Intelligent System to detect software defect in Critical Applications

Submitted By

Bhaumikkumar M. Thakkar

20MCEC17

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2022

Intelligent System to detect software defect in Critical Applications

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Bhaumikkumar M. Thakkar

(20MCEC17)

Guided By

Dr. Rupal Kapdi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2022

Certificate

This is to certify that the major project entitled **"Intelligent System to detect software defect in Critical Applications"** submitted by **Bhaumikkumar M. Thakkar (20MCEC17)**, towards the partial fulfillment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached the level required for being accepted for examination. To the best of my knowledge, the results embodied in this Major Project Part-II have not been submitted to any other university or Institution for the award of any degree or diploma.

Dr. Rupal Kapdi
Internal Guide & Assistant Professor
CSE Department
Institute of Technology
Nirma University, Ahmedabad

Dr. Sudeep Tanwar
Professor & PG Coordinator (M.Tech - CSE)
CSE Department
Institute of Technology
Nirma University, Ahmedabad

Dr Madhuri Bhavsar
Professor & Head
CSE Department
Institute of Technology
Nirma University, Ahmedabad

Dr Rajesh Patel
Director
Institute of Technology
Nirma University, Ahmedabad

Statement of Originality

I, **Bhaumikkumar M. Thakkar**, **20MCEC17**, give undertaking that the Major Project entitled **"Intelligent System to detect software defect in Critical Applications"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work I carried out, and I assure you that no plagiarism has been made. It contains no previously published material or written, except where a reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere, it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Dr. Rupal Kapdi

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Dr. Rupal Kapdi**, Associate Professor, Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad, for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted have motivated me to reach a higher goal. Her guidance has triggered and nourished the intellectual maturity that I will benefit from for a long time to come.

It gives me immense pleasure to thank **Dr. Madhuri Bhavsar**, Hon'ble Head of Computer Science And Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. Rajesh Patel**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad, for the unmentionable motivation he has extended throughout this work.

I would also thank the Institution, all faculty members of the Computer Science and Engineering Department, Nirma University, Ahmedabad, for their particular attention and suggestions towards the project work.

- Bhaumikkumar M. Thakkar

20MCEC17

Abstract

Intelligent systems are sophisticated machines that can sense and react to their surroundings. These systems investigate how these technologies interact with human users in constantly changing physical and social situations. Some of the applications of intelligent systems are traffic lights, smart meters, automobiles, digital television, and many more. In spite of it's wide success, there are many software defects in these existing systems namely, system crashes, hangs, undefined behaviour. Such defects are exploited by hackers for various security attacks. Many defects are discovered and addressed by various machine learning models. Hence, the prime focus of this article is to exhaustively review various software defects, methods to compare various approaches to address the detects. The article also compares various machine learning models (tree based gradient boosting, decision tree based gradient boosting, optimized distributed gradient boosting, Gaussian Naive Bayes, Multinomial Naive Bayes and Bernoulli Naive Bayes) on five PROMISE datasets including JM1, KC1, KC2, and PC1.

Abbreviation

Abbrevations	**Explanation**
UAV	Unmanned aerial vehicle
AV	Autonomous Vehicles
TM	Telemedicine
ITS	Intelligent Transportation system
IA	Intelligent Automation
AI	Artificial Intelligence
ML	Machine Learning
V2V	Vehicle to vehicle
V2X	Vehicle to any
OBD	On-board diagnostics
Wifi	Wireless Fidelity
CACC	Cooperative Adaptive Cruise Control
NB-IoT	Narrowband Internet of things

Table 1: List of Abbreviations

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This chapter discusses the critical applications and how it plays roles in the real world. This chapter discusses in brief critical applications, Automation, and bugs.

1.1 Critical applications

Critical applications depend highly on hardware, software, network, and environmental conditions. A critical system is one whose failure might jeopardize human lives and the environment. Various critical applications are aircraft flight control, nuclear systems, surveillance, telemedicine, voting system, and robotics.

1.2 Types of critical applications

These applications are classified into safety-critical, mission-critical, and business-critical applications. Safety-critical applications include fire alarms, fire sprinklers, electricity generation, transmission and distribution, and circuit breakers. Navigating systems, nuclear reactors, Railway/aircraft operating and control systems, electric power grid systems, and much more are examples of mission-critical applications [1]. Business-critical applications contain cloud-based data storage, networking systems, online banking systems, and more [1]. Any failure in these applications can cause trouble to humans and the environment in any form of software hardware and communication.

1.3 Failure scenarios of critical applications

Recent incidents have taken the place of the failure of critical applications due to software and hardware failure. According to the facts, from 2013 to 2021, many incidents

have happened with UAVs (Unmanned Aerial Vehicles). For example, in 2021, while approaching Buttonville Municipal Airport, a Cessna 172 of Canadian Flyers International Inc., registered as C-GKWL, collided with a drone operated by the York Regional Police. The Cessna landed safely, but with a bent air-box, broken engine cowling, and a propeller strike, it was substantially damaged. [2] [3]. It was a hardware failure incident. Some incidents regarding self-driving cars/autonomous vehicles occurred from 2018 to 2021. In 2021, Tesla's model Y went on the wrong lane in FSD (Full self-driving) mode and hit another driver in California [4]. The software failure occurred because some bug occurred at FSD mode and went rogue. The primary cause of failure in such critical applications is software, hardware, and security attacks.

1.4 Driving Force of critical applications

Software is the backbone of any critical applications controlled remotely based on the equation and conditions. Communication is via an open channel, the internet so that an attacker can compromise the data of any critical applications. *Hardware* is the microcontroller over which the software can run to complete the specific task with proper safety and security. Once the hardware has been designed, it is hard-bounded. We can install it into our critical applications, but the significant role is that the software needs to be updated regularly for security purposes. In every update, more security features have been added. Hence, the software must be up to date so that no one can attack systems. In addition, software needs to be adequately designed to be tested many times before deploying in critical applications.

1.5 Reasons for software failure of critical application

Software integration fails, glitches in software, IEC (Integrated electrical components) fails because of corrosion, heat, or other circumstances. The software can have bugs/exceptions, such as the wiring causing integrated electrical components to fail. The software is not integrating correctly with the hardware and can lead to the removal of essential safety features [5].

1.6 Software defects after deployment

It is crucial to track these things that can offer security to critical applications. Identifying software defects is very important in the initial phase. If there are no bugs in the preliminary stages, the system must be tested; once it has been upgraded. However, we might not include these in a critical application because many issues occur at the beginning phase of a critical application. Everyone knows its use for safety so that the attacker can breach the system. The software needs to apply security to every update of standard applications; however, no survey covers all such instances. The developer must test, develop, and integrate the best security mechanisms to critical applications several times. It has to be completed prior to delivery. Many software defects occur in every critical application to identify, and we need software defect solutions. If a critical application problem arises by chance, we may resolve it until then. Many researchers have to give solutions concerning specific tasks; however, none has made software bug identification related to critical applications. So motivated by this paper, we consolidate all the works at a single platform by presenting a comprehensive survey on Intelligent systems to detect software defects in critical applications.

Chapter 2

Scope of the Survey

The scope of survey contains detailed survey of software defect prediction [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. Here are a few of them: Dam et al. [6] created a unique prediction model that automatically learns attributes for describing source code and uses them to predict the defect. This model was created using tree-structured Long short-term memory (LSTM), a powerful deep learning approach that fits the abstract syntax tree structure of source code. The datasets used were from Samsung open-source projects and the PROMISE dataset.

Tong et al. [7] proposed an approach which is SDAEsTSE, combination of two approaches which are Stacked denoising autoencoders (SDAEs), which are powerful future learning, and ensemble learning, a two-staged ensemble (TSE). Initially, SDAEs were used to extract DPs (Deep Represenation) from the traditional software metrics, and TSE addresses the class imbalance problem. This model is used in 12 datasets of the National aeronautics and space administration (NASA) to demonstrate the effectiveness of Deep representation (DPs).

Qiao et al. [11] proposed an approach of deep learning techniques that predicts the number of defects in the software. A custom-built deep neural network model is used to organize the input data for the deep learning model, conduct data modelling, and forecast the number of faults. The publicly accessible dataset is first preprocessed, which includes log transformation and data normalization.

Wang et al. [13] suggested a defective approach that relies on Gated hierarchical long short-term memory networks (GH-LSTM), which uses hierarchical LSTM networks to extract semantic features from word embeddings of source code files. Hanon Tong et al. [16] proposed Subspace hybrid sampling ensemble (SHSE). This method combines hybrid sampling, ensemble learning, and subspace construction to build high-performance Software defect number prediction (SDNP) models and uses 27 types of different defect datasets which are publically available.

Author	Year	Objective	Technique	Cons
Dam et al. [6]	2018	A model that can acquire attributes for defining source code and use them to identify defects automatically	LSTM	Can be used with other web and mobile apps, as well as programming languages like C++ and PHP.
Tong et al. [7]	2018	SDAEsTSE is a new SDP(Software defect prediction) strategy that combines SDAEs with ensemble learning, precisely the suggested TSE method	SDAEsTSE	Cross-project defect prediction and class imbalance
Liang et al. [8]	2019	For fault prediction, it combines similar representation of words and deep learning approaches	LSTM	Extend Seml to forecast defects at a finer level.
Li et al. [10]	2020	Examine the application and efficiency of unsupervised learning approaches in the estimation of software defects	Unsupervised learning	It is uncertain which qualities are needed.
Qiao et al. [11]	2020	To estimate the amount of faults in software systems	Deep learning	To investigate the number of projected faults in software modules by including more projects created in other programming languages, and also big projects from industry.
Alsawalqah et al. [12]	2020	To develop expert and robust heterogeneous classification models	Heterogeneous Ensemble Classification	Explore more advanced clustering algorithms and investigate techniques that can automatically determine the best number of clusters for each dataset
Wang et al. [13]	2021	To extract semantic features from word embeddings of abstract syntax trees (ASTs) of source code files	GH-LSTM	Examine cross-project prediction performance and experiment with various languages such as C and C++.
Feng et al. [14]	2021	A unique oversampling approach that may concurrently achieve low pf and high pd	Oversampling Technique	Apply COSTE to various software datasets using a range of parameters, classifiers, and performance evaluations to evaluate the technique's generalizability.
Tong et al. [16]	2022	Introduces a new subspace hybrid sampling ensemble (SHSE) approach for creating high-performance SDNP models based on feature substructure creation, hybrid sampling, and ensemble learning.	Subspace hybrid sampling ensemble	To see if deep representations may assist enhance the performance of SDNP models by combining deep learning with software defect number prediction.

Table 2.1: Scope of the survey

Chapter 3

Critical application faults

This section describes faults in critical applications.

3.1 Autonomous Vehicles

An autonomous vehicle is a system that can operate independently or with little assistance from humans. Tesla, uber, autox, cruise, waymo, voyage, swift navigation embark trucks, zoox, and many other AVs have been launched in recent years [17]. Before, after, and during the deployment, culpability was created. These vulnerabilities result in unfortunate events that were not expected. In autonomous vehicles, many distinctive types of flaws can be generated. Many researchers have unique discoveries related to AVs. [18] [19], [20], [21] demonstrate the faults that might be generated in the sensors in their study. Hard-over, erratic, stuck, spike, and drift defects are among the issues. These are the five most common problems that autonomous vehicles may encounter. The definition for the faults are as follows:

1. **Hard-over fault**: Hard-over faults are simple to identify since they have unique features, peculiar qualities. For a very short instance of time, sensor value reaches the saturation point. No conventional driving strategy should generate this sensor output for sensors with a signal gradient approaching infinity. This fault leads to a complete depletion in the safety of a vehicle and causes crucial damage.

2. **Erratic fault**: Value of sensor fluctuates around the true value, and the amount of the variance may increase with time. AVs detect the object to be operated on. An erratic fault arises when the value is close to its true value, at that time the vehicle

hits or avoids the item, causing damage to public property and human life.

3. **Stuck fault**:The value of the sensor remains constant for a short period of time. As one specific aspect of stuck faults in the absence of noise dispenses following fault beginning, stuck faults indicate lower accuracy. In AVs, if a stuck fault materializes, the steering or the breaking malfunction for a small amount of time. At that point, an unexpected event may occur, along with a car operating in the wrong lane or within the traffic centre; it begins and loses control. Because the value of the fault rises to the top in that little amount of time, it endangers human life and different vehicles.

4. **Spike fault**: For a single point, the sensor value is much higher than the actual value, and the number of spike faults within the signal might arise with time. When the vehicle's user enters its destination to the navigation, and after putting the vehicle in Full self-driving (FSD) mode, the spike fault may change location. It's possible that the position will change in very short period. So, the user is unable to reach its destination.

5. **Drift fault**: Over time, the actual value of sensor deteriorates linearly. When drift errors initially arise, they might be hard to spot since they can look like typical sensor behaviour. However, the drift problem may not be severe enough to compromise vehicle safety remarkably. For example, AVs may have a problem starting the radio or playing songs if a drift fault happens. It may imitate normal conduct, as described. As a result, identifying the problem is challenging, and it may not be a severe sensor problem.

These are the main faults generated in autonomous vehicles. It includes the sensors, actuators, process components, and electrical components. Furthermore, classifying the component where the problem occurs might help differentiate faults.

3.2 Unmanned Aerial Vehicle

3.2.1 Background

A remote-controlled or self-contained vehicle is known as a Unnamed Aerial Vehicle (UAV). UAVs have gained popularity, commonly known as drones. They are employed

Applications	Objective
Smart agriculture [23]	Crop surveillance in the form of photos or video feeds
Smart healthcare [24]	To comprehend a patient's physiological state remotely, generate, monitor, and analyse health data utilising smart wearable devices.
Traffic monitoring [25]	Monitoring and analysis of real-time traffic data on a regular basis
Social Distancing [26]	The monitoring of data distance between persons over a chosen region or location.
Smart Parking [27]	Using unmanned aerial vehicles (UAVs) to investigate parking lots
Construction Project [28]	To keep track of the progress of construction on high-rise structures.
Disaster Management [29]	Drones can be used to identify and convey actions that occur before and after a disaster.

Table 3.1: Application and their Objectives in UAVs

in a variety of military and civilian settings. By 2028, the market for UAVs in the military is estimated to reach $26.11 billion [22]. According to numerous studies, the use of UAVs for non-military purposes may eventually exceed their usage for military purposes, perhaps reducing the need for future conflicts. A UAV can be controlled using one of two methods: self-control or ground control channel. The increased research and development in UAVs have enhanced their usability capabilities over the last few years. As technology evolves, there are several security solutions for UAV communication. However, most of these solutions are only ideas or are in the early stages of development.

3.2.2 Applications of UAVs

This section explores a variety of scenarios in which UAVs may be used to collect essential data, which can then be utilized for more meaningful analysis and intelligent real-time decision-making. Table 3.1 summarises several real-time UAV application scenarios, including healthcare, volcano monitoring, and agriculture.

3.2.3 Faults

Actuator jams, airframe damage, communication failure, accidents, and environmental influences are all possible reasons of UAV failure. These factors, which are regarded as common defects in UAV formations, may occur at the same time or trigger the occurrence of the others. The description of the specific faults is as follows [30]:

1. **UAVs component failure**: It conveys the failure of sensors, actuators, control systems, power systems, flight computers, and other components. If the UAV's power and control system breaks down, it will crash, causing significant damage. This fault may lead to a UAV crash that hurts public property and human lives.

2. **Air-frame damage**: This damage can happen due to UAVs' collision with other obstacles. When UAVs are on surveillance, if some obstacle accidentally hits the UAV, they may destroy other living or non-living things.

3. **Communication failure**: It is possible that information transmission and reception to others will be disrupted due to a fault. Hence, there might be permanent or temporary loss with connected users.

4. **Formation Collision**: This fault may harm other neighboring aircraft because if the UAV changes its behavior, such as the address to the wrong navigation or quicken or change in speed, that is an abnormal thing, and it leads to some unpredicted occurrence.

5. **Environmental Impact**: Climate change or other environmental effects cause the reduction in the performance of UAV sensors, actuators, and the communication system. If lightning strikes the UAVs, it may damage some components of the system or the whole UAVs.

There are a few other sensor failure types such as stuck with constant bias, drift or additive-type, and multiplicative-type [31].

- Stuck with constant bias: This failure can cause maximum communication or electrical problems in UAVs because the sensor is stuck at one position and remains constant until the output.

- Drift or additive-type: This is a typical problem with analog sensors. Due to the heat up/down, the internal temperature causes serious damage to the sensor.

- Multiplicative-type: A scaling issue in the sensor output causes this failure category.

3.3 Intelligent Transportation System

The faults are of Intelligent transportation systems such as high-speed trains as follows: IGBT, inter-turn, gear, gearbox, sensor, operation, motor gap, and braking faults. The discussion of some faults is as follows [32]:

1. **IGBT fault**: IGBT stands for Insulated gate bipolar transistor. This fault is the main root of electric faults in motor drives and inverters. It can cause in every transportation system because it may endanger human lives.

2. **Inter-turn fault**: As the name indicates, an inter-turn fault occurs when a fault takes place between two adjacent twists in a machine winding. It might be the result of insulation failure in the windings. If not maintained effectively, these inaccuracies can lead to inter-phase failures, phase to ground difficulties, and other serious issues. [33].

3. **Sensor fault**: Faults can occur in some sensors, such as the current sensor, speed sensor, temperature sensor, and voltage sensor. If that happens in voltage and current sensor fault, it can lead to a short-circuit of the electrical transportation system. For example, a speed sensor failure might cause the transportation system to lose control, putting human life in danger.

4. **Motor fault**: If this fault occurs, the motor may overheat or overload, preventing it from being used for other purposes. Some of the causes are voltage imbalance, operational overloads, shaft imbalance, shaft looseness, etc.

5. **Braking fault**: This fault is produced by leaking fluid in various transportation systems, which indicates that the breaks are not entirely functional. Hence this might be dangerous to human lives.

Chapter 4

Tools and Techniques to detect software defects

This section contains the taxonomy of tools and techniques to detect software defects in each application. Techniques include statistical techniques, supervised learning, semi-supervised learning, and unsupervised learning algorithms in Figure 4.1. The description of specific techniques and algorithms are as follows:

4.1 Statistical techniques

Naive Bayes classifier, linear regression, Bayesian networks, discriminant analysis, k-nearest neighbour, logistic regression and correlation analysis are some of the statistical approaches used. Here's a more detailed explanation of statistical methodology:

4.1.1 Linear regression

One or more variables appear to have a linear relationship using linear regression. Simple and multiple linear regression are the two formations of linear regression [34]

4.1.2 Logistic regression

A logistic function is used to express a binary variable which is dependent in the simplest form in logistic regression. There are, however, several more sophisticated variations. [35].

4.1.3 Naive bayes

It is a classification approach based on Bayes' Theorem and the hypothesis of predictor independence. In other words, the presence of one value in a class has no influence on the presence of succeeding values, according to a Naive Bayes classifier. [36].

4.1.4 Classifier

An algorithm that translates input data to a specified category is a classifier.

4.1.5 K-nearest neighbours

The k-Nearest-Neighbors (kNN) classification method is a non-parametric classification approach that is basic but successful in many situations. The kNN algorithm is skewed due to k. There are some ways to figure out the k value, but one of the easiest is to run the algorithm multiple times with different k values and choose the one that works best. [37].

4.1.6 Bayesian networks

A Bayesian Network is a pictorial representation of the numerous probabilistic interactions between random variables in a collection of data [38].

4.1.7 Discriminant analysis

As a pre-processing step for machine learning and pattern classification applications, Discriminant Analysis is a reasonably prevalent approach for dimensionality reduction challenges. Simultaneously, it is widely utilized as a black box, although it is (sometimes) poorly understood [39].

4.1.8 Correlation analysis

The approach of correlation analysis is extensively utilised for finding hidden patterns in data. These connections help us comprehend the importance of attributes with the expected target class [40].

4.2 Supervised learning algorithms

In supervised learning, training data is given to utilities that function as supervisors, emphasizing that the equipment properly forecast the output. In addition, the labeled

information indicates that some input data has already been marked with the relevant result.

4.2.1 Ensemble Based

Ensemble techniques are a machine learning approach that connects multiple base models to create a single optimal predictive model [41].

1. Random forest: Random Forest Models are similar to BAGGing but with a few differences. Based on randomly selected elements, Random Forest models decide where to split.Because each tree splits depending on diverse qualities rather than separating at comparable attributes throughout each node, Random Forest models give a level of differentiation.

2. Bagging Boosting: The word BAGGing stems from combining Bootstrapping and Aggregation into a single ensemble model. Several bootstrapped subsamples are created from a collection of data. Each of the bootstrap sub-samples is allotted its own tree.

4.2.2 ML Based

Machine Learning-based algorithms for fault detections are OneR, CART-LS, C4.5, Decision Tree, classification tree, and J48.

4.2.3 Fuzzy Techniques

According to the probability theorem, a model matches the link between inputs and outcomes. It can deal with situations where there are uncertainties, such as challenges requiring human perception and experience [42]. The essential component of a fuzzy logic decision-making system is the Fuzzy Inference System. The "IF...THEN" controls and connectors "OR" or "AND" are used to create basic decision rules. Fuzzy regression coefficients of model genetic algorithm are applied.

4.2.4 Instance Based

Instance-based learning systems memorize the training examples and then generalise to new cases using some similarity metric. It has called instance-based since the hypotheses are built from the training data. It is also known as sluggish learning or memory-based learning.

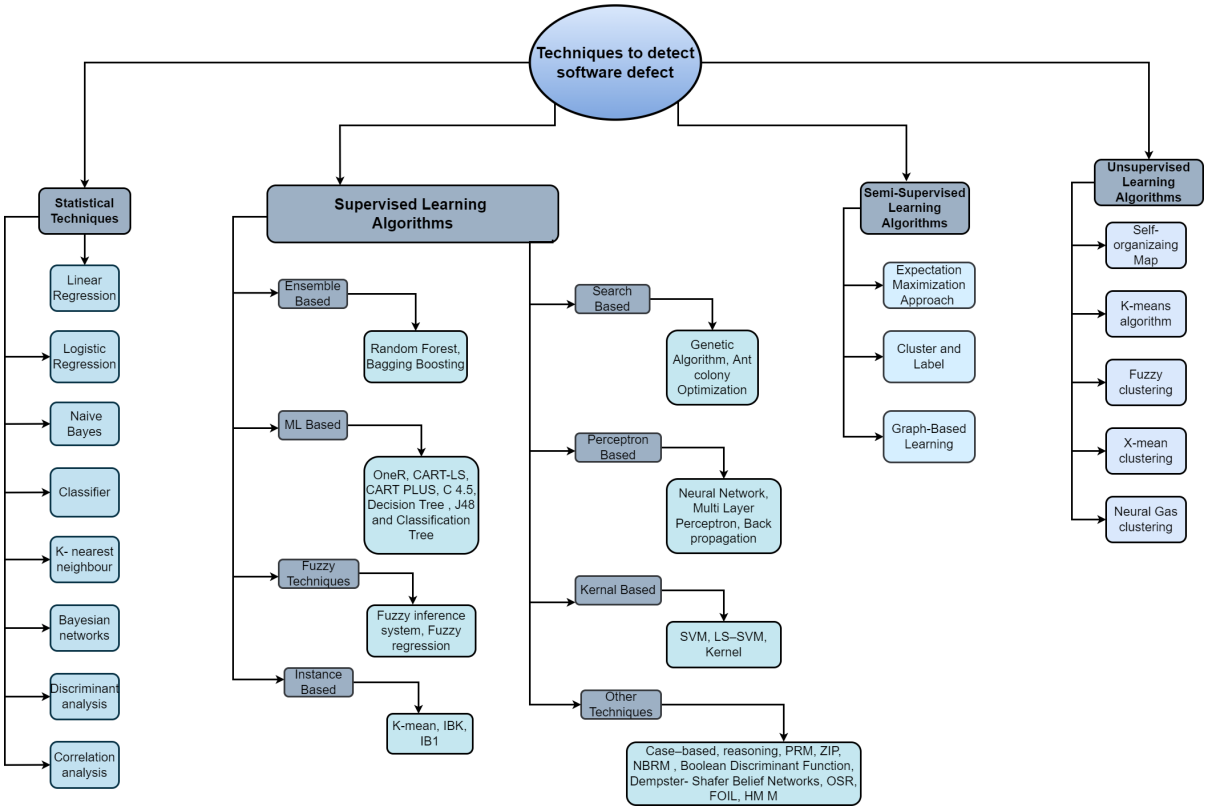

Figure 4.1: Techniques of software defect prediction

4.2.5 Search Based algorithm

1. Genetic Algorithm: Genetic Algorithm (GAs) are adaptive heuristic search algorithms that categorize the evolutionary algorithm. Genetic algorithms are established on the basis of selecting naturally and biological genetics. These are innovative uses of random search, supported by prior data, to guide the search to a solution space area with superior performance. They're usually used to suggest high-quality solutions for optimisation and search strategies.

2. Ant Colony Optimization: An ant's foraging activity when exploring for a passage between their colony and a food source is the basis for the Ant Colony Optimization approach. It is used to solve the well-known dilemma of the traveling salesman. It is then utilized to solve a variety of complex optimization problems.

4.2.6 Perceptron based model

A perceptron model is a guided learning strategy using binary classifiers in Machine Learning. The single-neuron perceptron model determines whether a function is an input or not and categorizes it accordingly.

1. Neural Network: By replicating the function of the human brain, a neural network is a group of algorithms that seeks to find hidden correlations in a block of data.In this respect, neural networks refer to biological or artificial systems of neurons.

2. Multi-Layer Perceptron: The most complicated artificial neural network architecture is the multi-layer perceptron. It is mainly made up of numerous layers of the perceptron. To fully understand what a multi-layer perceptron is, we must build one from scratch using Numpy.

3. Backpropagation: Backpropagation is one of the most crucial ideas in a neural network. We need to change the parameter and bias weights for this; To optimise the constant in the linear regression model, we utilize gradient descent. Backpropagation is also used to deploy the gradient descent technique.Our main aim is to classify our data as precisely as possible.

4.2.7 Kernel Based

The SVM is referred to as a "kernel" because it employs a set of mathematical functions to provide a window through which data may be changed. A kernel method accepts data and converts it into the format required for processing. As a consequence, the Kernel Function modifies the training data in order to convert a non-linear decision surface into a linear equation in a larger number of dimension spaces.

1. SVM: The SVM is a supervised learning approach used for regression and classification. The fundamental notion is that the algorithm tries to discover the best hyperplane based on the labeled data (training data) to categorize fresh data points. The hyperplane is a simple line in two dimensions.

2. LS-SVM: Least-squares support-vector machines (LS-SVM) are least-squares variations of support-vector machines (SVM), a series of related supervised learning algorithms for data analysis and pattern recognition for regression and classification problems.

4.2.8 Other Techniques

There are different machine learning techniques for fault detections in critical applications: Case-based, Reasoning, PRM, ZIP, NBRM, and Boolean [43].

4.3 Un-supervised learning algorithms

Ml algorithms are often used in unsupervised learning to analyse and categorize unlabeled data without any human input. The expectation-maximization technique, the cluster and label strategy, and the graph-based learning approach are all included in this algorithm. [43]. Furthermore, these algorithms reveal previously unknown patterns or data clusters. [44].

4.4 Semi-supervised learning algorithms

Semi-supervised learning is a machine learning technique that combines training with both labelled and unlabeled data. A self-organizing map, fuzzy clustering, x-mean clustering, k-means algorithm, and neural gas clustering are examples of semi-supervised learning methods. [43].

Chapter 5

Architecture

This section narrates the proposed software defect detection architecture in critical applications. First, we will understand all the layers that are presented in our architecture. The layers are critical applications, driving force, Machine learning (ML)/Deep Learning(DL) layer, and alert message in Figure 5.1.

5.1 Critical applications layer

This layer merely pivots on every critical application in the world. Here, critical applications such as CA_1 (Autonomous vehicles), CA_2 (Unmanned aerial vehicles), CA_3 (Telemedicine), CA_4 (Intelligent transportation system) and many more. $\{CA_1, CA_2, CA_3, CA_4, ..., CA_X\} \in CA_{all}$. Any $CA_a \in CA_{all}$, wherein 'a' is just any whole number, may generate a large quantity of data with a wide range of attributes like variety, volume, and velocity. The data engenders via CAs software components which are very much capable of producing a huge amount of data in any location and from each CAs. For example, the data from CA_1 (Autonomous vehicles) is generated via its sensor, actuators, gyroscope, camera, radar, lidar, and many more components. However, this layer is primarily responsible for data created from any given environment, as required by the application.

5.2 Driving force layer

Each $CA_a \in CA_{all}$ are made up of software and hardware components. Each hardware and software component can generate the data. Here, in this layer, we are entirely focused on the software components because we have to find out a defect in software

components. $\{CA_1, CA_2, CA_3, CA_4, ..., CA_n\}$ are the set of critical applications. From the software data of these CAs, they can generate the faults from sensors, actuators, gyroscope, camera, radar, lidar, storage, communication, runtime environment, and many more software components.

5.3 ML / DL layer

This layer distinguishes whether the software is defected or non-defected by using the unusual circumstance. Furthermore, in this Figure 5.2, the data generated from the above layers is transferred securely in this layer. How this layer works has been discussed further below:

5.3.1 Dataset

There are many datasets regarding software defect classification available on different sites. The datasets such as NASA's [45],PROMISE repository [46], AEEEM [47], Eclipse [48], and many more. These datasets are extensively used in many research projects for software defect classification.

5.3.2 Data preprocessing

Data preprocessing is requisite because it ensures correctness, consistency, punctuality, believability, and interpretation. For doing data preprocessing, there are a few tasks to do which are vital. The tasks are data cleaning, integration, reduction, and transformation.

- **Data cleaning**: Data cleaning is the process of eliminating erroneous, incomplete, or incorrect data from databases as well as restoring missing information.
- **Data integration**: Data integration, or the combining of disparate databases into a single dataset, is one of the most important parts of data management.
- **Data reduction**: This strategy allows for a reduction in data amount, which makes analysis easier while maintaining the same or almost comparable findings. It also saves storage space.
- **Data transformation**: The process of changing the format or arrangement of data is referred to as data transformation. This method might be simple or complex,

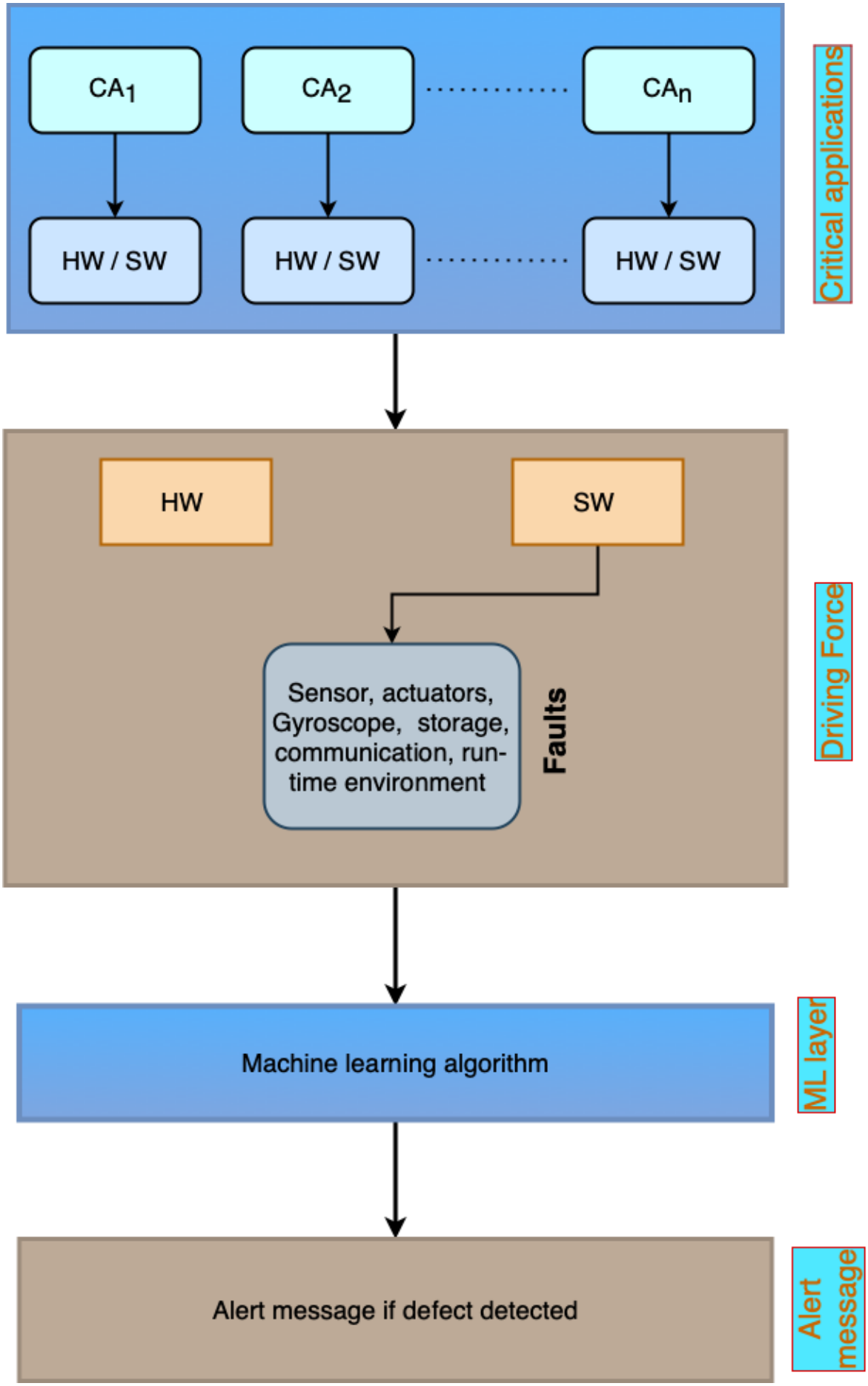

Figure 5.1: The proposed layered architecture

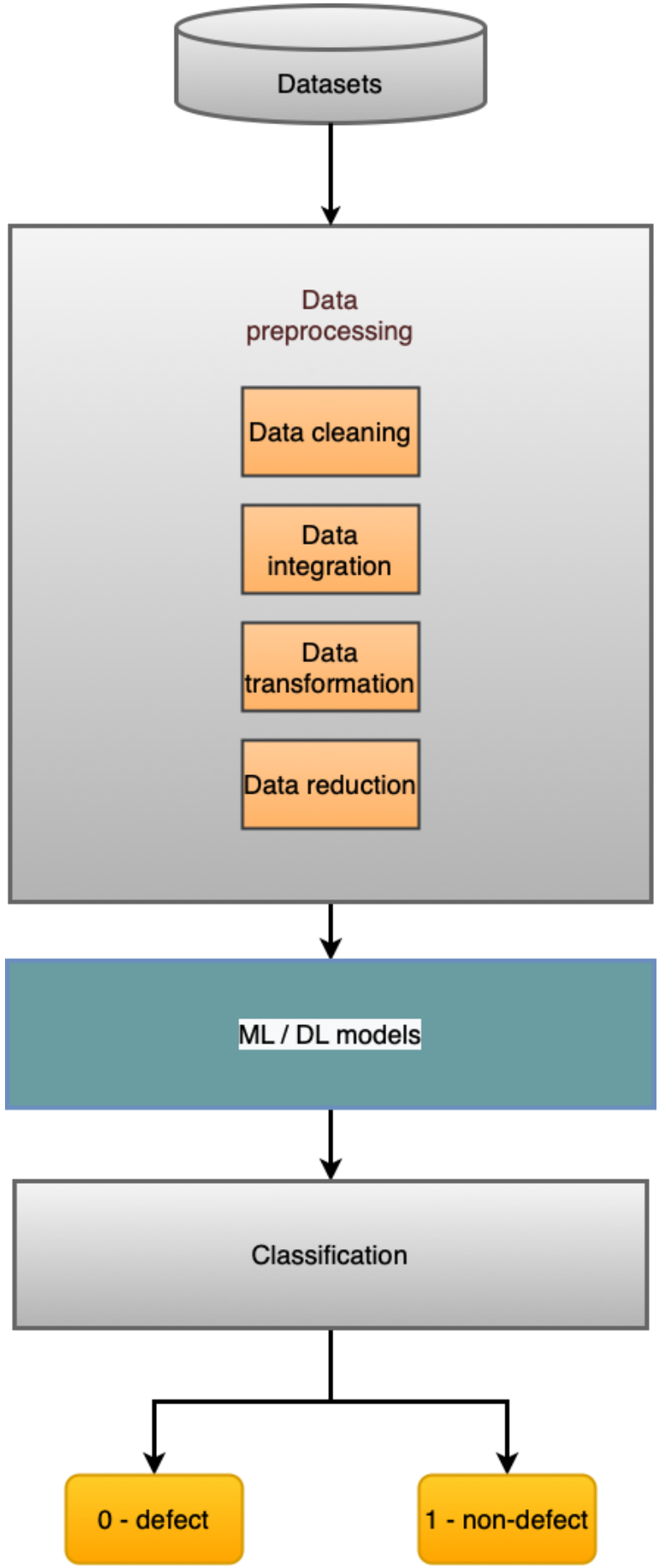

Figure 5.2: The ML layered architecture

depending on the demands.

5.3.3 Training models

Here, many models have been used for defect detection. ML models such as K nearest neighbor, Random forest, Decision tree, Naive Bayes, etc. DL models include LSTM, RNN, CNN, TCNN, DNN, GH-LSTM, etc.

5.3.4 Classification

After implementing ML and DL models to a certain dataset, it classifies into 0 and 1 labels. 0 for non-defective data and 1 for defective data. Each label contains metrics such as precision, recall, F1-score, and accuracy. For example, if accuracy contains 95-97%, the dataset is likely to be defective.

5.4 Alert message

Every critical application must be connected to some device that shows this CA has some faults and get a notification to stop the operation or repair the fault as soon as possible.

It is the proposed architecture for detecting software defects within any critical application.

Chapter 6

Analysis

In this analysis, we have used four datasets from the PROMISE repository, NASA's dataset, namely JM1, KC1, KC2, and PC1. These are the software defect classification datasets [46]. For fault categorization, we use the publicly available PROMISE data set to encourage repeatable, verifiable, refutable, and improveable software engineering prediction models. This dataset is made of various modules and repositories that meet our needs. JM1, KC1, KC2, and PC1 are the names of the four repositories. The dataset's description is as follows:

A real-time predictive ground system is written in "C" for the JM1. The KC1 is a "C++" system that receives and analyses data from the ground using storage management. KC2 is the result of using C++ functions. Data processing for scientific purposes; is part of the same project as KC1; not the same individuals as KC1. KC1 was given access to some third-party software libraries but no other applications. PC1 stores data from C functions used in-flight software for an earth-orbiting satellite [49]. These datasets contain the metrics to find out defects in the datasets. The types of metrics are McCabe, Halstead, Line count, operator/operand, and Branch.

In this table, each attribute contains its type and definition:

Each attribute contains its values from the function of the programming language C, C++, JAVA, and many more.

- Mccabe [50] : The McCabe metrics are a set of four software measures: essential complexity (ev(g)), cyclomatic complexity (v(g)), design complexity (iv(g)), and Line of code (LOC)

  .

Metric	Definition	Metric type
loc	Line Of Code	Mccabe [50]
v(g)	Cyclomatic Complexity	
ev(g)	Essential Complexity	
iv(g)	Design Complexity	
n	Length	Halstead [51]
v	Volume	
l	Level	
d	Difficulty	
i	Intelligent	
e	Effort	
b	Error Estimator	
t	Programming Time	
lOCode	Line Count	Line Count
lOComment	Count of Line of Comment	
lOBlank	Count of Blank lines	
lOCodeAnd Comment	Line of Code and Comment	
uniq_Op	Unique Operators	Operators & Operands
uniq_Opnd	Unique Operands	
total_Op	Total Operators	
total_Opnd	Total Operands	
branchCount	Total Branch Count	Branch

The number of linearly independent paths across a flow graph is measured by Cyclomatic Complexity, which is determined as follows:

$$\nu\left(G\right) = e - \upsilon + 2$$

where e = edge of the graph & v = vertices of the graph.

The amount to which a flowgraph can be "reduced" by dissecting all sub-flowgraphs of G is called essential complexity:

$$e(\nu\left(G\right)) = \nu\left(G\right) - m$$

Design Complexity, or

$$i(\nu\left(G\right))$$

, is the cyclomatic complexity of a module's reduced flowgraph. The module's flowgraph is reduced to eliminate any complexity that does not influence the interrelationship between design modules.

- Halstead [51]: The Halstead Measurements are another popular set of software metrics. They're named after the man who invented them, Maurice H. Halstead. Halstead believed that software (or the writing of software) might be linked to the themes discussed in the psychological literature at the time. He devised several measures to encompass these characteristics; these metrics may be retrieved using the McCabe IQ tool and are explained in more depth below [52]. The Halstead metrics are n(length), v(volume), l(level), d(difficulty), i(intelligent), e (effort), b(error estimate), and t (programming time).

  Using some of the values, we can find the other attributes of Halstead metrics. The values are as follows:

  n1 = number of unique operators
  n2 = number of unique operands
  N1 = total number of operators
  N2 = total number of operands
  n1* = potential operator count
  n2* = potential operand count

  With the possible exception of the future operator/operand counts, these six metrics are self-explanatory. First, Halstead defined the minimal number of operators and operands that a module can contain. This minimal number would occur in a (perhaps fictitious) language when the needed operation existed as a subroutine, function, or procedure. Because each function must include at least two operators: one for the function name and one for the assignment or grouping symbol, n1* = 2. n2* is the maximum number of arguments that must be passed to the function or method without being repeated.

  By using these measurements length of program calculate by :

  $$N = N1 + N2$$

  total number of unique operators and operands is:

$$n = n1 + n2$$

The volume of program is :

$$V = N * \ln n$$

Potential volume of program is :

$$V^* = (2 + n2^*) \ln (2 + n2^*)$$

Level of program is:

$$L = V^*/V$$

Difficult of program is:

$$D = 1/L$$

Estimator of L is:

$$\hat{L} = 1/D = 2/n1 * n2/N2$$

Intelligent content of a program is :

$$I = \hat{L} * V$$

Effort required to generate program is :

$$E = V/L$$

Programming time required by program P is:

$$T = E/18 seconds$$

Various models have been used to detect software defects, such as lightgbm, catboost, xgboost, Gaussian naive Bayes, muti-nomial naive Bayes, and Bernoulli naive Bayes. By using the dataset, we are training the model.

- Lightgbm: LightGBM outperforms XGBoost in various ways, including sparse optimization, simultaneous training, various loss functions, regularisation, bagging, and early stopping. The structure of trees distinguishes the two. Unlike most other implementations, LightGBM does not construct a tree level by level and row by row. Rather, it starts from the ground up, leaf by leaf, to form trees. It selects the leaf that it feels would reduce the greatest loss. Furthermore, unlike XGBoost and other implementations, LightGBM does not use the popular sorted-based decision tree learning technique, which looks for the optimal breakpoint among sorted attribute values. Instead, it develops trees leaf by leaf from the ground up. Exclusive Feature Bundling (EFB) and Gradient-Based One-Side Sampling are two unique tactics used by the LightGBM method to run quicker while retaining high accuracy (GOSS). LightGBM, on the other hand, makes use of a highly optimized histogram-based decision tree learning method that maximizes efficiency and memory consumption [53].

- Catboost:Yandex created CatBoost, an open-source software package[54]. It introduces a gradient boosting framework that, instead of using the traditional approach, intends to address Categorical features using a permutation-driven option. Native handling is available for categorical characteristics. Hurried GPU training, visualizations, and tools for model and feature analysis, Oblivious Trees or Symmetric Trees for faster execution, and Ordered Boosting to elude over-fitting. The project aims to develop a "Scalable, Portable, and Distributed Gradient Boosting Library" [54].

- Xgboost: XGBoost is an open-source software package that imparts a regularizing gradient boosting framework. It can encourage you to predict any type of data if you have formerly forecasted data. Any type of data can be classified. Tree boosting is a well-known and effective machine learning method. Data scientists use XGBoost, a scalable end-to-end tree boosting algorithm, to get cutting-edge results on a diversity of machine learning problems [55].

- Gaussian naïve Bayes: It is standard to assume that the constant values associated with each class follow a standard (or Gaussian) distribution when working with continuous data. We can not use Gaussian naive Bayes that portray discrete count. Gaussian naive Bayes is used where the values of features are in continuous nature. Mean, standard deviation, and variance is present in this formula of Gaussian naive Bayes [56].

- Multinomial naïve Bayes: Multinomial is Natural Language Processing (NLP), and Naive Bayes algorithm is a probabilistic learning method [57]. The Bayes theorem is predicted using this approach. It assesses each label's feasibility for a given sample and outputs the label with the most escalated probability for a text's tag, such as an email or a newspaper story.
  Naive Bayes classifier is a genus of methods that all follow the same principle: each feature to be classified is trivial to the others [58].

- Bernoulli Naive Bayes: BernoulliNB renders the naive Bayes training and classification algorithms multivariate Bernoulli distribution data. Several features may be, but each is a binary-valued variable. As an outcome, samples must be characterized as binary-valued element vectors; if given any other type of data, the BernoulliNB representative may binarize it. Word occurrence vectors can be utilized to train and apply this classifier in the point of text classification. BernoulliNB may outperform other algorithms on some datasets, distinctly ones with fewer documents. If time allows, it is recommended that analyze both models [59].

After implementing models to each datasets we utilize an f1-score, Precision and Recall. When there are uneven classes of data, precision and recall take precedence over accuracy. We need these parameters for assessment since the dataset has unstable classes. The comparative debate is made more accessible as a result.

$$\text{Precision} = \mathbb{TP} / \left(\mathbb{TP} + \mathbb{FP}\right) \tag{6.1}$$

Here TP and FP stand for True Positive and False Positive, respectively.

$$\text{Recall} = \mathbb{TP} / \left(\mathbb{TP} + \mathbb{FN}\right) \tag{6.2}$$

Here, FN is False Negative.

F1-score is calculated as:

$$\text{F1-score} = 2 * (\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall}) \tag{6.3}$$

These are all evaluation measures we have performed in each 4 of the datasets. Here, we have performed an analysis of the performance evaluation of each dataset with each model.

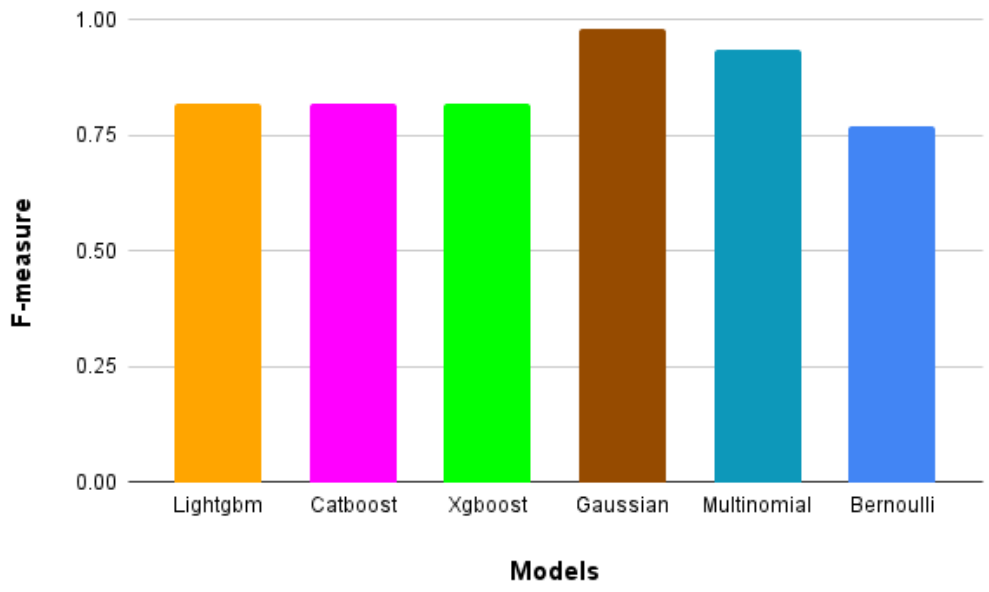

Figure 6.1: Performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for JM1 defect dataset

Fig. 6.1 shows the performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for the JM1 defect dataset. However, there is a low f1-score in the JM1 dataset, which is 77%. Once again, in the JM1 dataset, the gaussian f1-score outperformed all other models with 98%. The lightgbm, catboost, and xgboost got the same f1-score of 82%. The multinomial was placed second among all other models in the JM1 dataset with the f1-score of 93.3%.

Fig. 6.2 shows the performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for the KC1 defect dataset. The lightgbm, catboost, xgboost, and Bernoulli got the f1-score between 86% and 87%. In the KC1 dataset, multinomial outperformed all other models by the f1-score of 96%, and gaussian got 95% of the f1-score. Only in KC1 multinomial got the highest f1-score among all other datasets. The

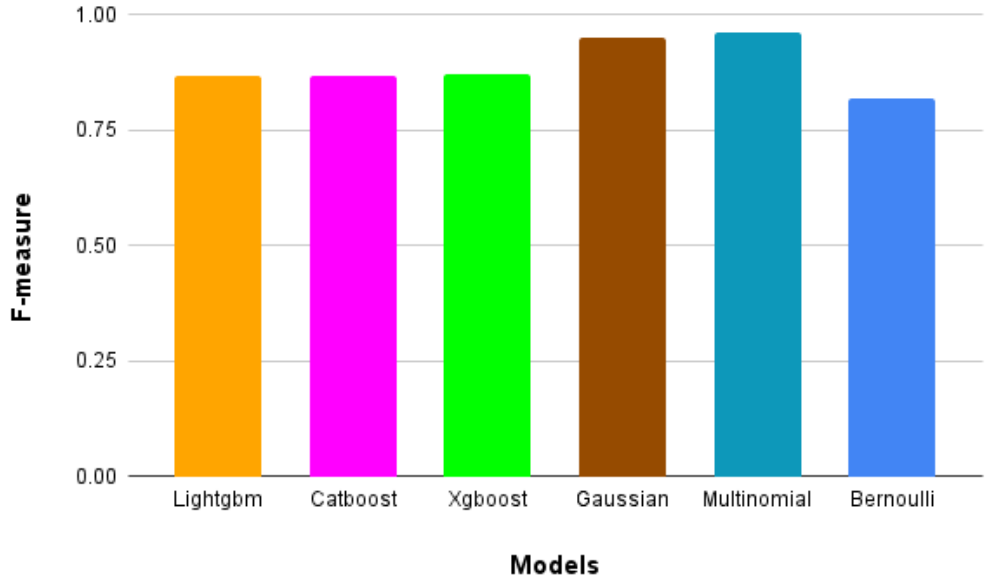

Figure 6.2: Performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for KC1 defect dataset

Bernoulli can learn the defect and non-defect separation up to 82%.

Fig 6.3 shows the performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for the KC2 defect dataset.From Figure 6.3, the Bernoulli has an F-measure of 0.79. Thus, Bernoulli can learn the defect and non-defect separation up to 80%. Similar to JM1, and KC1, the Bayes network also shows considerably low detection performance for the KC2 dataset. The Gaussian has an increased F-measure of 0.98. Thus the Gaussian F-measure has a combined Precision and Recall performance of 98%. Xgboost and lightgbm & catboost got f1-score of 86% and 84%, respectively.

Fig 6.4 shows the performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for the PC1 defect dataset. Lightgbm and catboost got the increment of 7-10% of f1-score in the PC1 dataset, which is 92%. The f1 score of gaussian is 95% which is the lowest among other datasets and the highest in the PC1 dataset. The lowest f-measure of Bernoulli is 76% among all other models.

According to the values regarding precision-recall and accuracy, we observe that the Gaussian naive Bayes is the best model for identifying software defects among all other models that we have compared. Here False and True defined the data as non-defected and defected. Here JM1 and KC2 datasets got the highest accuracy of 98% in the same model named Gaussian naive Bayes.

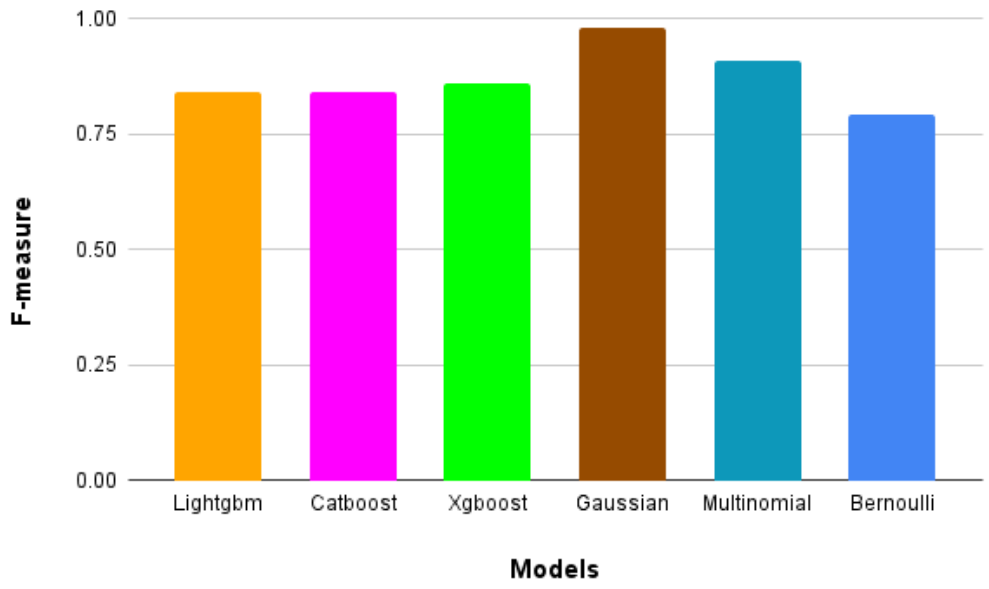

Figure 6.3: Performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for KC2 defect dataset

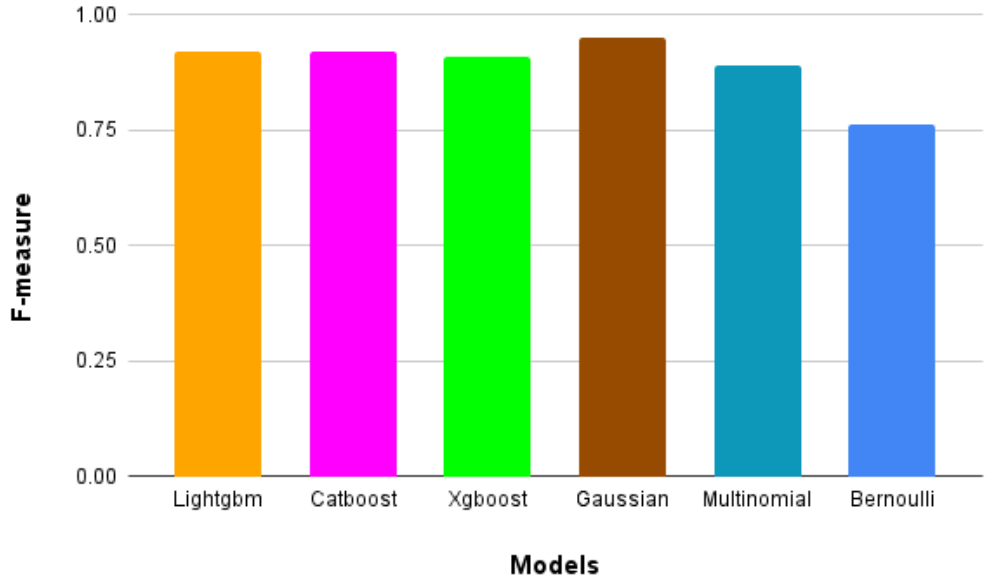

Figure 6.4: Performance evaluation of Lightgbm, Catboost, Xgboost, Gaussian, Multinomial, and Bernoulli for PC1 defect dataset

DATASET	MODELS	PRECISION	RECALL	F1-SCORE	ACCURACY
JM1	LightGBM	F = 0.84 T = 0.49	F = 0.97 T = 0.12	F = 0.90 T = 0.20	0.82
	Catboost	F = 0.84 T = 0.49	F = 0.97 T = 0.12	F = 0.90 T = 0.20	0.82
	Xgboost	F = 0.84 T = 0.48	F = 0.96 T = 0.16	F = 0.90 T = 0.24	0.819
	GaussianNB	F = 0.92 T = 0.99	F = 0.96 T = 0.98	F = 0.94 T = 0.99	0.98
	MultinomialNB	F = 0.87 T = 0.94	F = 0.65 T = 0.98	F = 0.75 T = 0.96	0.933
	Bernoulli NB	F = 0.37 T = 0.95	F = 0.76 T = 0.77	F = 0.49 T = 0.85	0.77
KC1	LightGBM	F = 0.89 T = 0.56	F = 0.97 T = 0.24	F = 0.93 T = 0.33	0.867
	Catboost	F = 0.89 T = 0.56	F = 0.97 T = 0.24	F = 0.93 T = 0.33	0.867
	Xgboost	F = 0.89 T = 0.59	F = 0.97 T = 0.29	F = 0.93 T = 0.39	0.87
	GaussianNB	F = 0.60 T = 1.00	F = 1.00 T = 0.95	F = 0.75 T = 0.97	0.95
	MultinomialNB	F = 0.69 T = 0.99	F = 0.91 T = 0.97	F = 0.78 T = 0.98	0.96
	Bernoulli NB	F = 0.29 T = 0.98	F = 0.84 T = 0.83	F = 0.43 T = 0.90	0.82
KC2	LightGBM	F = 0.83 T = 0.91	F = 0.99 T = 0.38	F = 0.90 T = 0.54	0.84
	Catboost	F = 0.83 T = 0.91	F = 0.99 T = 0.38	F = 0.90 T = 0.54	0.84
	Xgboost	F = 0.84 T = 1.00	F = 1.00 T = 0.42	F = 0.91 T = 0.59	0.86
	GaussianNB	F = 0.93 T = 0.99	F = 0.93 T = 0.99	F = 0.93 T = 0.99	0.98
	MultinomialNB	F = 0.88 T = 0.92	F = 0.47 T = 0.99	F = 0.61 T = 0.95	0.91
	Bernoulli NB	F = 0.40 T = 0.99	F = 0.93 T = 0.77	F = 0.56 T = 0.86	0.79
PC1	LightGBM	F = 0.92 T = 1.00	F = 1.00 T = 1.00	F = 0.96 T = 0.18	0.92
	Catboost	F = 0.92 T = 1.00	F = 1.00 T = 1.00	F = 0.96 T = 0.18	0.92
	Xgboost	F = 0.92 T = 0.50	F = 0.99 T = 0.15	F = 0.95 T = 0.23	0.91
	GaussianNB	F = 0.85 T = 0.98	F = 0.93 T = 0.96	F = 0.89 T = 0.97	0.95
	MultinomialNB	F = 0.72 T = 0.92	F = 0.67 T = 0.94	F = 0.69 T = 0.93	0.89
	Bernoulli NB	F = 0.40 T = 0.88	F = 0.50 T = 0.82	F = 0.44 T = 0.85	0.76

Table 6.1: Precision, Recall, F1-score and Accuracy of each datasets

Chapter 7

Open Issues and Challenges

7.1 Security and Privacy

For safe communication, the important application causes numerous security challenges to be addressed. Authentication and authorisation, secure data transfer, secure processing, and data storage are some of the topics covered. The data in the Most Critical Application is designed for certain user groups or organisations. Therefore , authentication and authorzsation are required. The information must be kept safe and processed safely. Routing layer attacks and eavesdropping attacks are examples of attacks that interrupt data communications. There are several threats, including ransomware, phishing, data loss, and hacking. Multi-factor authentication, password managers, and avoid reusing the same password are all possible solutions. To ensure secure data transfer, data encryption and backup, anti-malware protection, and a secure wireless network are all required.

7.2 Software bugs identification at initial stage

Before deploying the programme, the developer must identify issues in the initial stages. Unfortunately, many bugs take place at the initial step. SFunctional faults, performance flaws, security defects, syntax and logic errors, and system-level integration issues are all examples of software bugs that can occur during development. The software may be tested using advanced or autonomous testing, which allows the developers to find and fix errors.

7.3 Deployment trust issues

If developers are distributing software, they must have confidence that the critical application will be trusted wherever it is installed. They'll only deploy after that. As a result, there must be an evaluation and trustworthiness between important applications and their sensors.

7.4 Deployment environment

Any software created, such as autonomous cars or any other critical application like UAV or ITS, will not run in that critical application since it is hard-coded for that critical application alone. It is determined by the platform used for deployment. It could not be utilized for any other purpose. Making hybrid software that can run on any critical application might be the solution.

7.5 Data collection

Data is crucial in every use case. Beginners may practise machine learning with data easily available on Kaggle, the UCI ML Repository, and other places. The sensors of the crucial application record pedestrians, bicycles, traffic signs, traffic lights, road borders, traffic lanes, other infrastructure, and landscape during data collection. For critical applications, data collecting is not an easy task. It's also not easily available and hard to acquire.

7.6 Data preprocessing

It's feasible that the data will not ever be found. For example, it could be filled by mean and median, but we can't use so in a critical application since the data after mean and median filling may not be right; even a minor change in data might result in an unexpected incident. This causes its negative consequences. Developing a software architecture designed for edge components, evaluating the efficiency of the Data preprocessing solution and its influence on AI algorithms, and handling high throughput and low-latency requirements are the difficulties for data preprocessing.

7.7 Latency

When one system module transmits a message and then receives confirmation of receipt from another system module before sending further data, latency is often assessed as a round trip delay. The communication latency and reliability of traditional networks are incompatible with the crucial application. Even a 1ms delay is ineffective in preventing system failure, which might lead to devastating tragedy for people and property. This issue of critical application can be solved by 6-G deployment.

Chapter 8

Identify software defects in Autonomous vehicles: Case study

8.1 Identify the real-life problems

A self-driving automobile, also known as an autonomous vehicle, is a vehicle that reduces travel time, traffic deaths, and human stress. Many autonomous cars have been developed across the world in recent years. Everyone, whether human or machine, has a limitation. Before the launch, the engineers conducted extensive testing to identify any flaws or defects in the autonomous vehicle software. If a failure occurs in the AVs system, it may become uncontrollable as a result of the fault. For example, it may choose the incorrect lane instead of the original. When a defect occurs in an AV, it causes a number of significant issues. Some software generates defects automatically after installation, resulting in unexpected events such as risking human life or the environment.

8.2 Select significant problems in the case

Faults in the gyroscope, sensors, actuators, connectivity, and various software components are among the significant challenges. For example, defects in the sensor, might indicate problems with the camera, lidar, radar, or GPS. Assume that an error occurs in these sensor components. Many factors can interfere in this situation, including inability to identify an item, navigation system difficulties, inability to perceive the light or frequency, and many more. Other concerns develop, such as controller faults, which are controlled or run according to control algorithms, such as steering, braking, and acceleration. It

is possible that the AV user will be unable to manage steering, braking, or acceleration. There are also solutions available to resolve these significant problems.

8.3 Suggest solutions to these significant problems

Some solutions may be used to detect problems or deficiencies in autonomous vehicles. The answers are Machine learning and Deep learning. The techniques in Machine learning are frequently used to detect software defects such as SVM, DT, RF, XGBoost, KNN, NB, LR, and so on. Deep learning techniques include LSTM, R-CNN, Faster R-CNN, Pointnet, voxelnet, VGG-16, Adapnet, Resnet, and many more.

8.4 Recommend the best solution to be implemented

There are many techniques that are best at identifying software defects.However, our implementation primarily analyzes all machine learning algorithms, including lightgbm, catboost, xgboost, Gaussian, Bernoulli, and multinomial. After analysing all of these techniques, we determined that Gaussian had the best accuracy of any of them, at nearly 95%.

8.5 Detail how this solution should be implemented

The procedure is as follows: the dataset must be used to train the model, and then the process continues. In this, the datasets are from PROMISE and NASA's websites. The datasets are JM1, KC1, KC2, and PC1. The data is first correlated to see how the variables are connected to one another, and then it is used to forecast the target variables. The assessment procedure has been completed in the next section, and it appears that there is just one requirement that must be met if a defect exists; otherwise, the item is not defective. Following that, preprocessing occurs, which includes data cleansing, data integration, data transformation, and data reduction. Then we separate training and testing to determine precision, recall, f1-score, and accuracy using the pre-trained model Gaussian Naive Bayes classifier.

Chapter 9

Future work & Conclusion

To conclude, the survey regarding identifying software defects of critical applications such as Autonomous Vehicles (AVs), Unmanned Aerial Vehicles (UAVs), and Intelligent Transportation Systems (ITS). The simple error or the delay of milliseconds in this application leads to unexpected incidents. Many researchers have researched each application; however, every solution is imperfect. In the meantime, the intruders try to attack differently; hence it is not 100% safe.

Bibliography

[1] "Misson critical system." https://www.netmotionsoftware.com/blog/mobility/mission-critical-systems. 2019.

[2] "Damaged drone." https://toronto.ctvnews.ca/plane-damaged-after-being-hit-by-york-police-drone-at-buttonville-airport-1.5554617. 2021.

[3] "Crash drone." https://bit.ly/3ETab0X. 2021.

[4] "Tesla car crash." https://bit.ly/3sR8Wgh. 2021.

[5] "Software bug." https://www.tiredefectattorney.com/software-defects-in-auto-vehicles/. 2021.

[6] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, "A deep tree-based model for software defect prediction," *arXiv preprint arXiv:1802.00921*, 2018.

[7] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

[8] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic lstm model for software defect prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019.

[9] C. L. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," in *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, pp. 728–733, IEEE, 2020.

[10] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, p. 106287, 2020.

[11] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, 2020.

[12] H. Alsawalqah, N. Hijazi, M. Eshtay, H. Faris, A. A. Radaideh, I. Aljarah, and Y. Alshamaileh, "Software defect prediction using heterogeneous ensemble classification based on segmented patterns," *Applied Sciences*, vol. 10, no. 5, p. 1745, 2020.

[13] H. Wang, W. Zhuang, and X. Zhang, "Software defect prediction based on gated hierarchical lstms," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 711–727, 2021.

[14] S. Feng, J. Keung, X. Yu, Y. Xiao, K. E. Bennin, M. A. Kabir, and M. Zhang, "Coste: Complexity-based oversampling technique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, vol. 129, p. 106432, 2021.

[15] T. LEI, J. XUE, Y. WANG, Z. NIU, Z. SHI, and Y. ZHANG, "Wcm-wtra: A cross-project defect prediction method based on feature selection and distance-weight transfer learning," *Chinese Journal of Electronics*, pp. 1–13, 2022.

[16] H. Tong, W. Lu, W. Xing, B. Liu, and S. Wang, "Shse: A subspace hybrid sampling ensemble method for software defect number prediction," *Information and Software Technology*, vol. 142, p. 106747, 2022.

[17] "Autonomous vehicles." https://builtin.com/transportation-tech/self-driving-car-companies. 2021.

[18] J. Kullaa, "Detection, identification, and quantification of sensor fault in a sensor network," *Mechanical Systems and Signal Processing*, vol. 40, no. 1, pp. 208–221, 2013.

[19] S. U. Jan, Y.-D. Lee, J. Shin, and I. Koo, "Sensor fault classification based on support vector machine and statistical time-domain features," *IEEE Access*, vol. 5, pp. 8682–8690, 2017.

[20] L. Biddle and S. Fallah, "A novel fault detection, identification and prediction approach for autonomous vehicle controllers using svm," *Automotive Innovation*, pp. 1–14, 2021.

[21] J.-l. Yang, Y.-s. Chen, L.-l. Zhang, and Z. Sun, "Fault detection, isolation, and diagnosis of self-validating multifunctional sensors," *Review of scientific instruments*, vol. 87, no. 6, p. 065004, 2016.

[22] "Military drone market." https://www.fortunebusinessinsights.com/military-drone-market-102181. 2021.

[23] P. K. R. Maddikunta, S. Hakak, M. Alazab, S. Bhattacharya, T. R. Gadekallu, W. Z. Khan, and Q.-V. Pham, "Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges," *IEEE Sensors Journal*, 2021.

[24] R. Gupta, A. Shukla, P. Mehta, P. Bhattacharya, S. Tanwar, S. Tyagi, and N. Kumar, "Vahak: A blockchain-based outdoor delivery scheme using uav for healthcare 4.0 services," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 255–260, IEEE, 2020.

[25] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (uavs) for traffic monitoring," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 221–234, IEEE, 2013.

[26] Z. Shao, G. Cheng, J. Ma, Z. Wang, J. Wang, and D. Li, "Real-time and accurate uav pedestrian detection for social distancing monitoring in covid-19 pandemic," *IEEE Transactions on Multimedia*, 2021.

[27] M. D'Aloia, M. Rizzi, R. Russo, M. Notarnicola, and L. Pellicani, "A marker-based image processing method for detecting available parking slots from uavs," in *International Conference on Image Analysis and Processing*, pp. 275–281, Springer, 2015.

[28] J. G. Martinez, M. Gheisari, and L. F. Alarcón, "Uav integration in current construction safety planning and monitoring processes: Case study of a high-rise building construction project in chile," *Journal of Management in Engineering*, vol. 36, no. 3, p. 05020005, 2020.

[29] M. Erdelj and E. Natalizio, "Uav-assisted disaster management: Applications and open issues," in *2016 international conference on computing, networking and communications (ICNC)*, pp. 1–5, IEEE, 2016.

[30] J. Shen, Q. Zhu, X. Wang, and P. Chen, "Typical fault estimation and dynamic analysis of a leader-follower unmanned aerial vehicle formation," *International Journal of Aerospace Engineering*, vol. 2021, 2021.

[31] G. Heredia and A. Ollero, "Detection of sensor faults in small helicopter uavs using observer/kalman filter identification," *Mathematical Problems in Engineering*, vol. 2011, 2011.

[32] H. Chen and B. Jiang, "A review of fault detection and diagnosis for the traction system in high-speed trains," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 2, pp. 450–465, 2019.

[33] "Inter-turn fault." https://www.quora.com/What-is-inter-turn-fault-in-induction-motor 2019.

[34] D. Maulud and A. M. Abdulazeez, "A review on linear regression comprehensive in machine learning," *Journal of Applied Science and Technology Trends*, vol. 1, no. 4, pp. 140–147, 2020.

[35] J. Tolles and W. J. Meurer, "Logistic Regression: Relating Patient Characteristics to Outcomes," *JAMA*, vol. 316, pp. 533–534, 08 2016.

[36] "Naive bayes." https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/. 2017.

[37] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "Knn model-based approach in classification," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 986–996, Springer, 2003.

[38] "Bayesian network." https://www.geeksforgeeks.org/basic-understanding-of-bayesian-belief-networks/. 2017.

[39] A. Tharwat, T. Gaber, A. Ibrahim, and A. E. Hassanien, "Linear discriminant analysis: A detailed tutorial," *AI communications*, vol. 30, no. 2, pp. 169–190, 2017.

[40] S. Kumar and I. Chong, "Correlation analysis to identify the effective data in machine learning: Prediction of depressive disorder and emotion states," *International journal of environmental research and public health*, vol. 15, no. 12, p. 2907, 2018.

[41] "Ensemble methods." https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f 2017.

[42] P. Pongpaibool, P. Tangamchit, and K. Noodwong, "Evaluation of road traffic congestion using fuzzy techniques," in *TENCON 2007-2007 IEEE Region 10 Conference*, pp. 1–4, IEEE, 2007.

[43] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255–327, 2019.

[44] "Unsupervised learning." https://www.ibm.com/cloud/learn/unsupervised-learning. 2020.

[45] "Nasa dataset." https://data.nasa.gov/browse?limitTo=datasets.

[46] "Promise repository." http://promise.site.uottawa.ca/SERepository/datasets-page.html.

[47] "Aeeem dataset." https://zenodo.org/record/3362613#.YkFq3C8RpQI.

[48] "Eclipse dataset." https://bug.inf.usi.ch/index.php.

[49] R. U. Khan, S. Albahli, W. Albattah, and M. N. I. Khan, "Software defect prediction via deep learning," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, pp. 343–349, 3 2020.

[50] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[51] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[52] T. Menzies, J. S. Di Stefano, M. Chapman, and K. McGill, "Metrics that matter," in *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings.*, pp. 51–57, IEEE, 2002.

[53] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[54] R. Z. SAFAROV, Z. K. SHOMANOVA, Y. G. NOSSENKO, Z. G. BERDENOV, Z. B. BEXEITOVA, A. S. SHOMANOV, and M. MANSUROVA, "Solving of classification problem in spatial analysis applying the technology of gradient boosting catboost," *Folia Geographica*, vol. 62, no. 1, p. 112, 2020.

[55] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[56] H. Zhang, "The optimality of naive bayes," *Aa*, vol. 1, no. 2, p. 3, 2004.

[57] J. Rennie, L. Shih, J. Teevan, and D. Karger, "Tackling the poor assumptions of naive bayes classifiers (pdf)," *ICML.[Accessed: 10-Feb-2017]*, 2003.

[58] J. ús Cerquides and R. L. de Mantaras, "The indifferent naive bayes classifier," 2003.

[59] A. McCallum, K. Nigam, *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, Citeseer, 1998.

rg

ORIGINALITY REPORT

13%	6%	9%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ijitee.org Internet Source	1%
2	K. McGill. "Metrics that matter", 27th Annual NASA Goddard/IEEE Software Engineering Workshop 2002 Proceedings SEW-02, 2003 Publication	1%
3	T. Menzies, J.S. Di Stefano, M. Chapman, K. McGill. "Metrics that matter", 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings., 2003 Publication	1%
4	Submitted to Middlesex University Student Paper	1%
5	Vishakha K Ralegankar, Jagruti Bagul, Bhaumikkumar Thakkar, Rajesh Gupta, Sudeep Tanwar, Gulshan Sharma, I. E. Davidson. "Quantum Cryptography-as-a-Service for Secure UAV Communication: Applications, Challenges, and Case Study", IEEE Access, 2021 Publication	1%