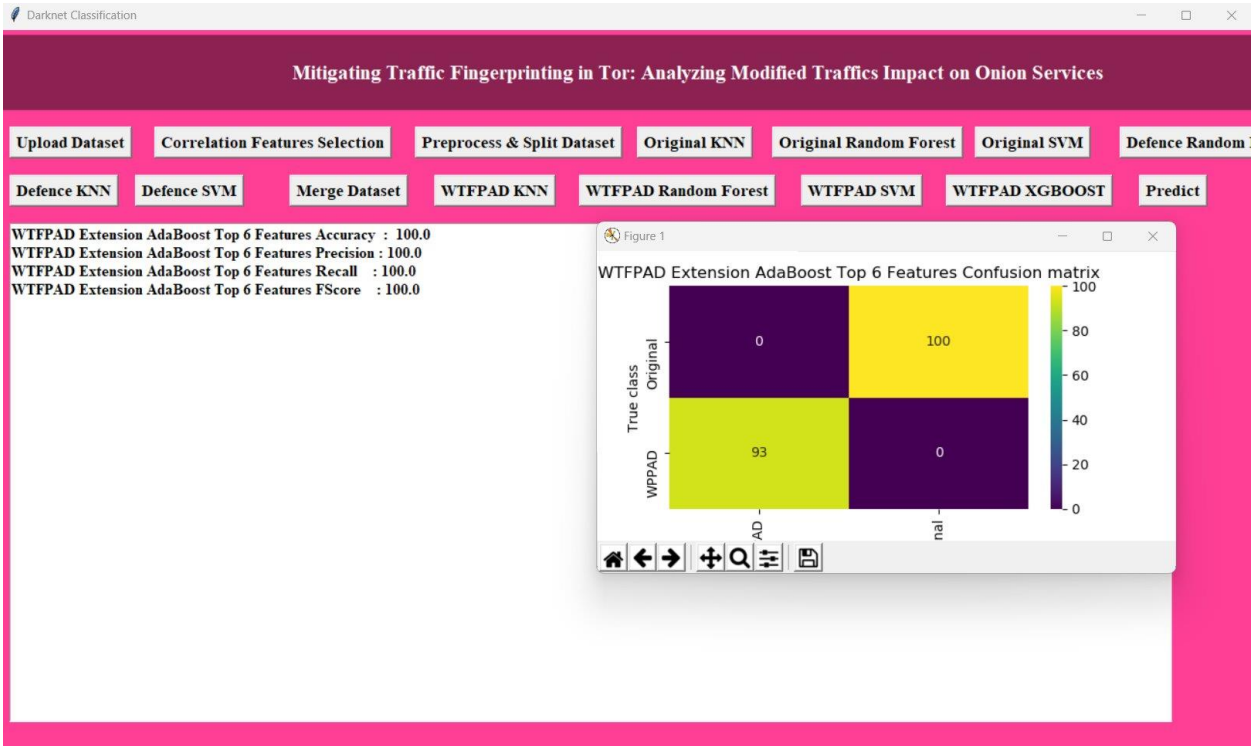
In the above screen training KNN merged with TOR and OS TOP 6 features selected dataset and then after training KNN got 99.0% accuracy. In confusion matrix graph x-axis represents predicted Original OS service label and WTFPAD service label and y-axis represents true labels. Yellow and light green boxes contain correct prediction count and blue boxes contain incorrect prediction count.



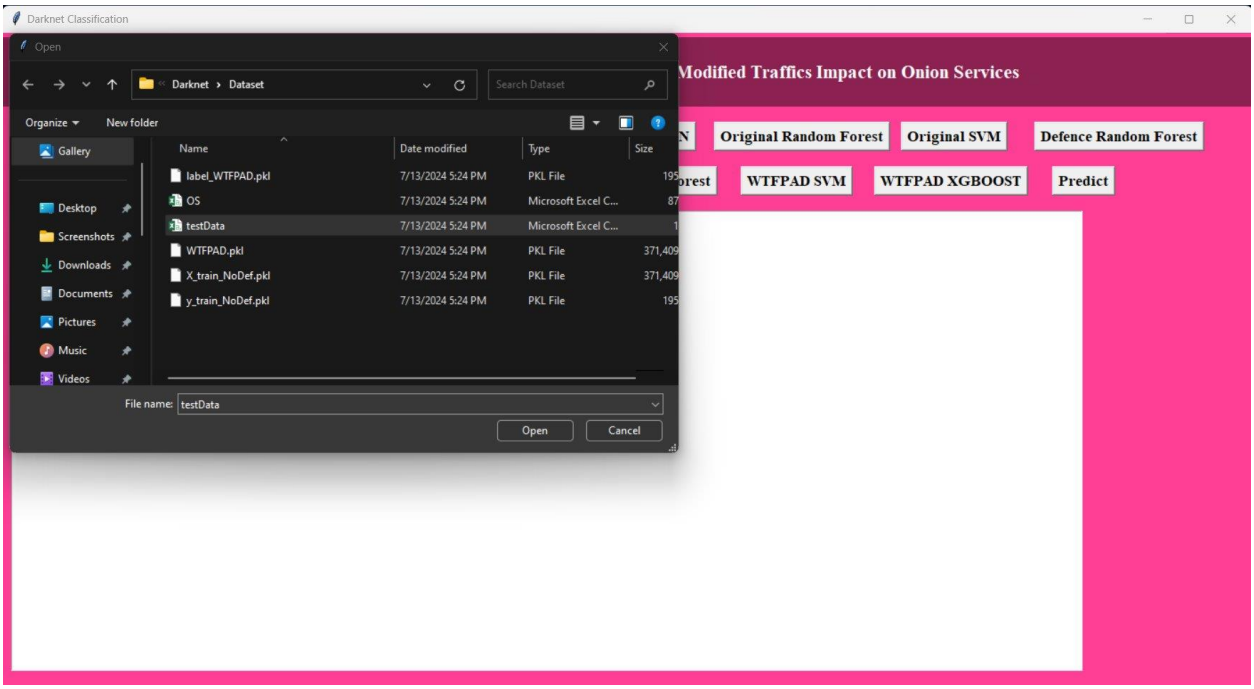
In the above screen Random Forest got 100% accuracy.



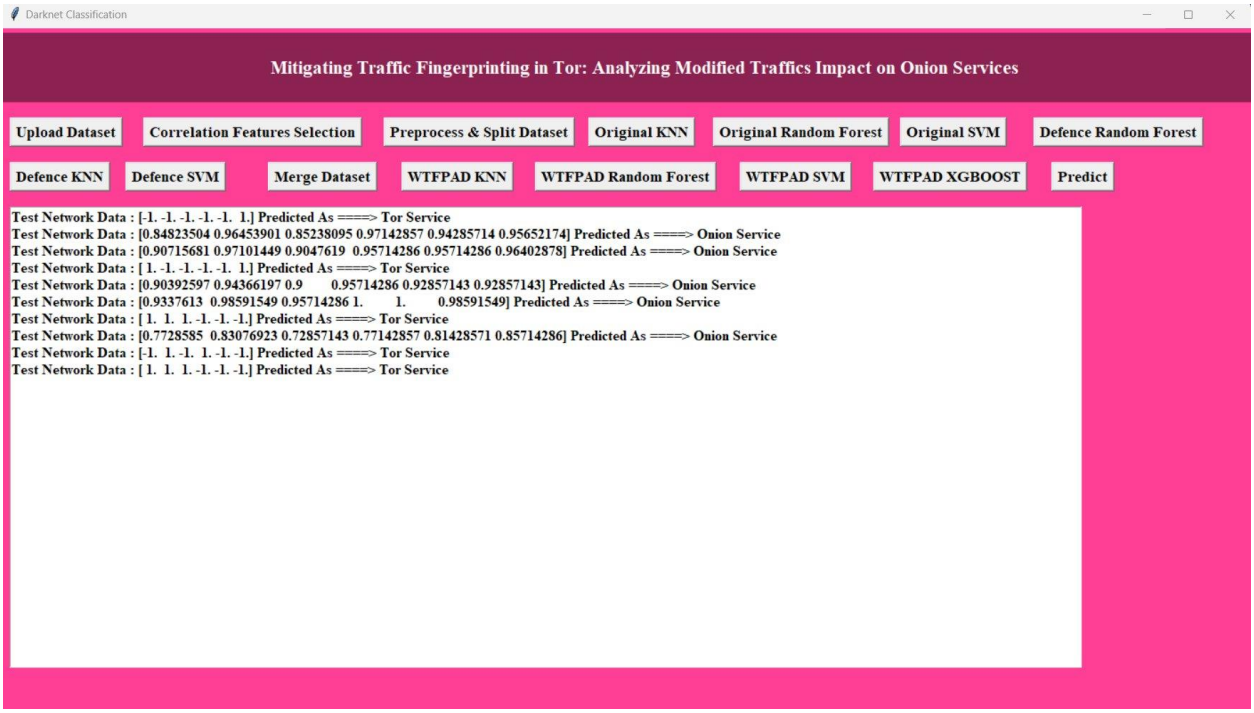
In the above screen SVM got 99.45% accuracy.



In the above screen training extension ADABOOST algorithm and then after training extension got 100% accuracy.



In the above screen uploading test data.



In the above screen reading test Network data and then using extension ADABOOST algorithm classifying network traffic as Tor or Onion services. In the square bracket we can see test data and after arrow symbol can see predicted service type.

CHAPTER 9
CONCLUSION

In this work, we answered three research questions focused on Onion Service traffic classification. We evaluated the applicability of supervised machine learning models to classify Onion Service traffic from other Tor traffic. We extracted fifty features from each traffic trace and used that feature set as input to the machine learning classifiers. Our results showed that KNN, RF, and SVM classifiers can distinguish Onion Service traffic from Tor traffic with 99% accuracy. Then, we tried to identify whether state-of-the-art Website Fingerprinting defences affect the classifiability of Tor traffic. These defences introduce different modifications to try and obfuscate information leakage from traffic, and we evaluated how those changes affect the Onion Service traffic classification. Finally, Machine learning models can effectively classify Onion Service traffic, even with modified Tor traffic.

CHAPTER 10
REFERENCES

[1] R. Dingledine, N. Mathewson, and P. Syverson, 'Tor: The second generation onion router,' in Proc. 13th USENIX Secur. Symp. (SSYM), San Diego, CA, USA, Aug. 2004, pp. 303–320.
[2] M. Al Sabah, K. Bauer, and I. Goldberg, ''Enhancing Tor's performance using real-time traffic classification,' in Proc. ACM Conf. Comput. Commun. Secur. (CCS), New York, NY, USA, Oct. 2012, pp. 73–84.
[3] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, ''Characterization of Tor traffic using time based features,'' in Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy (ICISSP), Porto, Portugal, Feb. 2017, pp. 253–262.
[4] M. Kim and A. Anpalagan, ''Tor traffic classification from raw packet header using convolutional neural network,'' in Proc. 1st IEEE Int. Conf. Know. Innov. Invention (ICKII), Jeju Island, South Korea, Jul. 2018, pp. 187–190.
[5] G. He, M. Yang, J. Luo, and X. Gu, ''Inferring application type information from Tor encrypted traffic,'' in Proc. 2nd Int. Conf. Adv. Cloud Big Data (CBD), Washington, DC, USA, Nov. 2014, pp. 220–227.
[6] A. Montieri, D. Ciuonzo, G. Aceto, and A. Pescapé, ''Anonymity services tor, I2P, JonDonym: Classifying in the dark (web),'' IEEE Trans. Dependable Secure Comput., vol. 17, no. 3, pp. 662–675, May 2020.