

# **ABSTRACT**

Credit card frauds are easy and friendly targets. E-commerce and many other online sites have increased the online payment modes, the classifier with a better rating score can be chosen to be one of the best methods to predict frauds. Thus, followed by a feedback mode, increasing the risk for online frauds. Increase in fraud rates, researchers started using different machine learning methods mechanisms to solve the problem of concept drift.

To detect and analyze frauds in online transactions. Due to rapid advancements in electronic commerce technology, the use of credit cards has dramatically increased. Since a credit card is the most popular mode of payment, the number of fraud cases associated with it is also rising. Here we implement different machine learning algorithms on an imbalanced dataset such as logistic regression, naïvebayes, random forest with ensemble classifiers using boosting techniques.

Here An review is done on the existing and proposed models for credit card fraud detection and has done a comparative study on these techniques. So Different classification models are applied to the data and the model performance is evaluated on the basis of quantitative measurements such as accuracy, precision, recall, f1 score, support, confusion matrix.

The conclusion of our study explains the best classifier by training and testing using supervised techniques that provides a better solution.

# TABLE OF CONTENTS

## Contents

<b>Chapter - 1: INTRODUCTION.....</b>	<b>9</b>
1.1 PROBLEM STATEMENT.....	10
1.2 EXISTING SYSTEM .....	10
1.3 PROPOSED SYSTEM .....	11
1.4 SOFTWARE REQUIREMENTS .....	12
1.5 HARDWARE REQUIREMENTS .....	12
1.6 MACHINE LEARNING.....	12
1.7 PYTHON TECHNOLOGY.....	18
THE JUPYTER NOTEBOOK INTRODUCTION .....	19
 <b>2. SYSTEM DESIGN .....</b>	 <b>23</b>
2.1 INTRODUCTION .....	24
2.3 UML DIAGRAMS .....	25
2.4 USE CASE DIAGRAM: .....	26
2.5 CLASS DIAGRAM:.....	27
2.6 SEQUENCE DIAGRAM: .....	28
2.7 DATA FLOW DIAGRAM:.....	29
 <b>CHAPTER - 3: DESIGN OF THE PROJECT.....</b>	 <b>30</b>
3.1 DATASET12.....	31
3.2 SAMPLING.....	31
3.3 DIVIDE THE DATASET.....	31
3.4 RANDOM FOREST.....	31
3.5 ALGORITHM STEPS FOR FINDING THE BEST ALGORITHM .....	32
3.6 TEST DATA.....	32
3.7 OUTCOME FOR TEST DATA .....	33
3.8 ACCURACY RESULTS.....	34
3.9 USECASE DIAGRAM .....	35
 <b>CHAPTER - 4: IMPLEMENTATIONS.....</b>	 <b>36</b>
4.1 LIBRARIES USED .....	37

4.2 CODE .....	38
<b>CHAPTER - 5: TESTING.....</b>	<b>41</b>
5.1 TESTING TECHNIQUES.....	41
Case1: UNIT TESTING	
Case2: INTEGRATION TESTING	
Case3:FUNCTIONAL TESTING	
Case4:SYSTEM TESTING	
<b>CHAPTER - 6: SYSTEM RESULTS.....</b>	<b>45</b>
6.1 PERFORMANCE.....	45
6.2 SCREENSHOTS .....	47
<b>CHAPTER - 7: CONCLUSION, LIMITATIONS AND FUTURE SCOPE.....</b>	<b>49</b>
7.1 CONCLUSION.....	50
7.2 LIMITATIONS.....	50
7.3 FUTURE SCOPE .....	51
<b>BIBLIOGRAPHY .....</b>	<b>52</b>
REFERENCES .....	52

# INTRODUCTION

Credit card refers to a card that is usually issued to a customer (cardholder) and usually allows a customer to purchase goods or services or withdraw cash in advance within their credit limit. Credit cards provide cardholders with a period of time, i.e., Customers can move on to the next billing cycle and pay their money later in a prescribed time. Credit card fraud is easy prey. Risk-free prices can be withdrawn in a short amount of time without the owner's knowledge.

In today's world, we are on the express train to a cashless society. In 2016 total non-cash transactions increased by 10.1% from 2015 for a total of 482.6 billion transactions. In 2017, there were 1,579 data breaches and about 179 million records, of which 133,015 were reported on credit card fraud. On the flip-side fraudulent transactions are on the rise as well. This must be stopped as soon as possible.

As it evolves, banks turn to EMV cards. It's a smart card that stores data in a compact circuit rather than a magnetic strip, making certain card payments more secure, but still leaving card-not-present frauds at higher rates. Criminals have shifted their focus on activities related to CNP transactions as the security of chip cards have increased. The below Figure shows the number of CNP fraud cases that were registered in respective years.

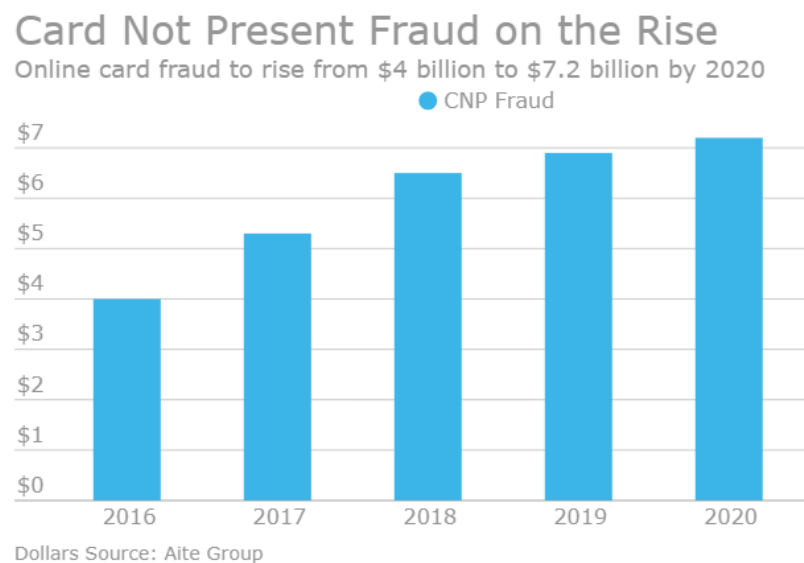


Fig 1.1: CNP Transactions

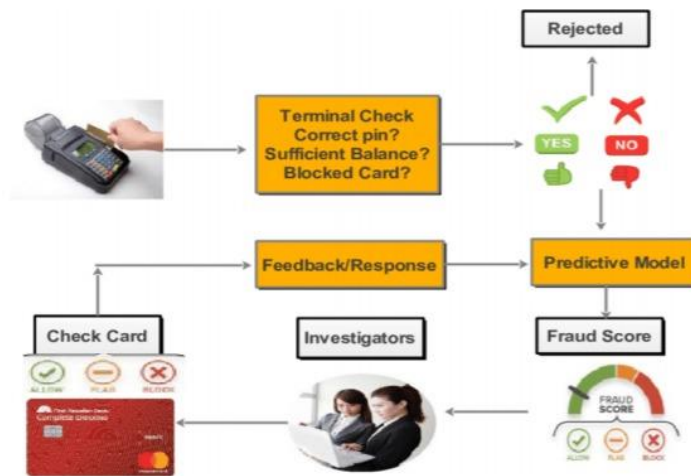


Fig 1.2: Card payment approval process

## 1.1 PROBLEM STATEMENT

Not all the doubtful transactions are considered fraudulent. It is commonly called as false positive (FP) which means that the case was not fraud although it was flagged as being potentially scam. This process of affirming each transaction that outliers from the cardholder's normal routine brings doubt about possible client disappointment. Additionally, the expenses related with exploring an enormous no. of false positives are high.

## 1.2 EXISTING SYSTEM

Since credit card fraud detection systems are a well-studied area, there are various algorithms and strategies for creating credit card fraud detection programs. One of the earliest CCFD systems is used. Card fraud detection systems include Critical Cost Tree (CSDT), Vector Assist Machine (SVM), Random Forest, and more. Follow the whale herd optimization algorithm to find available values. Use the BP network to correct the incorrect value. This credit card fraud scheme, which uses a whaling algorithm and a problem-solving approach, overcomes its shortcomings.

### 1.3 PROPOSED SYSTEM

Card transactions are still relatively low compared to past customer transactions. These variables lead to high data inequality. The main goal of our research is to pass on the concept of drift for use in real-world situations. This table shows the basic features that are captured when a transaction is executed.

Attribute name	Description
Transaction id	Identification number of a transaction
Cardholder id	Unique Identification number given to the cardholder
Amount	The amount transferred or credited in a particular transaction by the customer
Time	Details like time and date, to identify when the transaction was made
Label	To specify whether the transaction is genuine or fraudulent

Table 1.1: Raw features of credit card transactions

The database contains cardholder's actions for two days (Two days in the month of September). If there are 284,807 transactions, there are 492

fraudulent transactions. The database is highly unbalanced. Since providing a user's transaction history is considered private and translates most of the database's operations. Use PCA (Principal Component Analysis) as shown in the table. V1, V2, V3, ..., V28 are the active features of PCA, some are active PCA features.

SNo.	Feature	Description
1.	Time	Time in seconds to specify the elapses between the current transaction and the first transaction.
2.	Amount	Transaction amount
3.	Class	0 - not fraud 1 – fraud

Table 1.2: Attributes of dataset

#### 1.4 SOFTWARE REQUIREMENTS

Operating System: Windows 10, macOS, Linux.

Language: Python

#### 1.5 HARDWARE REQUIREMENTS

Hard disk: 32GB or more

Processor: Intel i3 or equivalent or more

## 1.6 Machine Learning

### What is Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. **Machine learning focuses on the development of computer programs** that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. **The primary aim is to allow the computers learn automatically** without human intervention or assistance and adjust actions accordingly.

But, using the classic algorithms of machine learning, text is considered as a sequence of keywords; instead, **an approach based on semantic analysis mimics the human ability to understand the meaning of a text.**

Hospitals, clinics and other healthcare organizations all around the world are working with software companies to develop administrative systems that are growingly digitized and automated. More importantly, scientists and researchers are using machine learning (ML) to churn out a number of smart solutions that can ultimately help in diagnosing and treating an illness. Patients are set to benefit the most as the technology can improve their outcome by analyzing the best forms of treatment for them. ML is capable of more accurately detecting a disease at an earlier stage, helping to reduce the number of readmissions in hospitals and clinics.

The technology has also come a long way in discovering and developing new drugs that have great potential in helping patients with complicated conditions. A cornerstone of ML is its ability to gather data and automate the output of smart solutions.



## **Applications of Machine Learning (ML) in the Healthcare Industry**

Here are four applications of ML in the healthcare industry:

### **1) Disease Identification**

One of the key components of a successful healthcare organization is its ability to identify a disease with speed and accuracy. With are hundreds of drugs currently on clinical trial, scientists and computationalists are entering the fray in high-need areas such as cancer identification and treatment. One such solution integrates cognitive computing with genomic tumor sequencing, while another uses ML to develop diagnostics and therapeutic treatments in multiple areas such as oncology. Another example is DeepMind Health, which is developing technology that can address macular degeneration in aging eyes.

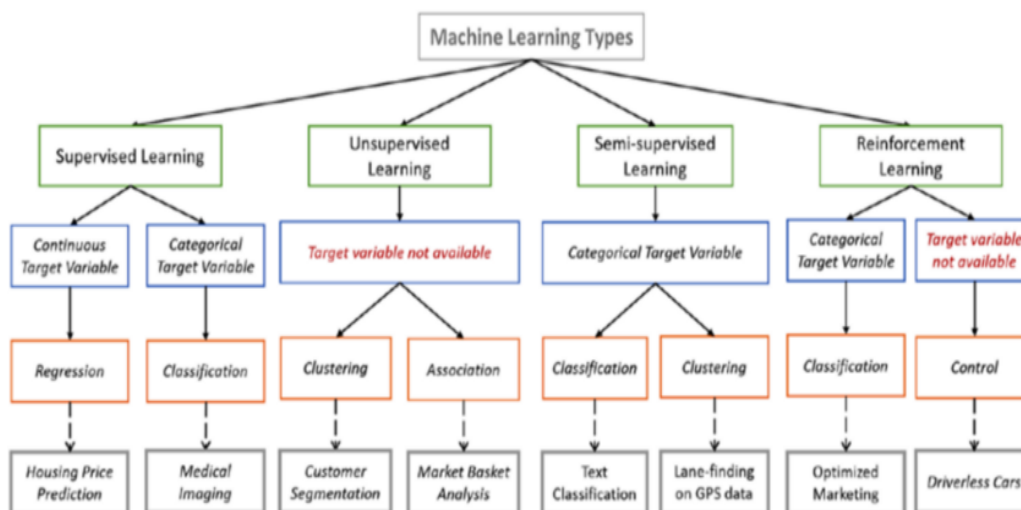
### **2) Diagnosis in Medical Imaging**

Another important element of diagnosing an illness is medical imaging and its ability to show a more complete image of an illness. Deep learning is playing a key role in this regard as it is becoming more accessible thanks to richer data sources that can be used in the diagnostic process. The technology has some limits as it is incapable of explaining how it arrived at its predictions, although these ML applications are correct a lot of the time. Nevertheless, the technology, combined with healthcare professionals, can offer treatment solutions quicker with these advanced diagnosis tools by interpreting a result and deciding whether the machine's treatment suggestions are correct or not.

## Types of Machine Learning Algorithms

Some Common Terms in ML:

- **Labeled data:** Data consisting of a set of *training examples*, where each example is a *pair* consisting of an input and a desired output value (also called the *supervisory signal*, *labels*, etc)
- **Classification:** The goal is to predict discrete values, e.g. {1,0}, {True, False}, {spam, not spam}.
- **Regression:** The goal is to predict continuous values, e.g. home prices.



There are some variations of how to define the types of Machine Learning Algorithms but commonly they can be divided into categories according to their purpose and the main categories are the following:

- a) Supervised algorithm
- b) Unsupervised algorithm
- c) Semi-supervised algorithm
- d) Reinforcement algorithm

## Supervised Learning

- I like to think of supervised learning with the concept of function approximation, where basically we train an algorithm and in the end of the process we pick the function that best describes the input data, the one that for a given  $X$  makes the best estimation of  $y$  ( $X \rightarrow y$ ). Most of the time we are not able to figure out the true function that always make the correct predictions and other reason is that the algorithm rely upon an assumption made by humans about how the computer should learn and this assumptions introduce a bias, Bias is topic I'll explain in another post.
- Here the human experts acts as the teacher where we feed the computer with training data containing the input/predictors and we show it the correct answers (output) and from the data the computer should be able to learn the patterns.
- Supervised learning algorithms try to *model relationships and dependencies between the target prediction output and the input features* such that we can predict the output values for new data based on those relationships which it learned from the previous data sets.

## List of Common Algorithms

- Nearest Neighbor
- Naive Bayes
- Decision Trees
- Linear Regression
- Support Vector Machines (SVM)
- Neural Networks

## 1. Unsupervised Learning

- The computer is trained with unlabeled data.
- These algorithms are particularly useful in cases where the human expert doesn't know what to look for in the data.
- are the family of machine learning algorithms which are mainly used in *pattern detection* and *descriptive modeling*. However, *there are no output categories or labels* here based on which the algorithm can try to model relationships. These algorithms try to use techniques on the input data to *mine for rules, detect patterns, and summarize and group the data points* which help in deriving meaningful insights and describe the data better to the users.

### List of Common Algorithms

- k-means clustering, Association Rules

## Semi-supervised Learning

In the previous two types, either there are no labels for all the observation in the dataset or labels are present for all the observations. Semi-supervised learning falls in between these two. In many practical situations, the cost to label is quite high, since it requires skilled human experts to do that. So, in the absence of labels in the majority of the observations but present in few, semi-supervised algorithms are the best candidates for the model building. These methods exploit the idea that even though the group memberships of the unlabeled data are unknown, this data carries important information about the group parameters.

## Reinforcement Learning

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised Learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only

- **List of Common Algorithms**

- Temporal Difference (TD)
- Deep Adversarial Networks

## 1.7 PYTHON TECHNOLOGY

1. Syntax and semantics
2. Main article Python syntax and Semantics
3. Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal
4. Indentation
5. Main article: Python Syntax , Indentation and Semantics
6. Python uses whitespaces indentation, rather than curly braces or keywords, to delimit blocks .An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. This feature is also sometimes termed the off-side rule
7. Statements and control flow
8. Python's statements include (among others):
9. The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, eg: `x = 2`, translates to "typed variable name `x` receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage allocation for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type . In the simplest case of Python assignment, using the same example, `x = 2`, translates to "(generic) name `x` receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., `x = 2; y = 2;` most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data Type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing .

10. The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
11. The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
12. The [while](#) statement, which executes a block of code as long as its condition is true.
13. The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
14. The class statement, which executes a block of code and attaches its local namespace to [class](#), for use in [object-oriented programming](#)
15. The def statement, which defines a [function](#) or [method](#) .
16. The with statement (from Python 2.5), which encloses a code block within a context manager (for example, acquiring a [lock](#) before the block of code is run and releasing the lock afterwards, or opening a [file](#) and then closing it), allowing [Resource Acquisition Is Initialization](#) (RAII)-like behavior.
17. The pass statement, which serves as a [NOP](#). It is syntactically needed to create an empty code block.
18. The [assert](#) statement, used during debugging to check for conditions that ought to apply.
19. The yield statement, which returns a value from a [generator](#) function. From Python 2.5, yield is also an operator. This form is used to implement [coroutines](#).
20. The import statement, which is used to import modules whose functions or variables can be used in the current program. There are two ways of using import . from <module name> import \* or import <module name>.
21. The print statement was changed to the print() function in Python 3.<sup>[56]</sup>
22. Python does not support [tail call](#) optimization or [first-class continuations](#), and, according to Guido van Rossum, it never will. However, better support for [coroutine](#) -like functionality is provided in 2.5, by extending Python's [generators](#). Before 2.5, generators were [lazy iterators](#); information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.
23. Expressions
24. Some Python [expressions](#) are similar to languages such as [C](#) and [Java](#) , while some are not:
25. Addition, subtraction, and multiplication are the same, but the behavior of division differs.

26. Python also added the `**` operator for exponentiation.
27. From Python 3.5, it enables support of [matrix multiplication](#) with the `@` operator
28. In Python, `==` compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the `equals()` method.) Python's `is` operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example `a <= b <= c`.
29. Python uses the words `and`, `or`, `not` for its boolean operators rather than the symbolic `&&`, `||`, `!` used in Java and C.
30. Python has a type of expression termed a [list comprehension](#). Python 2.4 extended list comprehensions into a more general expression termed a [generator](#) expression
31. [Anonymous functions](#) are implemented using [lambda expressions](#) ; however, these are limited in that the body can only be one expression.
32. Conditional expressions in Python are written as `x if c else y` (different in order of operands from the [c ? x : y](#) operator common to many other languages).
33. Python makes a distinction between [lists](#) and [tuples](#). Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be [immutable](#) in Python). Tuples are written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The `+` operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable `t` initially equal to `(1, 2, 3)`, executing `t = t + (4, 5)` first evaluates `t + (4, 5)`, which yields `(1, 2, 3, 4, 5)`, which is then assigned back to `t`, thereby effectively "modifying the contents" of `t`, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.
34. Python features sequence unpacking where multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left
35. Python has a "string format" operator `%`. This functions analogous to [printf](#) format strings in [C](#), e.g. `"spam=%s eggs=%d" % ("blah", 2)` evaluates to `"spam=blah eggs=2"`. In Python 3 and



2.6+, this was supplemented by the `format()` method of the `str` class, e.g. `"spam={0} eggs={1}".format("blah", 2)`, Python 3.6 added "f-strings": `f'spam={"blah"} eggs={2}'`.

36. Python has various kinds of [string literals](#) :

37. Strings delimited by single or double quote marks. Unlike in [Unix shells](#), [Perl](#) and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (`\`) as an [escape character](#) . [String interpolation](#) became available in Python 3.6 as "formatted string literals". <sup>[67]</sup>

38. Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like [here documents](#) in shells, Perl and [Ruby](#).

39. [Raw string](#) varieties, denoted by prefixing the string literal with an `r`. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as [regular expressions](#) and [Windows](#) -style paths. Compare "@-quoting" in [C#](#) .

40. Python has [array index](#) and [array slicing](#) expressions on lists, denoted as `a[key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are [zero-based](#), and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for

example `a[:]` returns a copy of the entire list. Each element of a slice is a [shallow copy](#) .

41. In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as [Common Lisp](#), [Scheme](#), or [Ruby](#). This leads to duplicating some functionality.

42. For example:

43. [List comprehensions](#) vs. for-loops

44. [Conditional](#) expressions vs. if blocks

45. The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements.

46. Statements cannot be a part of an expression, so list and other comprehensions or [lambda expressions](#), all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is syntactically valid (but probably unintended) C code but `if c = 1: ...` causes a syntax error in Python.

47. Methods are useful

48. [Methods](#) on objects are [functions](#) attached to the object's class; the syntax

instance.method(argument) is, for normal methods and functions, [syntactic sugar](#) for Class.method(instance, argument). Python methods have an explicit [self](#) parameter to access [instance data](#), in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., [C++](#) , [Java](#) , [Objective-C](#), or [Ruby](#) ).

#### 49. Typing

50. Python uses [duck typing](#) and has typed objects but untyped variable names. Type constraints are not checked at [compile time](#) ; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being [dynamically typed](#), Python is [strongly typed](#), forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

51. Python allows programmers to define their own types using [classes](#), which are most often used for [object-oriented programming](#) . New [instances](#) of classes are constructed by calling the class (for example, SpamClass() or EggsClass()),

and the classes are instances of the [metaclass](#) type (itself an instance of itself), allowing [metaprogramming](#) and [reflection](#) .

52. Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

53. The long term plan is to support [gradual typing](#) and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython.

## The Jupyter Notebook

### Introduction

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

**A web application:** a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.

**Notebook documents:** a representation of all content visible in the web application.

## **Chapter - 2: SYSTEM DESIGN**

### **2.1 INTRODUCTION**

UML stands for Unified Modeling Language. UML is a standardized general- purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

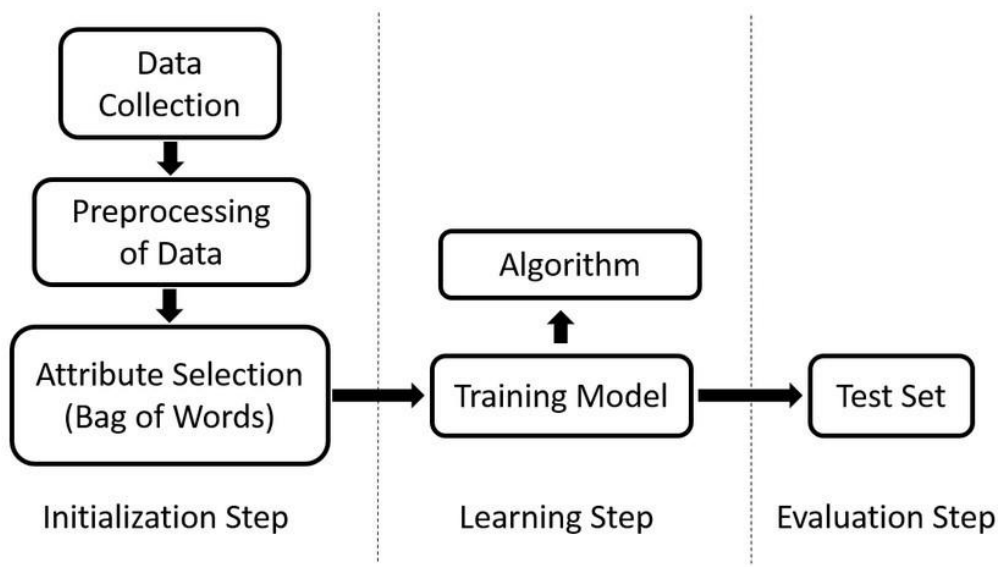
### **GOALS:**

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.

## **2.2 DATA FLOW DIAGRAM:**

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.
5. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
6. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
7. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
8. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.



## 2.3 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general- purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

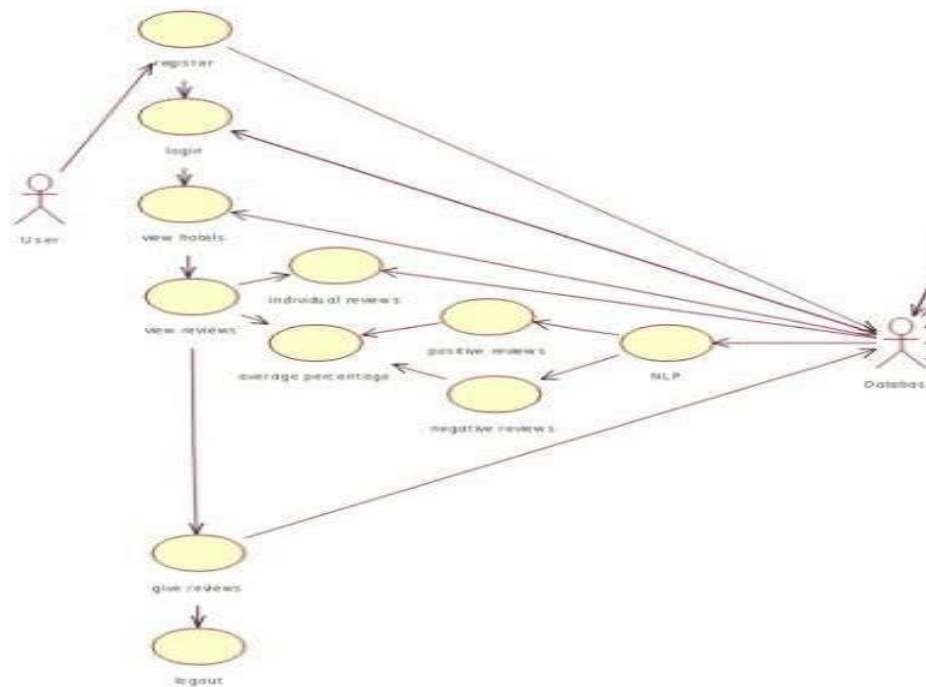
The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

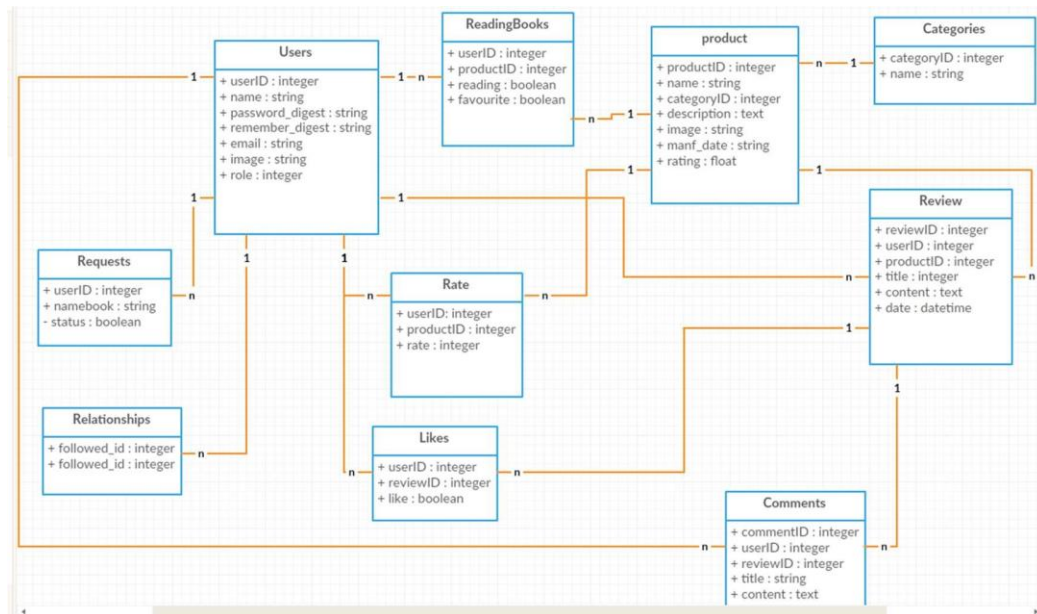
## 2.4 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted



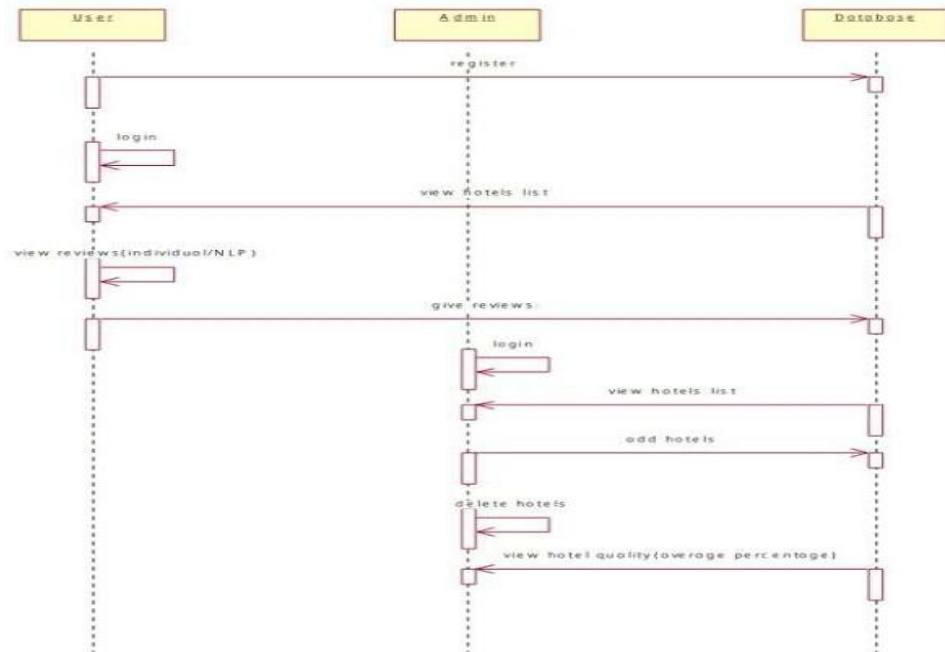
## 2.5 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

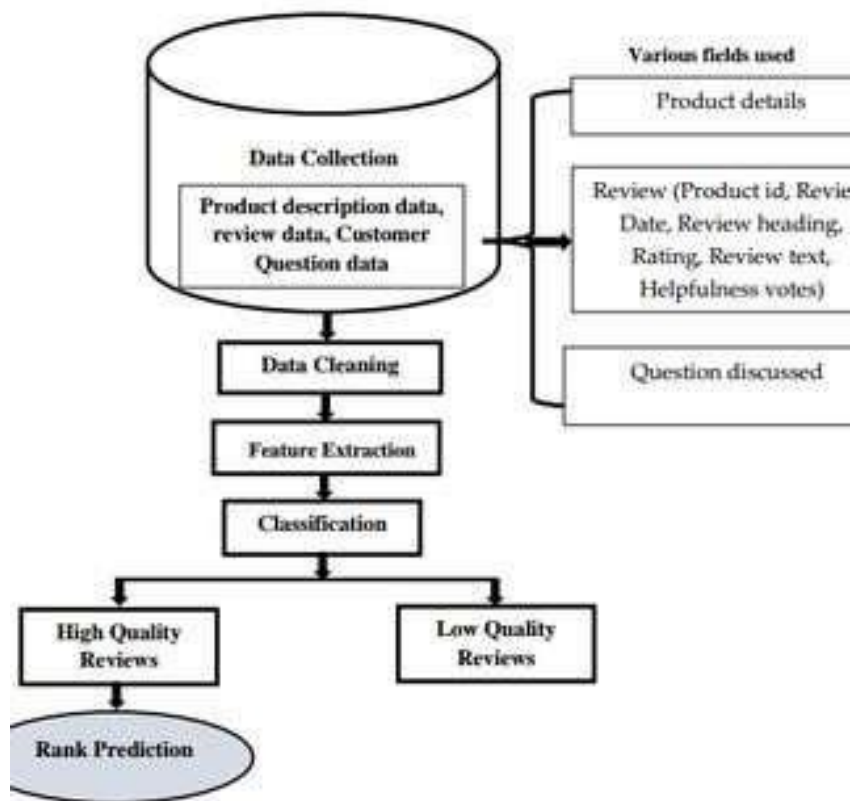


## 2.6 SEQUENCE DIAGRAM:

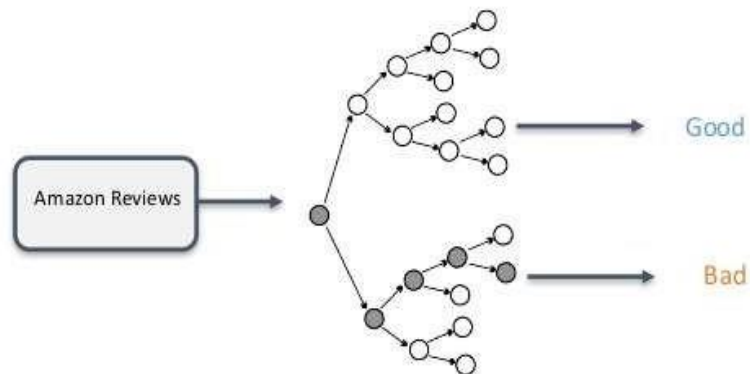
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagram are sometimes called event diagrams, event scenarios, and timing diagrams.







## Machine Learning Model



## CHAPTER - 3: DESIGN OF THE PROJECT

The figure below is the System Architecture of the project:

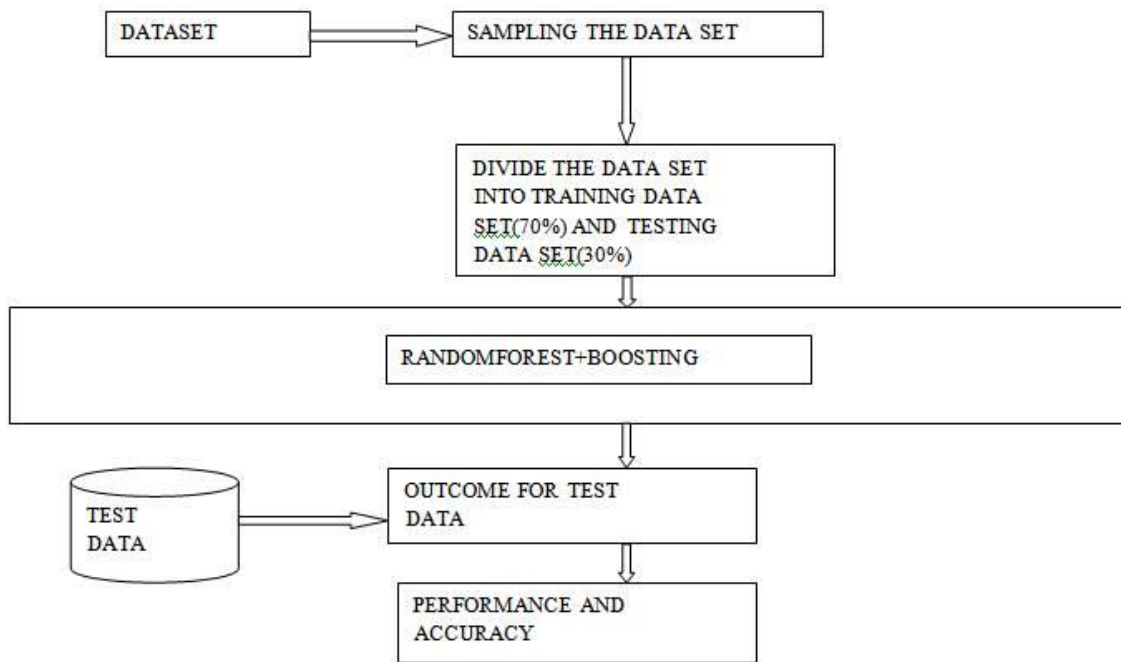


Fig 2.1: System Architecture

### 3.1 DATASET

In this article, we used the credit card fraud detection dataset you can download from Kaggle. This includes transactions made by European cardholders in September 2013. The database contains 31 numeric features. Since some of the input variables contain financial information, the PCA transformation of these input variables was performed in order to keep these data anonymous. Three of the given features weren't transformed. Feature "Time" shows the time between the first transaction and every other transaction in the dataset. Feature "Amount" is the amount of the transactions made by credit card. Feature "Class" represents the label and takes only 2 values: value 1 in case of fraud transaction and 0 otherwise.

### 3.2 SAMPLING

Further, the data set is minimized.

### 3.3 DIVIDE THE DATASET

The dataset is divided into a trained data set and test data set. 70% of the data set is under training and the remaining 30% is under testing. Here we are using the Random Forest algorithm.

### 3.4 RANDOM FOREST

First, start with the selection of random samples from a given dataset. Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree. Then voting will be performed for every predicted result. So finally, select the most voted prediction result as the final prediction result.

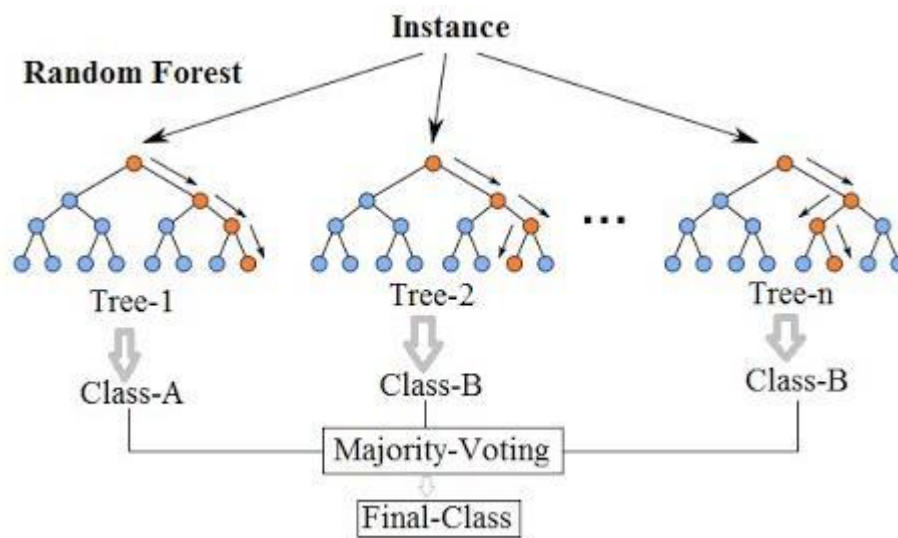


Fig 2.2: Random Forest Classification

## Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

## Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

## How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

**Step-1:** Select random K data points from the training set.

**Step-2:** Build the decision trees associated with the selected data points (Subsets).

**Step-3:** Choose the number N for decision trees that you want to build.

**Step-4:** Repeat Step 1 & 2.

**Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

## **Applications of Random Forest**

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

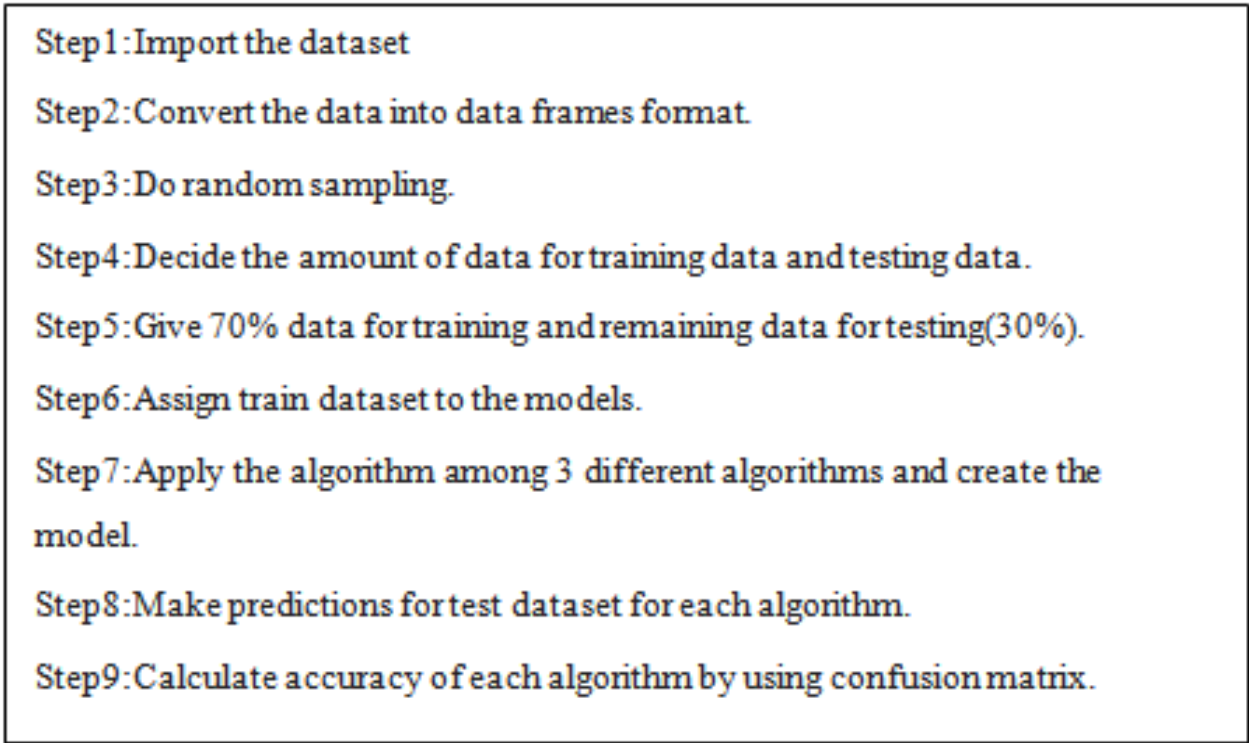
## **Advantages of Random Forest**

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

## **Disadvantages of Random Forest**

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

### 3.5 ALGORITHM STEPS FOR FINDING THE BEST ALGORITHM



Step 1: Import the dataset

Step 2: Convert the data into data frames format.

Step 3: Do random sampling.

Step 4: Decide the amount of data for training data and testing data.

Step 5: Give 70% data for training and remaining data for testing (30%).

Step 6: Assign train dataset to the models.

Step 7: Apply the algorithm among 3 different algorithms and create the model.

Step 8: Make predictions for test dataset for each algorithm.

Step 9: Calculate accuracy of each algorithm by using confusion matrix.

Fig 2.3: Steps

### 3.6 TEST DATA

After training is done on the dataset then the testing process takes place.

### 3.7 OUTCOME FOR TEST DATA

We will get the respective results for each algorithm and performance is displayed in graphs.

### 3.8 ACCURACY RESULTS

The final results are shown with accuracy and the best is identified.

### 3.9 USECASE DIAGRAM

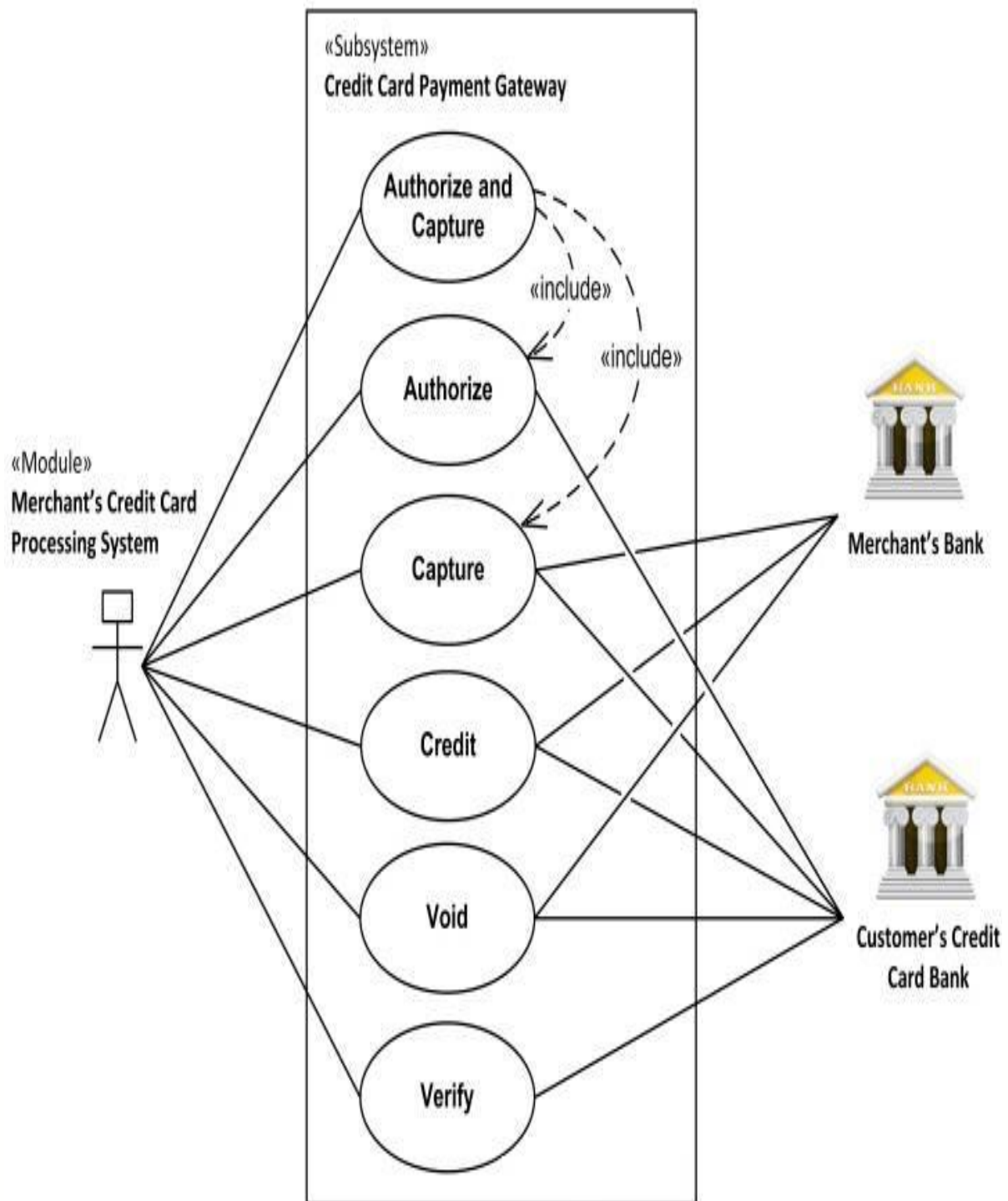


Fig 2.4: Usecase Diagram

## CHAPTER - 4: IMPLEMENTATION

### 4.1 LIBRARIES USED

Numpy:

Numpy is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform and matrices.

Pandas:

Pandas is a software library written for python programming language for data manipulation and analysis. In particular it offers data structures and operations for manipulating numerical tables and time series.

Matplotlib:

Matplotlib is a plotting library for python programming language and its numerical mathematics extension numpy. It provides an object oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter

Seaborn:

Seaborn is a library that uses matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Gridspec:

Gridspec contains classes that help to layout multiple axes in a grid-like pattern within a figure. It specifies the overall grid structure.



## 4.2 CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec

from google.colab import drive
drive.mount('/content/drive/')

data = pd.read_csv('/content/drive/My Drive/PROJECT/creditcard.csv')
data.shape

data.head()

data.columns

data.drop_duplicates(inplace=True)
data.shape

data.isnull().sum()

print(data.describe())

fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
```

```
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

```
data[data['Class']==1]
```

```
print('Amount details of the fraudulent transaction')
```

```
fraud.Amount.describe()
```

```
print('details of valid transaction')
```

```
valid.Amount.describe()
```

```
corrmat = data.corr()
```

```
fig = plt.figure(figsize = (15,15))
```

```
sns.heatmap(data.corr(),cmap = 'RdYlGn',annot= False,center =0)
```

```
plt.show()
```

```
X = data.drop(['Class'], axis = 1)
```

```
Y = data["Class"]
```

```
print(X.shape)
```

```
print(Y.shape)
```

```
# getting just the values for the sake of processing
```

```
# (its a numpy array with no columns)
```

```
xData = X.values
```

```
yData = Y.values
```

```
from sklearn.model_selection import train_test_split
```

```
# Split the data into training and testing sets
```

```
xTrain, xTest, yTrain, yTest = train_test_split(  
    xData, yData, test_size = 0.3, random_state = 42)
```

```
xTrain.shape
```

xTest.shape

```
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)
# predictions
yPred = rfc.predict(xTest)

# scoring in anything
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix

n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")

acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))

prec = precision_score(yTest, yPred)
print("The precision is {}".format(prec))

rec = recall_score(yTest, yPred)
print("The recall is {}".format(rec))

f1 = f1_score(yTest, yPred)
print("The F1-Score is {}".format(f1))
```

```
MCC = matthews_corrcoef(yTest, yPred)
print("The Matthews correlation coefficient is{}".format(MCC))
```

```
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(yTest, yPred)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt = "d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

## **CHAPTER - 5: TESTING**

### **5.1 TYPES OF TESTS**

#### **Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

#### **Integration testing**

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is

specifically aimed at exposing the problems that arise from the combination of components.

### **Functional test**

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.

### **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.

## 5.2 TESTING STRATEGIES

### White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its 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. 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 In the test cases below 0 is not a fraudulent transaction and 1 is a fraudulent transaction and it also describes the information in the dataset.

```
[18] data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.631
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.175

Fig 4.1.1: Test Cases

V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
1.468177	-0.470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.635558	0.463917	-0.114805	-0.183361	-0.145783	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
0.175121	-0.451449	-0.237033	-0.038195	0.803487	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Fig 4.1.2: Test Cases

```
[22] print(data.describe())
```

	Time	V1	...	Amount	Class
count	283726.000000	283726.000000	...	283726.000000	283726.000000
mean	94811.077600	0.005917	...	88.472687	0.001667
std	47481.047891	1.948026	...	250.399437	0.040796
min	0.000000	-56.407510	...	0.000000	0.000000
25%	54204.750000	-0.915951	...	5.600000	0.000000
50%	84692.500000	0.020384	...	22.000000	0.000000
75%	139298.000000	1.316068	...	77.510000	0.000000
max	172792.000000	2.454930	...	25691.160000	1.000000

```
[8 rows x 31 columns]
```

Fig 4.2: Description of the Data



## **CHAPTER – 6: SYSTEM RESULTS**

### **6.1 PERFORMANCE.**

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

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

#### Features to be tested

- 6.1.4 Verify that the entries are of the correct format
- 6.1.5 No duplicate entries should be allowed
- 6.1.6 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.

## 6.2 SCREENSHOTS

The following results were observed with random forest technique were evaluated against the data:

```
[25] data.columns

Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
      'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
      'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
      'Class'],
      dtype='object')

[43] data.drop_duplicates(inplace= True)
data.shape

(283726, 31)

[26] data.isnull().sum()


Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Fig 5.1: Details of the dataset

```
[28] fraud = data[data['Class'] == 1]
      valid = data[data['Class'] == 0]
      outlierFraction = len(fraud)/float(len(valid))
      print(outlierFraction)
      print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
      print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))

0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
```

Fig 5.2: Number of fraud and valid transactions



```
# scoring in anything
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix

n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")


acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))

prec = precision_score(yTest, yPred)
print("The precision is {}".format(prec))

rec = recall_score(yTest, yPred)
print("The recall is {}".format(rec))

f1 = f1_score(yTest, yPred)
print("The F1-Score is {}".format(f1))

MCC = matthews_corrcoef(yTest, yPred)
print("The Matthews correlation coefficient is{}".format(MCC))
```



The model used is Random Forest classifier  
The accuracy is 0.9995669627705019  
The precision is 0.9159663865546218  
The recall is 0.8014705882352942  
The F1-Score is 0.8549019607843137  
The Matthews correlation coefficient is0.856597434470664

Fig 5.3: Metrics for Random Forest Classifier model

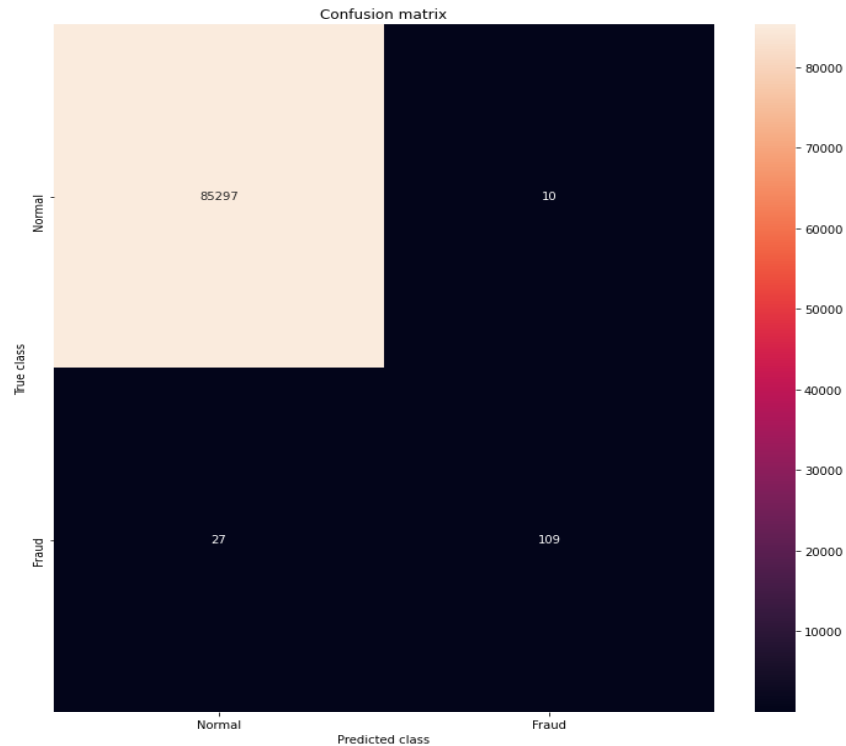


Fig 5.4: Confusion matrix

## CHAPTER - 7: CONCLUSION, LIMITATIONS AND FUTURE SCOPE

### 7.1 CONCLUSION

In this, we studied applications of machine learning like Random Forest Classifier and it shows that it proves accurate in deducting fraudulent transactions and minimizing the number of false alerts. If these algorithms are applied to a bank credit card fraud detection system, the probability of fraud transactions can be predicted soon after credit card transactions. And a series of anti-fraud strategies can be adopted to prevent banks from great losses and reduce risks. The objective of the study was taken differently than the typical classification problems in that we had a variable misclassification cost. Precision, recall, f1-score, support, and accuracy are used to evaluate the performance of the proposed system. Comparing Random Forest Classifier with Naive Bayes and Logistic Regression, it is more accurate and better.

## **7.2 LIMITATIONS**

Although the discovery of credit card fraud has received much attention, there are still some issues that investigators face and that are not adequately addressed. Hopefully, this information focuses on future research methods to provide an effective fraud investigation system. These issues are as follow:

i. Lack of adaptive credit card fraud detection systems:

Although lots of research has been investigated in the credit card fraud detection field, there are none or limited adaptive techniques which can learn data streams of transactions as they are conducted.

ii. Nonexistence of standard algorithm:

There is not any powerful algorithm known in credit card fraud literature that outperforms all others. Each Technique has its own advantages and disadvantages.

iii. Nonexistence of standard and comprehensive credit card dataset:

Credit card is inherently private property, so creating a proper benchmark for this purpose is very difficult. Incomplete datasets can cause fraud detection systems to learn fraud tricks or normal behavior partially.

iv. Nonexistence of suitable metrics:

The criteria for good measurement to evaluate the results of a fraudulent detection system have not yet been defined. Without these indicators, researchers and experts would not be able to compare different methods and prefer the most effective fraud detection system.

## **7.3 FUTURE SCOPE**

From the above analysis of the credit card fraud detection technique, it is clear that Random Forest Classifier technique performs best. But the drawback of this paper by using this algorithm is we cannot determine the names of fraud and unfraud transactions for the given dataset using machine learning. For the further development of the project, we can work to solve this problem by using various methods

## BIBLIOGRAPHY

We can find similar project details on the net through GitHub , Linkedin etc.

## REFERENCES

- [1] Machine Learning With Random Forests And Decision Trees: A Visual Guide For Beginners (Author Scott HartsHorn)
- [2] Andhavarapu Bhanusri "Credit card fraud detection using Machine learning algorithms" Quest Journals Journal of Research in gHumanities and Social Science, vol. 08(02), 2020.
- [3] <https://www.kaggle.com/mlg-ulb/creditcardfraud>
- [4] <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- [5] Perdisci, R., Lee, W., & Feamster, N. (2010, April). Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In NSDI (Vol. 10, p. 14).
- [6] Morales, J. A., Al-Bataineh, A., Xu, S., & Sandhu, R. (2010, September). Analyzing and exploiting network behaviors of malware. In International conference on security and privacy in communication systems (pp. 20-34). Springer, Berlin, Heidelberg.
- [7] Dadhia, R., & Bahl, P. (2012). U.S. Patent No. 8,256,003. Washington, DC: U.S. Patent and Trademark Office.
- [8] Saxe, J., & Berlin, K. (2015, October). Deep neural network based malware detection using two dimensional binary program features. In 2015 10th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 11-20). IEEE.
- [9] Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., & Elovici, Y. (2014). Mobile malware detection through analysis of deviations in application network behavior. Computers & Security, 43, 1-18.
- [10] Nari, S., & Ghorbani, A. A. (2013, January). Automated malware classification based on network behavior. In 2013 International Conference on Computing, Networking and Communications (ICNC) (pp. 642-647). IEEE.
- [11] Paolo Passeri, 'February 2019 Cyber Attacks Statistics', 2019 Published. [Online]. Available: <https://www.hackmageddon.com/2019/04/03/february-2019-cyber-attacks-statistics/>. [Accessed: 1- May-2019]

- [12] Kalafut, A., Acharya, A., & Gupta, M. (2006, October). A study of malware in peer-to-peer networks. In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (pp. 327-332). ACM.
- [13] Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., & Elovici, Y. (2014). Mobile malware detection through analysis of deviations in application network behavior. *Computers & Security*, 43, 1-18.
- [14] Morales, J. A., Al-Bataineh, A., Xu, S., & Sandhu, R. (2010, September). Analyzing and exploiting network behaviors of malware. In International conference on security and privacy in communication systems (pp. 20-34). Springer, Berlin, Heidelberg.
- [15] Bekerman, D., Shapira, B., Rokach, L., & Bar, A. (2015, September). Unknown malware detection using network traffic classification. In 2015 IEEE Conference on Communications and Network Security (CNS) (pp. 134-142). IEEE.
- [16] Saeed, I. A., Selamat, A., & Abuagoub, A. M. (2013). A survey on malware and malware detection systems. *International Journal of Computer Applications*, 67(16).