**HAZARD IDENTIFICATION AND DETECTION USING MACHINE LEARNING**

*A project submitted by*

**B.V.S. PAVAN KUMAR**

**(Reg. No: 321206416001)**

*In the partial fulfillment of the requirements for the award of the degree of*

**MASTER OF TECHNOLOGY**

in

**INFORMATION TECHNOLOGY**

Under the guidance of

**Dr. A. MARY SOWJANYA**

**Associate Professor**



**DEPARTMENT OF INFORMATION TECHNOLOGY AND COMPUTER APPLICATIONS**

**ANDHRA UNIVERSITY COLLEGE OF ENGINEERING (A)**

**ANDHRA UNIVERSITY**

**VISAKHAPATNAM – 530003**

**(2021 – 2023)**

**DEPARTMENT OF INFORMATION TECHNOLOGY AND COMPUTER APPLICATIONS**

**ANDHRA UNIVERSITY COLLEGE OF ENGINEERING (A)**

**ANDHRA UNIVERSITY**

**VISAKHAPATNAM – 530003**



**CERTIFICATE**

This is to certify that the project report entitled “**HAZARD IDENTIFICATION AND DETECTION USING MACHINE LEARNING**”, is the bonafide work carried out by **B.V.S. PAVAN KUMAR** with Reg.No: 321206416001 a student of MTech in ANDHRA UNIVERSITY COLLEGE OF ENGINEERING(A), ANDHRA UNIVERSITY, VISAKHAPATNAM, during the year 2021- 2023, in partial fulfillment of requirements for the award of degree of **MASTER OF TECHNOLOGY.**

**Project Guide**

**Dr. A. MARY SOWJANYA**

**Associate Professor**

**Dept of CS & SE**

**Head of the Department**

**Prof. KUNJAM NAGESWARA RAO**

**Professor**

**Dept of IT & CA**

**DECLARATION**

I declare that the report entitled “**HAZARD IDENTIFICATION AND DETECTION USING MACHINE LEARNING**” has been done by me in partial fulfillment of requirement for the award of degree of “**Master of Technology**”, during the academic year 2021 – 2023 under the guidance of “**Dr. A.MARY SOWJANYA”**, Department of Computer Science and Systems Engineering, Andhra University College of Engineering(A), Andhra University, Visakhapatnam. I hereby, declare that this project work has not been submitted to any other universities/institutions for the award of any degree.

**PLACE: VISAKHAPATNAM**

**DATE:**

**B.V.S. PAVAN KUMAR**

**Reg.No: 321206416001**

**ACKNOWLEDGEMENT**

I would like to express my deepest appreciation to **Dr. A. MARY SOWJANYA,** my guide, for her exceptional supervision, unwavering support, and invaluable assistance. Her guidance played a pivotal role in ensuring the smooth execution of the project. Without her backing, this project would not have come to fruition.

I extend my sincere gratitude to **Prof. K. Nageswara Rao**, the Head of the Department of Information Technology and Computer Applications at Andhra University College of Engineering(A), for his encouragement and valuable recommendations that helped shape this dissertation.

I am grateful to **Prof. G. Sasibhusana Rao**, the Principal of Andhra University College of Engineering(A), for his facilitation, which contributed significantly to the timely completion of the project.

My heartfelt thanks go to the teaching and non-teaching staff members of the Department of Computer Science and System Engineering at Andhra University College of Engineering(A), Visakhapatnam. Their consistent support was instrumental in the successful completion of my studies.

Lastly, I wish to acknowledge the profound contribution of my beloved parents and friends. Their unwavering support and encouragement were indispensable in bringing this work to fruition.

**B.V.S. PAVAN KUMAR**

**Reg.No: 321206416001**

**MTech (IT)**

**ABSTRACT**

**ABSTRACT**

In an ever-evolving technological landscape, the ability to accurately identify and detect hazards is of paramount importance to ensure safety across various domains. Hazards, encompassing potential threats and risks that can lead to harm or adverse events, necessitate proactive measures for timely intervention. This project focuses on harnessing the power of Machine Learning (ML) techniques to enhance hazard identification and detection processes.  
Internet browsing has seamlessly integrated into our everyday routines, and in order to engage users, different browser vendors constantly strive to introduce new functionalities and advanced features. Unfortunately, these enhancements also create opportunities for attackers to exploit vulnerabilities, posing risks to websites and users. Current security measures are insufficient in protecting surfers, necessitating the development of a fast and accurate model capable of distinguishing between benign and potentially harmful web pages. In this research paper, we develop a novel classification system utilizing machine learning classifiers, including random forest, support vector machine, naïve Bayes, and logistic regression. for the purpose of examining and identifying malicious web pages. By extracting features from special Uniform Resource Locators (URLs), we train the classifiers to predict whether a webpage is malicious or benign. Experimental results demonstrate that the random forest classifier compared to other machine learning classifiers, achieves an accuracy rate of 95%. Achieving a higher classification accuracy is crucial in enhancing web security and protecting users from potential cyber threats.

**TABLE OF CONTENTS**

**1. INTRODUCTION…………………………………………………………………….......1**

**1.1 OVERVIEW………………………………………………………………………...1**

**1.2 PROBLEM STATEMENT………………………………………………...............1**

**1.3 SCOPE OF THE PROJECT……………………………………………………....2**

**1.4 EXISTING SYSTEM………………………………………………………………2**

**1.5 PROPOSED SYSTEM……………………………………………………………..2**

**2. LITERATURE REVIEW………………………………………………………………..4**

**3. SYSTEM ANALYSIS AND DESIGN………………………………………………......6**

**3.1 ARCHITECTURE………………………………………………………………....7**

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS………………………….8**

**3.3 UML DIAGRAMS………………………………………………………………....8**

**3.3.1 USE CASE DIAGRAM………………………………………………………9**

**3.3.2 CLASS DIAGRAM…………………………………………………………..9**

**3.3.3 SEQUENCE DIAGRAM………………………………………........….…...10**

**3.3.4 COLLABORATION DIAGRAM…………………………………..………11**

**3.3.5 DEPLOYMENT DIAGRAM…………………………………………….....11**

**3.3.6 ACTIVITY DIAGRAM…………………………………………..…………11**

**3.3.7 COMPONENT DIAGRAM……………………………………………..….12**

**3.3.8 ER DIAGRAM……………………………………………………………....12**

**3.3.9 DATA FLOW DIAGRAM (DFD)…………………………………………..13**

**4. METHODOLOGY………………………………………………………………...……15**

**4.1 OVERVIEW OF MACHINE LEARNING CLASSIFIERS:……………...…..15**

**4.1.1 DECISION TREE……………………………………………………………15**

**4.1.2 K-NEAREST NEIGHBOURS (KNN)………………………………………15**

**4.1.3 XG BOOST…………………………………………………………………...17**

**4.1.4 ADABOOST………………………………………………………………….17**

**4.1.5 RANDOM FOREST…………………………………………………………19**

**4.1.6 EXTRA TREES……………………………………………………………...23**

**4.2 PERFORMANCE COMPARISON OF DIFFERENT CLASSIFIERS………23**

**5. IMPLEMENTATION…………………………………………………………………..25**

**5.1 PYTHON LIBRARIES AND TOOLS USED…………………………………..25**

**5.2 SOFTWARE INSTALLATION…………………………………….…………...25**

**5.3 SAMPLE CODE………………………………………………………………….28**

**6. TESTING………………………………………………………………………………...41**

**7. RESULT ANALYSIS …………………………………………………………………..44**

**CONCLUSION AND FUTURE SCOPE ………………………………………………...49**

**REFERENCES……………………………………………………………………………..51**

**JOURNAL CERTRIFICATES…………………………………………………………....52**

**LIST OF FIGURES**

Figure 1 : Architecture……………………………………………………...7

Figure 2 : Use Case Diagram……………………………………………….9

Figure 3 : Class Diagram…………………………………………………...10

Figure 4 : Sequence Diagram……………………………………………....10

Figure 5 : Collaboration Diagram…………………………………………..11

Figure 6 : Deployment Diagram…………………………………………....11

Figure 7 : Activity Diagram…………………………………………….......12

Figure 8 : Component Diagram…………………………………………….12

Figure 9 : ER Diagram……………………………………………………...13

Figure 10 : DFD Diagram…………………………………………………..13

Figure 11 : DFD Diagram..............................................................................14

Figure 12 : Ada Boost....................................................................................18

Figure 13 : Random Forest.............................................................................20

Figure 14 : Random Forest.............................................................................21

Figure 15 : Anaconda Installation..................................................................26

Figure 16 : Anaconda Navigator....................................................................27

Figure 17 : Anaconda Prompt........................................................................27

Figure 18 : Anaconda Package Installation....................................................28

Figure 19: Home Page....................................................................................44

Figure 20 : Upload Dataset.............................................................................44

Figure 21 : View Dataset................................................................................45

Figure 22 : Train/Test Dataset........................................................................45

Figure 23 : Model Performance......................................................................46

Figure 24 : Prediction.....................................................................................48

Figure 25 : Graph...........................................................................................49

**LIST OF TABLES**

Table 1 : Performance Comparison of Different Classifiers…………………………….24

**1.INTRODUCTION**

**1.1** **OVERVIEW:**

As the internet continues to experience rapid growth, an expanding array of services, including Online banking, electronic commerce, social media networking, bill settlement, and digital learning have become readily accessible to users through web browsers and web applications. However, with the advancement of browsers and their numerous features, there is a growing risk of users' personal and sensitive information being compromised. Many users, especially those who are not well-versed in online security, are unaware of various malware threats.

Consequently, they can easily fall prey to intruders by simply clicking on malicious websites. These websites enable attackers to detect vulnerabilities and inject harmful code to obtain remote access to the target's webpage. Thus, it is crucial to accurately identify web pages in this ever-expanding web landscape. To address these challenges, blacklisting services have been integrated into browsers.

Our work, we introduce a self-learning approach that relies on a streamlined set of features for the categorization of web pages. The primary objective is to utilize six machine learning classifiers to distinguish web pages into two distinct categories: benign and malicious.

* 1. **PROBLEM STATEMENT:**

In various industries and environments, the timely identification and detection of hazards are paramount to ensuring safety for individuals, assets, and the environment. Hazardous events can have catastrophic consequences, making it crucial to develop robust and efficient methods for hazard identification. Traditional methods often rely on human observation and rule-based systems, which may not always be effective or scalable.

The objective of this project is to design, develop, and evaluate a comprehensive hazard identification and detection system using state-of-the-art machine learning classifiers. The project will focus on six different classifiers, including Decision Trees, K-Nearest Neighbour’s (KNN),

XG Boost, AdaBoost, Random Forest, and Extra Trees, to harness the power of ensemble learning and tree-based algorithms.

* 1. **SCOPE OF THE PROJECT:**

In this specific context, we have designed and implemented a cutting-edge classification system tailored. To the examination and recognition of malicious web pages. Our system relies on the prowess of machine learning classifiers, including Decision tree, KNN, XG Boost, Ada Boost, Random Forest, Extra trees. The fundamental principle behind our approach involves the utilization of URL (Uniform Resource Locator) data. We extract pertinent features from these URLs and subsequently employ these features in the training of our classifiers.

Through an iterative training process, our system attains the capability to make precise predictions regarding the maliciousness or benignity of a web page. This means that it can effectively discern whether a web page poses a security threat, contributing to enhanced web security.

* 1. **EXISTING SYSTEM:**

In this research article, we introduce an innovative approach that leverages machine learning to identify and detect malicious URLs associated with a wide range of popular attack types. Diverging from existing methodologies that primarily target the recognition of malicious URLs within a single attack category, our proposed method is engineered to effectively discern malicious URLs across multiple attack categories. Furthermore, our approach transcends mere detection by striving to precisely ascertain the specific nature of the attack intended by a malicious URL.

Through the application of advanced machine learning techniques, our objective is to furnish a comprehensive and resilient solution to address the multifaceted challenges posed by diverse and constantly evolving malicious URL attacks. This approach aims to significantly enhance the security measures against a broad spectrum of cyber threats.

* 1. **PROPOSED SYSTEM:**

Our approach is founded upon the utilization of a diverse and comprehensive set of discriminative characteristics. These encompass various aspects involving textual attributes, link structures, webpage content, DNS data, and network traffic.

A noteworthy aspect of our approach is the inclusion of numerous original features, which have proven to exhibit high efficacy in our evaluations.

To substantiate the efficacy of our approach, we conducted extensive experiments utilizing a substantial dataset comprising 40,000 benign URLs and 32,000 malicious URLs, drawn from real-world internet data sources. The outcomes of our experimentation vividly illustrate the exceptional performance of our methodology. It attains an impressive accuracy rate, surpassing 98%, in effectively detecting malicious URLs, while also achieving a precision of over 93% in correctly identifying the specific attack types associated with these URLs.

Additionally, we provide comprehensive analyses of the effectiveness of each group of discriminative features incorporated into our methodology. Furthermore, we delve into an exploration of the potential susceptibilities of these features to evasion techniques, fostering a comprehensive comprehension of the system's strengths and constraints.

**2. LITERATURE REVIEW**

Researchers have proposed various techniques, for the identification of malicious web pages by employing methods, encompassing techniques such as blacklisting, static analysis, and dynamic analysis.

Tao et al. [1] introduced an innovative framework that leverages supervised machine learning to autonomously determine the nature of a web page, whether it is malicious or benign. Their classification relied on specific features, and they compiled a dataset of benign web pages for this purpose.

Adware and rami et al. [2] put forth a lightweight self-learning methodology for categorizing malicious web pages, utilizing a framework named MALURL. They utilized Genetic Algorithm (GA) for the training of classifiers designed to detect malicious web pages. Their training dataset included benign websites from Alexa and malicious ones from Phish Tank, resulting in an average system precision of 87%.

Hwang et al. [3] employed the Adaptive SVM (SVM) machine learning technique, known for its ability to effectively adapt to new training data, thereby reducing the risk of misclassifying novel web pages.

Yue et al. [4] introduced a method for classifying malicious web pages, utilizing 30 distinctive features with the assistance of machine learning algorithms such as K-NN and SVM. Their research indicated that the K-NN algorithm outperformed SVM. They implemented two classification models for the detection of malicious web pages and specific threat types.

Yoo et al. [5] introduced two distinct detection methods: misuse detection, which aims to identify known malicious web pages, and anomaly detection, designed to detect previously unidentified malicious web pages. In their experiments, using the RafaBot dataset within the WEKA tool, they achieved a notable detection rate of up to 98%. However, it's worth noting that the false positive rate was relatively high, reaching 30.5%.

Kim et al. [6] introduced Web Mon, an automated and minimally interactive malicious webpage detection tool. Web Mon leverages machine learning and YARA signatures to discern detrimental components within web resources loaded via WebKit2-based browsers.

**3. SYSTEM ANALYSIS AND DESIGN**

**Input Design:**

In an information system, the defined output is exposed to as input. Developers must consider input devices such as Windows, OCR, and Wrong. the penal during planning phase.

**Output Design:**

The most important tip towards every system is the performance design. During system analysis, developers determine the level of areas to improve, used and the distributes power and low fidelity report layouts.

**MODULES:**

1. User:

1.1 Homepage View:

Users can access the Hazards Classification web application and view its main page.

1.2 About Page View:

The about page provides information about hazard detection for users to learn more.

1.3 Input Data:

Users are required to provide input values for specific fields to obtain results.

1.4 Result Display:

Users can view the generated results from the model.

1.5 Score Display:

Users have the option to view the score represented as a percentage.

2. System:

2.1 Data Processing:

The system checks for the availability of data and loads it into CSV files.

2.2 Data Preprocessing:

Data undergoes preprocessing to enhance the accuracy of the model.

2.3 Data Training:

The data is divided into training and testing sets prior to being subjected to the designated algorithms.

2.4 Model Creation:

This module assists in building a model that predicts personality with higher accuracy.

2.5 Score Generation:

Users can view the generated score represented as a percentage.

2.6 Result Generation:

Machine learning algorithms are trained to predict the outcome.

**3.1 ARCHITECTURE:**

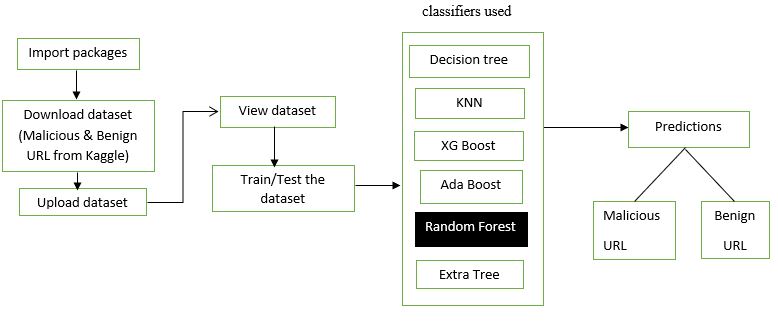
****

Fig.1

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS:**

**Hardware Configuration:**

Processor: Intel Core i3 or equivalent

Hard Disk: 160GB or higher

Keyboard: Standard Windows Keyboard

Mouse: Two or Three Button Mouse

Monitor: SVGA

RAM: 8GB

**Software Configuration:**

Operating System: Windows 10

Server-side Script: Python with Anaconda

IDE: PyCharm

Libraries Used: sklearn, pandas, Scikit-learn

Dataset: URL dataset containing benign and malicious web pages

Technology: Python 3.11

**3.3 UML DIAGRAMS:**

**3.3.1** USE CASE DIAGRAM

A use test switch is a form of software diagram that is tested and rated by a use-case study in the Uml. Its goal is to provide a sub map of a plan's physiology in view of founders, their scene (governed as use cases), and the interactions between use cases.

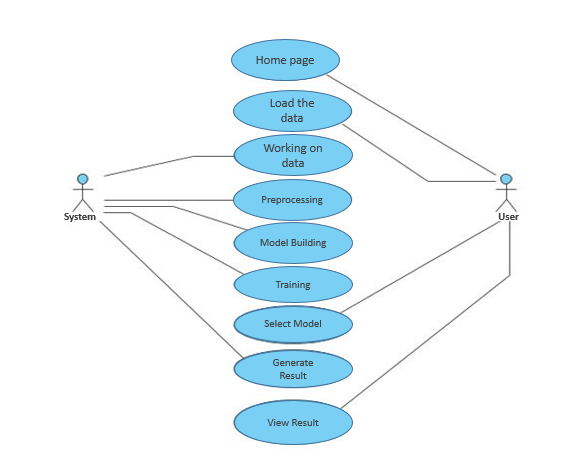
****

Fig.2

**3.3.2** CLASS DIAGRAM

A use context diagram is a form of group diagram called by and viewed via use-case basic research. His goal to provide an assigning of an app's function in spite of mentors, their objectives covered as use problems in today’s world.

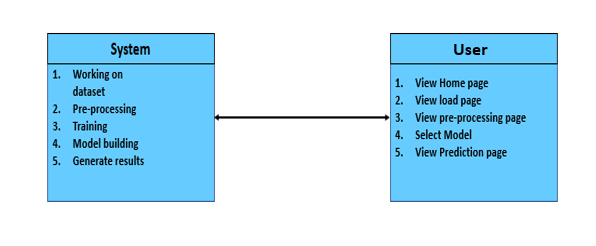


Fig.3

**3.3.3** SEQUENCE DIAGRAM

A sequence swap is a method vsm that suggests how skills link with us and why time may be urgent.

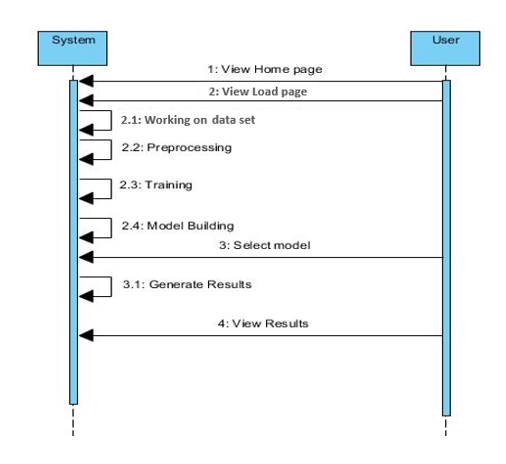


Fig.4

**3.3.4** COLLABORATION DIAGRAM

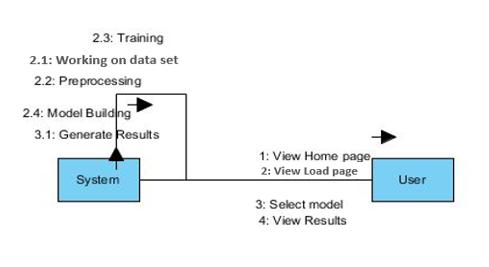
The bilateral and multilateral diagram is specified just use the same ERP system.

Fig.5

**3.3.5** DEPLOYMENT DIAGRAM

A deploy table describes a scheme's deployment view. It is conflated with the component diagram. Although the entity relationship diagrams will be used to manage the components.

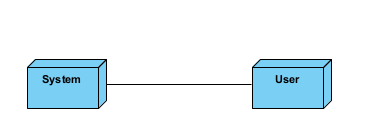


Fig.6

**3.3.6** ACTIVITY DIAGRAM

In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

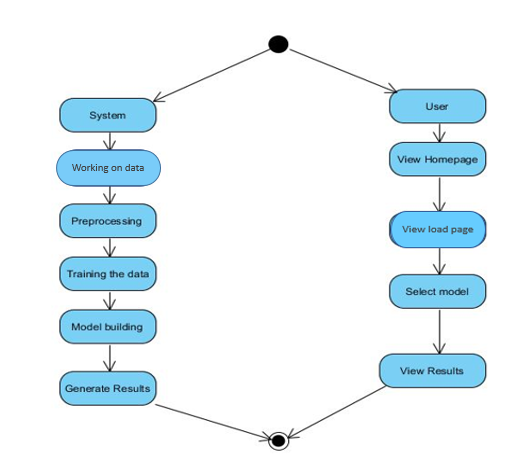


Fig.7

**3.3.7** COMPONENT DIAGRAM

A component diagram, may also designated as a Particles diagram, confirms how different components in an equation are done and tied simultaneously.

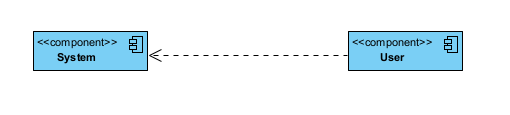


Fig.8

**3.3.8** ER DIAGRAM

To assess the control flow, an Equity theory should employ a diagram called as a Base Class. A diagram is a MySQL style or secret agenda be done as a file.

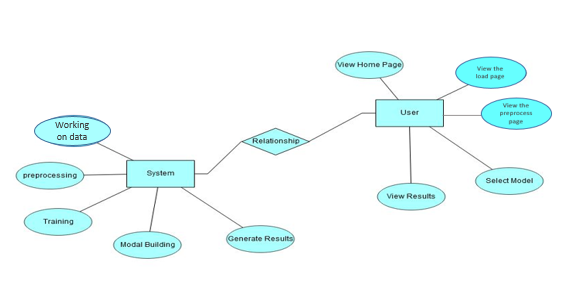


Fig.9

**3.3.9** DFD

A Data Storage Map is a major parameter to redefine the flow of data onto a system. A cool and simple DFD may vividly find a vast percentage software.

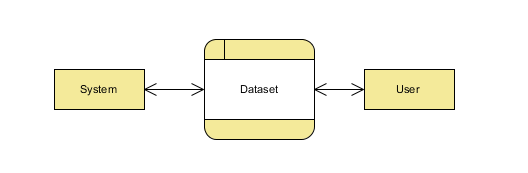
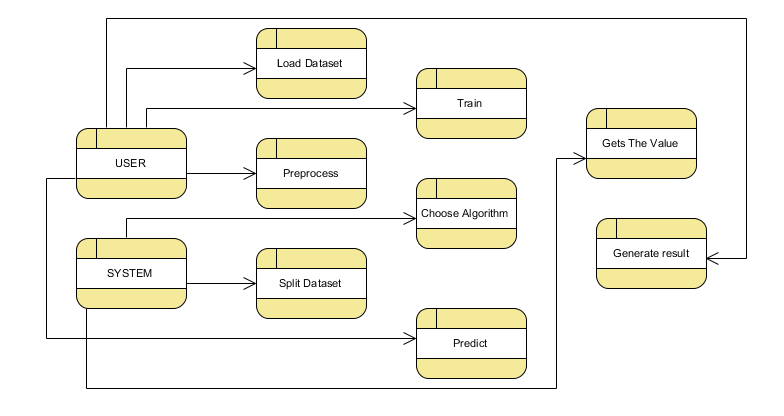


Fig.10



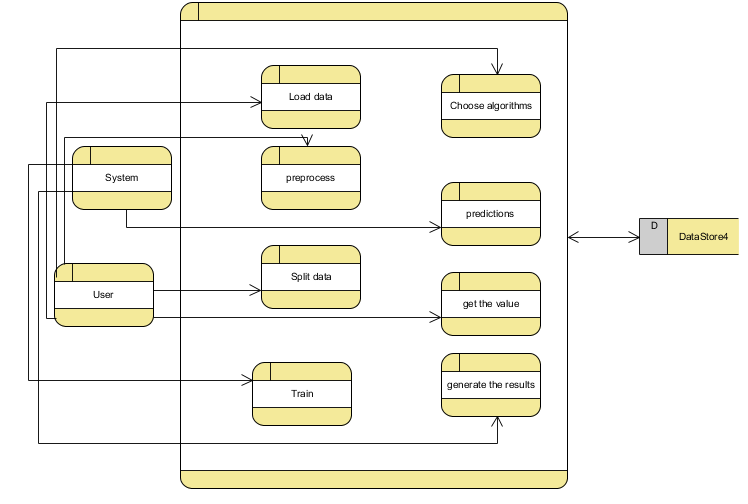


Fig.11

**4.METHODOLOGY**

**4.1 OVERVIEW OF ML CLASSIFIERS:**

**4.1.1** DECISION TREES:

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal.

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/internal node, based on which the tree splits into branches/ edges. The end of the branch that doesn’t split anymore is the decision/leaf, in this case, whether the passenger died or survived, represented as red and green text respectively.

Although, a real dataset will have a lot more features and this will just be a branch in a much bigger tree, but you can’t ignore the simplicity of this algorithm. The feature importance is clear and relations can be viewed easily. This methodology is more commonly known as learning decision tree from data and above tree is called Classification tree as the target is to classify passenger as survived or died. Regression trees are represented in the same manner, just they predict continuous values like price of a house. In general, Decision Tree algorithms are referred to as CART or Classification and Regression Trees.

So, what is actually going on in the background? Growing a tree involves deciding on which features to choose and what conditions to use for splitting, along with knowing when to stop. As a tree generally grows arbitrarily, you will need to trim it down for it to look beautiful. Let’s start with a common technique used for splitting.

**4.1.2** K-NEAREST NEIGHBOURS:

K-Nearest Neighbors is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data. Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset.

The K-NN working can be explained on the basis of the below algorithm:

* Step-1: Select the number K of the neighbors
* Step-2: Calculate the Euclidean distance of K number of neighbors
* Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
* Step-4: Among these k neighbors, count the number of the data points in each category.
* Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
* Step-6: Our model is ready.

**4.1.3** XG BOOST:

XG Boost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.

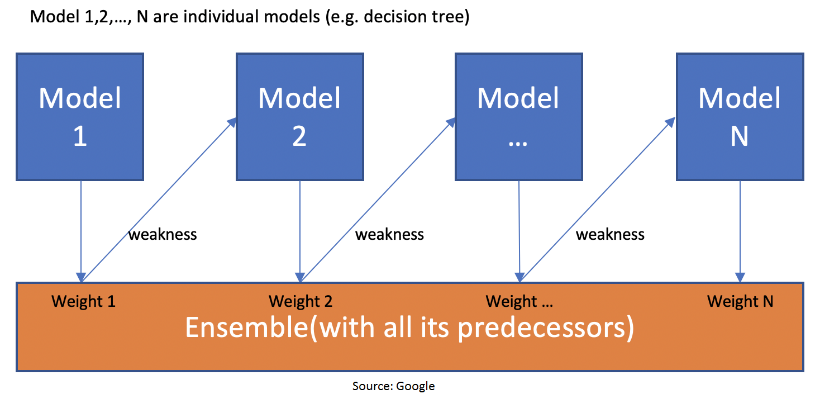
Bagging: Now imagine instead of a single interviewer, now there is an interview panel where each interviewer has a vote. Bagging or bootstrap aggregating involves combining inputs from all interviewers for the final decision through a democratic voting process.

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners (CARTs generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.

**4.1.4** ADA BOOST:

AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique that is used as an Ensemble Method in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights to incorrectly classified instances. Boosting is used to reduce bias as well as the variance for supervised learning. It works on the principle where learners are grown sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones. Adaboost algorithm also works on the same principle as boosting, but there is a slight difference in working.

It makes *n* number of decision trees during the training period of data. As the first decision tree/model is made, the record which is incorrectly classified during the first model is given more priority. Only these records are sent as input for the second model. The process will go on until we specify a number of base learners we want to create. Remember, the repetition of records is allowed with all boosting techniques.



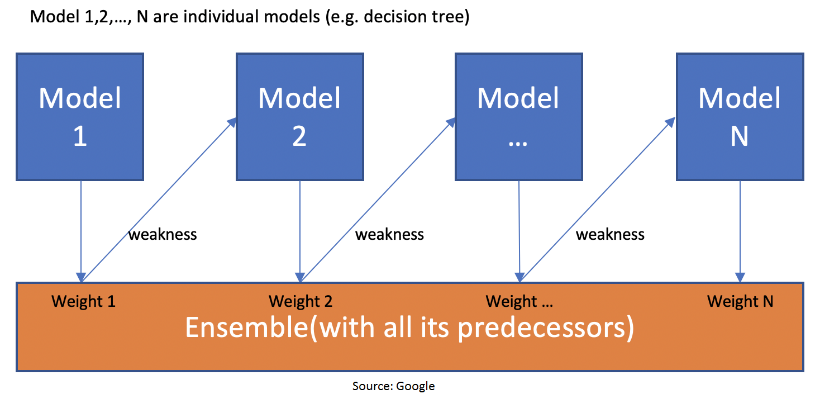


Fig.12

This figure shows that when the first model is made and the errors from the first model are noted by the algorithm, the record which is incorrectly classified is given as the input for the next

model. This process is repeated until the specified condition is met. As you can see in the figure, there are *n* number of models made by taking the errors from the previous model. This is how boosting works. The models 1,2, 3…, N are individual models that can be known as decision trees. All types of boosting models work on the same principle.

**4.1.5** RANDOM FOREST:

A random forest is a machine learning technique that’s used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

A random forest algorithm consists of many decision trees. The ‘forest’ generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.

The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the over fitting of datasets and increases precision. It generates predictions without requiring many configurations in packages (like [Scikit-learn](https://en.wikipedia.org/wiki/Scikit-learn)).

Features of a Random Forest Algorithm:

* It’s more accurate than the decision tree algorithm.
* It provides an effective way of handling missing data.
* It can produce a reasonable prediction without hyper-parameter tuning.
* It solves the issue of over fitting in decision trees.
* In every random forest tree, a subset of features is selected randomly at the node’s splitting point.

Decision trees are the building blocks of a random forest algorithm. A decision tree is a decision support technique that forms a tree-like structure. An overview of decision trees will help us understand how random forest algorithms work.

A decision tree consists of three components: decision nodes, leaf nodes, and a root node. A decision tree algorithm divides a training dataset into branches, which further segregate into other

branches. This sequence continues until a leaf node is attained. The leaf node cannot be segregated further.

The nodes in the decision tree represent attributes that are used for predicting the outcome. Decision nodes provide a link to the leaves. The following diagram shows the three types of nodes in a decision tree.

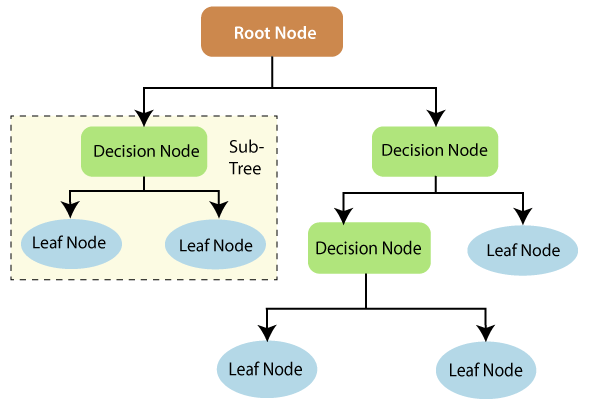


Fig.13

The information theory can provide more information on how decision trees work. Entropy and information gain are the building blocks of decision trees. An overview of these fundamental concepts will improve our understanding of how decision trees are built.

Entropy is a metric for calculating uncertainty. Information gain is a measure of how uncertainty in the target variable is reduced, given a set of independent variables.

The information gain concept involves using independent variables (features) to gain information about a target variable (class). The entropy of the target variable (Y) and the [conditional entropy](https://en.wikipedia.org/wiki/Conditional_entropy) of Y (given X) are used to estimate the information gain. In this case, the conditional entropy is subtracted from the entropy of Y.

Information gain is used in the training of decision trees. It helps in reducing uncertainty in these trees. A high information gain means that a high degree of uncertainty (information entropy) has been removed. Entropy and information gain are important in splitting branches, which is an important activity in the construction of decision trees.

Let’s take a simple example of how a decision tree works. Suppose we want to predict if a customer will purchase a mobile phone or not. The features of the phone form the basis of his decision. This analysis can be presented in a decision tree diagram.

The root node and decision nodes of the decision represent the features of the phone mentioned above. The leaf node represents the final output, either *buying* or *not buying*. The main features that determine the choice include the price, internal storage, and Random Access Memory (RAM). The decision tree will appear as follows.

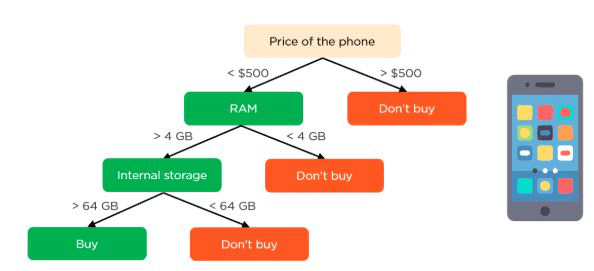


Fig.14

Applying decision trees in random forest

The main difference between the decision tree algorithm and the random forest algorithm is that establishing root nodes and segregating nodes is done randomly in the latter. The random forest employs the bagging method to generate the required prediction.

Bagging involves using different samples of data (training data) rather than just one sample. A training dataset comprises observations and features that are used for making predictions. The decision trees produce different outputs, depending on the training data fed to the random forest algorithm. These outputs will be ranked, and the highest will be selected as the final output.

Our first example can still be used to explain how random forests work. Instead of having a single decision tree, the random forest will have many decision trees. Let’s assume we have only four decision trees. In this case, the training data comprising the phone’s observations and features will be divided into four root nodes.

The root nodes could represent four features that could influence the customer’s choice (price, internal storage, camera, and RAM). The random forest will split the nodes by selecting features randomly. The final prediction will be selected based on the outcome of the four trees.

The outcome chosen by most decision trees will be the final choice. If three trees predict *buying*, and one tree predicts *not buying*, then the final prediction will be *buying*. In this case, it’s predicted that the customer will buy the phone.

**4.1.6** EXTRA TREE CLASSIFIER:

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output it’s classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.

Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, Each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees.

To perform feature selection using the above forest structure, during the construction of the forest, for each feature, the normalized total reduction in the mathematical criteria used in the decision of feature of split (Gini Index if the Gini Index is used in the construction of the forest) is computed. This value is called the Gini Importance of the feature. To perform feature selection, each feature is ordered in descending order according to the Gini Importance of each feature and the user selects the top k features according to his/her choice.

**4.2 PERFORMANCE COMPARISON OF DIFFERENT CLASSIFIERS:**

Various experiments have been carried out by implementing the classification algorithms such as Decision tree, K-Nearest Neighbors (KNN), Extreme Gradient (XG) Boost, Adaptive Boost, Random Forest, Extra Tree Classifiers. All the experiments were coded and tested in PyCharm Software [8] which is an interactive python environment for Machine Learning. With its integrated support for Pandas, Scikit-Learn, Matplotlib, markup language, plots, and tables, a much more appealing and understandable presentation of the flow of the code can be made. We then compare the performance of the Six machine learning classifiers.

We have used the precision, recall and accuracy to evaluate the detection performance because it correctly labels a webpage. So, to obtain the best results, accuracy performance metric plays a vital role. We notice that the machine learning classifier RF obtains higher accuracy, whose performance is better than the other classifiers on malicious web page detection.

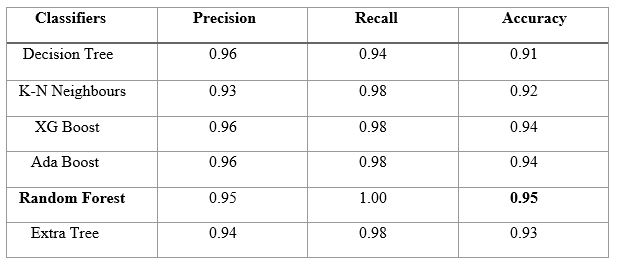


Table 1: Performance Comparison of different classifiers

**5.IMPLEMENTATION**

**5.1 PYTHON LIBRARIES AND TOOLS USED:**

Various Python libraries and tools are essential for data manipulation, model development, evaluation, and deployment. Here are some commonly used Python libraries and tools for such project.

1. NumPy: NumPy is a fundamental library for scientific computing in Python. It provides support for arrays and matrices, which are crucial for handling numerical data efficiently.

2. Pandas: Pandas is a powerful library for data manipulation and analysis. It offers data structures like Data Frames and Series, which are invaluable for preprocessing and exploring datasets.

3. Scikit-Learn (sklearn): Scikit-Learn is a comprehensive library for machine learning in Python. It includes a wide range of classifiers, preprocessing techniques, and evaluation metrics. Some specific modules include sklearn. tree for decision trees, sklearn. neighbors for KNN, and sklearn. ensemble for ensemble methods like Random Forest and AdaBoost.

4. XGBoost: XGBoost is a popular gradient boosting library known for its high-performance implementation. It's especially useful for structured data and is available as the XGBoost Python package.

5. Flask/Django: To deploy your hazard detection system as a web application, Flask or Django has used to create the backend API and web interface.

These libraries and tools provide a solid foundation for developing, deploying, and maintaining a hazard identification and detection system using machine learning in Python.

**5.2** **SOFTWARE INSTALLATION**:

*1*. Visit the Anaconda downloads page.

Go to the following link: [Anaconda.com/downloads](https://www.anaconda.com/download/)

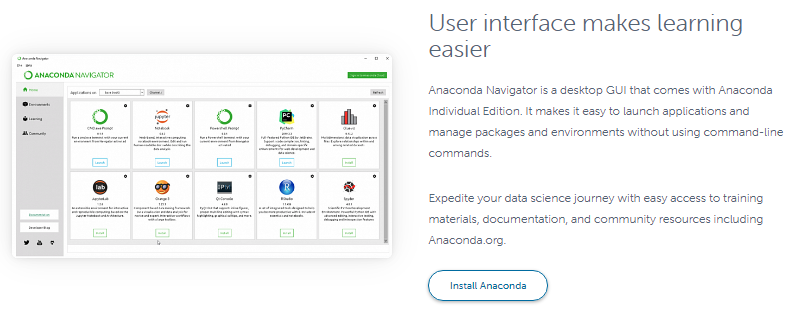


Fig.15

2. Select Windows.

Select Windows where the three operating systems are listed.

*3.* Download.

Choose Python 3.11 version, 64-bit graphical installer.

4. Let it download in an .exe format.

*5.* Open and run the installer.

Once the download completes, open and run the .exe installer.

6. Click on next, I agree, install.

7. Completion of the installation, open your windows start menu and select the Anaconda navigator.

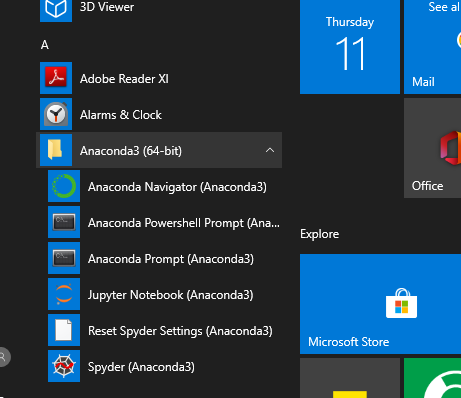


Fig.16

8. You need to install some packages to execute your project in a proper way.

9. Select windows start menu, right click on anaconda prompt, choose run as administrator.

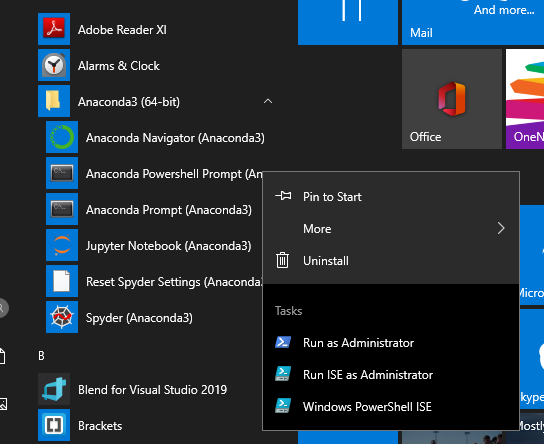


Fig.17

11. Anaconda prompt will get open, with specified path, type “Pip install package name” which you want to install (like Jumpy, Pandas, seaborne, sickie learn, matplotlib, pilot)

Ex: pip install NumPy.



Fig.18

**5.3 SAMPLE CODE:**

import pygal

from flask import Flask, render\_template, request, session, url\_for, Response

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn import linear\_model

from sklearn.metrics import classification\_report

import sys,os

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import ExtraTreeClassifier

filepath = os.getcwd()

app = Flask(\_\_name\_\_)

global LR1,RF1,NB1,SVM1

def f(x\_train,x\_test, y\_train, y\_test):

global X\_trains,X\_tests,y\_trains,y\_tests

X\_trains = pd.DataFrame(x\_train)

X\_tests = pd.DataFrame(x\_test)

y\_trains = pd.DataFrame(y\_train)

y\_tests = pd.DataFrame(y\_test)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/upload')

def uploaddataset():

return render\_template('uploaddataset.html')

@app.route('/traintest')

def traintestvalue():

print("hello")

return render\_template('traintestdataset.html')

@app.route('/modelperformance')

def modelperformances():

return render\_template('modelperformance.html')

@app.route('/uploaddataset',methods=["POST","GET"])

def uploaddataset\_csv\_submitted():

if request.method == "POST":

csvfile = request.files['csvfile']

result = csvfile.filename

filepath.replace("\\","/")

file = filepath +"\\" + result

print(file)

session['filepath'] = file

return render\_template('uploaddataset.html',msg='sucess')

return render\_template('uploaddataset.html'

@app.route('/viewdata',methods=["POST","GET"])

def viewdata():

session\_var\_value = session.get('filepath')

print("session variable is=====" + session\_var\_value)

df = pd.read\_csv(session\_var\_value)

global x

x = pd.DataFrame(df)

x=x.dropna(how="any",axis=0)

#le=preprocessing.LabelEncoder()

# x['CHARSET']=le.fit\_transform(x['CHARSET'])

#le2 = preprocessing.LabelEncoder()

#x['SERVER']=le2.fit\_transform(x['SERVER'].astype("str"))

# print(x.head(10))

# session['x'] = x

return render\_template("viewdataset.html",col=x.columns.values, row\_data=list(x.values.tolist()),zip=zip)

@app.route('/traintestdataset',methods=["POST","GET"])

def traintestdataset\_submitted():

if request.method == "POST":

value = request.form['traintestvalue']

value1=(value)

df1=x

df1["CONTENT\_LENGTH"]= df1["CONTENT\_LENGTH"].fillna(df1["CONTENT\_LENGTH"].mean())

X = df1.drop(["Result","URL"],axis=1)

y = df1['Result']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=float(value1))

f(X\_train,X\_test, y\_train, y\_test)

X\_train1 = pd.DataFrame(X\_train)

X\_trainlen=len(X\_train)

y\_test1 = pd.DataFrame(y\_test)

y\_testlen = len(y\_test)

return render\_template('traintestdataset.html',msg='sucess',data=X\_train1.to\_html(),X\_trainlenvalue=X\_trainlen,y\_testlenval=y\_testlen)

return render\_template('traintestdataset.html')

from xgboost import XGBClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import make\_scorer, accuracy\_score, precision\_score, recall\_score, f1\_score,roc\_auc\_score

@app.route('/modelperformance',methods=["POST","GET"])

def selected\_model\_submitted():

global accuracyscore,precisionscore,recallscore

if request.method == "POST":

selectedalg =int(request.form['algorithm'])

if selectedalg == 1:

global p33,p31,accuracyscore

model = DecisionTreeClassifier(criterion='entropy', random\_state=0)

model.fit(X\_trains, y\_trains)

y\_pred = model.predict(X\_tests)

accuracyscore = accuracy\_score(y\_tests, y\_pred)

p31 = precision\_score(y\_tests, y\_pred, average='macro')

p33 = recall\_score(y\_tests, y\_pred, average='macro')

p36 = f1\_score(y\_true=y\_tests, y\_pred=y\_pred, average='macro')

p37 = roc\_auc\_score(y\_true=y\_tests, y\_score=y\_pred, average='macro')

msg2 = classification\_report(y\_tests, y\_pred)

print('The classification\_report for Decision Tree')

print(msg2)

return render\_template('modelperformance.html', msg="accuracy\_score", score=accuracyscore,

model="Decision Tree",msg2=msg2)

elif selectedalg == 2:

model = KNeighborsClassifier(n\_neighbors=8)

model.fit(X\_trains, y\_trains)

y\_pred = model.predict(X\_tests)

accuracyscore = accuracy\_score(y\_tests, y\_pred)

p61 = precision\_score(y\_tests, y\_pred, average='macro')

p63 = recall\_score(y\_tests, y\_pred, average='macro')

p66 = f1\_score(y\_true=y\_tests, y\_pred=y\_pred, average='macro')

p67 = roc\_auc\_score(y\_true=y\_tests, y\_score=y\_pred, average='macro')

msg2 = classification\_report(y\_tests, y\_pred)

print('The classification\_report for K Nearest Neighbour')

print(msg2)

return render\_template('modelperformance.html', msg="accuracy\_score", score=accuracyscore,

model="K Nearest Neighbour")

elif selectedalg == 3:

model = XGBClassifier()

model.fit(X\_trains, y\_trains)

y\_pred = model.predict(X\_tests)

accuracyscore = accuracy\_score(y\_tests, y\_pred)

p91 = precision\_score(y\_tests, y\_pred, average='macro')

p93 = recall\_score(y\_tests, y\_pred, average='macro')

p96 = f1\_score(y\_true=y\_tests, y\_pred=y\_pred, average='macro')

p97 = roc\_auc\_score(y\_true=y\_tests, y\_score=y\_pred, average='macro')

msg2 = classification\_report(y\_tests, y\_pred)

print('The classification\_report for XGBoost')

print(msg2)

return render\_template('modelperformance.html', msg="accuracy\_score", score=accuracyscore,

model="XG Boost")

elif selectedalg==4:

from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier(n\_estimators=100, random\_state=0)

model.fit(X\_trains, y\_trains)

y\_pred = model.predict(X\_tests)

accuracyscore = accuracy\_score(y\_tests, y\_pred

msg2 = classification\_report(y\_tests, y\_pred)

print('The classification\_report for AdaBoost')

print(msg2)

return render\_template('modelperformance.html', msg="accuracy\_score", score=accuracyscore,

model="Ada Boost")

elif selectedalg==5:

model = RandomForestClassifier()

model.fit(X\_trains, y\_trains)

y\_pred = model.predict(X\_tests)

accuracyscore = accuracy\_score(y\_tests, y\_pred)

msg2 = classification\_report(y\_tests, y\_pred)

print('The classification\_report for RandomForestClassifier')

print(msg2)

return render\_template('modelperformance.html', msg="accuracy\_score", score=accuracyscore,

model="RandomForestClassifier")

elif selectedalg==6:

model = ExtraTreeClassifier()

model.fit(X\_trains, y\_trains)

y\_pred = model.predict(X\_tests)

accuracyscore = accuracy\_score(y\_tests, y\_pred)

msg2 = classification\_report(y\_tests, y\_pred)

print('The classification\_report for ExtraTreeClassifier')

print(msg2)

return render\_template('modelperformance.html', msg="accuracy\_score", score=accuracyscore,

model="ExtraTreeClassifier")

@app.route('/prediction', methods=["POST","GET"])

def prediction():

print("hello")

if request.method == "POST":

print("hi")

url = request.form['url\_len']

age = request.form['NUMBER\_SPECIAL\_CHARACTERS']

length = request.form['CONTENT\_LENGTH']

traffic = request.form['SOURCE\_APP\_BYTES']

anchor = request.form['APP\_PACKETS']

all\_obj\_vals = [[float(url), float(age), float(length), float(traffic), float(anchor)]]

model = DecisionTreeClassifier(criterion='entropy', random\_state=0)

model.fit(X\_trains, y\_trains)

predi = model.predict(all\_obj\_vals)

pre = predi

return render\_template('prediction.html',msg='predictsucess',predvalue=predi)

return render\_template('prediction.html')

@app.route("/bar\_chart")

def bar\_chart():

line\_chart = pygal.Bar()

line\_chart.title = 'DETECTION AND IDENTIFICATION USING MACHINE LEARNING APPROACH'

line\_chart.add('PRECISION', [p33])

line\_chart.add('RECALL', [p31])

line\_chart.add('ACCURACY', [accuracyscore])

graph\_data = line\_chart.render()

return render\_template('bar\_chart.html', graph\_data=graph\_data)

if \_\_name\_\_=='\_\_main\_\_':

app.secret\_key = ".."

app.run(debug=True)

**6.TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

***SYSTEM TEST***

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

***White Box Testing***

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

***Black Box Testing***

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

**Integration Testing:**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Acceptance Testing:**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Functional testing:**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**7.RESULTS**

Home page:



Fig.19

Upload dataset :

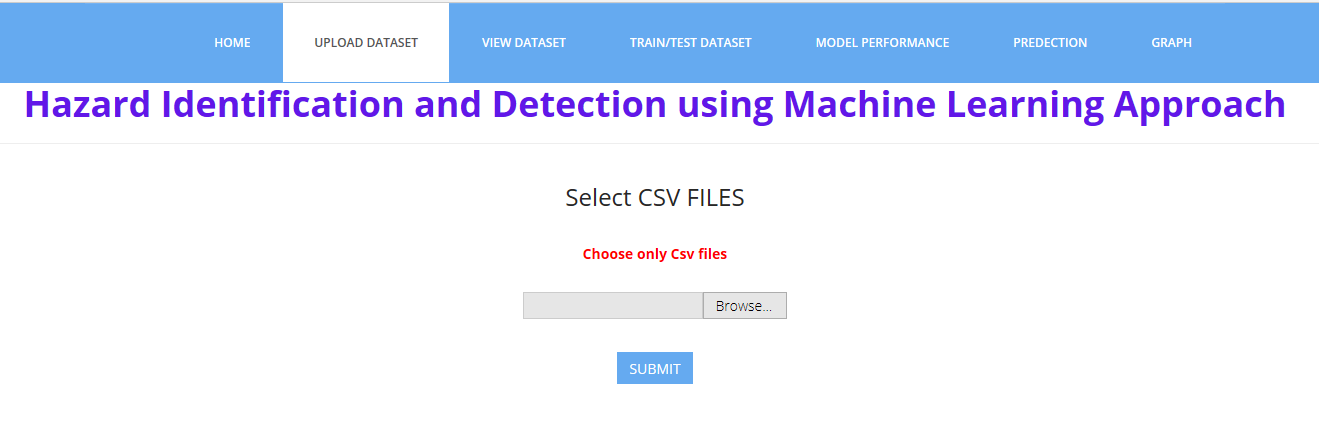


Fig.20

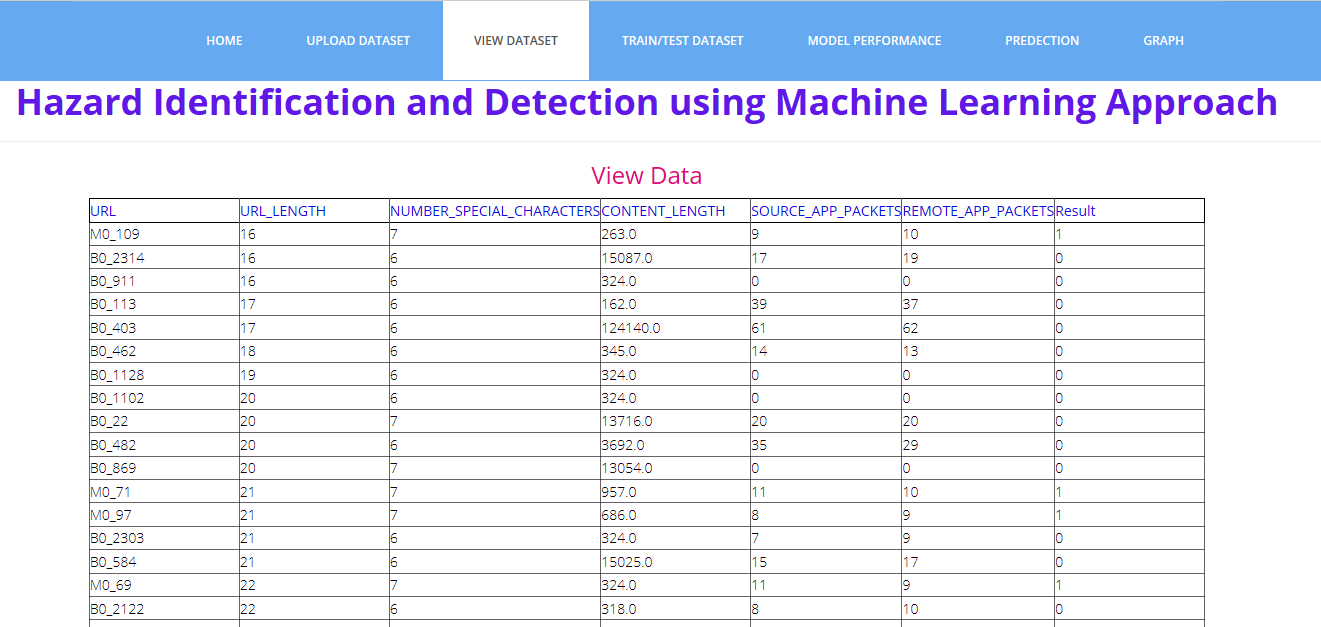
Viewdataset:

Fig.21

Train/test dataset size:

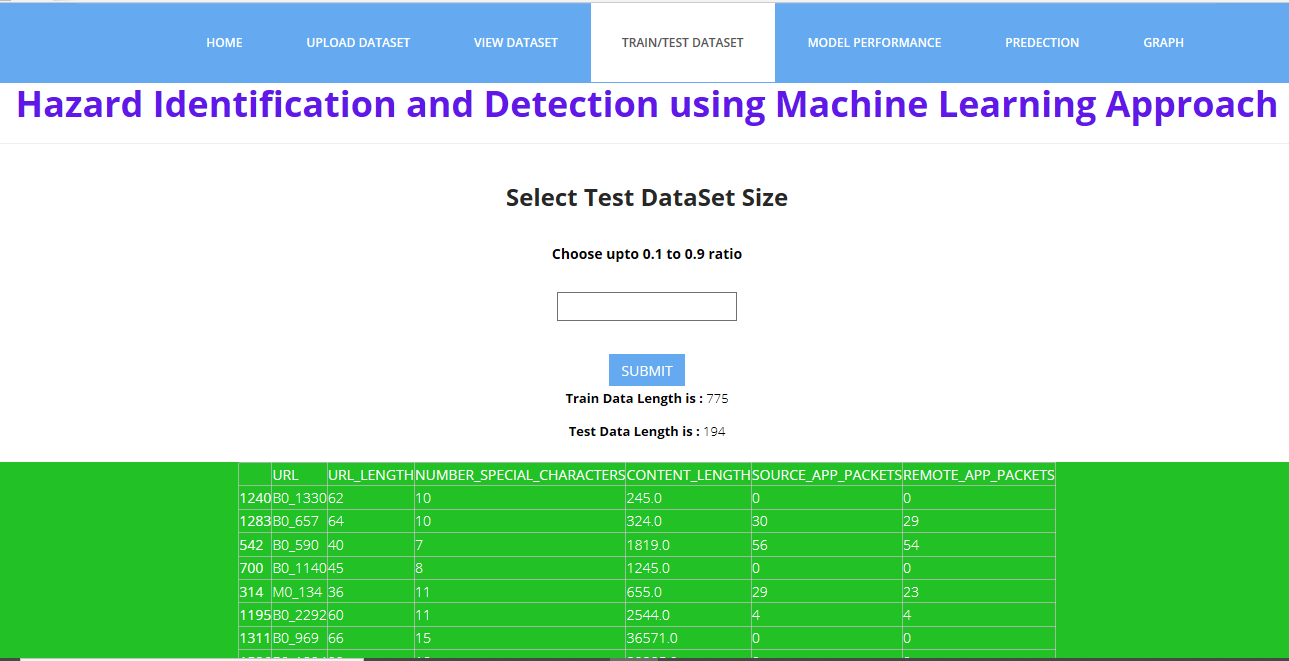
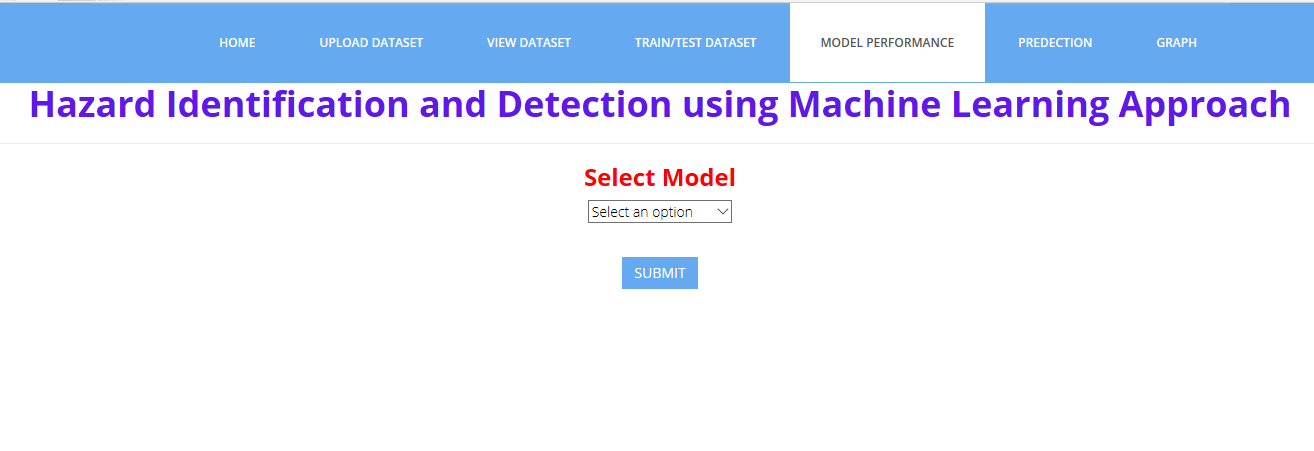
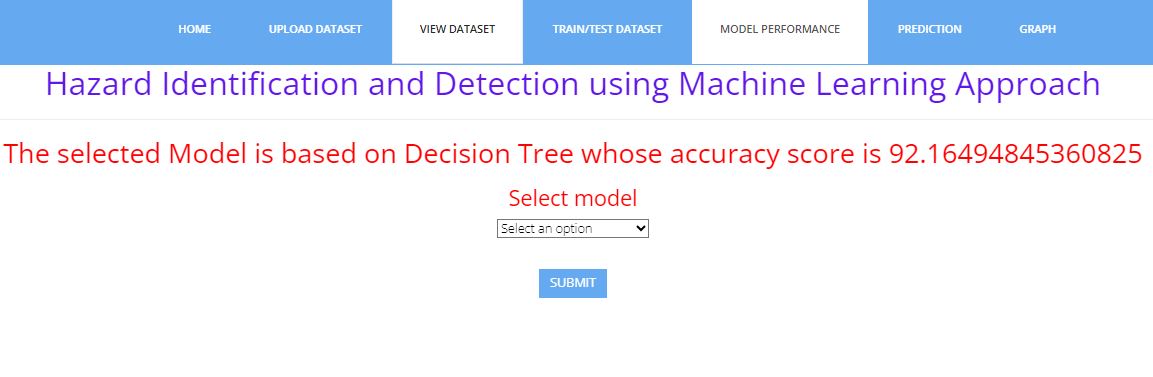


Fig.22

ModelPerformance:



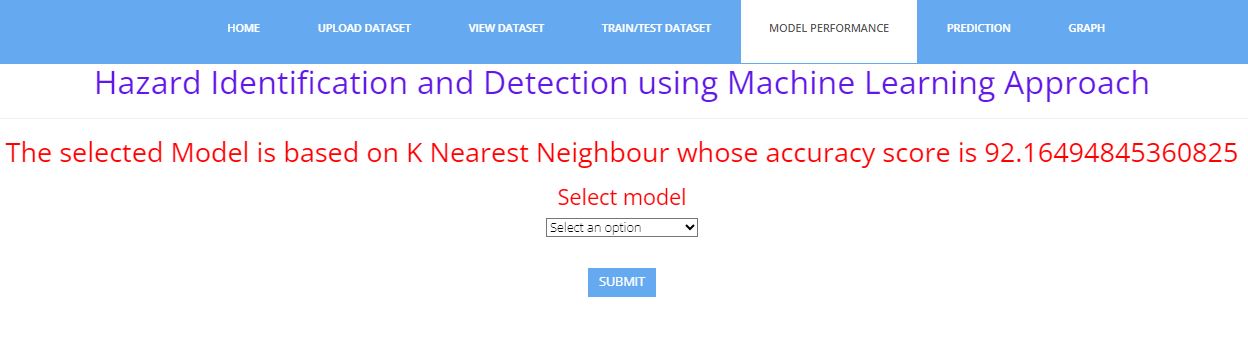
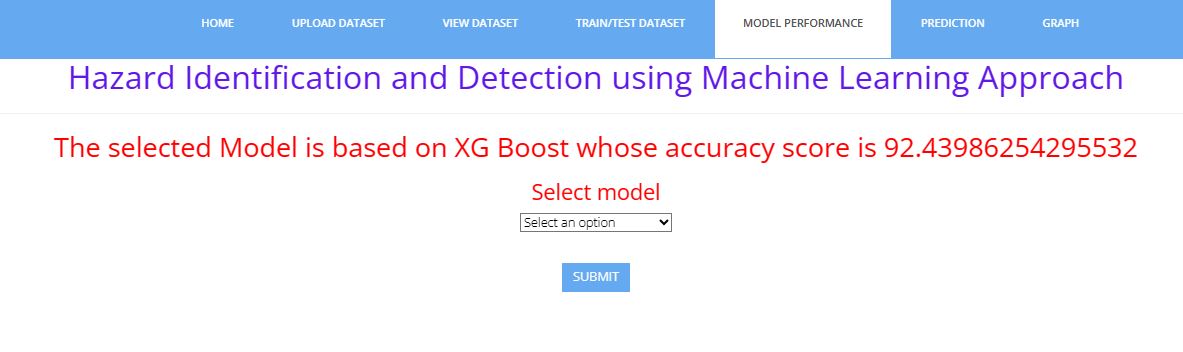
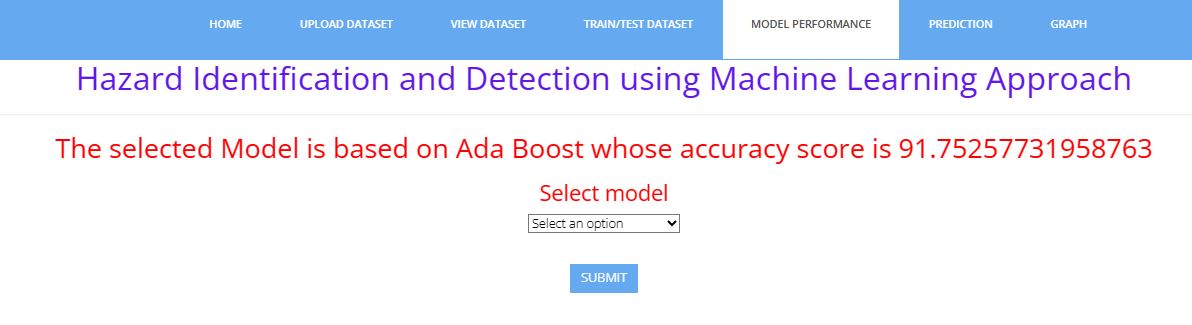
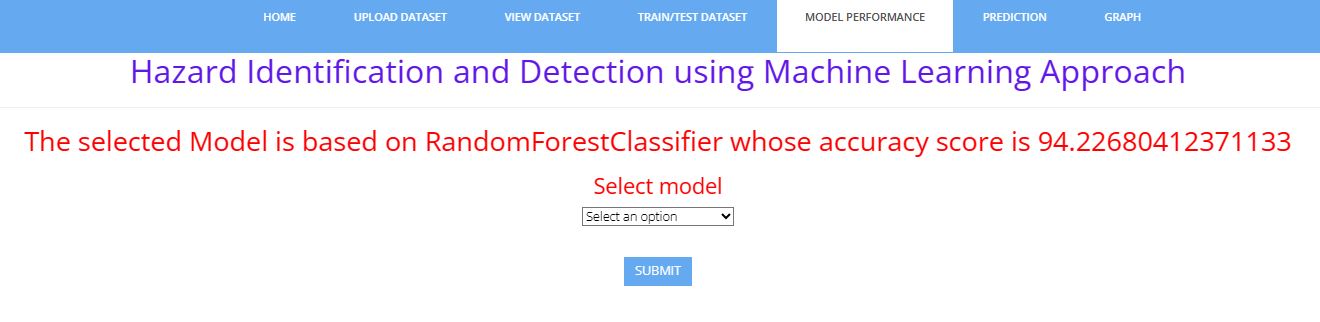


Fig.23







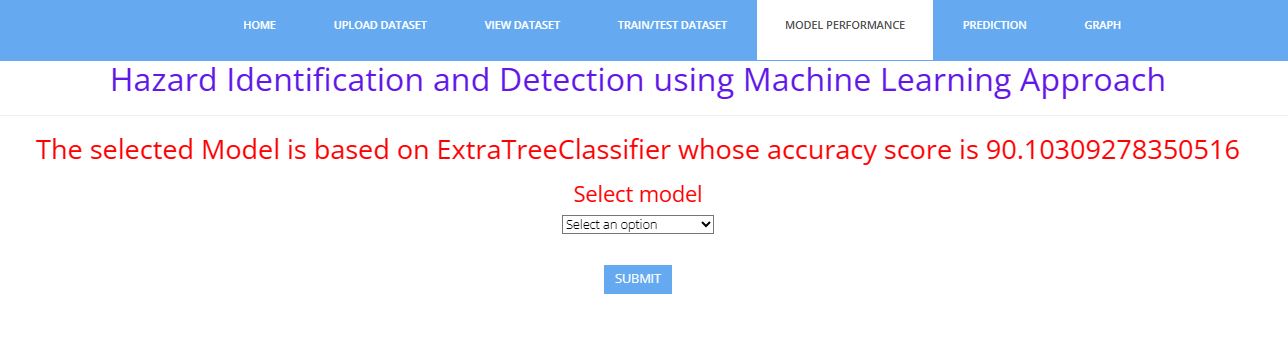
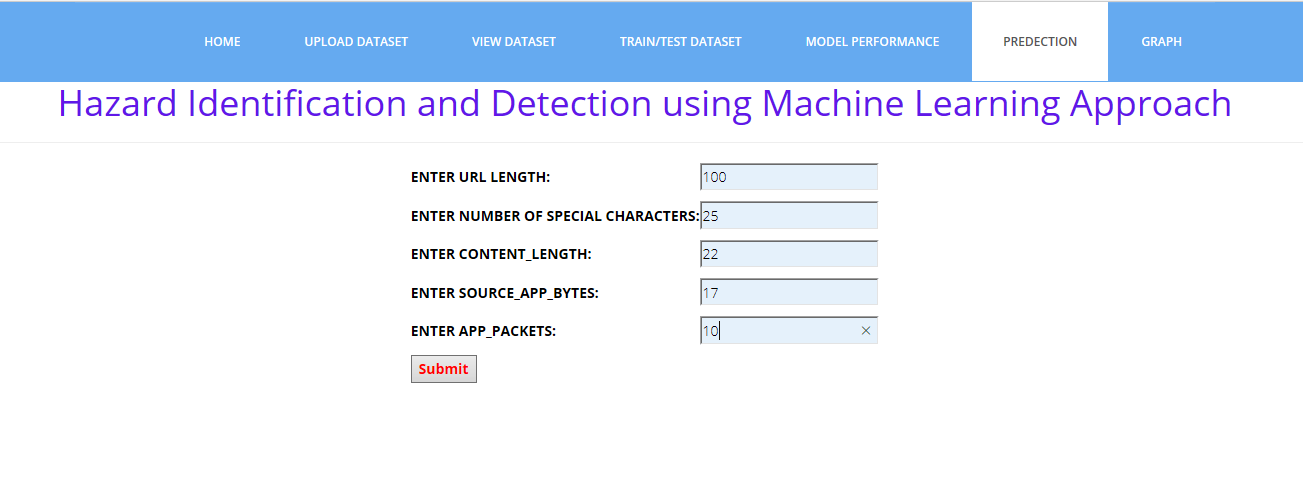
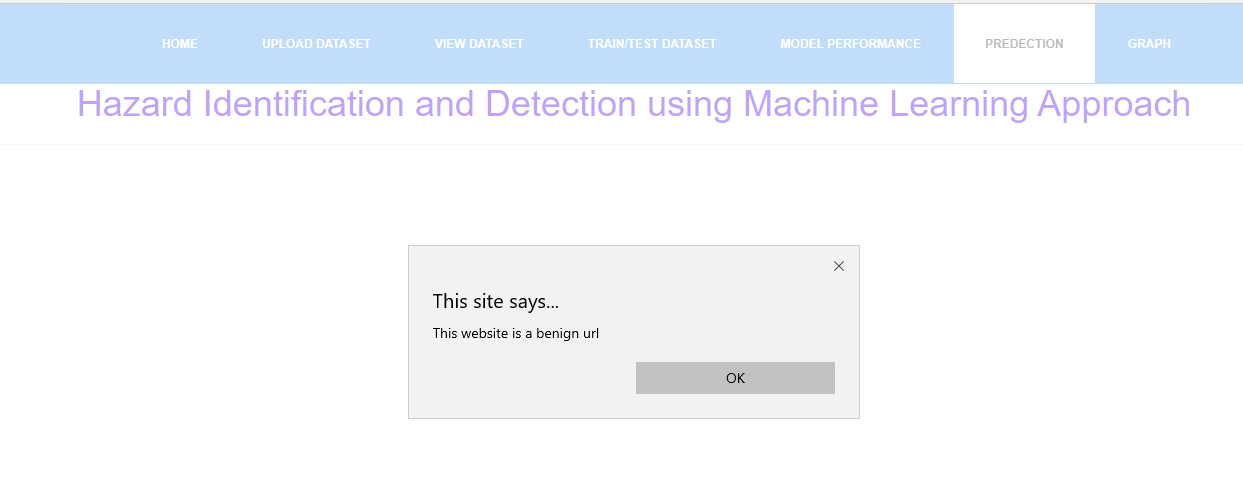


Fig.24

Prediction:





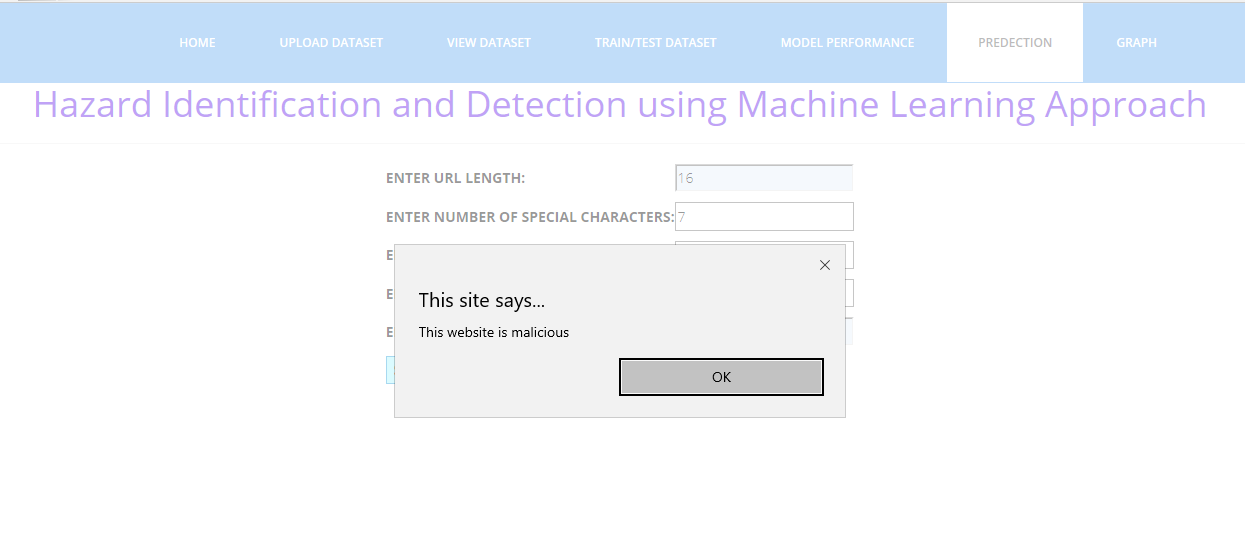


Fig.25

Graph:

**

Fig.26

**CONCLUSION AND FUTURE SCOPE:**

The identification of malicious web pages is a growing concern within the cybersecurity domain. While numerous research endeavors have been committed to addressing the complexities linked with identifying malicious web pages, these initiatives frequently entail substantial time and resource investments. In this research work, we introduce a pioneering website classification system that leverages URL attributes to anticipate the character of web pages, discerning between malign and benign ones by employing machine learning algorithms.

Among the machine learning models employed in our investigation, the Random Forest (RF) classifier distinguishes itself with exceptional performance, achieving an impressive accuracy rate of 95%.

The empirical results gleaned from our experiments unequivocally highlight the effectiveness of our approach in the successful identification of malicious web pages. These findings underscore its potential as an efficient and promising cybersecurity solution, capable of mitigating the risks posed by malicious online content.

FUTURE SCOPE:

The future scope for hazard identification and detection using ML is promising, with the potential to save lives, protect the environment, and enhance disaster preparedness and response. As ML technologies continue to advance and become more accessible, these systems will play an increasingly vital role in addressing a wide range of hazards and ensuring safety and resilience in various domains.

In future work, it has been planned to expand the feature sets and analysis using various sources of data to enhance the classifier performance.

**REFERENCES**

[1] Tao, Wang, Yu Shunzheng, and Xie Bailin. "A novel framework for learning to detect malicious web pages." In 2010 International Forum on Information Technology and Applications, vol. 2, pp. 353-357. IEEE, 2010.

[2] Aldwairi, Monther, and Rami AL Salman. "MalURLs: A lightweight malicious website classification based on URL features." Journal of Emerging Technologies in Web Intelligence 4, no. 2 (2012): 128-133.

[3] Hwang, Young Sup, Jin Baek Kwon, Jae Chan Moon, and Seong Je Cho. "Classifying malicious web pages by using an adaptive support vector machine." Journal of Information Processing Systems 9, no. 3 (2013): 395-404.

[4] Yue, Tao, Jianhua Sun, and Hao Chen. "Fine-grained mining and classification of malicious Web pages." In 2013 Fourth International Conference on Digital Manufacturing & Automation, pp. 616-619. IEEE, 2013.

[5] Yoo, Suyuan, Sehun Kim, Anil Choudhary, O. P. Roy, and T. Tuithung. "Two-phase malicious web page detection scheme using misuse and anomaly detection." International Journal of Reliable Information and Assurance 2, no. 1 (2014): 1-9.

[6] Kim, Sungjin, Jinkook Kim, Seok woo Nam, and Dohoon Kim. "WebMon: ML-and YARA-based malicious webpage detection." Computer Networks 137 (2018): 119-131.

[7] <https://archieve.ics.uci.edu/ml/dataset/>

[8] Website: https://www.jetbrains.com/pycharm/

**JOURNAL CERTIFICATES**

****

****



**HAZARD IDENTIFICATION AND DETECTION USING MACHINE LEARNING APPROACH**

**1 B.v.s. Pavan Kumar**

1M.Tech

1Department of IT&CA, 1Andhra University College of Engineering,

**2 A.Mary Sowjanya**

2Associate Professor

2Department of CS&SE, Visakhapatnam, Andhra Pradesh, India

**ABSTRACT**

In an ever-evolving technological landscape, the ability to accurately identify and detect hazards is of paramount importance to ensure safety across various domains. Hazards, encompassing potential threats and risks that can lead to harm or adverse events, necessitate proactive measures for timely intervention. This project focuses on harnessing the power of Machine Learning (ML) techniques to enhance hazard identification and detection processes. Internet browsing has seamlessly integrated into our everyday routines, and in order to engage users, different browser vendors constantly strive to introduce new functionalities and advanced features. Unfortunately, these enhancements also create opportunities for attackers to exploit vulnerabilities, posing risks to websites and users. Current security measures are insufficient in protecting surfers, necessitating the development of a fast and accurate model capable of distinguishing between benign and potentially harmful web pages. In this research paper, we develop a novel classification system utilizing machine learning classifiers, including random forest, support vector machine, naïve Bayes, and logistic regression. for the purpose of examining and identifying malicious web pages. By extracting features from special Uniform Resource Locators (URLs), we train the classifiers to predict whether a webpage is malicious or benign. Experimental results demonstrate that the random forest classifier compared to other machine learning classifiers, achieves an accuracy rate of 95%. Achieving a higher classification accuracy is crucial in enhancing web security and protecting users from potential cyber threats.

***Keywords*-** *malicious web page, machine learning, detection, URL, cyber threats.*

**INTRODUCTION**

As the internet continues to experience rapid growth, an expanding array of services, including Online banking, electronic commerce, social media networking, bill settlement, and digital learning have become readily accessible to users through web browsers and web applications. However, with the advancement of browsers and their numerous features, there is a growing risk of users' personal and sensitive information being compromised. Many users, especially those who are not well-versed in online security, are unaware of various malware threats. Consequently, they can easily fall prey to intruders by simply clicking on malicious websites. These websites enable attackers to detect vulnerabilities and inject harmful code to obtain remote access to the target's webpage. Thus, it is crucial to accurately identify web pages in this ever-expanding web landscape. To address these challenges, blacklisting services have been integrated into browsers. Our work, we introduce a self-learning approach that relies on a streamlined set of features for the categorization of web pages. The

primary objective is to utilize six machine learning classifiers to distinguish web pages into two distinct categories: benign and malicious.

**LITERATURE SURVEY**

Researchers have proposed various techniques, for the identification of malicious web pages by employing methods, encompassing techniques such as blacklisting, static analysis, and dynamic analysis.

Tao et al. [1] introduced an innovative framework that leverages supervised machine learning to autonomously determine the nature of a web page, whether it is malicious or benign. Their classification relied on specific features, and they compiled a dataset of benign web pages for this purpose.

Adware and rami et al. [2] put forth a lightweight self-learning methodology for categorizing malicious web pages, utilizing a framework named MALURL. They utilized Genetic Algorithm (GA) for the training of classifiers designed to detect malicious web pages. Their training dataset included benign websites from Alexa and malicious ones from Phish Tank, resulting in an average system precision of 87%.

Hwang et al. [3] employed the Adaptive SVM (SVM) machine learning technique, known for its ability to effectively adapt to new training data, thereby reducing the risk of misclassifying novel web pages.

Yue et al. [4] introduced a method for classifying malicious web pages, utilizing 30 distinctive features with the assistance of machine learning algorithms such as K-NN and SVM. Their research indicated that the K-NN algorithm outperformed SVM. They implemented two classification models for the detection of malicious web pages and specific threat types.

Yoo et al. [5] introduced two distinct detection methods: misuse detection, which aims to identify known malicious web pages, and anomaly detection, designed to detect previously unidentified malicious web pages. In their experiments, using the RafaBot dataset within the WEKA tool, they achieved a notable detection rate of up to 98%. However, it's worth noting that the false positive rate was relatively high, reaching 30.5%.

Kim et al. [6] introduced WebMon, an automated and minimally interactive malicious webpage detection tool. WebMon leverages machine learning and YARA signatures to discern detrimental components within web resources loaded via WebKit2-based browsers.

**METHODOLOGY**

In this section, we provide a detailed discussion about our proposed approach to identifying the malicious web page. To address the drawback of previous studies we design a new web site classification system based on the URL features to identify malicious websites which are shown in fig.1. In step 1 according to our requirements, we have imported packages and downloaded a dataset from the internet source contains both the malicious and benign web sites. In step 2 we have designed our dataset consisting of 7 URL features and 1782 records. Then we manually divide the dataset into two sets; one is for training set made up of 812 records and another is for testing set consists of 970 records. In step 3 machine learning classifiers are trained to create a Machine Learning (ML) model with the help of the training set. In the final step, the ML model is verified with the testing set to obtain our required result. If the Type attribute value contains 0 means the inputted URL is a benign web site else it is a malicious web site.

The outcome of our experimentation illustrates the performance of our methodology with an impressive accuracy rate, surpassing 98%, in effectively detecting malicious URLs, while also achieving a precision of over 93% in correctly identifying the specific attack types associated with these URLs.

Additionally, we provide comprehensive analyses of the effectiveness of each group of discriminative features incorporated into our methodology. Furthermore, we delve into an exploration of the potential susceptibilities of these features to evasion techniques, fostering a comprehensive comprehension of the system's strengths and constraints.

**ARCHITECTURE**

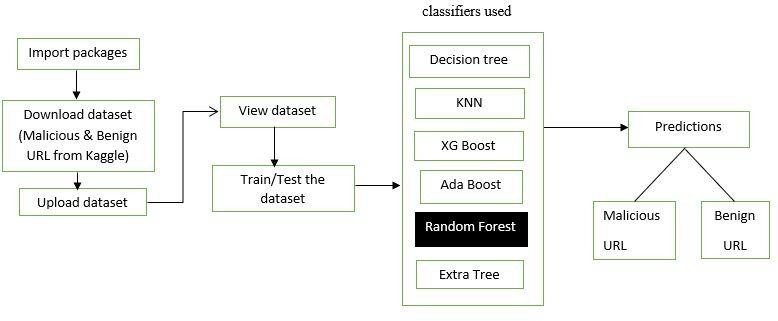


fig 1. Proposed Approach for Malicious Web Page Detection

**RESULT ANALYSIS**

1. *Dataset:*

The provided dataset, sourced from Kaggle database [7], encompasses a collection of attributes characterizing instances of cloud storage access. Each entry comprises features like URL attributes, URL length, the count of special characters, content length, source and remote application packets, culminating in an outcome label with 56 entries, the dataset exhibits a blend of binary classification outcomes (1 or 0), representing public integrity auditing results. While some fields exhibit missing values (NA), the dataset provides valuable insights into factors affecting cloud storage security. Derived from Kaggle, it offers a foundation for research and analysis in identity-based public integrity auditing, contributing to advancements in cloud data protection and privacy preservation.

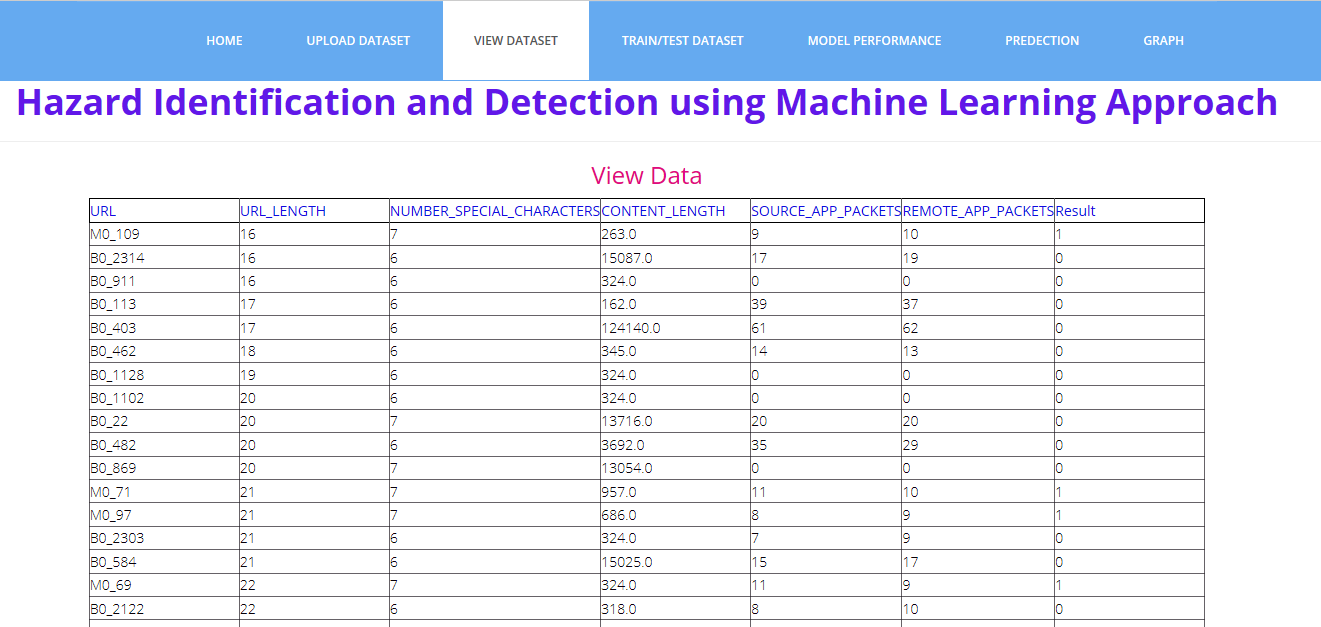


fig 2. A snapshot of dataset.

1. *Graph:*



fig 3. Graphical representation of RF approach

1. *Comparative Analysis of Classifiers Used:*

Various experiments have been carried out by implementing the classification algorithms such as Decision tree, K-Nearest Neighbours (KNN), Extreme Gradient (XG) Boost, Adaptive Boost, Random Forest, Extra Tree Classifiers. All the experiments were coded and tested in PyCharm Software [8] which is an interactive python environment for Machine Learning. With its integrated support for Pandas, Scikit-Learn, Matplotlib, markup language, plots, and tables, a much more appealing and understandable presentation of the flow of the code can be made. We then compare the performance of the Six machine learning classifiers. We have used the precision, recall and accuracy to evaluate the detection performance because it correctly labels a webpage. So, to obtain the best results, accuracy performance metric plays a vital role. We notice that the machine learning classifier RF obtains higher accuracy, whose performance is better than the other classifiers on malicious web page detection.

Table: Performance Comparison of different classifiers

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifiers** | **Precision** | **Recall** | **Accuracy** |
| Decision Tree | 0.96 | 0.94 | 0.91 |
| K-N Neighbours | 0.93 | 0.98 | 0.92 |
| XG Boost | 0.96 | 0.98 | 0.94 |
| Ada Boost | 0.96 | 0.98 | 0.94 |
| **Random Forest** | 0.95 | 1.00 | **0.95** |
| Extra Tree | 0.94 | 0.98 | 0.93 |

CONCLUSION

The identification of malicious web pages is a growing concern within the cybersecurity domain. While numerous research endeavours have been committed to addressing the complexities linked with identifying malicious web pages, these initiatives frequently entail substantial time and resource investments. In this research work, we introduce a pioneering website classification system that leverages URL attributes to anticipate the character of web pages, discerning between malign and benign ones by employing machine learning algorithms.

Among the machine learning models employed in our investigation, the Random Forest (RF) classifier distinguishes itself with exceptional performance, achieving an impressive accuracy rate of 95%.

The empirical results gleaned from our experiments unequivocally highlight the effectiveness of our approach in the successful identification of malicious web pages. These findings underscore its potential as an efficient and promising cybersecurity solution, capable of mitigating the risks posed by malicious online content.

REFERENCES

1. Tao, Wang, Yu Shunzheng, and Xie Bailin. "A novel framework for learning to detect malicious web pages." In 2010 International Forum on Information Technology and Applications, vol. 2, pp. 353-357. IEEE, 2010.
2. Aldwairi, Monther, and Rami AL Salman. "MalURLs: A lightweight malicious website classification based on URL features." Journal of Emerging Technologies in Web Intelligence 4, no. 2 (2012): 128-133.
3. Hwang, Young Sup, Jin Baek Kwon, Jae Chan Moon, and Seong Je Cho. "Classifying malicious web pages by using an adaptive support vector machine." Journal of Information Processing Systems 9, no. 3 (2013): 395- 404.
4. Yue, Tao, Jianhua Sun, and Hao Chen. "Fine-grained mining and classification of malicious Web pages." In 2013 Fourth International Conference on Digital Manufacturing & Automation, pp. 616-619. IEEE, 2013.
5. Yoo, Suyuan, Sehun Kim, Anil Choudhary, O. P. Roy, and T. Tuithung. "Two-phase malicious web page detection scheme using misuse and anomaly detection." International Journal of Reliable Information and Assurance 2, no. 1 (2014): 1-9.
6. Kim, Sungjin, Jinkook Kim, Seok woo Nam, and Dohoon Kim. "WebMon: ML-and YARA-based malicious webpage detection." Computer Networks 137 (2018): 119-131.
7. <https://archieve.ics.uci.edu/ml/dataset/>
8. Website: https:/[/www.jetbr](http://www.jetbrains.com/pycharm/)a[ins.com/pycharm/](http://www.jetbrains.com/pycharm/)