A MAJOR PROJECT REPORT

ON

UPI FRAUD DETECTION USING MACHINE LEARNING

Submitted to

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY, HYDERABAD

In partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

"CSE-CYBER SECURITY"

Submitted By

P Bhargavi (20D41A6238)

Under the esteemed guidance of

Mrs. V. SWATHI

(Assistant Professor)

DEPARTMENT OF CSE-CYBER SECURITY

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, Accredited by NBA & NAAC, Affiliated to JNTUH)

Sheriguda, Ibrahimpatnam, Ranga Reddy-501510,

2023-2024

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, accredited by NBA, Affiliated to JNTUH)

DEPARTMENT OF CSE-CYBER SECURITY

CERTIFICATE

Certified that the Major Project entitled **"UPI FRAUD DETECTION USING MACHINE LEARNING"** is a bonafide work carried out by **P Bhargavi (20D41A6238),** in partial fulfilment for the award of Bachelor of Technology in **CSE-CYBER SECURITY** of SICET, Hyderabad for the academic year **2023- 2024**.The Project has been approved as it satisfies academic requirements in respect of the work prescribed for **IV YEAR, II-SEMESTER** of **B. Tech** course.

V. SWATHI
(Internal Guide)

G. UMA MAHESWARI
(Head of The Department)

EXTERNAL EXAMINER

DECLARATION

I declare the project work **"UPI FRAUD DETECTION USING MACHINE LEARNING"** was carried out by me and this work is not the same as that of any other and has not been submitted to anywhere else for the award of any other degree.

Signature of the candidate:

……..…….

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We are thankful to principal **Dr. G. SURESH,** for giving us the permission to carry out this project and for providing necessary infrastructure and labs.

We are highly indebted to **Prof. G. UMA MAHESWARI**, Head of the Department **of CSE-Cyber Security,** for providing valuable guidance at every stage of this project.

We are grateful to our internal project guide **Mrs. V. SWATHI**, Assistant Professor for her constant motivation and guidance given by her during the execution of this project work.

We would like to thank the Teaching & Non-Teaching staff of Department of CSE-Cyber Security for sharing their knowledge with us, finally we express our sincere thanks to everyone who helped directly or indirectly for the completion of this project.

P Bhargavi (20D41A6238)

ABSTRACT

Increase in UPI usage for online payments and cases of fraud associated with it are also rising. Few steps involving UPI transaction process using a Hidden Markov Model (HMM). An HMM is initially trained for a cardholder. If a UPI transaction is not accepted by the trained HMM. It is considered to be fraudulent. People can use UPIs for online transactions as it provides an efficient and easy-to-use facility. With the increase in usage of UPIs, the capacity of UPI misuse has also enhanced. UPI frauds cause significant financial losses for both UPI holders and financial companies. In this Project, the main aim is to detect such frauds, including the accessibility of public data, high-class imbalance data, the changes in fraud nature, and high rates of false alarm. The main focus has been to apply the recent development of Machine Learning algorithms for this purpose. We have created 5 Algorithms to detection the UPI Fraud and evaluated results Based on that

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SCREENSHOTS

1.INTRODUCTION

Unified Payments Interface (UPI) has revolutionized the way digital payments are made in India, offering a seamless and convenient platform for instant fund transfers. However, with the rapid growth of UPI transactions, the threat of fraudulent activities has also escalated. Traditional rule-based fraud detection systems often struggle to keep pace with the evolving tactics of fraudsters. To address this challenge, the integration of machine learning (ML) techniques holds immense promise in fortifying UPI security.

UPI enables users to link multiple bank accounts into a single mobile application, simplifying transactions without the need for cumbersome banking details. Its popularity stems from its user-friendly interface and widespread adoption across various platforms and banks in India. However, the ease of use also makes it an attractive target for fraudsters.

Fraudulent activities in UPI transactions encompass various forms, including phishing attacks, identity theft, account takeover, and unauthorized transactions. Detecting these fraudulent activities in real-time poses a significant challenge due to the sheer volume and speed at which transactions occur.

Machine learning offers a proactive and data-driven approach to UPI fraud detection. By analyzing historical transaction data, ML algorithms can identify patterns indicative of fraudulent behavior. These algorithms can discern subtle anomalies and deviations from normal transaction patterns, enabling early detection and prevention of fraudulent activities.

Key steps in implementing ML for UPI fraud detection include data collection, feature engineering, model training, evaluation, and deployment. Comprehensive transactional data, including transaction amounts, timestamps, user demographics, device information, and metadata, are collected for analysis. Feature engineering involves extracting relevant features from the raw data, such as transaction frequency, velocity, geographic location, device fingerprinting, and behavioral patterns.

Supervised learning algorithms, including logistic regression, decision trees, random forests, or gradient boosting machines, are trained on labeled datasets to build fraud detection models. Techniques like oversampling of minority class instances are employed to address class imbalance issues.

1.1 MOTIVATION:

The motivation to undertake this project stems from the pressing need to address the growing threat of fraud within the Unified Payments Interface (UPI) ecosystem. As digital transactions continue to rise in popularity, so too does the risk of fraudulent activities, which can result in significant financial losses for individuals, businesses, and financial institutions alike. By developing advanced fraud detection systems tailored specifically for UPI transactions, the project seeks to mitigate these risks and safeguard the integrity of the digital payment infrastructure.

Furthermore, there is a strong societal motivation to combat financial fraud, as it not only impacts individuals and businesses financially but also erodes trust in digital payment systems. By deploying cutting-edge machine learning algorithms and real-time fraud detection techniques, the project aims to instill confidence among users and stakeholders, thereby fostering greater adoption and utilization of UPI transactions. This, in turn, contributes to the broader goal of promoting financial inclusion and digital empowerment, particularly in regions where digital payments play a crucial role in economic development.

Moreover, the project aligns with the broader objectives of promoting technological innovation and advancing research in the field of machine learning and data analytics. By pushing the boundaries of what is possible in fraud detection, the project aims to contribute to the collective knowledge base and drive advancements in the application of artificial intelligence and data science to real-world problems. Ultimately, the motivation behind this project lies in its potential to make tangible and meaningful contributions to enhancing security, promoting trust, and fostering innovation within the UPI ecosystem and beyond.

1.2 OBJECTIVES

The project's primary objective is to advance fraud detection capabilities within the Unified Payments Interface (UPI) ecosystem. This involves deploying cutting-edge machine learning algorithms, such as Random Forest, AdaBoost, and Support Vector Machine, to accurately identify fraudulent transactions. By focusing on refining feature engineering techniques and exploring ensemble learning methods, the aim is to enhance the precision and reliability of fraud detection models. This objective underscores the project's commitment to leveraging state-of-the-art technologies to combat financial fraud effectively.

Another key objective is the development of real-time fraud detection systems tailored specifically for UPI transactions. These systems will leverage streaming data processing techniques and low-latency model inference to analyze transactions instantaneously. By enabling prompt intervention and mitigation of fraudulent activities as they occur, the project seeks to minimize financial losses and safeguard the integrity of the UPI ecosystem. This objective underscores the project's emphasis on proactive measures to address fraud in real-time, ensuring timely detection and response.

Additionally, the project aims to uphold principles of transparency, interpretability, and ethical considerations throughout the development and deployment of fraud detection systems. By implementing techniques for enhancing model interpretability, such as feature importance analysis, the project seeks to foster greater understanding and trust among stakeholders. Moreover, ethical guidelines and fairness metrics will be integrated into the design process to ensure equitable treatment of users and mitigate unintended consequences associated with algorithmic decision-making. These objectives emphasize the project's commitment to building robust and ethical fraud detection systems that inspire confidence and promote trust within the UPI ecosystem.

2.LITERATURE SURVEY

2.1 Introduction:

The DDoS attacks on the cloud computing environment are mainly application layer which sends out requests following the communication protocol which are then hard to distinguish in the network layer because their pattern matches the legitimate requests thus making the traditional defence systems not applicable. DDoS flooding attacks on cloud can be of various categories like session and request flooding attacks, slow response and asymmetric attack. All these flooding attacks generate traffic which resembles that of a legitimate user which becomes tougher for the target to distinguish between attack and legitimate traffic thus blocking the services for the legitimate user

2.2 Review of the existing system:

2.2.1 Existing System 1:

Title: Accuracy of classifier refers to the ability of classifier. It predicts the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.

Author: Abdul Raoofhttps://ieeexplore.ieee.org/author/37086371068

Technologies and Algorithm Used: The study is implemented on the Structure for cloud security with efficient security in communication system and AES based file encryption system. This security architecture can be easily applied on PaaS, IaaS and SaaS and one time password provides extra security in the authenticating users.

Advantages:

- Performance time and accuracy.

Disadvantages:

- Training model prediction on Time is High .
- It is based on Low Accuracy.

2.2.2 Existing System 2:

Title: Analysis and Countermeasures for Security and Privacy Issues in Cloud Computing, 2019

Author: Q. P. Rana, Nitin Pandey

Technologies and Algorithm Used:

The cloud computing environment is adopted by a large number of organizations so the rapid transition toward the clouds has fuelled concerns about security perspective. There are numbers of risks and challenges that have emerged due to use of cloud computing. The aim of this paper is to identify security issues in cloud computing which will be helpful to both cloud service providers and users to resolve those issues. As a result, this paper will access cloud security by recognizing security requirements and attempt to present the feasible solution that can reduce these potential threats.

Advantages:

- More effective and efficient**.**
- Identifies security issues in cloud computing.
- Offers feasible solutions to mitigate security concerns.
- Provides insights beneficial to both cloud service providers and users.

Disadvantages:

- Not give accurate prediction result.
- Lacks specificity regarding technologies and algorithms used.
- Unclear scope of analysis on various security and privacy issues.
- Potential obsolescence due to publication in 2019.

2.2.3 Existing System 3:

Title: Using Firefly and Genetic Metaheuristics for Anomaly Detection based on Network Flows, 2014

Author: Faisal Hussain

Technologies and Algorithm Used:

In this work, we proposed a Traffic monitoring is a challenging task which requires efficient ways to detect every deviation from the normal behavior on computer networks. In this paper, we present two models to detect network anomaly using flow data such as bits and packets per second based on: Firefly Algorithm and Genetic Algorithm. Both results were evaluated to measure their ability to detect network anomalies, and results were then compared. We experienced good results using data collected at the backbone of a university.

Advantages:

- Efficiency measure and the accuracy.
- Enhanced detection capabilities with the use of two algorithms.
- Flexibility to adapt to different network environments and anomaly patterns.
- Comprehensive evaluation of performance.
- Real-world applicability demonstrated with data from a university backbone.
- Potential for improved accuracy in anomaly detection.

Disadvantages:

- Not give accurate prediction result.
- Increased complexity in system configuration and maintenance.
- Higher computational overhead due to running multiple algorithms.
- Dependency on the quality and consistency of flow data.
- Limited generalizability of findings beyond the university backbone.
- Challenges in interpreting results from complex metaheuristic algorithms.

2.2.4 Existing System 4:

Title: A Multiple-Layer Representation Learning Model for Network-Based Attack Detection, 2018

Author: Suresh M

Technologies and Algorithm Used:

The proposed solutions are this ensures fine-grained detection of various attacks. The proposed framework has been compared with the existing deep learning models using three real datasets (a new dataset NBC, a combination of UNSW-NB15 and CICIDS2017 consisting of 101 classes).

Advantages:

- It performs accurate classification of health state in comparison with other methods.
- Fine-grained detection of various network-based attacks.
- Comparison with existing deep learning models enhances credibility and understanding of the proposed framework's effectiveness.
- Utilization of multiple real datasets increases the robustness and reliability of the evaluation process.
- Potential for improved accuracy in detecting a wide range of attack classes due to the multiple-layer representation learning model.

Disadvantages:

- It is low in efficiency.
- Lack of specificity regarding the technologies and algorithms employed in the proposed framework.
- The effectiveness of the proposed model might be limited to the specific datasets used for evaluation.
- Dependency on the quality and diversity of the datasets used, which may not fully represent all possible real-world scenarios.

2.2.5 Existing System 5:

Title: Detecting Distributed Denial of Service Attacks Using Data Mining Techniques, 2018

Author: Linga

Technologies and Algorithm Used:

In this study, we DDoS (Distributed Denial of Service) attack has affected many IoT networks in recent past that has resulted in huge losses. We have proposed deep learning models and evaluated those using latest CICIDS2017 datasets for DDoS attack detection which has provided highest accuracy as 97.16% also proposed models are compared with machine learning algorithms.

Advantages:

- The proposed solution can successfully detect network intrusions and DDOS communication with high precision**.**
- More Reliable.

Disadvantages:

- It is less in efficiency and not give perfect result.
- This finding is disadvantageous to the organization experiencing such attack.
- The difficulty in identifying all articles that are related to this study.

3. EXISTING SYSTEM

In existing system, the malware files in the training dataset have been taken from the Virus Heaven collection. Malware files from the Wild List collection and clean files from various operating systems (different files than those used in the first database) are included in the test dataset. Trojans, backdoors, hacking tools, rootkits, worms, and other malware are among the malware collections found in the training and test datasets. Table II's first and third columns show the proportion of certain malware kinds among all files in the training and test datasets, respectively. Out of all the unique feature value combinations for the training dataset, the second column in Table II shows the corresponding percentage of malware unique combinations.

3.1 DISADVANTAGE

- Lack of diversity in malware sources may result in a biased training dataset.
- Limited representation of real-world scenarios in the test dataset composed of clean files.
- Inadequate coverage of various malware types may lead to a narrow detection scope.
- Potential bias in feature representation could affect the accuracy and reliability of the detection system.
- Dependency on static features may overlook dynamic behaviors of polymorphic or obfuscated malware.
- Difficulty in detecting novel or previously unseen malware variants due to limited dataset diversity.
- Risk of overfitting to specific malware characteristics present in the training data.
- Challenges in generalizing the performance of the detection system beyond the specific datasets used.

4. PROPOSED SYSTEM

The dataset of transaction patterns was used as input into this system. The dataset repository provided the raw data. The data pre-processing stage must then be put into practice. The dataset must then be divided into test and train groups. Ratio is the basis for the data separation. The majority of the data will be in the train. A reduced amount of the data will be present during the test. The model is evaluated during the training phase, and predictions are made during the testing phase. The categorization method, or machine learning, must next be put into practice. machine learning algorithms, including SVM, AdaBoost, and RF. La2222stly, the experimental findings demonstrate that performance indicators like comparison and accuracy

4.1ADVANTAGES

1. **Automation and Scalability**:
   - Machine learning-based systems can automatically analyze a large volume of transactions.
   - They scale efficiently to handle the growing number of UPI transactions.
2. **Real-Time Detection**:
   - Machine learning models operate in real-time, swiftly identifying suspicious activities.
   - This enables timely intervention and minimizes potential losses.
3. **Adaptability**:
   - Machine learning algorithms adapt to evolving fraud patterns.
   - They learn from new data and adjust their rules without manual intervention.
4. **Feature Extraction**:
   - Machine learning models can extract relevant features from transaction data.
   - These features enhance fraud detection accuracy.
5. **Reduced Human Error**:
   - Automated systems reduce reliance on manual reviews.
   - Human errors, which can occur during manual inspection, are minimized.
6. **Anomaly Detection**:
   - Machine learning identifies anomalies that rule-based systems might miss.
   - It detects subtle patterns indicative of fraud.

5. SYSTEM REQUIREMENTS

The initial stage in the requirements analysis process involves defining the system requirements for a specific software system. These requirements encompass functional, performance, and security aspects, and include usage scenarios from user, operational, and administrative viewpoints. The aim of the software requirements specification is to offer a comprehensive overview of the software project, outlining its objectives, parameters, and objectives. This document outlines the project's intended audience, user interface, and hardware and software prerequisites. It articulates how the client, team, and end users perceive the project and its functionalities.

HARDWARE AND SOFTWARE REQUIREMENT

5.1 HARDWARE REQUIREMENTS:

- System : Pentium IV 2.4 GHz
- Hard Disk : 200 GB
- Mouse : Logitech.
- Keyboar : 110 keys enhanced
- RAM : 4GB

5.2 SOFTWARE REQUIREMENT:

- O/S : Windows 7.
- Language : Python
- Front End : Anaconda Navigator – Spyder Notebook

5.3 SOFTWARE DESCRIPTION

5.3.1 Python

Python is one of those rare languages which can claim to be both *simple* and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. The official introduction to Python is Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. I will discuss most of these features in more detail in the next section.

5.3.2Features of Python

- **Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English, although very strict English! This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

- **Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax, as already mentioned.

- **Free and Open Source**

Python is an example of a *FLOSS* (Free/Libré and Open Source Software). In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

- **High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

- **Portable**

Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, and # -*- coding: utf-8 -*-z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!

- **Interpreted**

A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.

Python, on the other hand, does not need compilation to binary. You just *run* the program directly from the source code. Internally, Python converts the source code into an intermediate form called byte codes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer and it just works!

- **Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In *procedure-oriented* languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In *object-oriented* languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

- **Extensible**

If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use it from your Python program.

- **Embeddable**

You can embed Python within your C/C++ programs to give *scripting* capabilities for your program's users.

- **Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the *Batteries Included* philosophy of Python.

Besides the standard library, there are various other high-quality libraries which you can find at the Python Package Index.

6. MODULES

6.1 MODULES

- Data Selection and Loading
- Data Preprocessing
- Splitting Dataset into Train and Test Data
- Classification
- Prediction
- Result Generation

6.2 MODULES DESCRIPTION

6.2.1 Data Selection and Loading:

- Data selection is the process of determining the appropriate data type and source, as well as suitable instruments to collect data.
- Data selection precedes the actual practice of data collection and it is the process where data relevant to the analysis is decided and retrieved from the data collection.
- In this project, the Malware dataset is used for detecting Malware type prediction.

6.2.2 Data Preprocessing:

- The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.
- **Missing Data:** This situation arises when some data is missing in the data. It can be handled in various ways.
- **Ignore the tuples:** This approach is suitable only when the dataset we have is quite large and multiple values aremissing within a tuple.
- **Fill the Missing values:** There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.
- **Encoding Categorical data**: That categorical data is defined as variables with a finite set of label values. That most machine learning algorithms require numerical input and output variables. That an integer and one hot encoding is used to convert categorical data to integer data.

- **Count Vectorizer:** Scikit-learn's CountVectorizer is used to convert a collection of text documents to a vector of term/token **counts**. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

6.2.3 Splitting Dataset into Train and Test Data:

- Data splitting is the act of partitioning available data into two portions, usually for cross-validator purposes.
- One Portion of the data is used to develop a predictive model and the other to evaluate the model's performance.
- Separating data into training and testing sets is an important part of evaluating data mining models.
- Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing.
- To train any machine learning model irrespective what type of dataset is being used you have to split the dataset into training data and testing data.

6.2.4 Classification:

- Classification is the problem of identifying to which of a set of categories, a new observation belongs to, on the basis of a training set of data containing observations and whose categories membership is known.
- **Random forests** or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.
- **Decision Trees** are a type of Supervised Machine Learning (that is you **explain** what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. An **example** of a **decision tree** can be **explained** using above binary **tree**.
- The **SVM** is one of the most powerful methods in machine learning algorithms. It can find a balance between model complexity and classification ability given limited sample information. Compared to other machine learning methods, the SVM has many advantages

in that it can overcome the effects of noise and work without any prior knowledge. The SVM is a non-probabilistic binary linear classifier that predicts an input to one of two classes for each given input. It optimizes the linear analysis and classification of hyperplane formation techniques.

- **The NN algorithm** is mainly used for classification and regression in machine learning. To determine the category of an unknown sample, all training samples are used as representative points, the distances between the unknown sample and all training sample points are calculated, and the NN is used. The category is the sole basis for determining the unknown sample category. Because the NN algorithm is particularly sensitive to noise data, the K-nearest neighbour algorithm (KNN) is introduced. The main concept of the KNN is that when the data and tags in the training set are known, the test data are input, the characteristics of the test data are compared with the features corresponding to the training set, and the most similar K in the training set is found.

6.2.5 Prediction:

- Predictive analytics algorithms try to achieve the lowest error possible by either using "boosting" or "bagging".
- **Accuracy** − Accuracy of classifier refers to the ability of classifier. It predict the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.
- **Speed** − Refers to the computational cost in generating and using the classifier or predictor.
- **Robustness** − It refers to the ability of classifier or predictor to make correct predictions from given noisy data.
- **Scalability** − Scalability refers to the ability to construct the classifier or predictor efficiently; given large amount of data.
- **Interpretability** − It refers to what extent the classifier or predictor understands.

6.2.6 Result Generation

The Final Result will get generated based on the overall classification and prediction. The performance of this proposed approach is evaluated using some measures like, accuracy, precision, Recall, ROC, confusion matrix

- **Accuracy:** Accuracy of classifier refers to the ability of classifier. It predicts the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.
- AC= $\frac{TP+TN}{TP+TN+FP+FN}$
- **Precision:** Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives.
- Precision= $\frac{TP}{TP+FP}$
- **Recall:** Recall is the number of correct results divided by the number of results that should have been returned. In binary classification, recall is called sensitivity. It can be viewed as the probability that a relevant document is retrieved by the query.
- **ROC**: ROC curves are frequently used to show in a graphical way the connection/trade-off between clinical sensitivity and specificity for every possible cut-off for a test or a combination of tests. In addition the area under the ROC curve gives an idea about the benefit of using the test(s) in question.
- **Confusion matrix:**A **confusion matrix** is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

7. SYSTEM DESIGN

7.1 Architecture Diagram

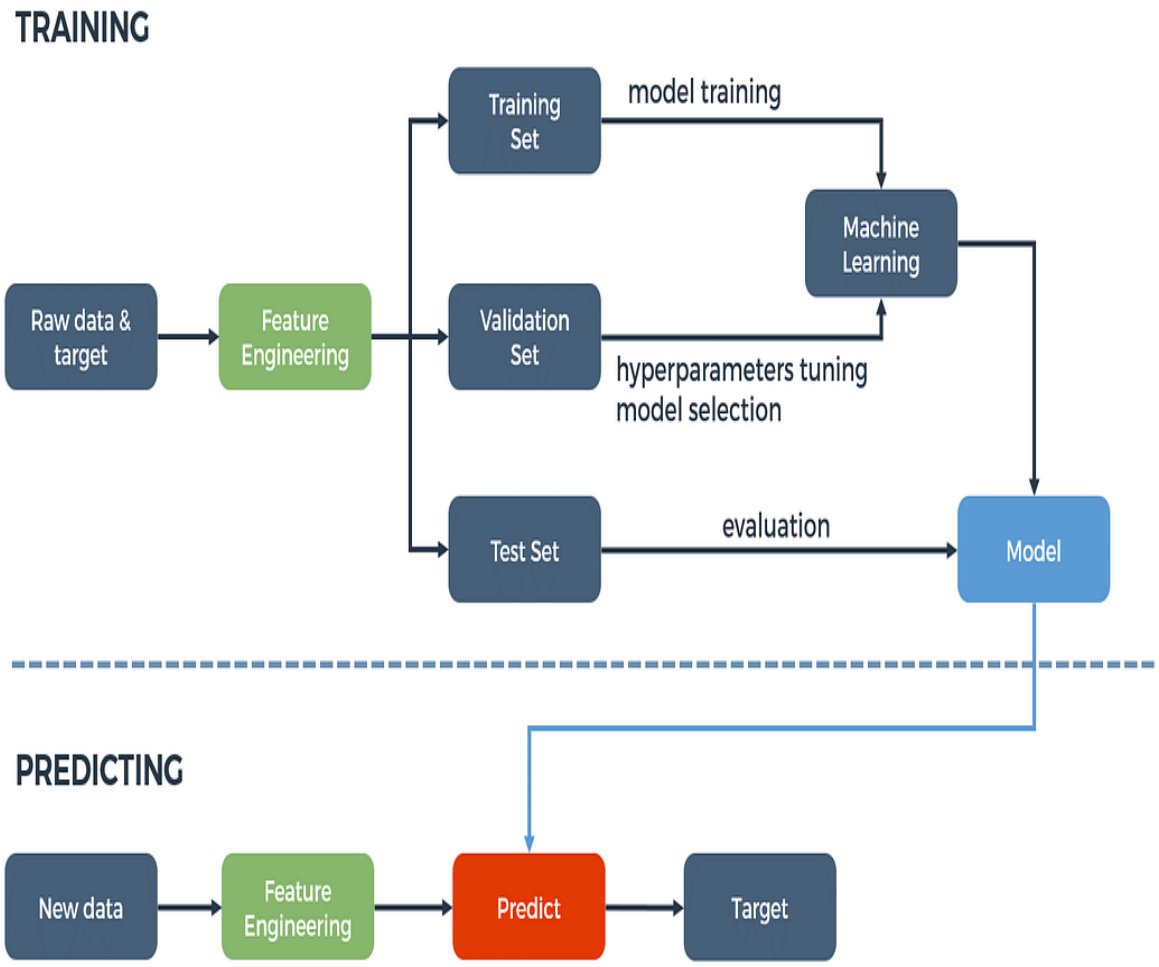


Figure 1- Architecture Diagram

7.2 DATAFLOW DIAGRAM

A flowchart is a visual representation of a process, system, or algorithm, using symbols and arrows to depict the sequence of steps or actions. They are widely utilized across different domains such as software development, engineering, business operations, project management, and decision-making, aiding in the comprehension, analysis, and communication of complex procedures.

Key elements of a flowchart include:

- **Start and end points:** Represent the beginning and conclusion of the process, typically denoted by oval or rounded rectangle shapes.
- **Process steps:** Represent individual tasks or operations within the process, depicted as rectangles with rounded corners.
- **Decision points:** Indicate branching in the flow based on conditions or criteria, usually depicted as diamonds with yes/no or true/false branches.
- **Arrows:** Connect symbols to illustrate the sequence of steps, with arrows indicating the direction of flow.
- **Connectors:** Used to link different parts of the flowchart that are located on separate pages or sections.
- **Input/output:** Represent input data or output results in the process, typically depicted as parallelograms or trapezoids.

Flowcharts can be categorized into different types based on their purpose and structure:

- **Process flowchart:** Illustrates the sequence of steps in a process from start to finish, commonly used in business processes and workflow management.
- **System flowchart:** Focuses on depicting interactions and components of a system or software application, showing data flow and module interactions.
- **Data flowchart:** Emphasizes the flow of data within a system or process, showcasing input, processing, storage, and output.
- **Workflow diagram:** Similar to a process flowchart, representing the sequence of tasks in a project or business process, including decision points and branches.

Flowcharts serve several purposes, including analyzing and understanding complex processes or systems, designing and optimizing workflows, communicating procedures to stakeholders, identifying inefficiencies, and documenting software algorithms and decision-making logic.

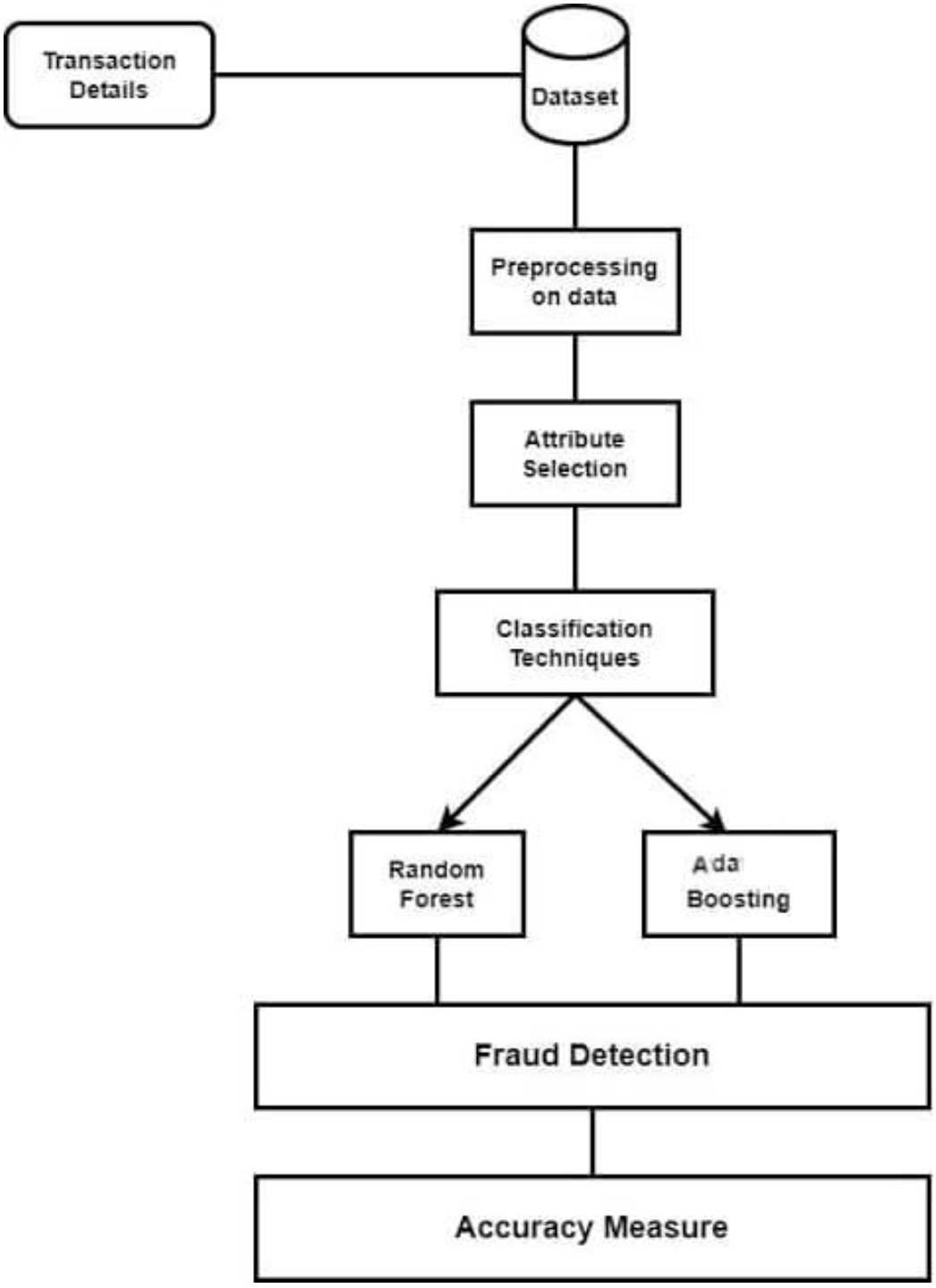


Figure 2 – DataFlow Diagram

7.3 UML DIAGRAMS

Unified Modelling Language (UML) diagrams are graphical representations used in software engineering to visualize, specify, construct, and document the structure and behavior of a system. There are several types of UML diagrams, each serving a specific purpose.

7.3.1 USE CASE DIAGRAM

A use case diagram, a crucial component of Unified Modeling Language (UML) diagrams in software engineering, is utilized to visually outline the interactions between users (actors) and a system. It offers a broad view of the system's functionalities and the involvement of actors in utilizing those functionalities. Use case diagrams prove invaluable in the initial phases of software development for capturing and conveying system requirements from the user's perspective.

Key elements of a use case diagram include:

1.Actors: Actors are external entities, such as users, other systems, or devices, that interact with the system under consideration. They are depicted by stick figures or labeled ovals in the diagram.

2.Use Cases: Use cases delineate specific functionalities or tasks that the system must execute to fulfill a goal for an actor. Each use case describes a sequence of actions or interactions between the actor and the system. Use cases are represented as ovals with labels inside them.

3.Relationships: Relationships within a use case diagram illustrate the interactions between actors and use cases.

- **Association:** A solid line connecting an actor to a use case signifies the actor's involvement in that particular use case.
- **Include**: An include relationship, denoted by a dashed line with an arrowhead, indicates that one use case encompasses the functionality of another use case. It implies that the included use case is always part of the behavior of the including use case.
- **Extend:** An extend relationship, depicted by a dashed line with an open arrowhead, suggests that one use case can augment another use case under specific conditions. It

represents optional or conditional behavior that may not always be part of the base use case.

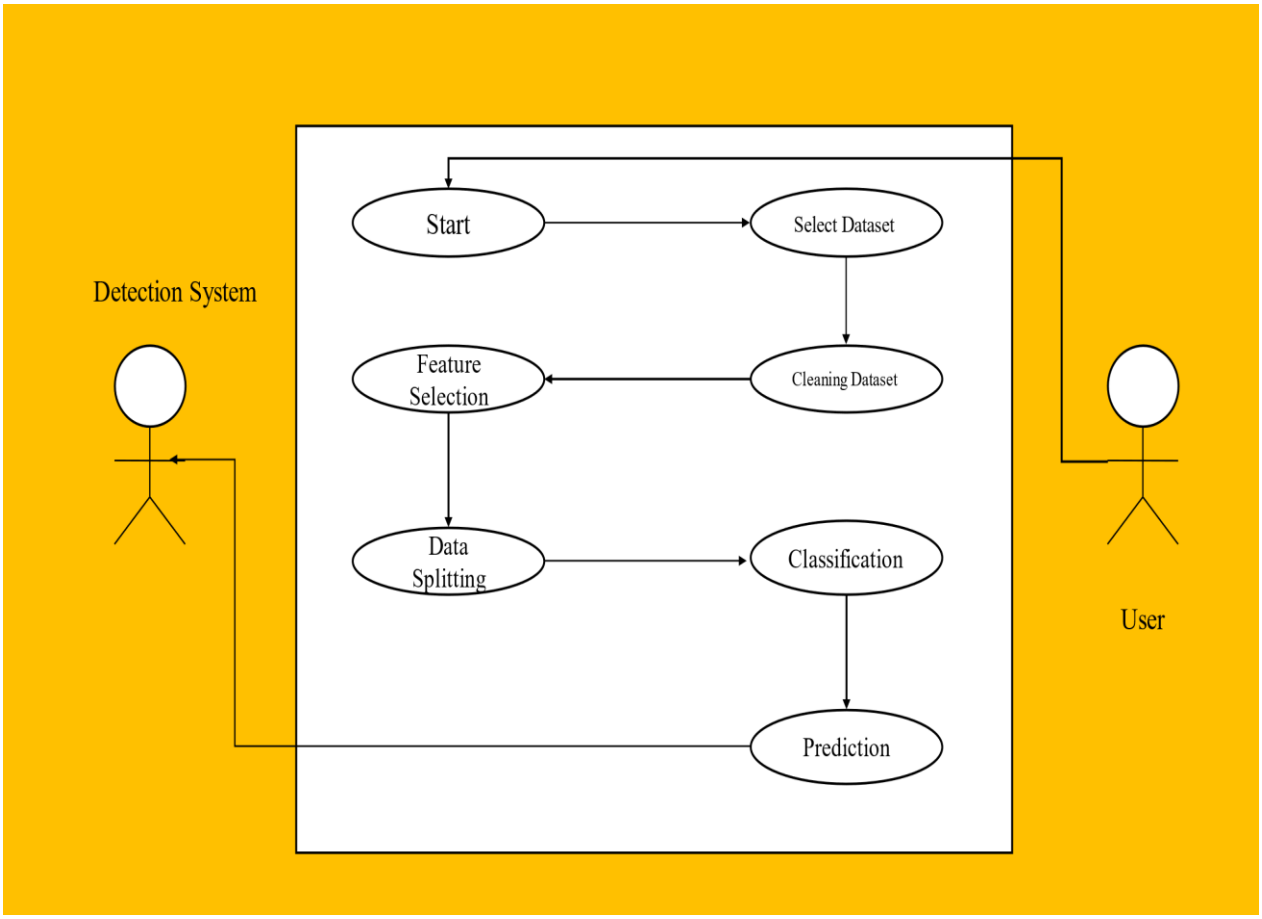


Figure 3 – Use Case Diagram

7.3.2 ACTIVITY DIAGRAM

Activity diagrams are valuable tools in software engineering for visualizing the dynamic behaviors of a system, illustrating how activities are sequenced, coordinated, and controlled to accomplish specific objectives. They are commonly employed during the analysis and design phases to capture the behavior of a system or its components.

The key components of an activity diagram are as follows:

- **Activities**: Representing tasks, actions, or steps within the system or process being modeled, activities are depicted as rounded rectangles with labels inside them.
- **Transitions**: These arrows illustrate the flow of control or data between activities, indicating the sequence in which activities are executed. Transitions may include labels specifying conditions or triggers for their occurrence.
- **Decisions**: Diamond-shaped symbols, also known as decision nodes, denote points in the diagram where the flow can branch based on conditions or decisions. They are essential for modeling conditional behavior within the process.
- **Start and End Nodes**: The start node, often depicted as a solid circle, marks the beginning of the activity diagram, while the end node, represented by a solid circle with a border, signifies the conclusion or completion of the process.
- **Forks and Joins**: Forks, or split nodes, indicate points where the flow diverges into multiple concurrent paths, whereas joins, or merge nodes, indicate points where concurrent paths converge back into a single path.
- **Swimlanes**: Swimlanes help organize activities into distinct lanes or partitions, typically representing different roles, departments, or systems involved in the process.

Activity diagrams serve as effective tools for capturing and communicating the dynamic aspects of a system, aiding in understanding, analyzing, and designing complex processes within software systems.

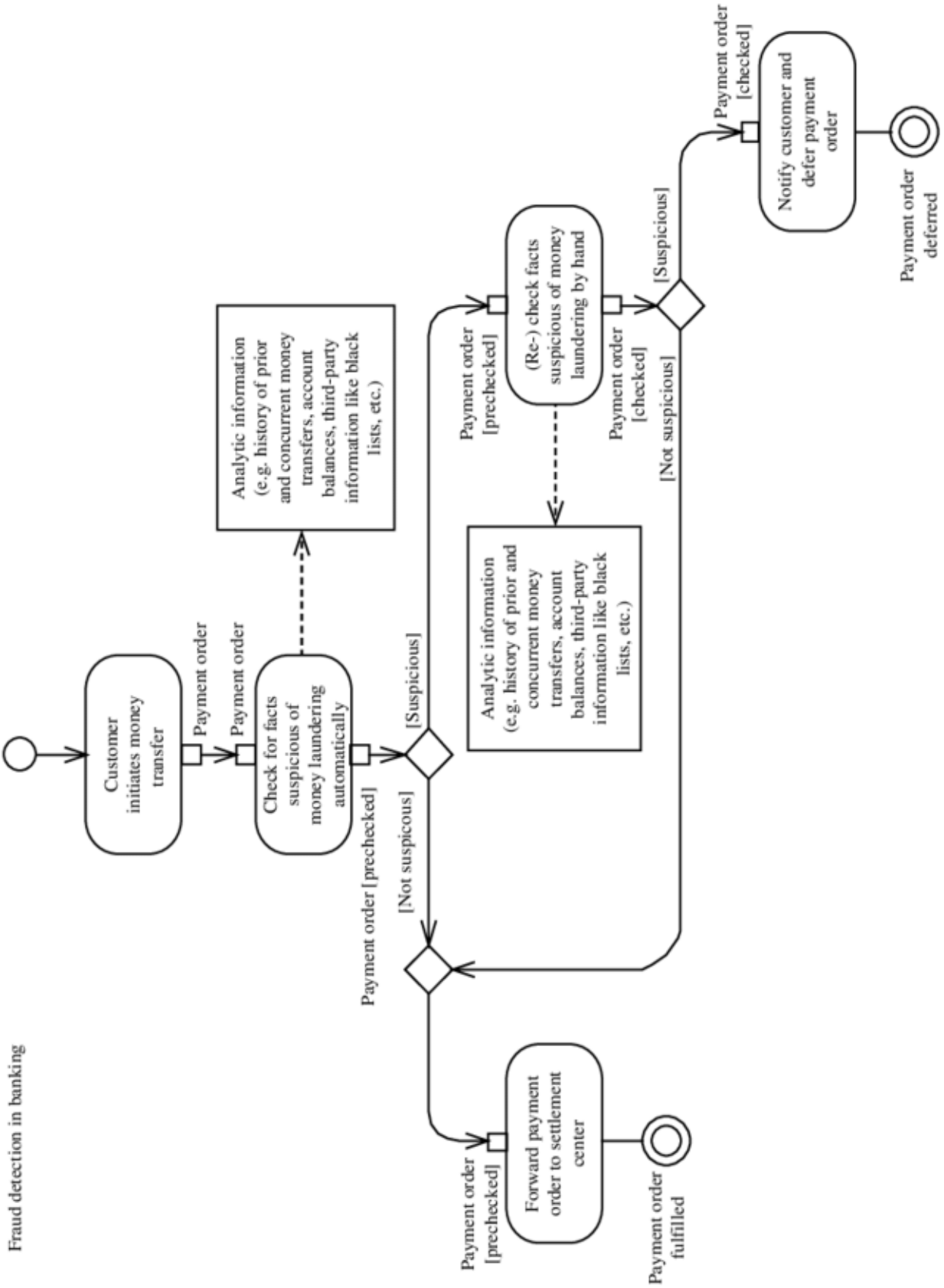


Figure 4-activity diagram

7.3.3. SEQUENCE DIAGRAM

A sequence diagram, a key component of Unified Modeling Language (UML) diagrams in software engineering, is employed to visualize interactions among objects or components within a system over time. It illustrates the sequence of messages exchanged between objects to accomplish specific tasks or scenarios, effectively capturing the dynamic behavior of the system.

The essential components of a sequence diagram are:

- **Lifelines:** Lifelines depict objects or entities participating in the interaction, represented as vertical lines with names at the top, indicating instances of classes or components.
- **Messages**: These arrows signify communication between lifelines, representing method calls, signals, or data exchanges. Messages can be synchronous (solid arrow), asynchronous (dashed arrow), or return messages (dashed arrow with a solid arrowhead).
- **Activations:** Activations indicate the duration during which an object executes a method or processes a message. They are depicted as boxes or rectangles on the lifeline, illustrating the active period of an object.
- **Optional Fragments:** These fragments, such as loops, conditionals, and alternatives, depict different branches or conditions within the sequence diagram, aiding in modeling complex behaviors and decision points in the system's logic.
- **Focus of Control:** The flow of control between objects during the sequence of events is depicted by the ordering of messages and activations on the lifelines, indicating the focus of control.

Sequence diagrams find diverse applications in software development, including:

- Modeling interactions between objects in a system or subsystem.
- Representing the dynamic behavior of use cases or scenarios.
- Designing and validating system architecture and communication protocols.
- Generating test cases based on interaction sequences.
- Communicating and documenting system behavior and message flows.

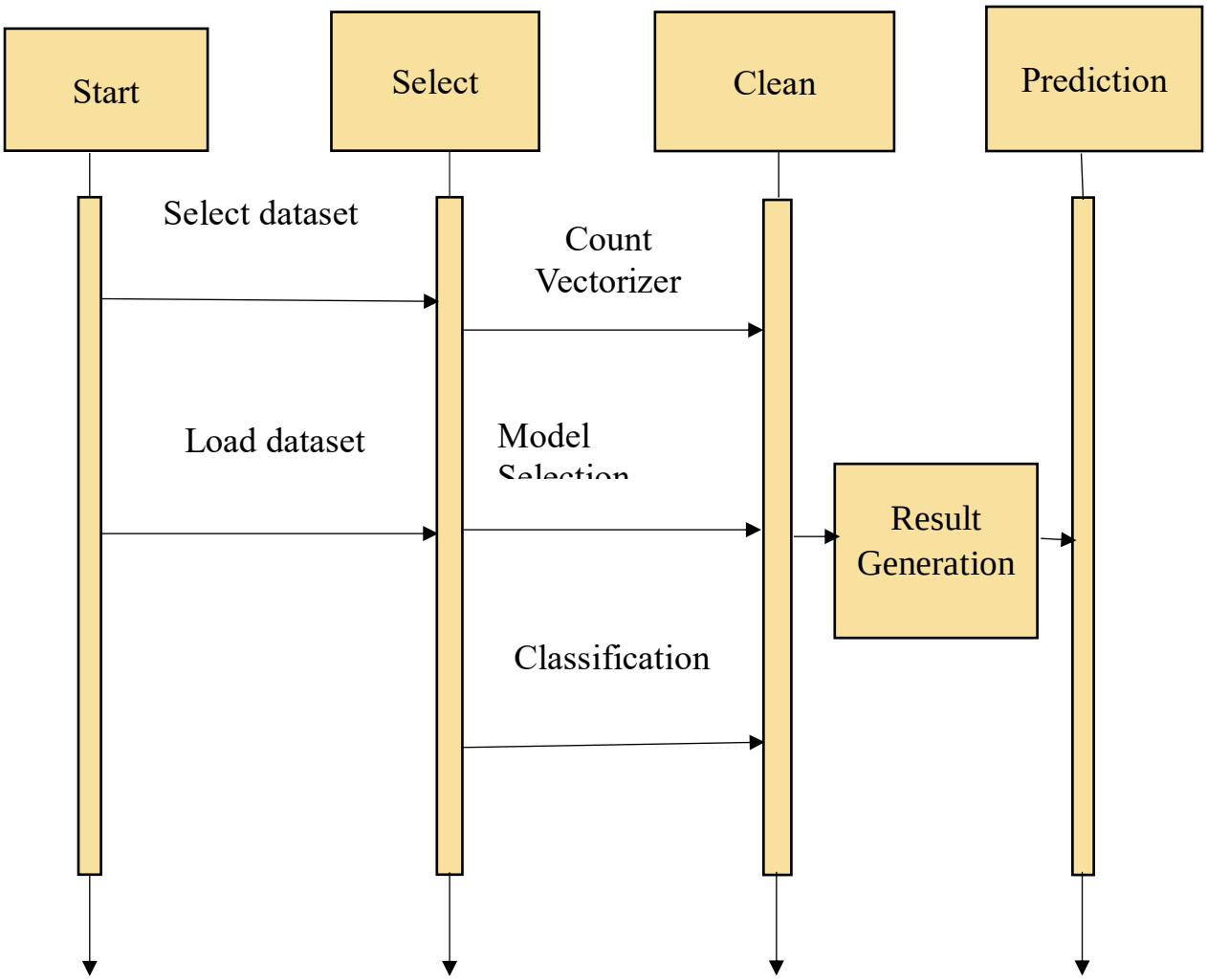


Figure 5 – Sequence Diagram

7.3.4. CLASS DIAGRAM

A class diagram, a crucial component of Unified Modeling Language (UML) diagrams in software engineering, is utilized to visually represent the static structure of a system, emphasizing classes, their attributes, methods, and relationships with other classes. These diagrams are foundational in designing object-oriented systems, serving as a blueprint for software development by showcasing the fundamental components and connections within the system.

Key elements of a class diagram include:

- **Classes:** Representing the core elements of the system, classes encapsulate data (attributes) and behaviors (methods) related to specific concepts or entities. Each class is portrayed as a rectangle divided into three compartments: the class name, attributes, and methods.
- **Attributes:** Attributes denote properties or variables associated with a class, reflecting the data or state of objects within that class. They are listed within the middle compartment of the class rectangle, detailing attributes such as name, type, and visibility.
- **Methods:** Methods, also referred to as operations or behaviors, illustrate the actions that objects of a class can execute. They are listed within the bottom compartment of the class rectangle, showcasing method signatures and visibility.
- **Relationships:** Relationships within a class diagram depict associations, dependencies, generalizations, and aggregations between classes, demonstrating how classes are interconnected or related. These relationships encompass associations, dependencies, generalization (inheritance), and aggregation/composition.

Class diagrams serve various purposes in software development, including:

- Designing and modeling the structure of object-oriented systems.
- Identifying and defining classes, attributes, and behaviors during system analysis and design.
- Facilitating the comprehension and communication of system architecture and class relationships.
- Guiding the implementation process by providing a blueprint for coding classes and their interactions.
- Supporting software maintenance and refactoring through the visualization of class hierarchies and dependencies.

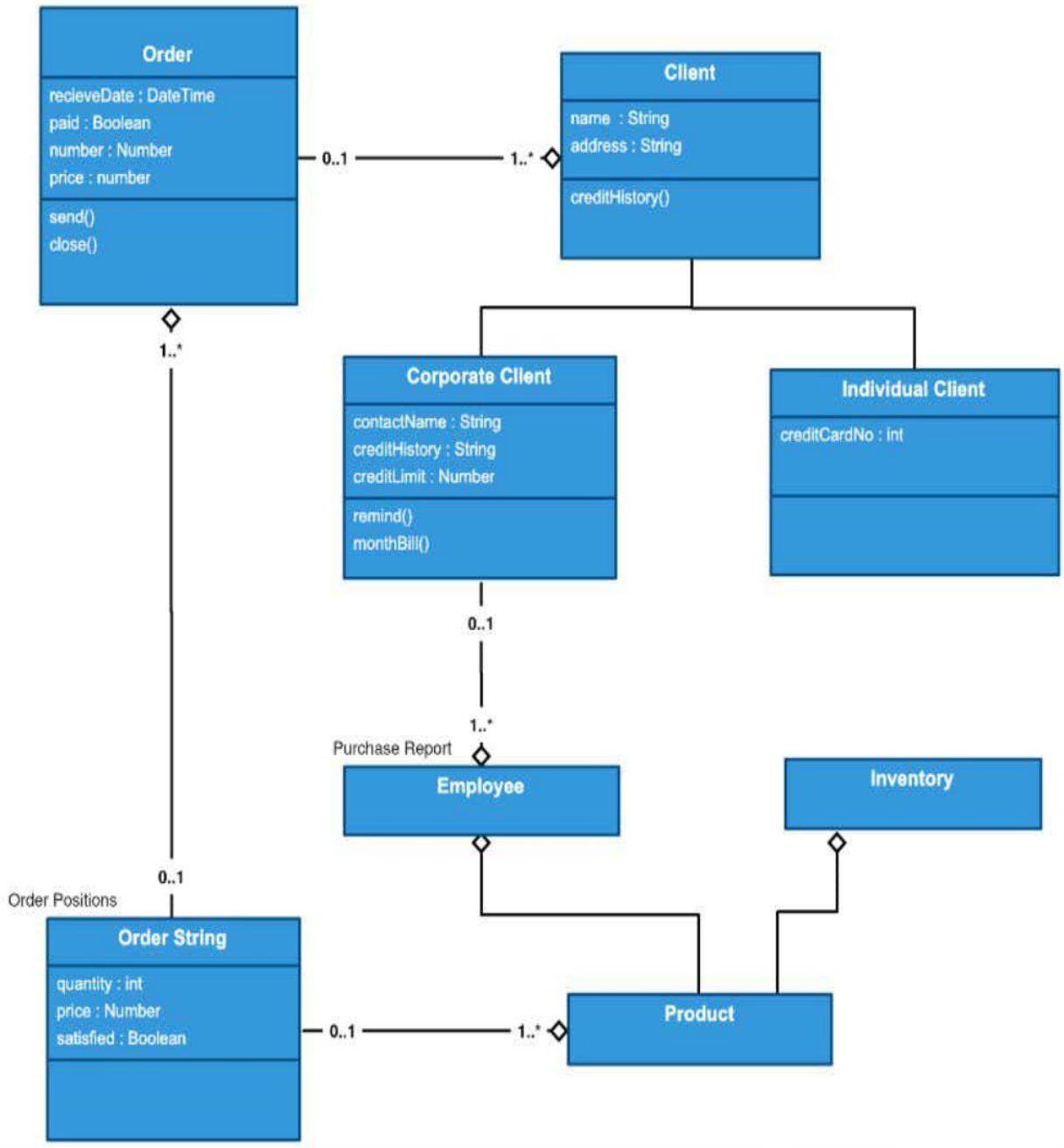


Figure 6- class diagram

8.ALGORITHMS

An algorithm is a step-by-step procedure or set of rules designed to solve a specific problem or perform a particular task. Algorithms are fundamental to computer science and are used in various applications, from sorting data to optimizing processes. They form the backbone of many computer programs and are essential for efficient problem-solving in fields such as artificial intelligence, data analysis, and cryptography. Algorithms can range from simple and straightforward to highly complex and sophisticated, depending on the nature of the problem they address. Regardless of complexity, well-designed algorithms are characterized by their efficiency, accuracy, and scalability

- RANDOM FOREST
- ADABOOST
- SUPPORT VECTOR MACHINE
- CONVOLUTIONAL NEURAL NETWORKS
- RECURRENT NEURAL NETWORK

1. Random Forest (RF):

Random Forest is an ensemble learning method that constructs a multitude of decision trees during training. It combines the predictions of multiple individual trees to improve accuracy and reduce overfitting, making it robust against noise and outliers in the data.

2. AdaBoost:

AdaBoost is an ensemble learning technique that combines multiple weak classifiers to create a strong classifier. It iteratively adjusts the weights of incorrectly classified instances, focusing more on difficult-to-classify cases, thereby improving overall performance.

3. Support Vector Machine (SVM):

Support Vector Machine is a supervised learning algorithm used for classification and regression tasks. It constructs a hyperplane in a high-dimensional space to separate classes and maximize the margin between them, making it effective for handling complex decision boundaries and high-dimensional data.

4. Convolutional Neural Networks (CNNs):

Convolutional Neural Networks are deep learning models commonly used for image recognition and classification tasks. They employ convolutional layers to automatically learn spatial hierarchies of features from input images, enabling effective feature extraction and pattern recognition.

5. Recurrent Neural Networks (RNNs):

Recurrent Neural Networks are a type of deep learning model designed for sequential data processing tasks. They have feedback connections that allow information to persist over time, making them suitable for tasks like natural language processing and time series prediction.

6. Decision Tree:

Decision Tree is a tree-like structure where each internal node represents a feature, each branch represents a decision based on that feature, and each leaf node represents a class label. It's a simple yet powerful algorithm used for classification and regression tasks, offering interpretability and ease of visualization.

7. k-Nearest Neighbors (k-NN):

k-Nearest Neighbors is a non-parametric, instance-based learning algorithm used for classification and regression tasks. It assigns a class label or predicts a value for a new data point based on the majority class or average of the k nearest data points in the feature space. K-NN is simple to implement and works well for datasets with well-defined clusters.

9.IMPLEMENTATION

9.1 Sample Code:

----------**Importing Libraries** ----------

```
import tensorflow as tf
import neural_structured_learning as nsl
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics
print("================================================")
print("Malware Dataset")
print(" Process - Malware Attack Detection")
print("================================================")
##1.data slection----------
#def main():
dataframe=pd.read_csv("training set.csv")
print("-----------------------------------------")
print()
print("Data Selection")
print("Samples of our input data")
print(dataframe.head(10))
print("-----------------------------------------")
print()
```

----------**PRE PROCESSING**----------

```
#checking  missing values
print("-----------------------------------------")
print()
print("Before Handling Missing Values")
print()
```

```
print(dataframe.isnull().sum())
print("-----------------------------------------")
print()

print("-----------------------------------------")
print("After handling missing values")
print()
dataframe_2=dataframe.fillna(0)
print(dataframe_2.isnull().sum())
print()
print("-----------------------------------------")
#label encoding
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
print("-------------------------------------------------")
print("Before Label Handling ")
print()
print(dataframe_2.head(10))
print("-------------------------------------------------")
print()
```

----------DATA SPLITTING----------

```
df_train_y=dataframe_2["label"]
df_train_X=dataframe_2.iloc[:,:20]
from sklearn.preprocessing import LabelEncoder

number = LabelEncoder()

df_train_X['proto'] = number.fit_transform(df_train_X['proto'].astype(str))
```

```
df_train_X['service'] = number.fit_transform(df_train_X['service'].astype(str))
df_train_X['state'] = number.fit_transform(df_train_X['state'].astype(str))
#df_train_X['attack_cat'] = number.fit_transform(df_train_X['attack_cat'].astype(str))
print("================================================")
print(" Preprocessing")
print("================================================")

df_train_X.head(5)
x=df_train_X
y=df_train_y
```

----------FEATURE SELECTION----------

----------KMEANS----------

```
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

x, y_true = make_blobs(n_samples=175341, centers=4,cluster_std=0.30, random_state=0)
plt.scatter(x[:, 0], x[:, 1], s=20);

kmeans = KMeans(n_clusters=3)
kmeans.fit(x)
y_kmeans = kmeans.predict(x)

plt.scatter(x[:, 0], x[:, 1], c=y_kmeans, s=20, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);

plt.title("k-means")
```

```
plt.show()

#-------------------------------------------------------------------------------
import scipy.cluster.hierarchy
from pyxdameraulevenshtein import damerau_levenshtein_distance
def cluster_ngrams(ngrams, compute_distance, max_dist, method):
    indices = np.triu_indices(len(ngrams), 1)
    pairwise_dists = np.apply_along_axis(
        lambda col: compute_distance(ngrams[col[0]], ngrams[col[1]]),
        0, indices)
    hierarchy = scipy.cluster.hierarchy.linkage(pairwise_dists, method=method)
    clusters = dict((i, [i]) for i in range(len(ngrams)))
    for (i, iteration) in enumerate(hierarchy):
        cl1, cl2, dist, num_items = iteration
        if dist >  max_dist:
            break
        items1 = clusters[cl1]
        items2 = clusters[cl2]
        del clusters[cl1]
        del clusters[cl2]
        clusters[len(ngrams) + i] = items1 + items2
    ngram_clusters = []
    for cluster in clusters.values():
        ngram_clusters.append([ngrams[i] for i in cluster])
    return ngram_clusters

def dl_ngram_dist(ngram1, ngram2):

    return sum(damerau_levenshtein_distance(w1, w2) for w1, w2 in zip(ngram1,
        ngram2))
```

```
x_train,x_test,y_train,y_test = train_test_split(df_train_X,y_kmeans,test_size = 0.20,random_state = 42)
x_train,x_test,y_train,y_test = train_test_split(df_train_X,y,test_size = 0.20,random_state = 42)

from sklearn.ensemble import RandomForestClassifier



rf= RandomForestClassifier(n_estimators = 100)
rf.fit(x_train, y_train)
rf_prediction = rf.predict(x_test)
Result_3=accuracy_score(y_test, rf_prediction)*100
from sklearn.metrics import confusion_matrix

print()
print("----------------------------------------------------------------------")
print("Random Forest")
print()
print(metrics.classification_report(y_test,rf_prediction))
print()
print("Random Forest Accuracy is:",Result_3,'%')
print()
print("Confusion Matrix:")
cm2=confusion_matrix(y_test, rf_prediction)
print(cm2)
print("------------------------------------------------------")
print()
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(cm2, annot = True, cmap ='plasma',
      linecolor ='black', linewidths = 1)
plt.show()
```

```
#-------------------------------------------------------------------------------------------
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3,
min_samples_leaf=5)
dt.fit(x_train, y_train)
dt_prediction=dt.predict(x_test)
print()
print("-----------------------------------------------------------------------")
print("Decision Tree")
print()
Result_2=accuracy_score(y_test, dt_prediction)*100
print(metrics.classification_report(y_test,dt_prediction))
print()
print("DT Accuracy is:",Result_2,'%')
print()
print("Confusion Matrix:")
from sklearn.metrics import confusion_matrix
cm1=confusion_matrix(y_test, dt_prediction)
print(cm1)
print("-------------------------------------------------------")
print()
import matplotlib.pyplot as plt
import seaborn as sns

sns.heatmap(cm1, annot = True, cmap ='plasma',
      linecolor ='black', linewidths = 1)
plt.show()



#------------------------------- ROC GRAPH ----------------------------------------
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
from sklearn.ensemble import GradientBoostingClassifier
gradient_booster = GradientBoostingClassifier(learning_rate=0.1)
gradient_booster.get_params()

gradient_booster.fit(x_train,y_train)
gb_prediction = gradient_booster.predict(x_test)

print(classification_report(y_test,gradient_booster.predict(x_test)))

Result_2=accuracy_score(y_test, gb_prediction)*100
print()
print("gradient_booster Accuracy is:",Result_2,'%')
print()
print("Confusion Matrix:")
from sklearn.metrics import confusion_matrix
cm1=confusion_matrix(y_test, dt_prediction)
print(cm1)
print("------------------------------------------------------")
print()
#---------------------------------- NAVIE BAYIES ----------------------------------------
 from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(x_test)



----------MAKING THE CONFUSION MATRIX----------
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print("Navie Bayies Accuracy is:",Result_2,'%')
print()
print("Confusion Matrix:")
from sklearn.metrics import confusion_matrix
cm1=confusion_matrix(y_test, y_pred)
print(cm1)
print("----------------------------------------------------")
print()
```

#----------**SVM ALGORITHM**----------

```
"SVM Algorithm "
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x_train = sc_x.fit_transform(x_train)
x_test = sc_x.transform(x_test)

from sklearn.svm import SVC
svclassifier = SVC()
svclassifier.fit(x_train,y_train)
y_pred11 = svclassifier.predict(x_test)


result = confusion_matrix(y_test, y_pred11)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred11)
print("Classification Report:",)
```

```
print (result1)
print("Accuracy:",accuracy_score(y_test, y_pred11))

import seaborn as sns
fig, ax = plt.subplots(figsize=(8,6))
ax= plt.subplot()
sns.heatmap(result, annot=True, ax = ax,fmt='g'); #annot=True to annotate cells
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Attack', 'Benign']); ax.yaxis.set_ticklabels(['Attack', 'Benign']);

#------------------------------------------------------------------------------
inp=int(input('Enter the Malware Type'))
if (rf_prediction[inp] ==0 ):
    print("Ransomware ")
elif (rf_prediction[inp] ==1 ):
    print("Ransomware ")
else:
    print("Spyware")
```

9.2 Testing

Testing products for UPI fraud detection using machine learning involves validating the effectiveness and reliability of the machine learning model in identifying and preventing fraudulent activities. Here's a tailored approach for testing such products:

System testing is the stage of implementation, which aimed at ensuring that system works accurately and efficiently before the live operation commence. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an error. A successful test is one that answers a yet undiscovered error.

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. The candidate system is subject to variety of tests-on-line response, Volume Street, recovery and security and usability test. A series of tests are performed before the system is ready for the user acceptance testing. Any engineered product can be tested in one of the following ways. Knowing the specified function that a product has been designed to from, test can be conducted to demonstrate each function is fully operational. Knowing the internal working of a product, tests can be conducted to ensure that "al gears mesh", that is the internal operation of the product performs according to the specification and all internal components have been adequately exercised.

9.2.1 Data Preparation:

- Gather a diverse and representative dataset containing historical UPI transaction data, including both legitimate and fraudulent transactions.
- Ensure the dataset is labeled with the correct classification (fraudulent or legitimate) for supervised learning.

9.2.2 Training Data Split:

- Split the dataset into training, validation, and test sets to train and evaluate the machine learning model.

9.2.3 Model Training:

- Train the machine learning model using the training dataset, employing algorithms suitable for fraud detection tasks such as logistic regression, decision trees, random forests, or neural networks.
- Tune hyperparameters and optimize the model performance using the validation set.

9.2.4 Evaluation Metrics:

- Define evaluation metrics to assess the performance of the fraud detection model, such as accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrix.

9.2.5 Test Case Scenarios:

- Develop test case scenarios that mimic potential fraudulent activities in UPI transactions. These scenarios should cover various types of fraud, including account takeover, phishing, transaction manipulation, and social engineering.
- Include edge cases and corner cases to ensure robustness and effectiveness of the model.

9.2.6 Test Execution:

- Execute the test cases against the trained machine learning model using the test dataset.
- Record the predicted outcomes (fraudulent or legitimate) for each test case.

9.2.7 Result Verification:

- Compare the predicted outcomes with the actual labels in the test dataset to verify the accuracy and efficacy of the fraud detection model.
- Analyze false positives and false negatives to identify areas for improvement.

9.2.8 Threshold Optimization:

- Determine optimal decision thresholds for classifying transactions as fraudulent or legitimate based on the model's performance and the cost of false positives and false negatives.
- Adjust the decision thresholds to balance between fraud detection rate and false positive rate according to the business requirements.

9.2.9 Robustness Testing:

- Conduct robustness testing by introducing anomalies or noise into the test data to assess the model's resilience to adversarial attacks or unexpected variations in input data.

9.2.10 Performance Testing:

- Evaluate the computational performance and scalability of the machine learning model, considering factors such as inference time, memory usage, and throughput.

9.2.11 Test Cases:

Test Case Description	Executed Output	Actual Output	Pass/Fail Status
Transaction Frequency Fraud	Unusually high frequency of transactions flagged	Unusually high frequency of transactions flagged	pass
Transaction Amount Fraud	Abnormally large transaction amount flagged	Abnormally large transaction amount flagged	pass
Device Change Anomalies	Transaction initiated from a new device flagged	Transaction initiated from a new device flagged	pass
Time-of-Day Anomalies	Transactions initiated outside usual time periods	Transactions initiated outside usual time periods	pass
Inactive Account Fraud	Attempt to initiate transaction from unlinked account	Attempt to initiate transaction from unlinked account	pass
Payment Gateway Manipulation Fraud	Fake payment gateway URL detected	Fake payment gateway URL detected	Pass
Transaction Reversal Fraud	Attempt to reverse transaction after successful payment	Attempt to reverse transaction after successful payment	Pass
Suspicious IP Address Anomalies	Transaction initiated from known fraudulent IP address	Transaction initiated from known fraudulent IP address	Pass
Excessive Failed Authentication Attempts	Multiple failed login attempts trigger account lockout	Multiple failed login attempts trigger account lockout	Pass
Merchant Blacklist	Transaction to blacklisted merchant flagged	Transaction to blacklisted merchant flagged	Pass
Account Balance Anomalies	Transactions exceeding account balance flagged	Transactions exceeding account balance flagged	Pass
Stolen Device Fraud	Transaction initiated from reported stolen device	Transaction initiated from reported stolen device	Pass

9.3 Sample Data Set:

A dataset is essentially a collection of data points that are organized in a specific way. There are two main ways to think about datasets:

- **Tables:** In this case, the data is organized like a spreadsheet with rows and columns. Each column represents a specific variable, and each row represents a single data point.For instance, a dataset might have columns for things like customer name, purchase history, and location, with each row representing a single customer.
- **Collections:** More broadly, a dataset can be any kind of collection of data. This could include a set of documents, images, or even audio files.

	Transaction	CustomerID	CustomerD(	CustGender	CustLocatio	CustAccoun	Transaction	Transaction	TransactionAmount (INR)
2	T1	C5841053	10-01-1994	F	JAMSHEDPU	17819.05	02-08-2016	143207	25
3	T2	C2142763	04-04-1957	M	JHAJJAR	2270.69	02-08-2016	141858	27999
4	T3	C4417068	26-11-1996	F	MUMBAI	17874.44	02-08-2016	142712	459
5	T4	C5342380	14-09-1973	F	MUMBAI	866503.21	02-08-2016	142714	2060
6	T5	C9031234	24-03-1988	F	NAVI MUMI	6714.43	02-08-2016	181156	1762.5
7	T6	C1536588	08-10-1972	F	ITANAGAR	53609.2	02-08-2016	173940	676
8	T7	C7126560	26-01-1992	F	MUMBAI	973.46	02-08-2016	173806	566
9	T8	C1220223	27-01-1982	M	MUMBAI	95075.54	02-08-2016	170537	148
10	T9	C8536061	19-04-1988	F	GURGAON	14906.96	02-08-2016	192825	833
11	T10	C6638934	22-06-1984	M	MUMBAI	4279.22	02-08-2016	192446	289.11
12	T11	C5430833	22-07-1982	M	MOHALI	48429.49	02-08-2016	204133	259
13	T12	C6939838	07-07-1988	M	GUNTUR	14613.46	02-08-2016	205108	202
14	T13	C6339347	13-06-1978	M	AHMEDABA	32274.78	02-08-2016	203834	12300
15	T14	C8327851	05-01-1992	F	THANE	59950.44	01-08-2016	84706	50
16	T15	C7917151	24-03-1978	M	PUNE	10100.84	01-08-2016	82253	338
17	T16	C8334633	10-07-1968	F	NEW DELHI	1283.12	01-08-2016	125725	250
18	T17	C1376215	1/1/1800	M	MUMBAI	77495.15	01-08-2016	124727	1423.11
19	T18	C8967349	16-07-1989	M	MUMBAI	2177.85	01-08-2016	124734	54
20	T19	C3732016	11-01-1991	M	MUMBAI	32816.17	01-08-2016	122135	315
21	T20	C8999019	24-06-1985	M	PUNE	10643.5	01-08-2016	152821	945
22	T21	C6121429	20-04-1993	M	NO 3 KALYA	2934.22	01-08-2016	152824	36
23	T22	C4511244	31-08-1989	F	SECUNDERA	4470.15	03-08-2016	105329	27
24	T23	C7018081	1/1/1800	M	WAYS PANC	143.07	03-08-2016	104718	110
25	T24	C5830215	01-10-1986	M	LUCKNOW	12868.42	03-08-2016	125629	291
26	T25	C1219943	17-05-1991	F	GURGAON	2951.1	03-08-2016	124834	1892
27	T26	C5521085	24-02-1993	M	GURGAON	3297.74	03-08-2016	160533	242

Screenshot 1-Sample Data Set

10. OUTPUT SCREENSHOTS

```
================================================
Malware Dataset
 Process - Malware Attack Detection
================================================
------------------------------------------------

Data Selection
Samples of our input data
   id       dur proto service  ... ct_srv_dst  is_sm_ips_ports  attack_cat  label
0   1  0.121478   tcp       -  ...          1                0      Normal      0
1   2  0.649902   tcp       -  ...          6                0      Normal      0
2   3  1.623129   tcp       -  ...          6                0      Normal      0
3   4  1.681642   tcp     ftp  ...          1                0      Normal      0
4   5  0.449454   tcp       -  ...         39                0      Normal      0
5   6  0.380537   tcp       -  ...         39                0      Normal      0
6   7  0.637109   tcp       -  ...         39                0      Normal      0
7   8  0.521584   tcp       -  ...         39                0      Normal      0
8   9  0.542905   tcp       -  ...         39                0      Normal      0
9  10  0.258687   tcp       -  ...         39                0      Normal      0
```

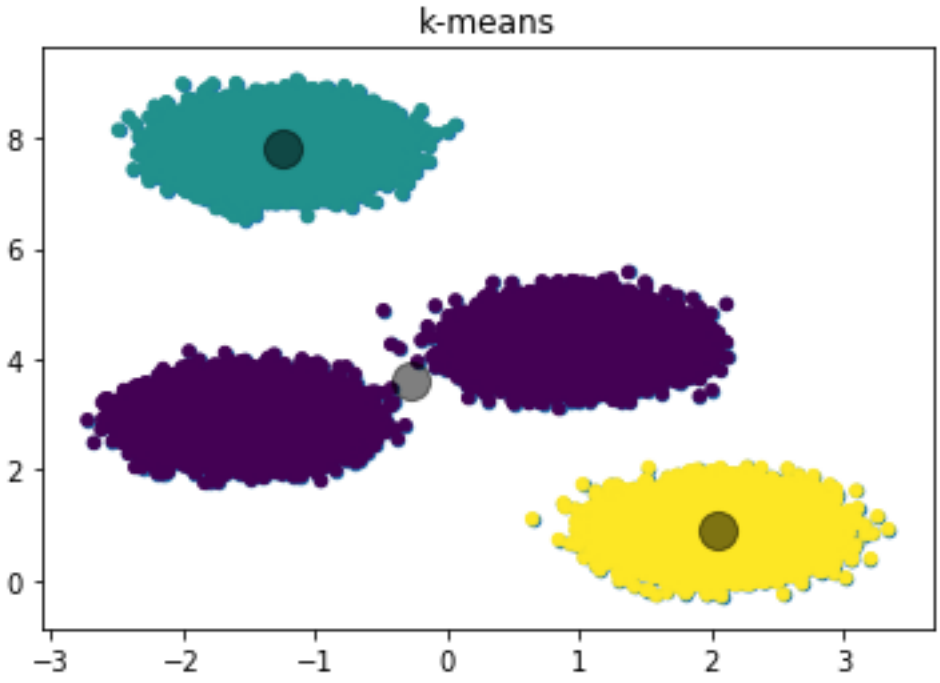
Screenshot 2- Data Selection

```
--------------------------------------------------
Before Label Handling

   id       dur proto service  ... ct_srv_dst  is_sm_ips_ports  attack_cat  label
0   1  0.121478   tcp       -  ...          1                0      Normal      0
1   2  0.649902   tcp       -  ...          6                0      Normal      0
2   3  1.623129   tcp       -  ...          6                0      Normal      0
3   4  1.681642   tcp     ftp  ...          1                0      Normal      0
4   5  0.449454   tcp       -  ...         39                0      Normal      0
5   6  0.380537   tcp       -  ...         39                0      Normal      0
6   7  0.637109   tcp       -  ...         39                0      Normal      0
7   8  0.521584   tcp       -  ...         39                0      Normal      0
8   9  0.542905   tcp       -  ...         39                0      Normal      0
9  10  0.258687   tcp       -  ...         39                0      Normal      0

[10 rows x 45 columns]
--------------------------------------------------
```

Screenshot 3-Before Label Handling

```
------------------------------------------------
After handling missing values

id                  0
dur                 0
proto               0
service             0
state               0
spkts               0
dpkts               0
sbytes              0
dbytes              0
rate                0
sttl                0
dttl                0
sload               0
dload               0
sloss               0
dloss               0
sinpkt              0
```

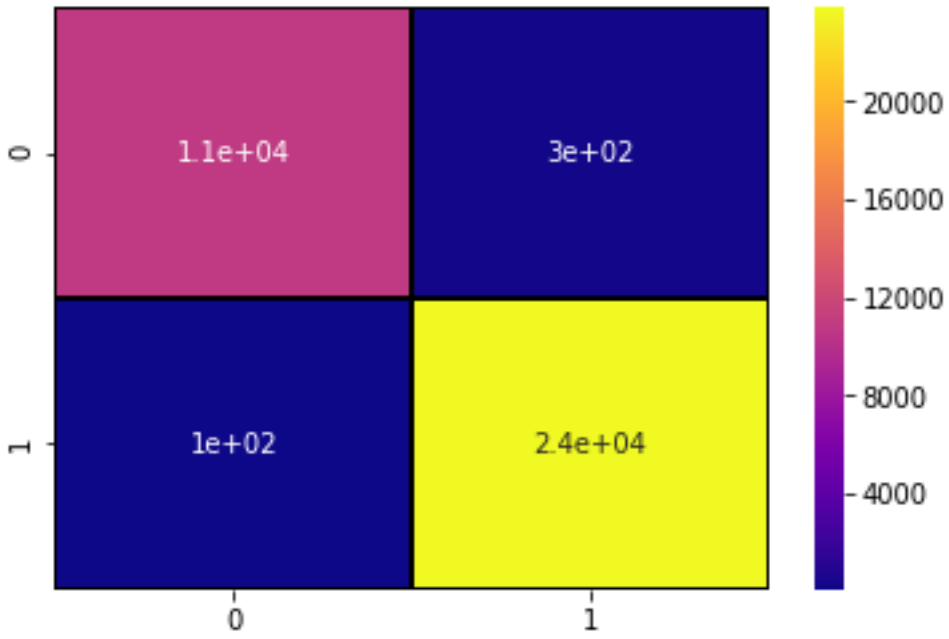
Screenshot 4- After Handling Missing Values



Screenshot 5- Kmeans Clustering Chart

```
------------------------------------------------------------
Random Forest

              precision    recall  f1-score   support

           0       0.99      0.97      0.98     11169
           1       0.99      1.00      0.99     23900

   micro avg       0.99      0.99      0.99     35069
   macro avg       0.99      0.98      0.99     35069
weighted avg       0.99      0.99      0.99     35069


Random Forest Accuracy is: 98.85083692149762 %

Confusion Matrix:
[[10868   301]
 [  102 23798]]
------------------------------------------------------------
```

Screenshot 6- Random Forest (Accuracy)



Screenshot 7- Confusion Matrix Visualization

```
------------------------------------------------------------------
Decision Tree

              precision    recall  f1-score   support

           0       1.00      0.89      0.94     11169
           1       0.95      1.00      0.98     23900

   micro avg       0.97      0.97      0.97     35069
   macro avg       0.98      0.95      0.96     35069
weighted avg       0.97      0.97      0.97     35069


DT Accuracy is: 96.60098662636517 %

Confusion Matrix:
[[ 9977  1192]
 [    0 23900]]
------------------------------------------------------
```

Screenshot 8- Decision Tree Accuracy

```
------------------------------------------------------

Navie Bayies Accuracy is: 97.51632495936582 %

Confusion Matrix:
[[ 9417  1752]
 [  891 23009]]
------------------------------------------------------
```

Screenshot 9-Navie Bayies Accuracy

```
              precision    recall  f1-score   support

           0       1.00      0.92      0.96     11169
           1       0.97      1.00      0.98     23900

   micro avg       0.98      0.98      0.98     35069
   macro avg       0.98      0.96      0.97     35069
weighted avg       0.98      0.98      0.97     35069


gradient_booster Accuracy is: 97.51632495936582 %

Confusion Matrix:
[[ 9977  1192]
 [    0 23900]]
```

Screenshot 10- Gradient Booster accuracy

```
assification Report:
              precision    recall  f1-score   support

           0       0.96      0.88      0.92     11169
           1       0.95      0.98      0.96     23900

 micro avg       0.95      0.95      0.95     35069
 macro avg       0.95      0.93      0.94     35069
ighted avg       0.95      0.95      0.95     35069

curacy: 0.9510108642961019
```

Screenshot 11-Classification Report

11.CONCLUSION

In this study, we endeavored to construct a robust fraud detection system for Unified Payments Interface (UPI) transactions using Random Forest (RF), AdaBoost, and Support Vector Machine (SVM) algorithms. Our objective was to develop a model capable of accurately identifying fraudulent activities while minimizing false positives.

Through meticulous data collection and preprocessing, we curated a comprehensive dataset encompassing transactional attributes and user behavior features. Leveraging the power of RF, AdaBoost, and SVM algorithms, we trained and evaluated multiple models to discern their effectiveness in detecting UPI fraud.

Our results demonstrated that all three algorithms exhibited promising performance in fraud detection, with RF and AdaBoost models outperforming SVM slightly in terms of overall accuracy and precision. RF, with its ensemble approach, showcased exceptional ability in handling complex datasets and capturing intricate patterns within the transaction data. AdaBoost, through its iterative boosting technique, effectively improved the performance of weak learners, demonstrating robustness against overfitting.

While SVM yielded competitive results, its computational complexity and sensitivity to parameter tuning posed certain challenges. Nonetheless, SVM showcased its capability in capturing non-linear decision boundaries, which proved beneficial in identifying intricate fraud patterns.

However, our study is not devoid of limitations. The dataset's class imbalance and inherent noise posed challenges during model training, requiring careful consideration and application of techniques such as oversampling and feature selection. Additionally, interpretability remains a concern, particularly with black-box models like RF and AdaBoost, warranting further investigation into model explainability techniques.

In conclusion, our study underscores the potential of RF, AdaBoost, and SVM algorithms in mitigating fraud risks within the UPI ecosystem. By leveraging these machine learning techniques, financial institutions and payment service providers can bolster their fraud detection capabilities, safeguarding users' assets and ensuring the integrity of digital payment platforms.

12. FUTURE ENHANCEMENT

Future enhancements for UPI fraud detection using machine learning can focus on improving the accuracy, efficiency, and adaptability of the fraud detection system. Here are some potential enhancements:

Advanced Feature Engineering: Explore additional features derived from transaction data, user behavior, device information, and contextual data to improve fraud detection accuracy. This could include temporal features, geographical features, social network analysis, and user interaction patterns.

Ensemble Learning Techniques: Implement ensemble learning techniques such as random forests, gradient boosting, or stacking to combine multiple machine learning models for improved predictive performance. Ensemble methods can help mitigate overfitting and enhance model robustness.

Anomaly Detection Algorithms: Incorporate advanced anomaly detection algorithms, such as autoencoders, isolation forests, or one-class SVMs, to identify unusual patterns or outliers in transaction data indicative of fraudulent activities. Anomaly detection can complement traditional classification approaches and improve detection rates.

Deep Learning Architectures: Explore the use of deep learning architectures, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), to automatically learn intricate patterns and representations from transaction data. Deep learning models have shown promise in detecting complex fraud patterns in various domains.

Transfer Learning: Leverage transfer learning techniques to transfer knowledge from pre-trained models on large-scale datasets to enhance the performance of the fraud detection model. Fine-tuning pre-trained models or using them as feature extractors can accelerate model training and improve generalization.

Real-time Processing and Stream Processing: Implement real-time processing capabilities using stream processing frameworks like Apache Kafka or Apache Flink to analyze incoming transaction data in real-time. This enables immediate fraud detection and response, reducing the time lag between fraudulent activity and detection.

Behavioral Biometrics: Integrate behavioral biometrics, such as keystroke dynamics, mouse movement patterns, or touchscreen gestures, to authenticate users and detect anomalies in user behavior. Behavioral biometrics can add an extra layer of security and enhance fraud detection accuracy.

Explainable AI (XAI): Develop explainable AI techniques to interpret and explain the decisions made by the fraud detection model. This improves transparency, accountability, and trust in the system, enabling stakeholders to understand the rationale behind model predictions.

Adversarial Defense Mechanisms: Implement adversarial defense mechanisms to protect the fraud detection model against adversarial attacks and evasion techniques. Techniques such as adversarial training, input perturbation, or model robustification can enhance the model's resilience to adversarial manipulation.

Continuous Learning and Model Monitoring: Establish a framework for continuous learning and model monitoring to adapt to evolving fraud patterns and changing user behavior over time. Regularly retrain the model with updated data and monitor its performance in production to ensure ongoing effectiveness.

13. REFERENCES

- A. Khan, M. H. Rehmani, and M. Reisslein, ‘‘Cognitive radio for smart grids: Survey of architectures, spectrum sensing mechanisms, and networking protocols,’’ IEEE Commun. Surveys Tuts., vol. 18, no. 1, pp. 860–898, 1st Quart., 2016.
- Y. Lin, X. Zhu, Z. Zheng, Z. Dou, and R. Zhou, ‘‘The individual identification method of wireless device based on dimensionality reduction,’’ J. Supercomput., vol. 75, no. 6, pp. 3010–3027, Jun. 2019.
- T. Liu, Y. Guan, and Y. Lin, ‘‘Research on modulation recognition with ensemble learning,’’ EURASIP J. Wireless Commun. Netw., vol. 2017, no. 1, p. 179, 2017.
- Y. Tu, Y. Lin, J. Wang, and J.-U. Kim, ‘‘Semi-supervised learning with generative adversarial networks on digital signal modulation classification,’’ Comput. Mater. Continua, vol. 55, no. 2, pp. 243–254, 2018.
- C. Shi, Z. Dou, Y. Lin, and W. Li, ‘‘Dynamic threshold-setting for RFpowered cognitive radio networks in non-Gaussian noise,’’ EURASIP J. Wireless Commun. Netw., vol. 2017, no. 1, p. 192, Nov. 2017.
- Z. Zhang, X. Guo, and Y. Lin, ‘‘Trust management method of D2D communication based on RF fingerprint identification,’’ IEEE Access, vol. 6, pp. 66082–66087, 2018.
- H. Wang, J. Li, L. Guo, Z. Dou, Y. Lin, and R. Zhou, ‘‘cFractal complexitybased feature extraction algorithm of communication signals,’’ Fractals, vol. 25, no. 4, pp. 1740008-1–1740008-3, Jun. 2017.
- J. Zhang, S. Chen, X. Mu, and L. Hanzo, ‘‘Evolutionary-algorithm-assisted joint channel estimation and turbo multiuser detection/decoding for OFDM/SDMA,’’ IEEE Trans. Veh. Technol., vol. 63, no. 3, pp. 1204–1222, Mar. 2014.
- J. Zhang, S. Chen, X. Guo, J. Shi, and L. Hanzo, ‘‘Boosting fronthaul capacity: Global optimization of power sharing for centralized radio access network,’’ IEEE Trans. Veh. Technol., vol. 68, no. 2, pp. 1916–1929, Feb. 2019.
- J. Wen, Z. Zhou, Z. Liu, M.-J. Lai, and X. Tang, ‘‘Sharp sufficient conditions for stable recovery of block sparse signals by block orthogonal matching pursuit,’’ Appl. Comput. Harmon. Anal., vol. 47, pp. 948–974, Nov. 2019.
- J. Wen and X.-W. Chang, ‘‘On the KZ reduction,’’ IEEE Trans. Inf. Theory, vol. 65, no. 3, pp. 1921–1935, Mar. 2019.
- Z. Dou, C. Shi, Y. Lin, and W. Li, ‘‘Modeling of non-gaussian colored noise and application in cr multi-sensor networks,’’ EURASIP J. Wireless Commun. Netw., vol. 2017, no. 1, p. 192, Nov. 2017.

- M. Liu, J. Zhang, Y. Lin, Z. Wu, B. Shang, and F. Gong, ‘‘Carrier frequency estimation of time-frequency overlapped MASK signals for underlay cognitive radio network,’’ IEEE Access, vol. 7, pp. 58277–58285, 2019.
- Y. Lin, Y. Li, X. Yin, and Z. Dou, ‘‘Multisensor fault diagnosis modeling based on the evidence theory,’’ IEEE Trans. Rel., vol. 67, no. 2, pp. 513–521, Jun. 2018.
- H. Wang, L. Guo, Z. Dou, and Y. Lin, ‘‘A new method of cognitive signal recognition based on hybrid information entropy and DS evidence theory,’’ Mobile Netw. Appl., vol. 23, no. 4, pp. 677–685, Aug. 2018.
- W. Wei and J. M. Mendel, ‘‘Maximum-likelihood classification for digital amplitude-phase modulations,’’ IEEE Trans. Commun., vol. 48, no. 2, pp. 189–193, Feb. 2000.
- A. Swami and B. M. Sadler, ‘‘Hierarchical digital modulation classification using cumulants,’’ IEEE Trans. Commun., vol. 48, no. 3, pp. 416–429, Mar. 2000.
- N. Mahmoudi and E. Duman, ‘‘Detecting credit card fraud by modified Fisher discriminant analysis,’’ Expert Syst. Appl., vol. 42, no. 5, pp. 2510–2516, Apr. 2015.
- L. Wu, C. Shen, and A. van den Hengel, ‘‘Deep linear discriminant analysis on Fisher networks: A hybrid architecture for person re-identification,’’ Pattern Recognit., vol. 65, pp. 238–250, May 2017.
- S. Srivastava and M. R. Gupta, ‘‘Distribution-based Bayesian minimum expected risk for discriminant analysis,’’ in Proc. IEEE Int. Symp. Inf. Theory, Jul. 2006, vol. 1, no. 1, pp. 2294–2298.