A Project Report

on

Detecting Fraudulent Vehicle Insurance Claims

Submitted in partial fulfilment of the requirements for the award of degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

RAMA KRISHNA PARJANYA VARANASI (160118733038)

VENKATA SAI TEJA GADDAM (160118733055)

Department of Computer Science and Engineering,
Chaitanya Bharathi Institute of Technology (Autonomous),
(Affiliated to Osmania University, Hyderabad)
Hyderabad, TELANGANA (INDIA) – 500075
[2021-2022]

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)

Kokapet (Village), Gandipet, Hyderabad, Telangana-500075. www.cbit.ac.in

ISO Certified 9001:2015

COMMITTED TO RESEARCH, INNOVATION AND EDUCATION 43 years

CERTIFICATE

This is to certify that the project titled "**Detecting Fraudulent Vehicle Insurance Claims**" is the bonafide work carried out by **Rama Krishna Parjanya Varanasi (160118733038) and Venkata Sai Teja Gaddam (160118733055)**, students of B.E.(CSE) of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2021-2022, submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering** (**Computer Science and Engineering**) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Supervisor
Sri R. Srikanth
Assistant Professor

Head, CSE Dept.
Dr. Y. Rama Devi
Professor

Place: HYDERABAD

Date:

DECLARATION

We hereby declare that the project entitled "**Detecting Fraudulent Vehicle Insurance Claims**" submitted for the B.E (CSE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

RAMA KRISHNA PARJANYA VARANASI (160118733038)

VENKATA SAI TEJA GADDAM (160118733055)

Place: HYDERABAD

Date:

ABSTRACT

Insurance claim investigations are challenging as there might be a hit on the customer satisfaction because of delayed payouts during the period of stress. On the other hand, Incorrect payouts might reduce profitability and encourage other policyholders to engage in similar delinquent behavior. As a result, the insurance industry has an immediate need to build a competence that can accurately identify suspected frauds.

This project aims to solve the problem using machine learning classification i.e., if a vehicle insurance claim is fraudulent or not. A model is built over a relatively smaller dataset to overcome this challenge. The model includes preprocessing, balancing the data set using SMOTE (Synthetic Minority Oversampling Technique) algorithm. Classification is performed using Decision Tree, Random Forest, KNN, AdaBoost and XGBoost algorithms and their accuracies are analyzed.

Machine learning models are good for smaller datasets. The performance of the traditional machine learning algorithms will be degraded if data increases. Hence, for a larger dataset, deep learning techniques i.e., Multi-Layer Perceptron model improve the performance of the model by interpreting the data features and relations automatically irrespective of the domain. Hence, the proposed project implements deep learning models and compares them with traditional machine algorithms.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding source of inspiration towards the completion of the project. We would like to express our sincere gratitude and indebtedness to our project guide, **Sri R. Srikanth**, Assistant Professor who has supported us throughout our project with patience and knowledge. We are also thankful to the Head of the Department **Dr. Y. Ramadevi**, for providing excellent infrastructure and atmosphere for completing this project successfully. We are also extremely thankful to our Project coordinators **Dr. K. Sagar**, Professor and **Smt. I. Srujana**, Asst. Professor, and Dept. of CSE, for their valuable suggestions and interest throughout the course of this project. We would like to express our deep sense of gratitude to our principal**, Dr. P. Ravinder Reddy,** Professor and the management of CBIT for having designed an excellent learning atmosphere. We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

LIST OF FIGURES

Figure Number	Figure Name	Page Number
3.1.1	Block diagram of Proposed System	9
3.3.1.1	Decision Tree Classifier	11
3.3.2.1	Random Forest Classifier	12
3.3.6.1	Architecture of Stacking	13
3.3.7.1	Multilayer Perceptron Model	14
3.3.8.1	SMOTE Approach	15
4.1.1	Flowchart of Proposed System	16
4.1.2	Level 0 Data Flow Diagram of Proposed System	17
4.1.3	Level 1 Data Flow Diagram of Proposed System	17
4.1.4	Use Case Diagram of Proposed System	18
4.3.1	Small Dataset Heatmap	21
4.3.2	Large Dataset Heatmap	22
4.3.3	Sigmoid Activation Function	26
4.3.4	ReLU Activation Function	26
5.1	Box Plot - Accuracies	32
5.2	Confusion Matrix	33
5.3	Confusion Matrix - KNN Classifier	34
5.4	Confusion Matrix - XGBoost Classifier	34
5.5	Confusion Matrix - Decision Tree Classifier	34
5.6	Confusion Matrix - Random Forest Classifier	35
5.7	Confusion Matrix - AdaBoost Classifier	35
5.8	Confusion Matrix - Stacking Classifier	35

5.9	Flask App Home Page - Small Dataset	36
5.10	Flask App Input Screen - Small Dataset	37
5.11	Flask App Output Screen - Fraudulent Input - Small Dataset	37
5.12	Flask App Output Screen - Genuine Input - Small Dataset	38
5.13	Flask App Home Page - Large Dataset	39
5.14	Flask App Input Screen - Large Dataset	39
5.15	Flask App Output Screen - Fraudulent Input - Large Dataset	40
5.16	Flask App Output Screen - Genuine Input - Large Dataset	40

LIST OF TABLES

Table Number	Table Name	Page Number
4.4.1	Small Dataset Description	30
4.4.2	Large Dataset Description	30
5.1	Accuracies and Standard Deviation Values for Small Dataset	32

Table of Contents

	Title Page	i
	Certificate of the Guide	ii
	Declaration of the Student	iii
	Abstract	iv
	Acknowledgement	v
	List of Figures	vi
	List of Tables	viii
1.	**INTRODUCTION**	**1**
	1.1 Problem Definition including the significance and objective	1
	1.2 Methodologies	2
	1.3 Outline of the results	2
	1.4 Scope of the project	2
	1.5 Organization of the report	3
2.	**LITERATURE SURVEY**	**4**
	2.1 Introduction to the problem domain terminology	4
	2.2 Existing solutions	4
	2.3 Related works	5
	2.4 Tools/Technologies used	7
3.	**DESIGN OF THE PROPOSED SYSTEM**	**9**
	3.1 Block Diagram	9
	3.2 Module Description	10
	3.3.Theoretical Algorithms	11
	3.3.1 Decision Tree Classifier	11
	3.3.2 Random Forest Classifier	11
	3.3.3 KNN Classifier	12
	3.3.4 AdaBoost Classifier	12
	3.3.5 XGBoost Classifier	13
	3.3.6 Stacking Classifier	13
	3.3.7 Multilayer Perceptron Model	14
	3.3.8 SMOTE Algorithm	15
4.	**IMPLEMENTATION OF THE PROPOSED SYSTEM**	**16**
	4.1 Flowcharts / DFDs / Use Case	16
	4.2 Design and Test Steps / Criteria	19
	4.3 Implementation Details	20
	4.4 Data Set description	28
	4.5 Testing Process	31
5.	**RESULTS / OUTPUTS AND DISCUSSIONS**	**32**

6. CONCLUSIONS / RECOMMENDATIONS **41**
6.1 Conclusions 41
6.2 Limitations 41
6.3 Future Scope 42

REFERENCES **43**

APPENDIX **45**

1. INTRODUCTION

Insurance fraud is a broad phrase that refers to a variety of unethical actions taken by a person in order to obtain a favourable outcome from an insurance company. This could include orchestrating the incident, lying about the facts, such as the relevant actors and the incident's cause, as well as the extent of the damage caused.

Every year, fraudulent car claims cost the insurance business a significant amount of money. Claims padding accounts for almost 90% of all auto insurance fraud (which means to add damages, injuries and fictitious passengers to insurance claims). The remaining 10% of insurance fraud is caused by coordinated accident staging. In India, it is estimated that every year a sum of INR 30 Crore is lost to false claims and scams under car insurance. As a result, insurance companies perform thorough investigation of the claims before releasing claim amounts. However, these investigations take anywhere around 30 days to 6 months to conclude as they are mainly performed manually and this delay may cause problems to the people with genuine claims. Therefore, our project addresses the above issues.

1.1 Problem Definition including the significance and objective

This project's goal is to create a model that can detect fraudulent vehicle insurance claims. The difficulty with machine learning fraud detection is that frauds are significantly less common than genuine insurance claims. Unbalanced class classification is the term for this type of issue. Frauds are immoral and cost insurance companies money. We can reduce insurance company losses by developing a model that can classify automobile insurance claims. This project aims to classify claims related to vehicle insurance fraud.

1.2 Methodologies

In insurance fraud detection, we first have to deal with the imbalance problem. Insurance fraud datasets have an imbalance in them as the number of observations of reported/confirmed frauds is significantly less in number. Therefore, initially, we will balance the dataset using the SMOTE (Synthetic Minority Oversampling Technique). Next, we will build various machine learning models such as Decision Tree, Random Forest, AdaBoost, and XGBoost, and a Multi-Layer Perceptron model which is a deep learning model. Then, we will compare the performance of the above models. Finally, we will select the best model to use in the flask application to deploy the fraudulent insurance claim prediction application on the web.

1.3 Outline of Results

The proposed model takes the input from the user and predicts whether the claim data provided by the user is fraudulent or genuine. If the probability is less than fifty percent, the claim is classified as genuine. Conversely, if the probability is greater than fifty percent, the claim is classified as fraudulent.

1.4 Scope of the Project

The scope of our project is confined to vehicle insurance claims. The main objective is to classify whether a particular claim is fraudulent or not based on the input claim. Release of claim amounts without proper investigation causes losses to the insurance companies. On the other hand, prolonged investigations during times of grief or urgency will cause their customers to lose trust in the insurance company. Thereby, leading to a gradual decline in customer satisfaction. Therefore, insurance companies will be able to use our model as a preliminary step in their investigation process, leading to quicker problem resolution.

1.5 Organisation of the Report

The project has been divided into the following components:
Chapter 1 - Introduction. Describes the main idea i.e., objective and methodologies of this project.
Chapter 2 - Literature Survey. This section provides a clear idea about the existing system
and the motivation that leads us to the present project.
Chapter 3 - Design of Proposed System. Gives basic idea of the process and methods to be followed in order to implement this project.
Chapter 4 - Implementation of Proposed System. Gives a detailed description of individual processes within the project.
Chapter 5 - Results and Discussion. This section will show the outputs obtained as a result of implementing the procedures explained in previous sections.
Chapter 6 - Conclusion and Future Work. We have given our observations on the results obtained and what future work we intend to do to continue this project.
Chapter 7 - References

2. LITERATURE SURVEY

In this chapter, the terminology related to the problem domain is discussed. Further, review of existing systems and related works is done. Finally, the tools and technologies that would be required to carry out the project are included.

2.1 Introduction to the Problem Domain Terminology

Various machine learning and deep learning algorithms are used to solve many types of problems such as classification, regression, clustering, time-series forecasting, etc. Insurance fraud detection is a classification problem and we will be using various algorithms to classify the claims. Insurance is a contract, represented by a policy, in which an individual or entity receives financial protection or reimbursement against losses from an insurance company. Insurance fraud involves any misuse of insurance policies or applications in order to illegally gain or benefit. It is usually an attempt to exploit an insurance contract for financial gain.

2.2 Existing Systems

Traditional insurance fraud detection procedures are time-consuming and complicated. Expert scrutiny, adjusters, and special investigation services are the mainstays of their operations. However, the number of experts in the investigation teams are negligible compared to the increasing number of claims that the insurance companies receive on a daily basis. As a result, the case descriptions can only be extracted, analysed, and evaluated to a limited extent by a few experts. As a result, the average amount of time spent by experts analysing a case is insufficient. Even for the same scenario, individual experts' conclusions may be very different due to their differing points of view. Therefore, manual detection incurs additional resources and may produce erroneous findings. Additionally, the people involved in the investigation process can be compromised, due to this, the investigation officials may classify the claims incorrectly. Moreover, insurance companies may incur additional losses as a result of late decisions.

2.3 Related Works

[1] Detecting Fraudulent Insurance Claims Using Random Forests and Synthetic Minority Oversampling Technique

Authors - Sonakshi Harjai, Sunil Kumar Khatri and Gurinder Singh

Over the last few years, there has been a considerable increase in the number of policyholders engaging in fraudulent activities. Deliberately deceiving insurance providers by missing facts and concealing information when filing insurance claims has resulted in severe financial and consumer value loss. A proper structure for judiciously monitoring insurance fraud is necessary to keep these risks under control. They present a novel strategy for developing a machine-learning-based auto-insurance fraud detector that can identify fraudulent claims from a dataset of over 15,420 car-claim records in this work. The model is constructed using the synthetic minority oversampling technique (SMOTE), which eliminates the dataset's class imbalance. To classify the claim records, the model uses Random Forest classification algorithm. The data they used in their experiment came from a publicly available dataset on auto insurance. On the basis of numerous performance measures, the results of their approach were compared to those of other existing models.

Methodology: The research outlined a method for developing a machine learning-based auto-insurance fraud detector that can identify fraudulent insurance claims from a dataset of over 15,420 entries. The suggested model is based on the synthetic minority oversampling method (SMOTE), which eliminates the dataset's class imbalance. To classify the claim records, they employed the Random Forests classification approach. Support Vector Machine (SVM), Decision-Tree, and Multi-layer Perceptron were the three supervised classifiers that the model outperformed (MLP). Additional data balancing approaches or other classifiers that are not affected by the class imbalance can be used to improve the model. Support Vector Machines (SVMs) are rudimentary, and using more advanced categorization methods can improve performance.

[2] Automobile Insurance Fraud Detection using Supervised Classifiers

Authors - Iffa Maula Nur Prasasti, Arian Dhini and Enrico Laoh

Automobile fraud has a number of ramifications for both the company and the insured. The existing detection system is ineffective and expensive. The goal of this study is to use a machine learning approach to develop a prediction model for detecting vehicle insurance fraud. The research was based on real-world data from an Indonesian vehicle insurance business. The dataset has a highly skewed distribution of data from policyholders who commit fraud vs data from honest policyholders.

Methodology: The study uses the Synthetic Minority Oversampling Technique (SMOTE) and undersampling methods to deal with the imbalanced dataset problem. Multilayer Perceptron (MLP), Decision Tree C4.5, and Random Forest are the proposed supervised classifiers. The confusion matrix, ROC Curve, and characteristics such as sensitivity are used to assess model performance. Random Forest topped the findings of other classifiers with 98.5 percent accuracy, according to this study.

[3] Detecting Fraudulent Motor Insurance Claims Using Support Vector Machines

Authors - Charles Muranda, Ahmed Ali, Thokozani Shongwe

Unbalanced training sets cause problems on classification systems. When it comes to detecting fraudulent claims in the insurance industry, fraud instances are uncommon compared to real claims. As a result, algorithms for detecting fraud have fewer positive training samples, resulting in lower performance metrics than when there are equal cases. They present a machine learning strategy for detecting fraudulent claims in the study.

Methodology: To reduce imbalances in the dataset, the suggested method employs the adaptive synthetic sampling method (ADASYN). The claim cases are classified using Support Vector Machines (SVM). The algorithm's results are compared to unbalanced datasets and other existing approaches.

2.4 Tools/Technologies used

Below are the tools and technologies that would be required to carry out the project

2.4.1 TensorFlow

TensorFlow is an open-source end-to-end machine learning platform. It's a symbolic maths toolkit that employs dataflow and differentiable programming to handle a variety of tasks related to deep neural network training and inference. It helps developers to build machine learning applications utilising a variety of tools, libraries, and open-source resources.

2.4.2 Keras

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. It is intended to facilitate rapid deep neural network research, with a focus on being user-friendly, modular, and extendable. Keras includes many implementations of standard neural-network building blocks like layers, objectives, activation functions, optimizers, and a slew of other tools to make working with image and text data easier while also reducing the amount of coding required to write deep neural network code. Keras supports convolutional and recurrent neural networks in addition to standard neural networks. It supports other common utility layers like dropout, batch normalisation, and pooling.

2.4.3 Pandas

Pandas is an open-source library designed to make it simple and intuitive to work with relational or labelled data. It comes with a number of data structures and methods for manipulating numerical data and time series. This library is based on the NumPy library. Pandas is fast and provides users with excellent performance and productivity.

2.4.4 Numpy

NumPy is a free and open-source array processing package. It includes a high-performance multidimensional array object as well as utilities for manipulating them. It is the most important Python package for scientific computing.

2.4.5 Matplotlib

Matplotlib is a Python toolkit for creating two-dimensional graphs and plots with Python scripts. It features a module called pyplot that simplifies charting by allowing users to adjust line styles, font characteristics, and axes formatting, among other things.

2.4.6 Seaborn

Seaborn is a data visualisation library for Python that is developed on top of matplotlib and tightly linked with pandas data structures. Visualisation is a key component of Seaborn, as it aids data exploration and comprehension.

2.4.7 Scikit-learn

Scikit-learn is an open-source Python library that implements a range of machine learning, pre-processing, cross-validation, and visualisation algorithms using a unified interface. It features various classification, regression and clustering algorithms such as random forests, gradient boosting, k-means, etc.

2.4.8 Imbalanced-Learn

It is a Python package for balancing datasets that are heavily skewed or biassed toward certain classes. As a result, it aids in resampling classes that are otherwise over or undersampled. When the imbalance ratio is significant, the output is skewed toward the class with the most examples.

2.4.9 Flask

Flask is a Python-based microweb framework. It is referred to as a microframework because it does not necessitate the usage of any specific tools or libraries. The Werkzeug WSGI toolkit and the Jinja2 template engine are the foundations of Flask.

3. DESIGN OF THE PROPOSED SYSTEM

In this chapter, the design of the proposed system is illustrated with the help of a block diagram. Each block in the block diagram is explained in the module description section. Finally, review of the algorithms that are used in the project is done.

3.1 Block Diagram

A block diagram is a system diagram in which the major components or functions are represented by blocks connected by lines that depict the relationships between the blocks. They are usually used for high-level, less-detailed descriptions that are meant to illustrate the overall concepts without getting into implementation details.

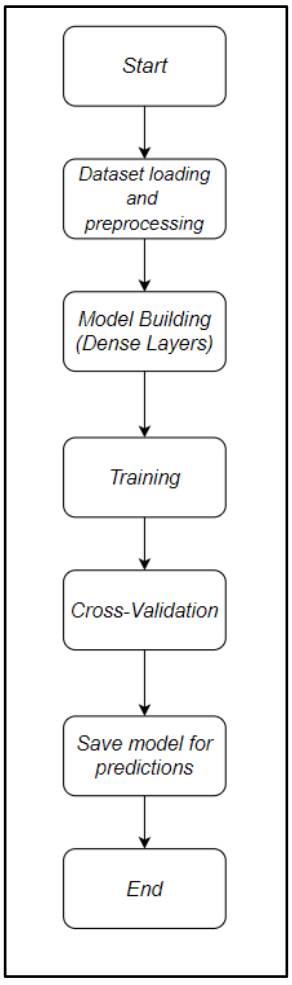


Fig 3.1.1: Block Diagram of Proposed System

3.2 Module Description

The above block diagram illustrates the basic workflow of the project.

3.2.1 Dataset Preprocessing

In this module, the data frame is searched for missing values and replaced with NaN (Not a Number). Then, Label encoding is performed for non-numeric values. Since classification involves selecting input parameters and a target variable, we create a heatmap for the same and select the top correlated attributes as input dimensions. The data frame is further checked for class imbalance and is removed using SMOTE technique.

3.2.2 Model Building

In this module, both machine learning and deep learning techniques are used to construct a suitable model. Since the correlation values from heatmap are really low , we make sure to build tightly-coupled models to suit our use-case. For example, in the MLP model, we add a good number of Dense layers to provide maximum concatenation of the result during propagation.

3.3.3 Training and Testing

This module is responsible for training the model with the dataset obtained from the previous modules. It involves splitting the datasets into training, testing, cross-validation, monitoring the loss and calculating performance metrics after the training.

3.3 Theoretical Algorithms

3.3.1 Decision Tree Classifier

Decision Tree is a supervised learning technique that can be used to address classification and regression problems, however it is typically used for classification. It's a tree-structured classifier with internal nodes that represent dataset attributes, branches that represent decision rules, and leaf nodes that represent the result.

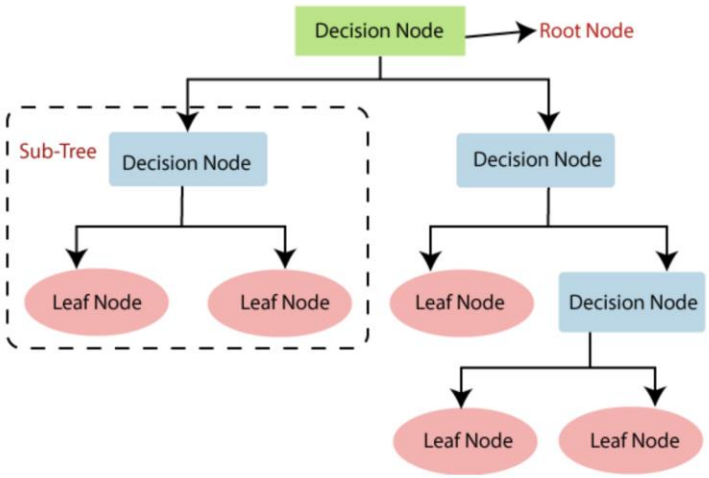


Fig 3.3.1.1 Decision Tree Classifier

There are two types of nodes, decision nodes and leaf nodes. Decision nodes are used to make any decision and can have multiple branches. Leaf nodes are the outputs of those decisions and do not contain any further branches.

3.3.2 Random Forest Classifier

Random Forest is a supervised learning technique. It is based on the notion of ensemble learning, which is a strategy for solving a complicated problem and improving the model's performance by combining multiple classifiers. Random Forest is a classifier that averages the results of several Decision Trees on distinct subsets of a dataset to improve the predictive accuracy.

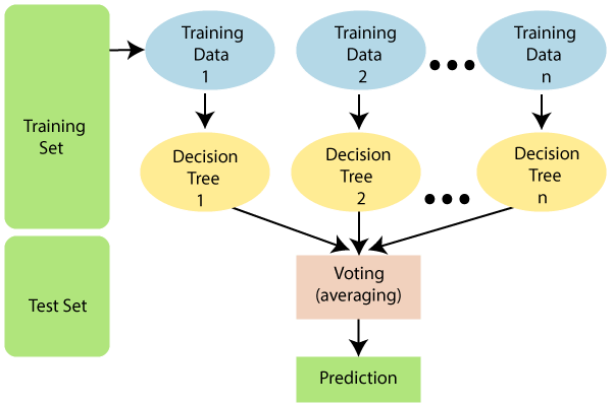


Fig. 3.3.2.1 Random Forest Classifier

Therefore, rather than depending on a single Decision Tree, Random Forest considers the predictions from each tree and predicts the final output based on the majority votes of predictions. As a result, this leads to higher accuracy and prevents the problem of overfitting.

3.3.3 K Nearest Neighbour Classifier

K Nearest Neighbour is a supervised learning technique. It assumes that the new case/data and old cases are comparable, and it assigns the new case to the category that is closest to the current ones. It keeps all of the available data and classifies new data points depending on how closely they resemble the existing data. During the training phase, the KNN algorithm saves the dataset, and when it receives new data, it classifies it into a category that is quite similar to the new data.

3.3.4 AdaBoost Classifier

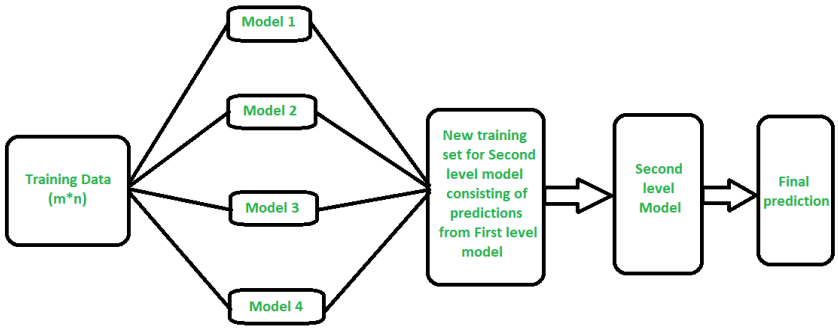
AdaBoost, which stands for Adaptive Boosting, is based on the boosting principle. Boosting is an ensemble modelling approach for generating a strong classifier from a set of weak ones. It's done by putting together a model from a series of weak models. To begin, the training data is used to develop a model. The second model is then developed, with the goal of correcting the prior model's flaws. This procedure is repeated until the full training dataset is accurately predicted or the maximum number of models is added.

3.3.5 XGBoost Classifier

XGBoost, which stands for Extreme Gradient Boosting, is an implementation of Gradient Boosted decision trees. Each predictor in gradient boosting corrects the error of its predecessor. When compared to Adaboost, the training instance weights are not changed; instead, each predictor is trained using the predecessor's residual errors as labels.

3.3.6 Stacking Classifier

In stacking, an algorithm takes the outputs of sub-models as input and tries to figure out how to combine the input predictions in the best way possible to get a superior output prediction. The stacking model's architecture is intended to comprise two or more base/learner models, as well as a meta-model that combines the base models' predictions. The meta-model is referred to as level 1, whereas the basic models are referred to as level 0. The Stacking ensemble approach includes original (training) data, primary level models, primary level prediction, secondary level model, and final prediction.



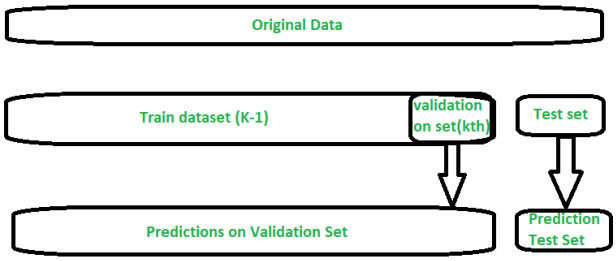


Fig. 3.3.6.1: Architecture of Stacking

3.3.7 Multilayer Perceptron Model

There are at least three levels of nodes in a Multilayer Perceptron: an input layer, a hidden layer, and an output layer. Each node, with the exception of the input nodes, is a neuron with a nonlinear activation function. Multilayer Perceptron is a feedforward algorithm because the inputs are combined with the initial weights in a weighted sum and applied to the activation functions. The algorithm would be unable to learn the weights that optimise the cost function if it just computed the weighted sums in each neuron, transmitted the results to the output layer, and then stopped there.

BackPropagation

Through a method known as chain rule, this algorithm is utilised to successfully train a neural network. In simple words, backpropagation takes a backward pass through a network after each forward pass while modifying the model's parameters (weights and biases), and this weight updation is done using the errors calculated at each layer of the neural network. The Adam optimizer is used to obtain these weight adjustments.

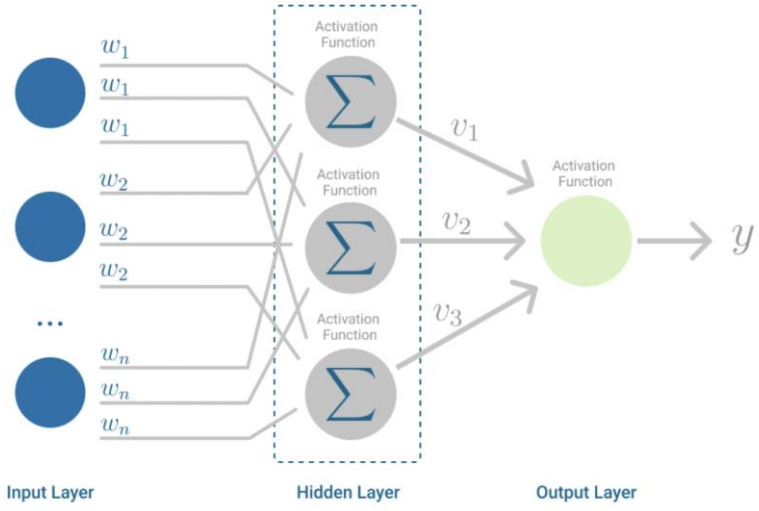


Fig 3.3.7.1: Multilayer Perceptron Model

3.3.8 SMOTE (Synthetic Minority Oversampling Technique)

One of the most often used oversampling strategies to overcome the imbalance problem is SMOTE (Synthetic Minority Oversampling Technique). It seeks to balance the distribution of classes by replicating minority class samples at random.

SMOTE creates new minorities between existing minorities. It creates virtual training records for the minority class using linear interpolation. These synthetic training records are created for each example in the minority class by randomly selecting one or more of the k-nearest neighbours. After the oversampling method, the data is reconstructed, and many classification models can be applied to the processed data.

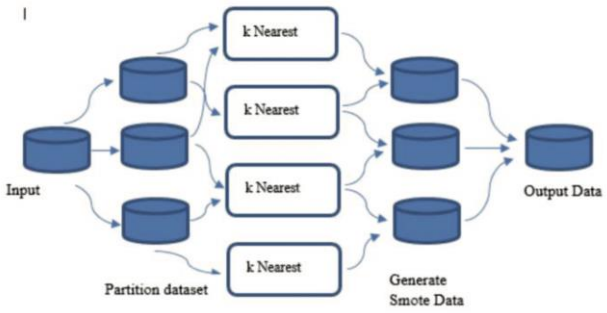


Fig. 3.3.8.1: SMOTE Approach

SMOTE Algorithm

O is the original data set

P is the set of positive instances (minority class instances)

For each instance x in P

Find the k-nearest neighbours (minority class instances) to x in P

Obtain y by randomising one from k instances

difference = x-y

gap = random number between 0 and 1

n = x + difference * gap

Add n to O

End For

4. Implementation of the Proposed System

The chapter discusses the implementation of the proposed system with the help of Flowchart, DFDs and Use Case diagrams.

4.1 Flowcharts/ DFDs/ Use Case Diagrams

Flowchart

A flowchart is a diagram that depicts an algorithm. It is frequently used by programmers as a problem-solving technique. It employs symbols that are linked together to represent the flow of information and processing.

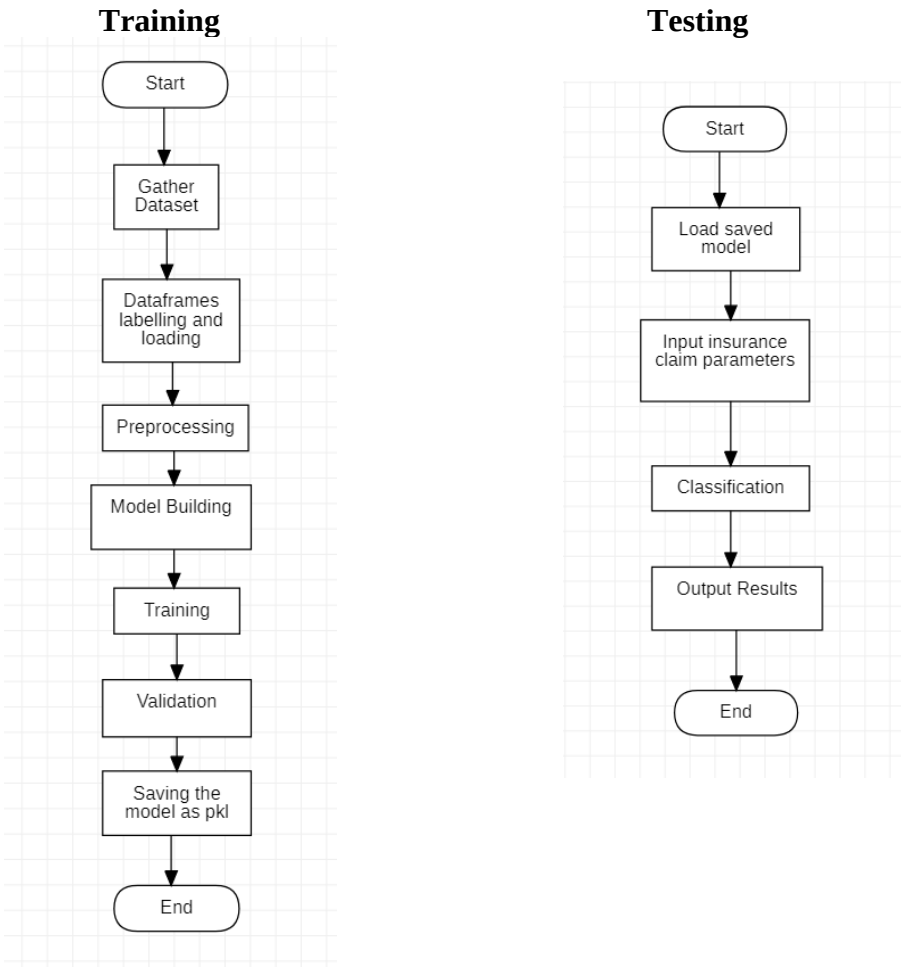


Fig 4.1.1: Flowchart of Proposed System

Data Flow Diagram

A Data Flow Diagram (DFD) is a classic visual representation of how data moves through a system. It depicts how data enters and exits the system, as well as what changes the data and where it is stored. A DFD's goal is to demonstrate the scope and bounds of a system as a whole. The visual representation makes it a good communication tool between a user and a system designer. A DFD's structure allows you to start with a broad overview and work your way down to a hierarchy of specific diagrams.

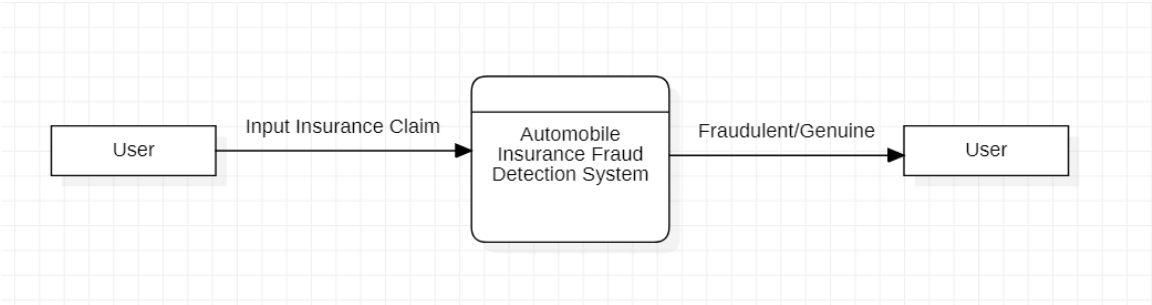


Fig 4.1.2: Level 0 Data Flow Diagram of Proposed System

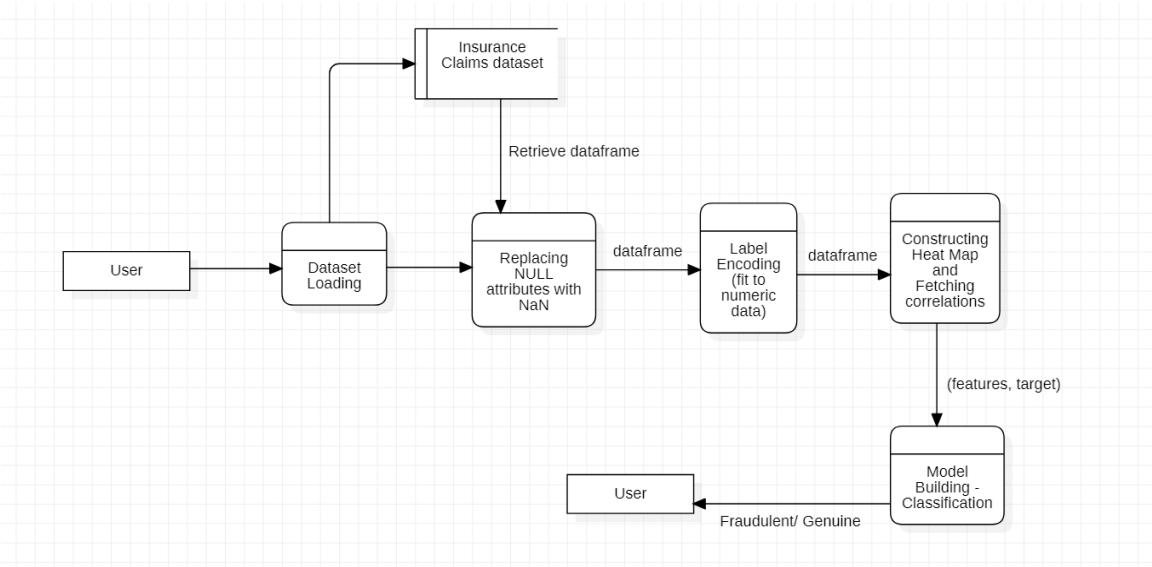


Fig 4.1.3: Level 1 Data Flow Diagram of Proposed System

UML Diagrams

The Unified Modelling Language, or UML, is a method for visually representing the architecture, design, and implementation of complex software systems. When writing code, an application can have thousands of lines, making it challenging to keep track of relationships and hierarchies within a software system. That software system is divided into components and subcomponents using UML diagrams.

Use Case Diagram

A use case diagram is a visual representation of a system's dynamic behaviour. It incorporates use cases, actors, and their interactions to encapsulate the system's functionality. It denotes the activities, services, and functionalities that a system/subsystem of an application requires. It represents a system's high-level functionality as well as the user's interaction with it.

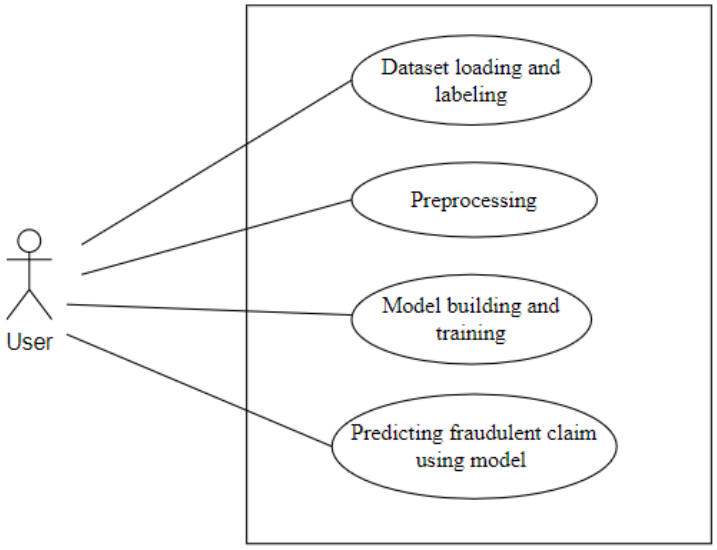


Fig 4.1.4: Use Case Diagram of Proposed System

4.2 Design and Test Steps/Criteria

There are various phases involved in the design of our project. In the first phase, we have gathered two datasets, a small dataset of 1000 rows and 39 columns and a large dataset of 100000 rows and 52 columns.

Initially, we loaded the datasets into the dataframe using the pandas library. Next, we looked for any missing values in the datasets and replaced them with NaN (Not a Number). And then, we performed label encoding, which involves converting labels in string form to numeric form. This concludes the preprocessing phase.

In the second phase, we built a seaborn heatmap and have drawn the correlations for the given data frame by keeping fraud_reported as the target variable for the small dataset. From the heatmap, we selected 20 features. Similarly, for the large dataset, we have drawn the correlations for the given data frame by keeping NB_Claim as the target variable for the large dataset and selected 33 features.

In the third phase, we built various machine learning classification models such as Decision Tree, Random Forest, KNN, AdaBoost, XGBoost, and Stacking models. Along with them, we have also built a Multiple Layer Perceptron model, which is a deep learning classification model.

In the fourth phase, we validated the models using stratified k-fold cross validation method and compared the performance of the above models.

Finally, we saved the models as pickle files and used the best model in the flask application for deploying the fraudulent insurance claim prediction application.

4.3 Implementation Details

Required packages for data preprocessing,

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

For the Machine Learning models, the required packages are,

```
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from numpy import mean
from numpy import std
```

For the Multilayer Perceptron model, the required packages are,

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
```

Dataset Balancing

In the small dataset, the number of observations in fraud_reported is as follows,
Not Fraud = 753 and Fraud = 247, therefore, we need to balance the dataset and it is done by the following code.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
```

Feature Selection

For the small dataset, we built a seaborn heatmap and have drawn the correlations for the given data frame by keeping fraud_reported as the target variable. From the heatmap, we selected 20 features.

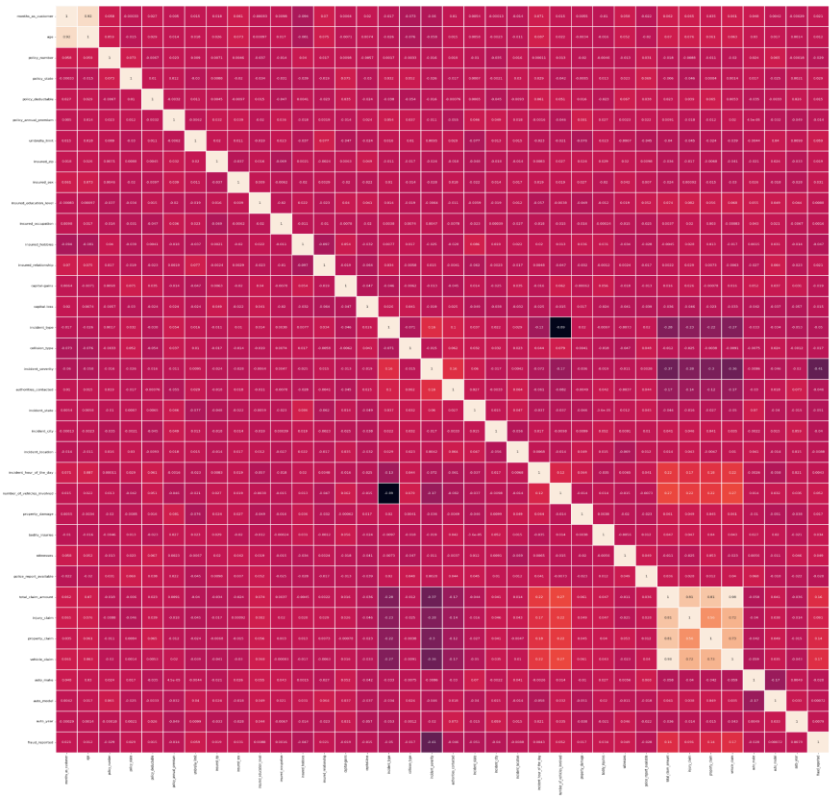


Fig 4.3.1: Small Dataset Heatmap

The selected features are:

```
corrs = df.corr()['fraud_reported']
columns = corrs[corrs > .001].index
corrs = corrs.filter(columns)
corrs.sort_values(ascending=False)
```

Fraud_reported	1.000000
vehicle_claim	0.170049
total_claim_amount	0.163651
property_claim	0.137835
injury_claim	0.090975
umbrella_limit	0.058622
number_of_vehicles_involved	0.051839
witnesses	0.049497
bodily_injuries	0.033877
insured_sex	0.030873
policy_state	0.029432
insured_relationship	0.021043
months_as_customer	0.020544
insured_zip	0.019368
property_damage	0.017202
policy_deductable	0.014817
age	0.012143
insured_education_level	0.008808
auto_year	0.007928
incident_hour_of_the_day	0.004316
insured_occupation	0.001564

Similarly, for the large dataset,

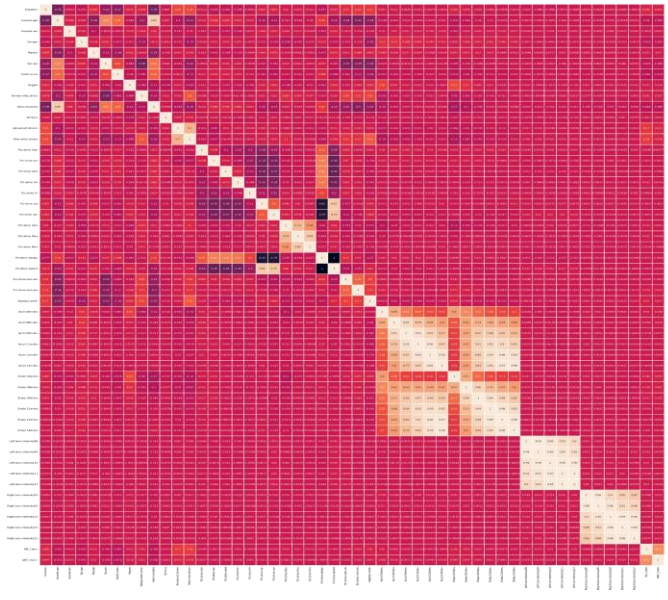


Fig 4.3.2: Large Dataset Heatmap

The selected features are:

```
corrs = df.corr()['NB_Claim']
columns = corrs[corrs > .001].index
corrs = corrs.filter(columns)
corrs.sort_values(ascending=False)
```

```
NB_Claim                1.000000
AMT_Claim               0.528515
Total.miles.driven      0.181060
Annual.pct.driven       0.168464
Duration                0.082083
Avgdays.week            0.050272
Brake.06miles           0.045633
Annual.miles.drive      0.041297
Brake.08miles           0.038756
Accel.06miles           0.030259
Pct.drive.rush pm       0.028721
Pct.drive.2hrs          0.022346
Region                  0.022334
Marital                 0.020760
Brake.09miles           0.016684
Left.turn.intensity08   0.013435
Pct.drive.thr           0.012478
Left.turn.intensity09   0.012205
Pct.drive.3hrs          0.010584
Left.turn.intensity10   0.010072
Left.turn.intensity12   0.009975
Left.turn.intensity11   0.009796
Right.turn.intensity10  0.009557
Right.turn.intensity11  0.009304
Accel.08miles           0.009168
Right.turn.intensity09  0.008940
Right.turn.intensity08  0.008840
Right.turn.intensity12  0.007664
Brake.11miles           0.006244
Pct.drive.wkday         0.005085
Pct.drive.mon           0.004896
Pct.drive.rush am       0.004401
Accel.09miles           0.001756
Pct.drive.4hrs          0.001414
Name: NB_Claim, dtype: float64
```

Model Building

We built six machine learning models, namely, Decision Tree, Random Forest, KNN, AdaBoost, XGBoost and Stacking.

- For the Decision Tree classifier, the maximum depth of the tree is set to 10 and the random state, which controls the randomness of the estimator, is set to 5.
- For the Random Forest classifier, the number of trees in the forest is set to 2000 trees.
- For the KNN classifier, the number of neighbours is set to 5.
- For the AdaBoost classifier, the maximum number of estimators at which boosting is terminated is set to 500.
- In the Stacking model, the base models/level 0 models are Decision Tree, Random Forest, KNN, AdaBoost and XGBoost and the level 1 model, which is the model that combines the predictions of the base models is Logistic Regression.

Along with the above models, we have also built a Multilayer Perceptron model.
In the MLP model, the Sequential function is used to create a stack for storing the layers. It also provides training and inference features on this model. The MLP model is built using 6 dense layers. A dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. In the first dense layer, the units parameter is set to 256 and the input dimensions parameter is set to 20 dimensions as 20 features have been selected. As discussed earlier, each of the 20 input dimensions is mapped to each of the 256 neurons. In the next layer, the unit parameter is set to 64 and in this layer, each of the 256 neurons from the previous layer is mapped to each of the 64 neurons. This process is repeated in the next 4 layers and the unit parameter is reduced by the powers of 2. For all the layers except the last one, the ReLU activation function is used to prevent the neurons from being activated for negative inputs. For the last layer, the Sigmoid activation is used to predict the probability of fraud. Similar process is used to implement the MLP model for the large dataset.

Adam Optimizer

Adam optimizer is a hybrid model of the below two models and they are:

- **Adaptive Gradient Algorithm (AdaGrad)** that maintains a per-parameter learning rate that improves performance on problems with sparse gradients.
- **Root Mean Square Propagation (RMSProp)** is a learning algorithm that adapts per-parameter learning rates based on the average of recent gradient magnitudes for the weight.

Instead of using the average first moment (the mean) like in RMSProp, Adam uses the average of the second moments of the gradients to adjust the parameter learning rates (the uncentered variance).

The technique creates an exponential moving average of the gradient and the squared gradient, with the decay rates of these moving averages controlled by the constants beta1 and beta2.

Moment estimations are biassed towards zero when the moving averages' starting value is close to 1.0 and the beta1 and beta2 values are close to 1.0. This prejudice is eliminated by generating bias-corrected estimates first, then bias-corrected estimates.

MinMax Scaler

This estimator scales and translates each feature independently so that it falls within the training set's specified range, for instance, between 0 and 1.
The transformation is given by:

X_Std = (X - X.Min(Axis=0)) / (X.Max(Axis=0) - X.Min(Axis=0))

X_Scaled = X_Std * (Max - Min) + Min

where Min, Max = Feature_Range
This transformation is frequently used as a substitute for unit variance scaling with zero mean.

Binary Cross Entropy

Each of the projected probabilities is compared to the actual class output, which can be either 0 or 1. The score is subsequently calculated, which penalises the probabilities based on their deviation from the expected value. This refers to how close or far the value is to the actual value.

Sigmoid Activation Function

Sigmoid function is defined as: $\frac{1}{(1+e^{-x})}$

Sigmoid function exists between 0 and 1 and is used in models where we have to predict the probability as an output. It is used in such models as probability of anything exists between 0 and 1 only.

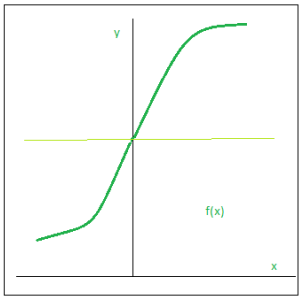
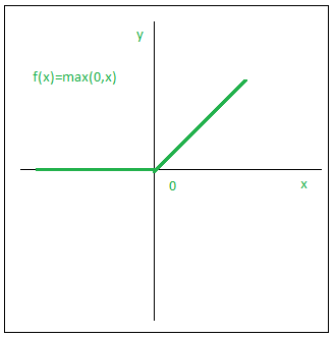


Fig 4.3.3: Sigmoid Activation Function

ReLU (Rectified Linear Unit) Activation Function

Rectified Linear Unit is defined as: $f(x) = \max(0, x)$



4.3.4: ReLU Activation Function

The ReLU activation function is a linear function that outputs the input directly if the input is positive, and zero if the input is negative. It has become the default activation function for many types of neural networks since a model that uses it is faster to train and generally generates better performance.

Machine Learning Models

```
def get_stacking():
  # define the base models
  level0 = list()
  level0.append(('DT', DecisionTreeClassifier(max_depth=10, random_state=5)))
  level0.append(('RF', RandomForestClassifier(n_estimators=2000)))
  level0.append(('KNN', KNeighborsClassifier(5)))
  level0.append(('ADA', AdaBoostClassifier(n_estimators=500)))
  level0.append(('XGB', XGBClassifier(objective= 'binary:logistic', eval_metric='logloss')))
  level1 = LogisticRegression(solver='lbfgs', max_iter=5000)
  # define the stacking ensemble
  model = StackingClassifier(estimators=level0, final_estimator=level1, cv=10)
  return model

# get a list of models to evaluate

def get_models():
  models = dict()
  models['Decision Tree'] = DecisionTreeClassifier(max_depth=10)
  models['Random Forest'] = RandomForestClassifier(n_estimators=2000)
  models['KNN'] = KNeighborsClassifier(5)
  models['ADA'] = AdaBoostClassifier(n_estimators=500)
  models['XGBoost'] = XGBClassifier()
  models['Stacking'] = get_stacking()
  return models
```

Multilayer Perceptron Model

```
def build_model():
  model = Sequential()
  model.add(Dense(256, input_dim=33, activation='relu'))
  model.add(Dense(64, activation='relu'))
  model.add(Dense(32, activation='relu'))
  model.add(Dense(16, activation='relu'))
  model.add(Dense(8, activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model
```

4.4 Dataset Description

For this project, we have gathered two datasets, a small dataset (containing an attribute for fraud/genuine) and a large dataset.

4.4.1 Small Dataset

This dataset, “Auto Insurance Claims Data” was collected from Kaggle. It features binary classification, that is, fraud or not fraud. This dataset has 1000 rows and 39 columns. The variables in this dataset are -

S.No	Variable	Description
1	months_as_customer	Number of months as insured
2	age	Age of the insured
3	policy_number	Insurance policy number
4	policy_bind_date	The date when the insurance goes into effect
5	policy_state	The state from which the insurance policy is taken
6	policy_csl	Combined single limits
7	policy_deductable	The amount that a insured has to pay before the insurance company starts paying up
8	policy_annual_premium	The total amount of premium paid annually

9	umbrella_limit	It is the extra liability insurance coverage that goes beyond the limits of the insured's auto insurance
10	insured_zip	Insured’s zip code
11	insured_sex	Insured’s sex
12	insured_education_level	Insured’s educational qualification
13	insured_occupation	Insured’s occupation
14	insured_hobbies	Insured’s hobbies
15	insured relationship	Relationship status of the insured
16	capital-gains	It is the profit accrued to the insured from the sale of their capital assets.
17	capital-loss	It is the loss incurred when a capital asset is sold for less than the price it was purchased for.
18	incident_date	Date of the incident
19	incident_type	Type of incident
20	collision_type	Type of collision
21	incident_severity	Severity of the damage
22	authorities_contacted	Department of the emergency services contacted
23	incident_state	The state in which the incident took place
24	incident_city	The city in which the incident took place
25	incident_location	The address at which the incident took place
26	incident_hour_of_the_day	The hour of the day at which the incident took place
27	number_of_vehicles_involved	Number of vehicles involved in the incident
28	property_damage	Whether property was damaged or not
29	bodily_injuries	Number of injuries
30	witnesses	Number of witnesses
31	police_report_available	Whether the police report is available or not
32	total_claim_amount	Total claim amount
33	injury_claim	Injury claim amount

34	property_claim	Property claim amount
35	vehicle_claim	Vehicle claim amount
36	auto_make	Vehicle company
37	auto_model	Vehicle model
38	auto_year	Year of manufacture of the vehicle
39	fraud_reported	Whether fraud was reported or not

Table 4.4.1: Small Dataset Description

4.4.2 Large Dataset

This dataset, "Synthetic Dataset Generation of Driver Telematics" was collected from the website of the University of Connecticut's Mathematics department. It features multi classification. The column NB_Claims, which is our target variable, has four possible values [0,1,2,3]. From this, we have considered the value 0 as fraud and the rest as not fraud. This dataset has 100,000 rows and 52 columns. The variables in this dataset are,

Type	Variable	Description
Traditional	`Duration`	Duration of the insurance coverage of a given policy, in days
	`Insured.age`	Age of insured driver, in years
	`Insured.sex`	Sex of insured driver (Male/Female)
	`Car.age`	Age of vehicle, in years
	`Marital`	Marital status (Single/Married)
	`Car.use`	Use of vehicle: Private, Commute, Farmer, Commercial
	`Credit.score`	Credit score of insured driver
	`Region`	Type of region where driver lives: rural, urban
	`Annual.miles.drive`	Annual miles expected to be driven declared by driver
	`Years.noclaims`	Number of years without any claims
	`Territory`	Territorial location of vehicle
Telematics	`Annual.pct.driven`	Annualized percentage of time on the road
	`Total.miles.driven`	Total distance driven in miles
	`Pct.drive.xxx`	Percent of driving day xxx of the week: mon/tue/.../sun
	`Pct.drive.xhrs`	Percent vehicle driven within x hrs: 2hrs/3hrs/4hrs
	`Pct.drive.xxx`	Percent vehicle driven during xxx: wkday/wkend
	`Pct.drive.rushxx`	Percent of driving during xx rush hours: am/pm
	`Avgdays.week`	Mean number of days used per week
	`Accel.xxmiles`	Number of sudden acceleration 6/8/9/.../14 mph/s per 1000miles
	`Brake.xxmiles`	Number of sudden brakes 6/8/9/.../14 mph/s per 1000miles
	`Left.turn.intensityxx`	Number of left turn per 1000miles with intensity 08/09/10/11/12
	`Right.turn.intensityxx`	Number of right turn per 1000miles with intensity 08/09/10/11/12
Response	`NB_Claim`	Number of claims during observation
	`AMT_Claim`	Aggregated amount of claims during observation

Table 4.4.2: Large Dataset Description

4.5 Testing Process

For testing our models, we used the Cross Validation method. Cross validation is a technique for assessing a model's efficiency by training it on a subset of input data and then testing it on previously unseen input data. It's also considered as a method for determining how well a statistical model generalises to a new dataset. For the Machine Learning models, we used Repeated Stratified K-Fold Cross Validation and for the MLP model, we used Stratified K-Fold Cross Validation.

K-Fold Cross Validation divides the data set into k subsets (called folds), then trains them all while evaluating the model on one (k-1) subset. We iterate k times with a different subset reserved for testing reasons each time in this technique. Assume that there are a total of 25 instances. In the first iteration, we use the first 20% of data for assessment and the remaining 80% for training ([1-5] testing and [5-25] training), and in the second iteration, the second 20% for evaluation and the remaining three subsets for training ([5-10] testing and [1-5 and 10-25] training), and so on.

Stratified k-fold cross validation is the same as k-fold cross validation, except instead of random sampling, stratified k-fold cross validation uses stratified sampling. In Repeated stratified k-fold cross validation, we simply repeat the cross-validation procedure multiple times and report the mean result across all folds from all runs. For the stratified k-fold cross validation, we used 10 Folds. We repeated the cross validation procedure 3 times for the repeated stratified k-fold cross validation.

5. RESULTS / OUTPUTS AND DISCUSSIONS

The models have been trained and tested on the dataset. The following table and box plot illustrates the performance statistics of the aforementioned models obtained by applying stratified k-fold cross validation.

Model	Accuracy	Standard Deviation
Decision Tree	70%	0.0511
Random Forest	83%	0.0326
KNN	70%	0.0393
AdaBoost	71%	0.0306
XGBoost	78%	0.0324
Stacking	84%	0.0292
MLP	78%	4.04

Table 5.1: Accuracies and Standard Deviation Values for Small Dataset

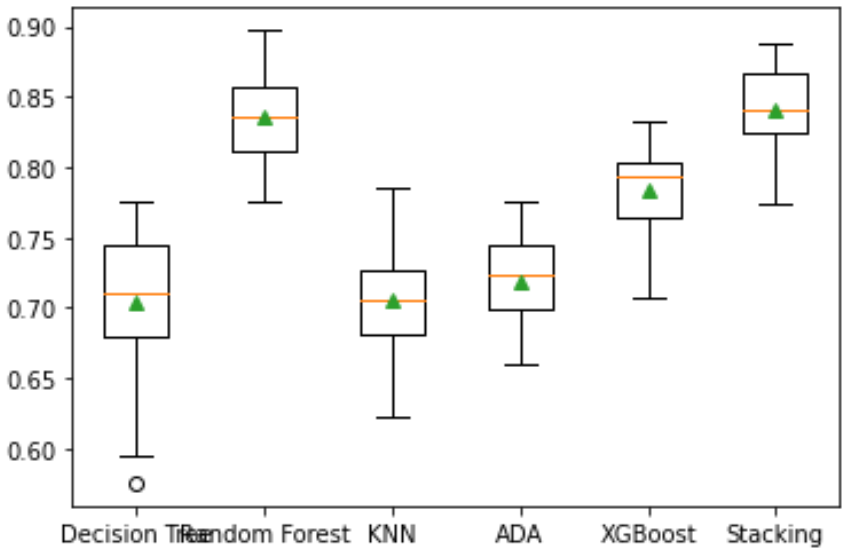


Fig 5.1: Box Plot - Accuracies

In addition to measuring the performance of the models using stratified k-fold cross validation, we have also used some other metrics to evaluate the performance of the models.

Assume [p, n] are the positive and negative classes of the testing set, and [P, N] are the prediction classes of the learning method. The positive class refers to the minority class of fraud cases in this scenario, whereas the negative class relates to non-fraud cases. As shown below, the data can be utilised to build a table of the findings. This table is known as a confusion matrix.

		Prediction class	
		N	***P***
True class	***n***	True Negatives (TN)	False Positives (FP)
	p	False Negatives (FN)	True Positives (TP)

Fig 5.2: Confusion Matrix

Using the data from the table, we can calculate,

1. Overall Accuracy: It is the measure of the overall accuracy of the algorithm. It is calculated using the formula: (TP + TN)/total
2. Precision: When the algorithm predicts yes, how often is it correct? It is calculated using the formula: TP/(TP + FP)
3. Recall: It is a measure of how many positive classes we predicted correctly from all the positive classes. It is calculated using the formula: TP/(TP + FN)
4. F1 Score: The F1 score states the equilibrium between the precision and the recall. It is calculated using the formula: (2*precision*recall) / (precision + recall)

```
Accuracy of KNN Classifier is : 0.6933333333333334
[[200  20]
 [ 72   8]]
              precision    recall  f1-score   support

           0       0.74      0.91      0.81       220
           1       0.29      0.10      0.15        80

    accuracy                           0.69       300
   macro avg       0.51      0.50      0.48       300
weighted avg       0.62      0.69      0.64       300
```

Fig 5.3: Confusion Matrix - KNN Classifier

```
Accuracy of XGBoost Classifier is : 0.7266666666666667
[[214   6]
 [ 76   4]]
              precision    recall  f1-score   support

           0       0.74      0.97      0.84       220
           1       0.40      0.05      0.09        80

    accuracy                           0.73       300
   macro avg       0.57      0.51      0.46       300
weighted avg       0.65      0.73      0.64       300
```

Fig 5.4: Confusion Matrix - XGBoost Classifier

```
Accuracy of Decision Tree Classifier is : 0.68
[[186  34]
 [ 62  18]]
              precision    recall  f1-score   support

           0       0.75      0.85      0.79       220
           1       0.35      0.23      0.27        80

    accuracy                           0.68       300
   macro avg       0.55      0.54      0.53       300
weighted avg       0.64      0.68      0.66       300
```

Fig 5.5: Confusion Matrix - Decision Tree Classifier

```
Accuracy of Random Forest Classifier is : 0.7333333333333333
[[220   0]
 [ 80   0]]
              precision    recall  f1-score   support

           0       0.73      1.00      0.85       220
           1       0.00      0.00      0.00        80

    accuracy                           0.73       300
   macro avg       0.37      0.50      0.42       300
weighted avg       0.54      0.73      0.62       300
```

Fig 5.6: Confusion Matrix - Random Forest Classifier

```
Accuracy of AdaBoost Classifier is : 0.67
[[184  36]
 [ 63  17]]
              precision    recall  f1-score   support

           0       0.74      0.84      0.79       220
           1       0.32      0.21      0.26        80

    accuracy                           0.67       300
   macro avg       0.53      0.52      0.52       300
weighted avg       0.63      0.67      0.65       300
```

Fig 5.7: Confusion Matrix - AdaBoost Classifier

```
Accuracy of Stacking model is : 0.7333333333333333
[[220   0]
 [ 80   0]]
              precision    recall  f1-score   support

           0       0.73      1.00      0.85       220
           1       0.00      0.00      0.00        80

    accuracy                           0.73       300
   macro avg       0.37      0.50      0.42       300
weighted avg       0.54      0.73      0.62       300
```

Fig 5.8: Confusion Matrix - Stacking Classifier

From the above performance statistics, we have ascertained that both Random Forest and Stacking models had outperformed other models and had the same statistics. In the flask application, we have used the Stacking model.

Large Dataset:

For the large dataset, the accuracy of the MLP model is 98% with 0.2848 standard deviation and implemented the flask application using the same.

Flask Application (Small Dataset)

We have implemented the flask application for detecting fraudulent vehicle insurance claims using the stacking model that was trained on the small dataset. Output screenshots are shown below.

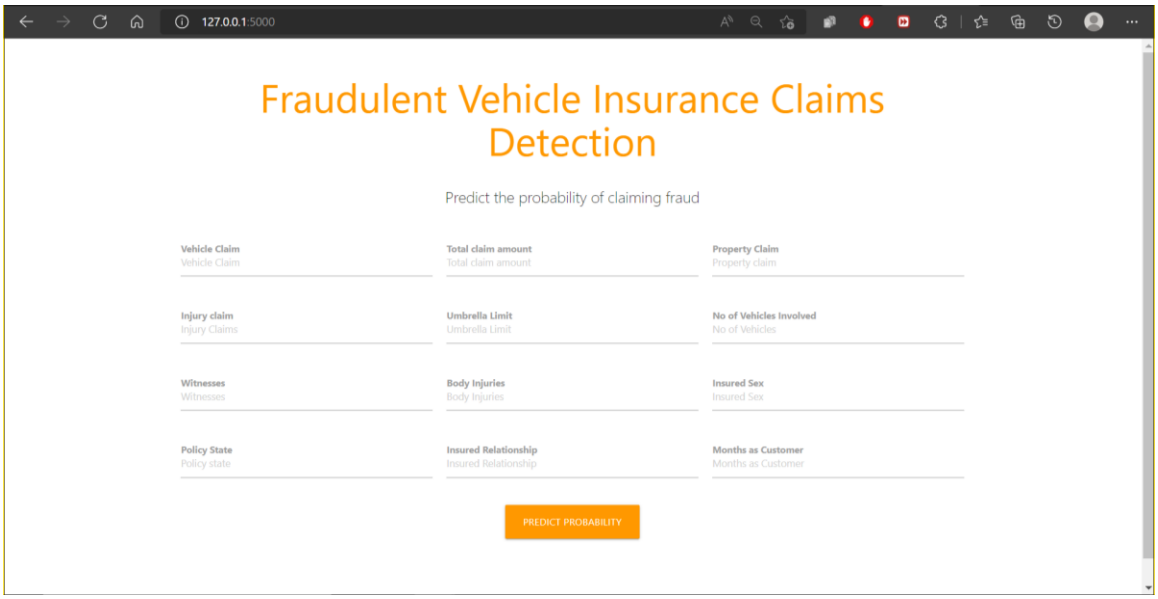


Fig 5.9: Flask App Home Page - Small Dataset

The input labels are the same as the labels that we have chosen as features for the model.

For the given input, the application processes the input, makes a prediction and displays whether the claim is fraudulent or genuine along with the probability of the claim being fraudulent. If the prediction probability is less than 50% then, that claim is classified as genuine. On the contrary, if the prediction probability is greater than 50% then, that claim is classified as fraudulent.

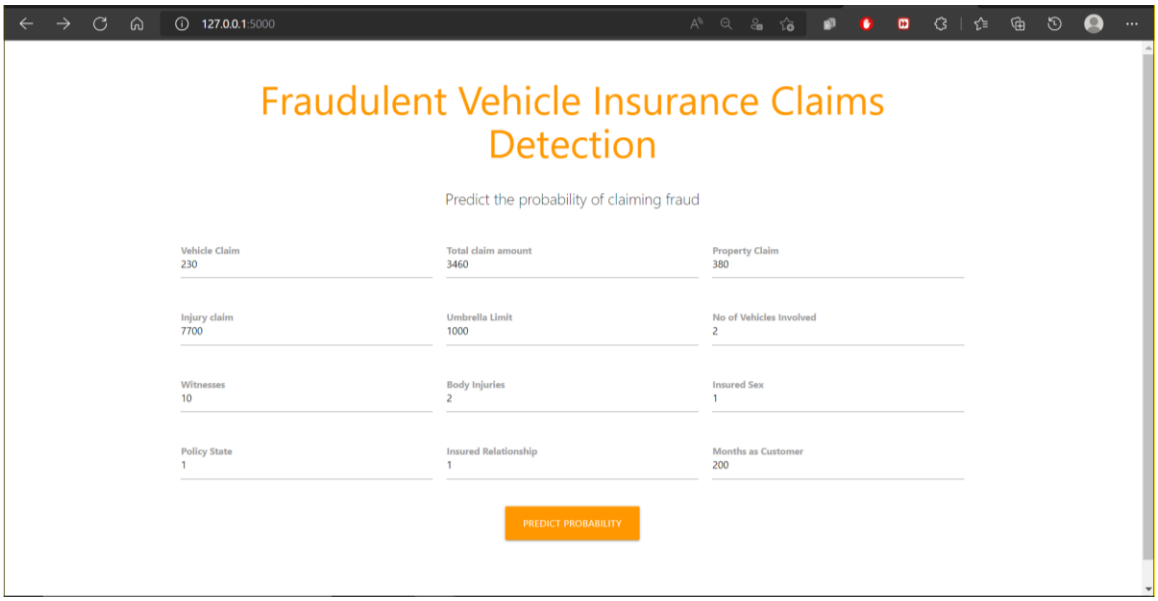


Fig 5.10: Flask App Input Screen - Small Dataset

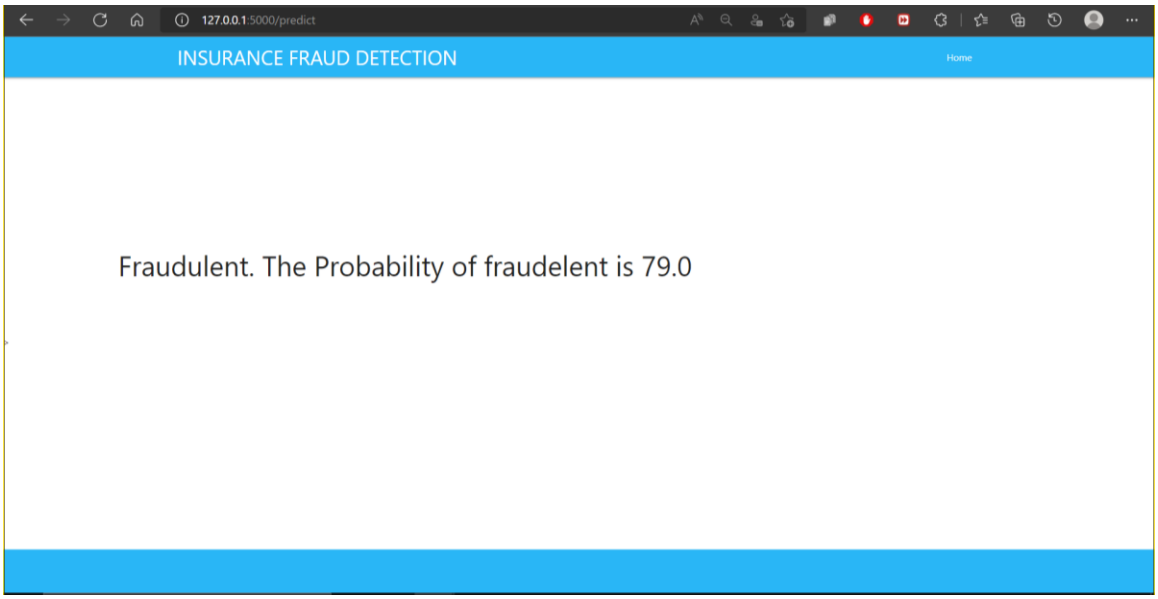


Fig 5.11: Flask App Output Screen - Fraudulent Input - Small Dataset

For the input given above, the application has classified that the claim is 79% fraudulent.

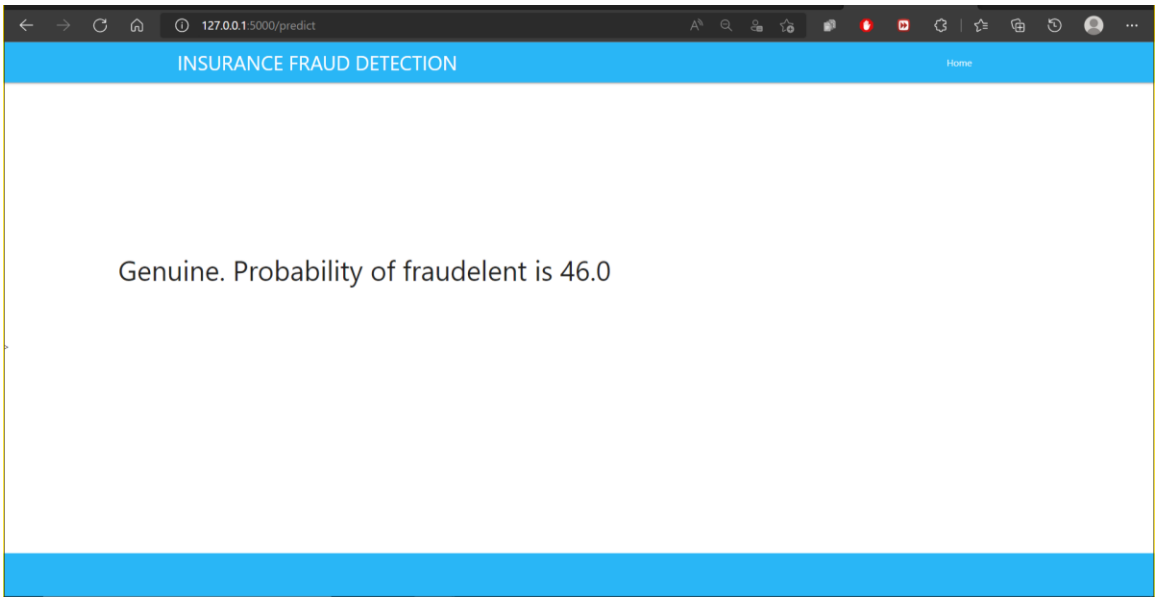


Fig 5.12: Flask App Output Screen - Genuine Input - Small Dataset

Similarly, for another input, the application has classified that the claim is genuine because the probability of it being fraudulent is less than 50%.

Flask Application (Large Dataset)

Similarly, We have implemented the flask application for detecting fraudulent vehicle insurance claims using the MLP model that was trained on the large dataset. Output screenshots are shown below.

The input labels are the same as the labels that we have chosen as features for the model.

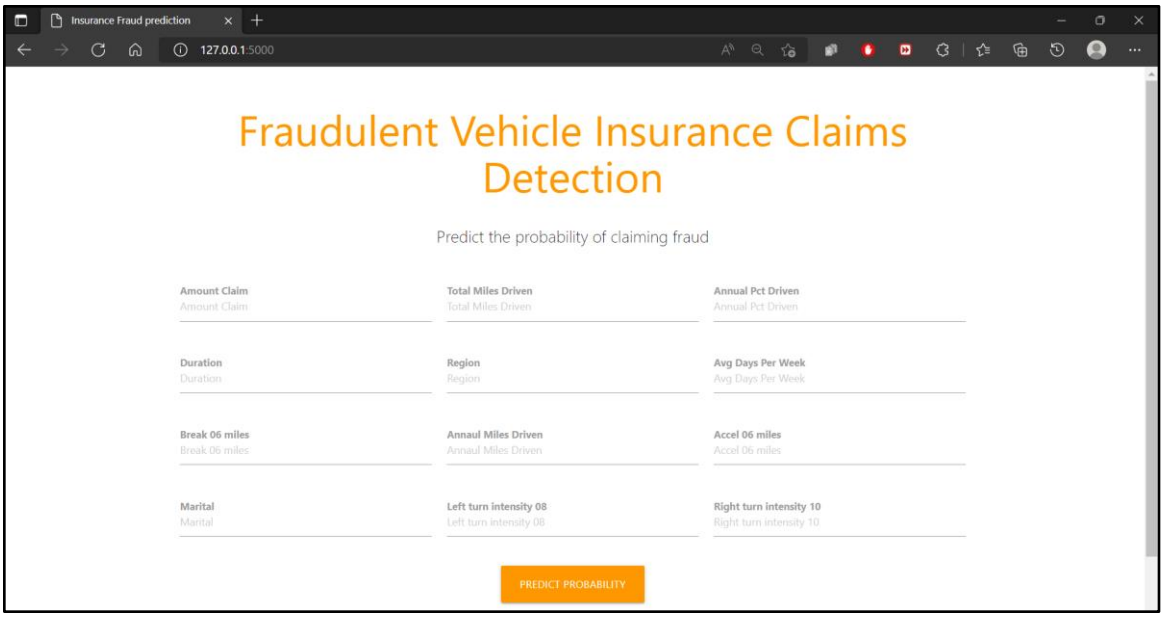


Fig 5.13: Flask App Home Page - Large Dataset

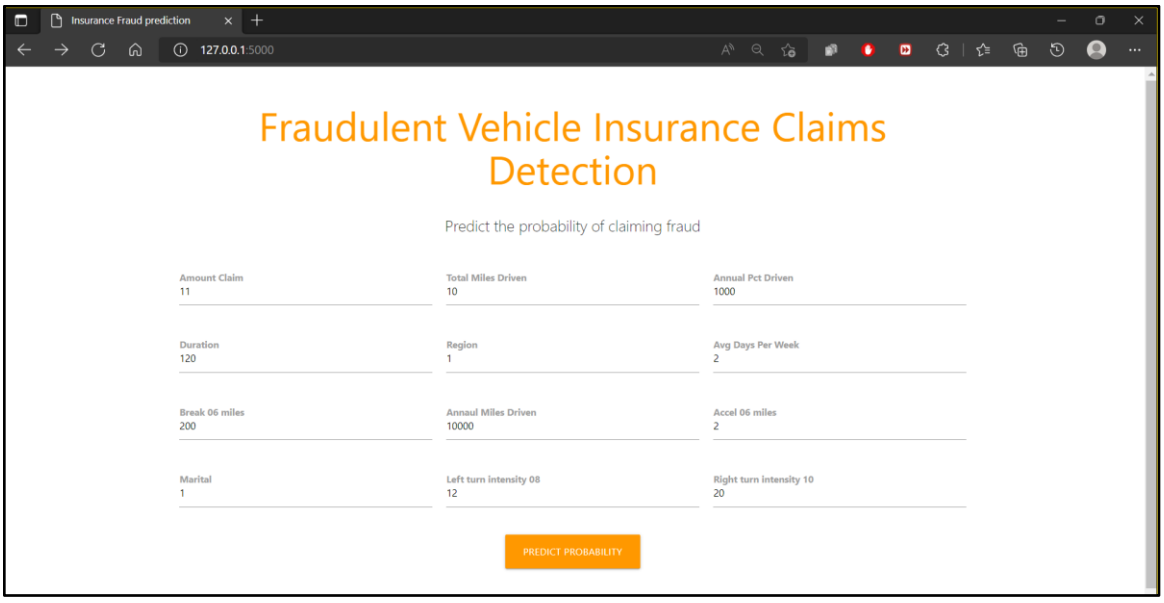


Fig 5.14: Flask App Input Screen - Large Dataset

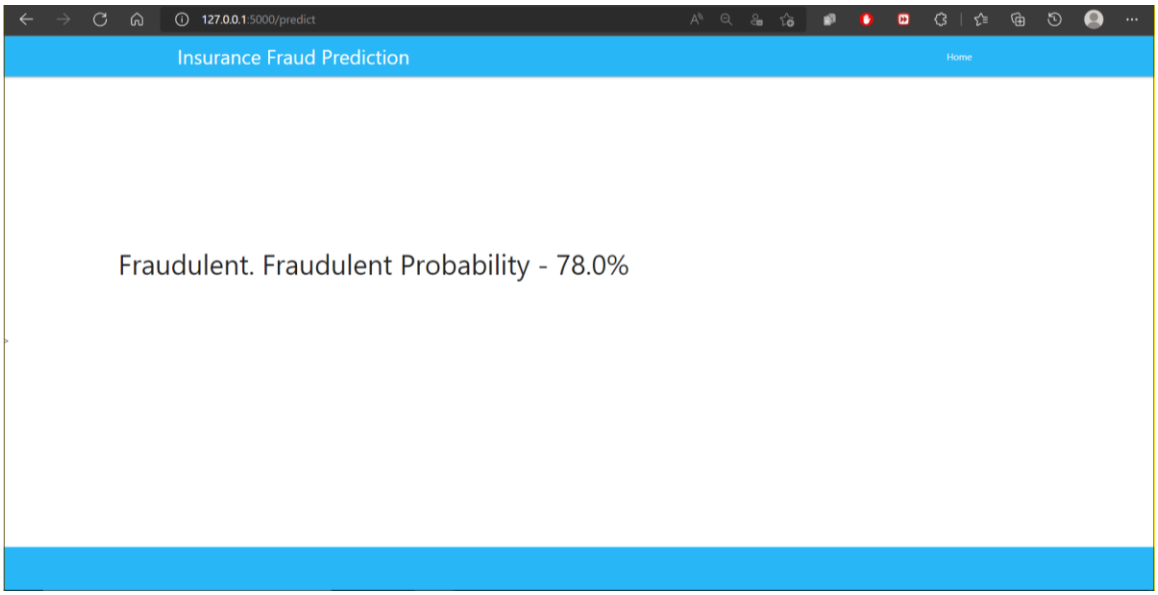


Fig 5.15: Flask App Output Screen - Fraudulent Input - Large Dataset

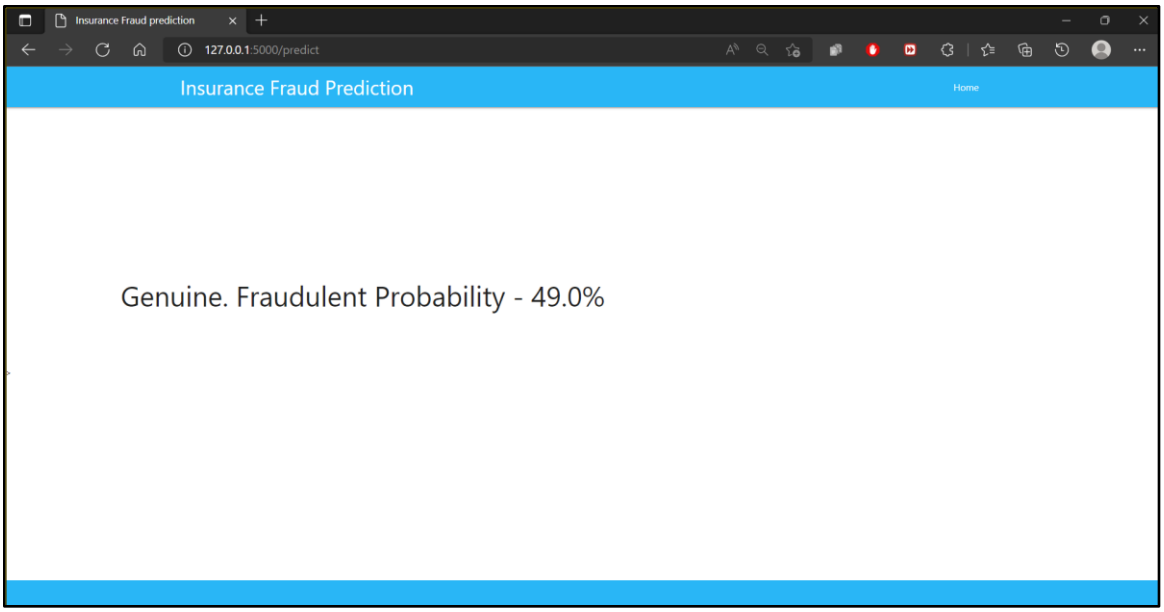


Fig 5.16: Flask App Output Screen - Genuine Input - Large Dataset

6. CONCLUSION AND FUTURE WORK

The chapter concludes the project and discusses the limitations. Further, the future scope of the project is discussed.

6.1 Conclusions

The use of prediction models in insurance fraud investigations will streamline the investigation process and facilitate quicker problem resolution. In this project, we have addressed the class imbalance problem that is usually present in fraud datasets using the SMOTE algorithm. Then, we implemented various machine learning models and a multilayer perceptron model for both the datasets.

After implementing and comparing the performance of the models, we found out that the Stacking classifier has outperformed the other classifiers. We have implemented two flask applications, one using the stacking classifier trained on the small dataset and the other using the multilayer perceptron classifier trained on the large dataset. The prediction results for the given claim inputs have been promising and based on that, we can conclude that the application can classify the fraudulent claims with a good accuracy.

6.2 Limitations

- Dataset quality plays a major role in the performance of the various machine learning and deep learning models. The accuracy of the predictive models varies when trained using different datasets. Therefore, it is imperative to train the models using datasets with desired attributes.
- Model training might become difficult if there are varying fraud patterns.

6.3 Future Work

- Since we have implemented the machine learning algorithms and the MLP model using non standard datasets, the results have varied greatly. However, by using an industry standard dataset, the models can be trained more efficiently and can be more applicable to the insurance companies.
- This project can be extended for detection of various other frauds such as credit card frauds, health insurance frauds, etc.

REFERENCES

[1]. M. Nur Prasasti, A. Dhini and E. Laoh, "Automobile Insurance Fraud Detection using Supervised Classifiers," *2020 International Workshop on Big Data and Information Security (IWBIS)*, 2020, pp. 47-52, doi: 10.1109/IWBIS50925.2020.9255426.

[2]. S. Harjai, S. K. Khatri and G. Singh, "Detecting Fraudulent Insurance Claims Using Random Forests and Synthetic Minority Oversampling Technique," 2019 4th International Conference on Information Systems and Computer Networks (ISCON), 2019, pp. 123-128, doi: 10.1109/ISCON47742.2019.9036162.

[3]. C. Muranda, A. Ali and T. Shongwe, "Detecting Fraudulent Motor Insurance Claims Using Support Vector Machines with Adaptive Synthetic Sampling Method," 2020 61st International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), 2020, pp. 1-5, doi: 10.1109/ITMS51158.2020.9259322.

[4]. R Guha, Shreya Manjunath, Kartheek Palepu, “Comparative Analysis of Machine Learning Techniques for Detecting Insurance Claims Fraud”, https://www.wipro.com/analytics/comparative-analysis-of-machine-learning-techniques-for-detectin/

[5]. S. Subudhi and S. Panigrahi, "Effect of Class Imbalanceness in Detecting Automobile Insurance Fraud," 2018 2nd International Conference on Data Science and Business Analytics (ICDSBA), 2018, pp. 528-531, doi: 10.1109/ICDSBA.2018.00104.

[6]. Kowshalya, G. & Nandhini, M.. (2018). Predicting Fraudulent Claims in Automobile Insurance. 1338-1343. 10.1109/ICICCT.2018.8473034.

[7]. N. Rai, P. K. Baruah, S. S. Mudigonda, and P. K. Kandala, “Fraud Detection Supervised Machine Learning Models for an Automobile Insurance,” International Journal of Scientific and Engineering Research (IJSER), vol. 9, no. 11, pp. 473–479, 2018.

[8]. S. Harjai, S. K. Khatri and G. Singh, "Detecting Fraudulent Insurance Claims Using Random Forests and Synthetic Minority Oversampling Technique," 2019 4th International Conference on Information Systems and Computer Networks (ISCON), 2019, pp. 123-128, doi: 10.1109/ISCON47742.2019.9036162.

[9]. Dina Elreedy and Amir F. Atiya. 2019. A Comprehensive Analysis of Synthetic Minority Oversampling Technique (SMOTE) for handling class imbalance. Inf. Sci. 505, C (Dec 2019), 32–64. https://doi.org/10.1016/j.ins.2019.07.070

[10]. B. Shah, "Auto Insurance Claims Data," Kaggle, Available: https://www.kaggle.com/buntyshah/auto-insurance-claims-data

[11]. B., Boucher, J-P, Valdez, EA, Synthetic Dataset Generation of Driver Telematics,

http://www2.math.uconn.edu/~valdez/data.html

APPENDIX

Smaller Data set

#Input Parameters for the model

```
X = dataframe[feat]
y = dataframe.fraud_reported

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1, stratify=y)

smt = SMOTE()
X_train, y_train = smt.fit_resample(X_train, y_train)
np.bincount(y_train)
```

#Training the ML models

```
from xgboost import XGBClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from numpy import mean
from numpy import std
# get a stacking ensemble of models
def get_stacking():
    # define the base models
    level0 = list()
    level0.append(('DT', DecisionTreeClassifier(max_depth=10, random_state=5)))
    level0.append(('RF', RandomForestClassifier(n_estimators=2000)))
    level0.append(('KNN', KNeighborsClassifier(5)))
```

```
    level0.append(('ADA', AdaBoostClassifier(n_estimators=500)))
    level0.append(('XGB', XGBClassifier(objective= 'binary:logistic', eval_metric='logloss')))
    level1 = LogisticRegression(solver='lbfgs', max_iter=5000)
    # define the stacking ensemble
    model = StackingClassifier(estimators=level0, final_estimator=level1, cv=10)
    return model

# get a list of models to evaluate
def get_models():
    models = dict()
    models['Decision Tree'] = DecisionTreeClassifier(max_depth=10)
    models['Random Forest'] = RandomForestClassifier(n_estimators=2000)
    models['KNN'] = KNeighborsClassifier(5)
    models['ADA'] = AdaBoostClassifier(n_estimators=500)
    models['XGBoost'] = XGBClassifier()
    models['Stacking'] = get_stacking()
    return models
# get the models to evaluate
models = get_models()

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
    return scores

# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('%s %.4f (%.4f)' % (name, mean(scores), std(scores)))
```

```
plt.boxplot(results, labels=names, showmeans=True)
plt.show()
pickle.dump(model, open("insurance_ml2.pkl", "wb"))

#Training MLP model
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from tensorflow.keras.utils import plot_model

# smaller model

def build_model():
    # create model

    model = Sequential()
    model.add(Dense(256, activation='relu',input_dim=20))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    plot_model(model, 'model.png', show_shapes=True)
    return model
```

```
estimators = []
estimators.append(('standardize', MinMaxScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=build_model, epochs=1000, batch_size=50,
verbose=1)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(pipeline, X_train, y_train, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

#**App.py**

```
from flask import Flask,request, url_for, redirect, render_template
import pickle
import pandas as pd

app = Flask(__name__)
model = pickle.load(open("insurance_dd3.pkl", "rb"))

@app.route('/')
def hello_world():
    return render_template("index.html")

@app.route('/predict',methods=['POST','GET'])
def predict():
    text1 = request.form['1']
    text2 = request.form['2']
    text3 = request.form['3']
    text4 = request.form['4']
    text5 = request.form['5']
    text6 = request.form['6']
    text7 = request.form['7']
    text8 = request.form['8']
    text9 = request.form['9']
    text10 = request.form['10']
    text11 = request.form['11']
```

```
    text12 = request.form['12']

    row_df =
pd.DataFrame([pd.Series([text1,text2,text3,text4,text5,text6,text7,text8,text9,text10,text11,text12
])])
    print(row_df)
    prediction=model.predict_proba(row_df)
    output='{0:.{1}f}'.format(prediction[0][1], 2)
    output= float(output)*100
    if str(output)>str(50):
        return render_template('result.html',pred=f'Fraudulent. The Probability of fraudelent is
{str(output)}')
    else:
        return render_template('result.html',pred=f'Genuine.\n Probability of fraudelent is
{str(output)}')

if __name__ == '__main__':
    app.run(debug=True)
```

Larger Data set

#Training MLP model

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

# smaller model
def build_model():
    # create model
```

```
    model = Sequential()
    model.add(Dense(256, input_dim=33, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # plot_model(model, 'model.png', show_shapes=True)
    return model
estimators = []
estimators.append(('standardize', MinMaxScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=build_model, epochs=10, batch_size=32,
verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=5, shuffle=True)
results = cross_val_score(pipeline, X_train, y_train, cv=kfold)
print("Accuracy: %.4f%% (%.4f%%)" % (results.mean()*100, results.std()*100))
```

#App.py

```
from flask import Flask,request, url_for, redirect, render_template
import pickle
from matplotlib.pyplot import flag
import pandas as pd

app = Flask(__name__)
model = pickle.load(open("insurance_dd2.pkl", "rb"))

@app.route('/')
def hello_world():
    return render_template("index.html")

@app.route('/predict',methods=['POST','GET'])
```

```
def predict():
    text1 = request.form['1']
    text2 = request.form['2']
    text3 = request.form['3']
    text4 = request.form['4']
    text5 = request.form['5']
    text6 = request.form['6']
    text7 = request.form['7']
    text8 = request.form['8']
    text9 = request.form['9']
    text10 = request.form['10']
    text11 = request.form['11']
    text12 = request.form['12']

    row_df =
pd.DataFrame([pd.Series([text1,text2,text3,text4,text5,text6,text7,text8,text9,text10,text11,text12
])])
    prediction=model.predict_proba(row_df)
    output='{0:.{1}f}'.format(prediction[0][1], 2)
    output= float(output)*100
    if str(output)>str(50):
        return render_template('result.html',pred=f'Fraudulent. Fraudulent Probability -
{str(output)+"%"}')
    else:
        return render_template('result.html',pred=f'Genuine.\n Fraudulent Probability -
{str(output)+"%"}')

if __name__ == '__main__':
    app.run(debug=True)
```