Predictive Models for Bug Fixing Time: An In-depth Analysis with Hidden Markov Model

A project report submitted in partial fulfillment of the requirement for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

By

GURRALA MADHU MOHAN VAMSI

21VV1F0012

Under the Esteemed Guidance of

Dr. B. Tirimula Rao, M.Tech, Ph.D.

Assistant Professor & HOD

Department of Information Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUGV College of Engineering Vizianagaram (Autonomous)

Jawaharlal Nehru Technological University, Gurajada Vizianagaram

Dwarapudi, Vizianagaram-535003, Andhra Pradesh, India

2022-2023

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTU-GV COLLEGE OF ENGINEERING

VIZIANAGARAM

CERTIFICATE

This is to certify that the Dissertation report entitled Predictive Models for Bug Fixing Time: An In-depth Analysis with Hidden Markov Model that is being submitted by Gurrala Madhu Mohan Vamsi bearing registration number 21VV1F0012 in partial fulfillment for the degree of Master of Computer Applications (MCA) from Jawaharlal Nehru Technological University Gurajada Vizianagaram - College of Engineering Vizianagaram.This bonafide work was carried out by him under my guidance and supervision during the year 2022 - 2023.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of Project Guide
Dr. B. TIRIMULA RAO
Assistant Professor & HOD
Dept. of Information Technology

Signature of Head of the Department
Dr.B.TIRIMULA RAO
Assistant Professor & HOD
Dept. of Information Technology

(Signature of External Examiner)

madhumohanvamsi123@gmail.com.pdf

ORIGINALITY REPORT

19%	12%	8%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.coursehero.com Internet Source	2%
2	Submitted to Manchester Metropolitan University Student Paper	1%
3	Submitted to University of Salford Student Paper	1%
4	Submitted to University of Warwick Student Paper	1%
5	Submitted to CTI Education Group Student Paper	1%
6	Anisha M. Lal, S. Margret Anouncia. "Semi-supervised change detection approach combining sparse fusion and constrained k means for multi-temporal remote sensing images", The Egyptian Journal of Remote Sensing and Space Science, 2015 Publication	<1%
7	Submitted to Brunel University Student Paper	<1%

DECLARATION

I **Gurrala Madhu Mohan Vamsi (Reg. No.: 21VV1F0012)**, declare that this written submission represents my ideas in my own words and where others' ideas or words have been included. I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea data fact source in my submission. I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

G M M Vamsi

(21VV1F0012)

Date :

Place :

DEPARTMENT OF INFORMATION TECHNOLOGY
JNTUGV College of Engineering Vizianagaram (Autonomous)
Jawaharlal Nehru Technological University, Gurajada Vizianagaram
Dwarapudi, Vizianagaram – 535003, Andhra Pradesh, India
2022 - 2023

Website: www.jntukucev.ac.in

Subject Name: Dissertation **Regulation: R20**

Subject Code: MCA4103 **Academic Year: 2023**

CO'S

Course Outcomes

Course Outcomes	
CO1	To develop the work practice in students to apply theoretical and practical tools/techniques to solve real life problems related to industry and current research.
CO2	Complete an independent research project, resulting in at least a thesis publication and research outputs in terms of publications in high impact factor journals, conference proceedings and patents.
CO3	Demonstrate knowledge of contemporary issues in their chosen field of research.
CO4	Demonstrate an ability to present and defend their research work to panel of experts.
CO5	Be able to apply the knowledge of computing tools and techniques in the selected field for solving real world problems encounter in the software industries.

CO-PO Mapping

Mapping of Course Outcomes (COs) with Program Outcomes (POs)

Course Outcomes	Program Outcomes (POs)														
	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	3	1	2	2	-	-	-	3	3	2	3	3	3	2
CO2	3	3	2	2	2	-	-	-	2	3	3	2	3	2	2
CO3	3	3	2	1	3	-	-	1	2	2	3	3	3	3	2
CO4	2	3	1	2	2	-	-	1	2	3	2	2	2	3	1
CO5	3	2	2	3	1	-	-	-	2	3	1	3	1	2	1

Enter correlation levels 1,2 and 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High) If there is no correlation, put "-"

Signature of the Project Guide

ACKNOWLEDGEMENT

This acknowledgment transcends the reality of formality when I express deep gratitude and respect to all those people behind the screen who inspired and helped us in the completion of this project work.

I take the privilege to express my heartfelt gratitude to my guide **Dr. B. TIRIMULA RAO**, Assistant Professor HOD, Department of Information Technology, JNTUGV-CEV for his valuable suggestions and constant motivation that greatly helped me in the successful completion of the project. Wholehearted cooperation and the keen interest shown by him at all stages are beyond words of gratitude.

With great pleasure and privilege, I wish to express my heartfelt sense of gratitude and indebtedness to **Dr. B.TIRIMULA RAO**, Assistant Professor, Head of the Department of Information Technology, JNTUGV-CEV, for his supervision.

I extend heartfelt thanks to our principal **Prof. Dr. K. SRIKUMAR** for providing intensive support throughout my project.

I express my sincere thanks to Project Coordinator **W. ANIL**, Assistant Professor Department of Information Technology, JNTU-GURAJADA VIZIANAGARAM for his continuous support.

I am also thankful to all the Teaching and Non-Teaching staff of the Information Technology Department, JNTUGV-CEV, for their direct and indirect help provided to me in completing the project.

I extend my thanks to my parents and friends for their help and encouragement in the success of my project

G M M Vamsi
(21VV1F0012)

ABSTRACT

Machine learning techniques have grown and evolved significantly, providing adaptable solutions to a wide range of problem areas. The performance evaluation of Hidden Markov Models (HMM) in the context of machine learning is the focus of this research. For jobs requiring a grasp of dynamic patterns, HMM is known for its ability to represent sequential data and capture temporal dependencies. The project examines HMM's performance in comparison to several well-known machine learning techniques during an eight-year period from 2015 to 2022, including Multinomial Naive Bayes (MNB), Random Forest Classifier (RFC), Support Vector Machines (SVM), and Non-linear Support Vector Machines (N-SVM).

By continually demonstrating strengths in precision and recall, HMM successfully captured sequential patterns, according to our analysis. When compared to its competitors, it showed potential for development in terms of overall accuracy and area under the curve (AUC). As time went on, HMM maintained its competitive edge in precision, but it was up against fierce competition from other techniques that showed improvements in accuracy and performance as a whole. These patterns demonstrate how machine learning is dynamic, with changes in processing power and algorithmic methods influencing the performance landscape.

While HMM excels at tasks that rely for the modelling of dynamic, time-varying data, such as speech recognition and gesture analysis, its changing performance highlights the significance of continual improvements to assure its viability in industries where precision and recall are crucial.

This project offers insights into the benefits, drawbacks, and prospective routes for improvement of HMM in machine learning at this critical juncture. In order to satisfy the changing needs of data-driven applications, it emphasises the crucial role that context and objectives play in the selection of machine learning techniques.

Contents

List of Figures

List of Tables

Abbreviations

HMM	Hidden Markov Model
MNB	Multinomial Naive Bayes
RFC	Random Forest Classifier
SVM	Linear Support Vector Machine
N-SVM	Non Linear Support Vector Machine

Chapter 1

Introduction

1.1 Introduction to Bugs

A software bug is a flaw that results in a programme crashing or producing incorrect output. The issue is brought on by faulty or insufficient reasoning. An error, omission, flaw, or defect that could lead to failure or a departure from desired outcomes is referred to as a bug.
Some flaws might not have a significant impact on the program's functionality and might be unnoticed for a long period. When major bugs go unfixed, a programme may crash. Another type of issue known as a security bug could let a hostile user get around access controls and gain unauthorised access to privileges.
The informal term for flaws, or bugs, is when a piece of software or an application does not function as intended.

A software bug may also be referred to as an issue, error, fault, or failure in software testing. Any blunder or mistake done by the developers while creating the product led to the bug.

The test engineer might not achieve the desired outcome when running the test cases or testing the application. And the bug went by many names in different firms, including error, problems, issues, fault, and mistake, among others.
The time needed to resolve a bug is a crucial aspect of software development that has an impact on both the overall quality of the product and the effectiveness of human resource management. When bugs are fixed on schedule, software development teams may better allocate their resources and produce high-quality software within the allotted time. Bug

fixing time estimation is a difficult undertaking, though, since it depends on a number of variables, including the complexity of the bug, the developer's experience, and the availability of resources.
The process by which any programming error becomes known as a bug is described here.

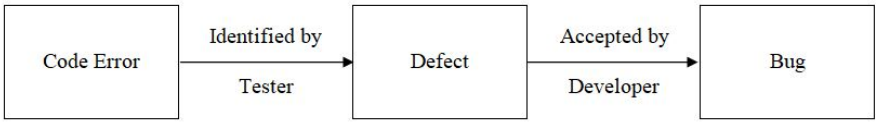

FIGURE 1.1: Code to Bug journey

The triage, assignment, and tracking tasks are related to bug fixing. The triage's goal is to assign a particular bug to a certain developer. It just considers priority and severity. By allocating different developers to different bugs, it may solve many issues simultaneously. Assignment, on the other hand, seeks to match a bug with an appropriate developer. When the allocated developer cannot resolve a specific bug because of their incapacity or improper assignment, another developer is brought on board.
This procedure will be carried out until the bug is assigned to a specific developer. And this method is known as "bug tossing." The bug tracker also records reported software flaws.
Large software projects also adopted bug repositories to keep track of all the information connected to defects, including textual descriptions of the bugs and updates based on how well they are being fixed. The ability to facilitate simpler issue discovery and repair based on analogous well-known occurrences is a key benefit of the bug repository.
However, the procedure of fixing bugs becomes very difficult. On the one hand, it necessitates the fusion of a small team of software developers with a large number of bugs (hundreds or thousands). On the other hand, software developers invest a large amount of time in locating and resolving bugs. Therefore, bug fix time estimation methods appear to have the promise of giving team managers timely and precise information that could facilitate realistic planning and efficient resource management procedures. Otherwise, time prediction depends on the team members' subjective estimates, which makes it more likely to be inaccurate or unreliable.
However, it calls for merging a small team of software developers with a large number of defects. Software engineers, on the other hand, put in a lot of time. Consequently, the estimating methods for repairing bugs become accurate.

1.2 Introduction to Bug Life Cycle

Another name for the bug life cycle is the defect life cycle. It is a stage of a defect that lives in several states at different times. It begins when a testing device discovers a new fault and concludes when it eliminates that problem, making sure that it does not recur. It's time to comprehend the real process of a faulty life cycle using the fundamental diagram that is presented below.

Every reported bug goes through a lifetime until it is fixed. A bug's journey from the time it is formed until it is fixed and closed is depicted by a bug life cycle. Below is a general bug lifecycle explained:

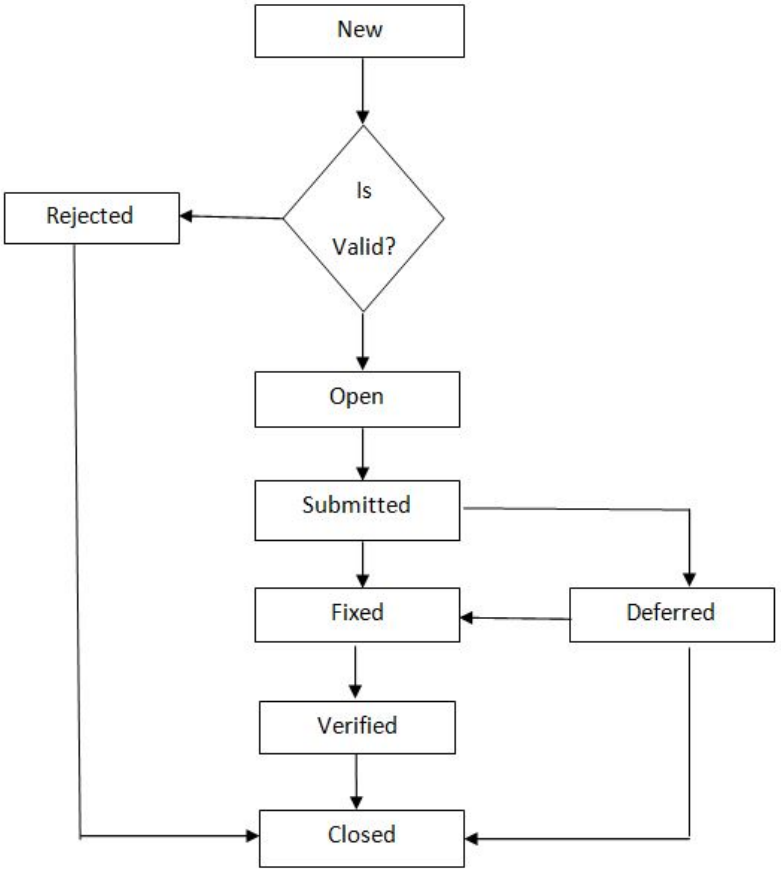

FIGURE 1.2: Simplified Bug Life Cycle

Let's examine each phase of the bug life cycle one by one.

- **Open :** The coder starts the process of analysing the bug and attempts to fix it. The mistake may be sent to one of the following four states: Reject or Not, which is Duplicate, if the programmer believes the flaw to be insufficient.
- **New :** The first stage of the bug life cycle is represented by this classification condition. If a new defect is found, validation and testing are conducted on these bugs in the latter stages of the bug life cycle.
- **Valid :** After New State, this is the next step.It merely determines whether the bug is true or false. If it is, the state changes to Open; else, it is closed.
- **Assigned :** A newly formed fault is given to the development team to work on the issue at this level. The project manager or the head of the team will assign this to a designer.
- **Deferred :** After the defect has been fixed, the designer will send the tester the error for retesting the error, and the defect will remain in the pending retest state while the tester works on retesting the error.
- **Fixed :** The defect status can be changed to "Fixed" if the developer successfully completes the work of fixing the issue by implementing the required changes.
- **Verified :** The defect status is set to "confirmed" if the tester finds no issues with the flaw after the designer assigned the flaw to the testing equipment and assumed it would be fixed properly.
- **Reopen :** The programmer will then be told to check again, and the defect status will be reopened, if there is still a problem with the flaw.
- **Closed :** The tester updates the defect status to "Closed" if the flaw is not present.
- **Duplicate :** Developers modify a defect's status to "duplicate" if they believe it to be similar to another defect or if the definition of the defect overlaps with another defect.

Chapter 2

Literature Survey

2.1 Literature Report of this Project

This project offers a succinct literature review on estimating the time it will take to resolve bugs. It notes that numerous efforts have been made in the past to develop learning models for estimating bug-fix durations. The survey shows that the majority of the published research have employed machine learning methods like regression analysis, decision trees, and neural networks to forecast the amount of time needed to correct bugs. These models, however, fall short in their ability to represent the temporal order of activities involved in the bug-fixing process.

The application of machine learning techniques has grown significantly over time and in a number of different fields. Five well-known machine learning methods—Hidden Markov Models (HMM), Multinomial Naive Bayes (MNB), Random Forest Classifier (RFC), Support Vector Machines (SVM), and Non-linear Support Vector Machines (N-SVM)—are thoroughly examined in this survey of the literature. Based on how well these techniques performed over several years (2015–2022), they were assessed.

This project also describes how Hidden Markov Models (HMMs) have been employed in several recent research to forecast problem fixing times. A statistical model called an HMM can be used to represent the temporal relationships between events in a series. As they can capture the temporal sequence of actions involved in the bug fixing process, HMMs have demonstrated promising results in predicting bug fixing time.

The review of the literature also identifies some of the shortcomings of earlier studies on bug fixing time estimation. For instance, some of the models that are now in use have simply employed a small number of features to forecast issue repair times, such as the

severity of the bug or the developer's expertise. These models could produce unreliable forecasts because they don't fully account for the complexity of bug fixing.

Additionally, this effort makes notice of the fact that some previous studies have made use of data from closed-source software initiatives, which may not be accessible to other researchers. This restricts the results' ability to be replicated and makes evaluating the effectiveness of various models challenging.

It's a science in and of itself to assign bugs to the appropriate developer or team; it's not just a routine administrative task. The amount of time needed to fix the problem can be significantly decreased by effective bug assignment. Manual assignment based on arbitrary judgements has given way to more data-driven, computational techniques throughout time. Advanced algorithms can estimate which developer is best suited to address a particular defect by examining historical data on bug resolutions, developer competence, previous code contributions, and even team communication patterns. This proactive method optimises resource allocation while simultaneously hastening resolution.

Every bug has a tale to tell. While some faults are quickly resolved, others might persist for weeks or even years. Such discrepancies have a variety of causes. The complexity of the bug itself, how it affects software functioning, or even how well-versed the developer is in the relevant code section are all possible contributing factors. Researchers try to identify patterns and trends by examining and classifying bugs based on their life cycle, severity, and other characteristics. The tactics, tools, and practises used to proactively address or even avoid particular bug categories can be informed by these findings.

The methods and tools for managing bugs have grown along with technological breakthroughs. Modern systems use machine learning algorithms to create predictions about bug features, such as their potential severity, affected modules, or expected fixing time. These algorithms analyse enormous volumes of historical data. Additionally, developers can benefit from visualisation tools like bug relationship graphs, which provide a more comprehensive knowledge of the interdependencies between issues and the health of the product.

The approaches used for effective bug management go beyond software engineering. Numerous industries can benefit from the fundamental ideas of data-driven decision-making, methodical prioritisation, and predictive analytics. Similar approaches could be used, for example, in healthcare to forecast and manage patient health outcomes or in finance to predict market movements.

While SVM and RFC showed versatility and durability across a range of applications, they competed with HMM, which was particularly good at capturing sequential patterns. RFC's ensemble learning strategy consistently produced precision, accuracy, and AUC that

were competitive. SVM, on the other hand, consistently outperformed other approaches in terms of maximising classification accuracy and overall performance. N-SVM displayed performance traits that were comparable to those of SVM, demonstrating the robustness of the SVM family of algorithms.

The comparative examination of machine learning techniques demonstrates that the particular goals and characteristics of the task should guide the selection of an effective method. HMM excels at properly handling temporal dependencies and sequential data. It is still a useful technique for applications like speech recognition and gesture analysis where recording dynamic patterns is essential.

However, it is crucial to recognise that machine learning is a dynamic field, and that performance patterns are continuously being shaped by improvements in computer power and algorithms. Although HMM's performance is strong in some situations, it might use some improvements to improve its overall accuracy and compete with other approaches in applications where recall and precision are crucial.

The literature review, taken as a whole, offers a thorough overview of the available works on bug fixing time prediction and draws attention to the shortcomings of the existing models. The survey lays the groundwork for the proposed HMM-based model, which seeks to overcome these constraints and offer precise estimates of bug fixing time.

Chapter 3

Software and Hardware Requirements

3.1 Software Requirements

- **Environment :** Python
- **Tool :** Jupyter Notebook
- **Version :** 3.9.13
- **Operating System :** Windows 10
- **Edition :** Windows 10 Home Single Language
- **Serial Number :** FV3DFT2
  Library Versions :
- **Numpy :** 1.21.5
- **Pandas :** 1.4.4
- **hmmlearn :** 0.2.8
- **scikit learn :** 1.0.2
- **matplotlib :** 3.5.2

3.2 Hardware Requirements

- **Processor :** Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz
- **RAM :** 4.00 GB (3.87 GB usable)
- **Storage :** 1 TB
- **System Type :** 64-bit operating system, x64-based processor

Chapter 4

Data-Set

4.1 Data Collection

In general we have six bug reports from Bugzilla repositories which are fairly used namely Eclipse, Freedesktop, GCC, Gnome, Mozilla and WineHQ. In our project, it is based on Mozilla repository

4.2 Source

(https://www.kaggle.com/datasets/azizullah444/firefox-bugs) This dataset was obtained from the Kaggle website and describes the Firefox bugs. Firefox bugs data were selected for a 8 year period ranging from 2015 to 2022. It contains information about 8063 bugs.

4.3 Input Attributes

The following are the input attributes for this Mozilla Firefox Bugs Dataset.

1. Reporting
    - Beginner
    - Intermediate
    - Advanced

2. Assignee
    - Particular
    - Random
3. Status
    - New
    - Unconfirmed
4. Notification
    - Normal
    - High

	Bug ID	Reporting	Summary	Product	Component	Assignee	Status	Notification	Updated
0	1116989	Beginner	Guest users should not see introduction info	Firefox	Toolbars and Customization	Random	Unconfirmed	Normal	1/2/2015 12:31
1	1047673	Beginner	URL checksums (for "securely" linking third-pa...	Firefox	General	Random	Unconfirmed	Normal	1/2/2015 14:16
2	1108341	Intermediate	Context menu entries use the original URI inst...	Firefox	Menus	Random	Unconfirmed	Normal	1/3/2015 8:26
3	1111278	Intermediate	A strip with all(?) possible icons is showing ...	Firefox	Toolbars and Customization	Random	Unconfirmed	High	1/5/2015 22:18
4	992209	Beginner	invoking firefox with command line argument(s)...	Firefox	General	Random	Unconfirmed	High	1/6/2015 5:17
...	...	...	...	...	...	...	...	...	...
8058	878812	Beginner	Make PDF Viewer's search highlight in a higher...	Firefox	PDF Viewer	Random	New	Normal	9/10/2022 19:12
8059	1782360	Intermediate	paramountplus.com page keeps refreshing, after...	Firefox	Untriaged	Random	Unconfirmed	Normal	9/10/2022 20:03
8060	1530394	Beginner	Turn each private browsing window into a separ...	Firefox	Private Browsing	Random	New	Normal	9/10/2022 22:53
8061	1790233	Intermediate	Firefox keeps crashing on Windows 11	Firefox	Untriaged	Random	Unconfirmed	Normal	9/11/2022 2:48
8062	1786207	Intermediate	Firefox sends invalid If-None-Match request he...	Firefox	Untriaged	Random	Unconfirmed	Normal	9/11/2022 2:58

8063 rows × 9 columns

FIGURE 4.1: Firefox Bugs Dataset

Chapter 5

Architecture

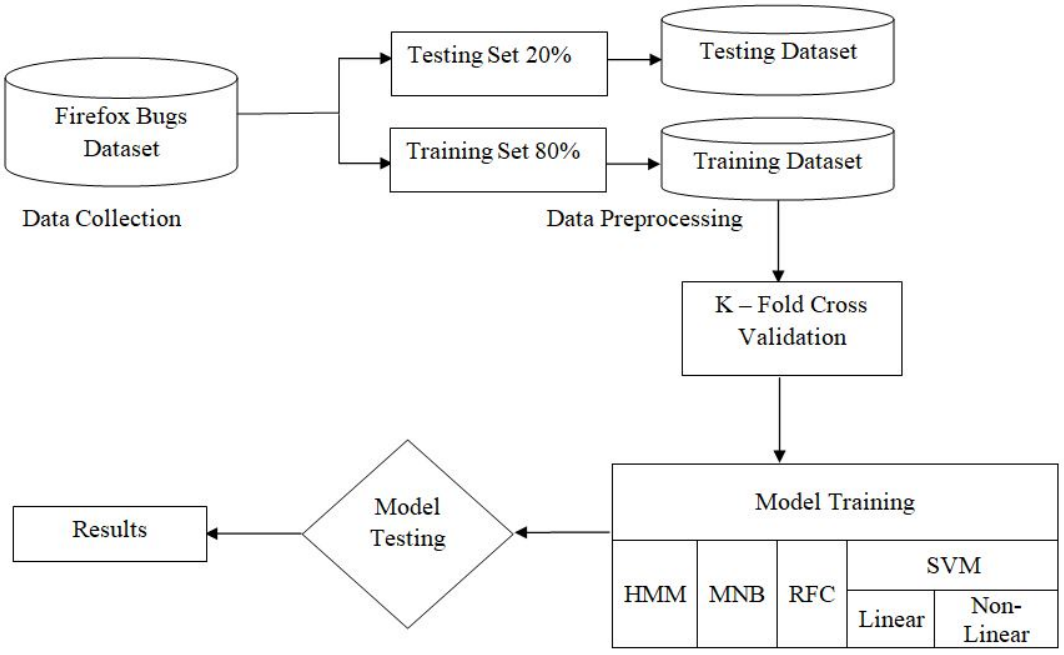

FIGURE 5.1: Architecture

Data Collection refers to the raw data that will be used in the project should be sourced. Data from databases, APIs, web scraping, etc. may be used for this.The systematic process of acquiring and analysing information on particular variables of interest in a predetermined, methodical way is known as data collection. This technique enables one to analyse outcomes and respond to pertinent inquiries.

In general, Data Preprocessing refers to Cleaning and converting raw data into a format that algorithms can easily understand and use, particularly in the context of machine learning, is known as data preprocessing.

- Data Cleaning: Handle any noise in the data, outliers, and missing numbers.
- Data Transformation: Transform the data so that it is suitable for the following steps (such as normalisation and encoding of categorical variables).
- Data Reduction: Delivering the same or similar analytical results while reducing the volume.
- Data Discretization: Creating categorical traits out of continuous ones.
- Data Splitting:Divide the data into training and testing datasets by using data splitting.

A statistical technique used to gauge the effectiveness of machine learning models is K-Fold Cross Validation. The dataset is split into k subsets, often known as "folds". The model is evaluated on the remaining fold after being trained on k-1 of these folds. Each fold acts as the test set once throughout this process, which is repeated k times.
In a k-fold arrangement, utilising 80% of the data for training indicates that 80% of the data is utilised for training and 20% for validation during each iteration of the k-fold process. It should be noted that this differs from a straightforward train-test split in which data is separated only once; in k-fold, the split takes place k times.
The process of feeding data into a machine learning algorithm (or "model") to discover patterns, correlations, or structures within that data is known as model training. The ultimate objective is to have the model make predictions or judgements on its own, without being expressly programmed to do so. Making predictions with the model repeatedly while altering model parameters to boost prediction accuracy is referred to as "training".

- HMM(Hidden Markov Model): Learning transition probabilities between states and the probabilities that observable data will emit given a state are both part of the training process.
- MNB (Multinomial Naive Bayes): Based on the frequency of occurrences in the training data, training entails generating prior probabilities for each class and likelihoods for each feature given a class.

- RFC (Random Forest Classifier): A Random Forest Classifier is a machine learning ensemble approach that mixes numerous decision trees to produce highly reliable classification task predictions, lowering overfitting and improving generalisation performance.

- Linear - SVM (Linear Support Vector Machine): By maximising the margin between classes, a Linear Support Vector Machine (Linear SVM) is an efficient machine learning model for binary classification tasks. It identifies the ideal linear border (hyperplane) to separate data points belonging to various classes.

- Non Linear - SVM (Non Linear Support Vector Machine): A Non-Linear Support Vector Machine (Non-Linear SVM) is a development of the Support Vector Machine (SVM) algorithm that handles complex, non-linear patterns in data by transforming the input features into a higher-dimensional space using kernel functions. This enables efficient classification in both linear and non-linear situations.

Model testing is necessary to assess models' performance after they have been trained on new data. The goal is to comprehend how the model will behave in practical situations when it comes in contact with data that it hasn't seen before.
To determine how well the model performed, performance indicators are assessed following model testing. Various measures, such as accuracy, precision, recall, F1 score, etc., are utilised depending on the situation at hand. The outcomes can serve as a roadmap for additional model improvement or serve as a sign of deployment readiness.

Chapter 6

Methodology

6.1 Hidden Markov Model

The Hidden Markov Model (HMM), a statistical model (thus the name "hidden"), depicts systems that are controlled by an underlying process that is not readily observable. One way to conceptualise this underlying process is as being in one of a number of states at any given moment. Each state produces an output that may be observed in accordance with a specific probability distribution. Transition probabilities control the transitions between states.

A class of statistical model called HMMs can model systems with unobservable or hidden states. Observable outputs (or observations) are generated by these hidden states depending on specific probability distributions.

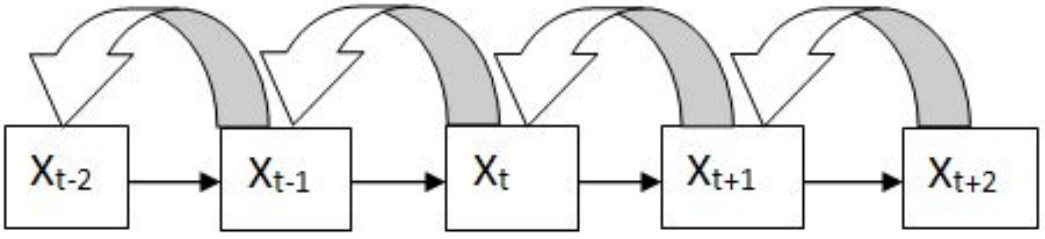

FIGURE 6.1: Hidden Markov Model

$$P(Hidden\ Markov\ Model) = \frac{P(X_t)}{P(X_{t-1})} \tag{6.1}$$

where,
X_t = current state
X_{t-1} = previous state
X_{t+1} = next state

This means outcome that satisfies X_{t-1}, then how likely it satisfies X_t. And here, X_t is dependent on X_{t-1}.

A statistical model called an HMM assumes that the system being modelled is a markov process with hidden states. The markov property states that the likelihood of a following state depends only on the prior state. It is referred to as a memory-less process because it never depends on any other possibilities. Each of the states in the HMM has a predetermined number of transactions and emissions restrictions. Each interstate exchange has a given probability. Every model begins in the start state and concludes in the end state. There are 3 probability and 2 states. These are the 2 states:

1. **Hidden State :** Hidden State: These cannot be seen directly, but the observation symbols that they emit allow for their detection. i.e. X1, X2, X3
2. **Observable State :** A state that can be viewed or seen is an observable state. i.e. Y1, Y2, Y3

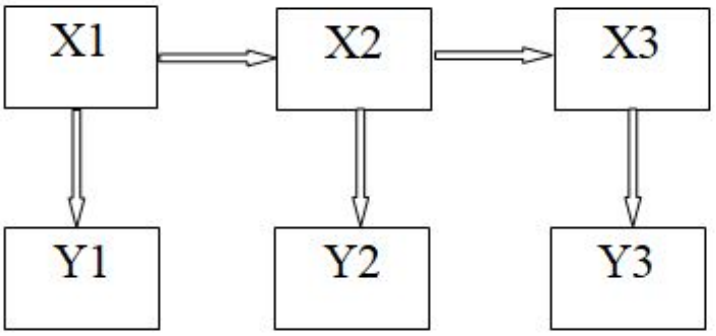

FIGURE 6.2: States in HMM

And the following are the three probabilities.

1. **Initial Probability:** Starting state probability π_i is the state that the markov chain will start in state i.
$$\pi_i = P(S_i) \tag{6.2}$$
2. **Transition Probability:** Each a_{ij} resenting the probability of moving from state i to state j.
$$a_{ij} = \frac{P(S_i)}{P(S_j)} \tag{6.3}$$
3. **Emission Probability:** A sequence of observation likelihoods.
$$P(Emission\ Matrix) = \frac{P(Value\ in\ Observable\ state)}{P(Value\ in\ Hidden\ state)} \tag{6.4}$$

Parameters :

1. **n_components :** The amount of states there are in the model.
2. **covariance_type :** Provides information on the kind of covariance parameters to be utilised. Typical choices include:
    - **spherical :** The variance is the same for all features.
    - **diag :** Every feature has a unique variation.
    - **full :** There is a full covariance matrix for each state.
    - **tied :** The covariance matrix is the same for all states.
3. **n_iter :** The most iterations that can be used to train the model.
4. **tol :** Convergence standard. When the log probability improvement during training is less than tol, iteration terminates.
5. **init_params :** The settings are initialised before training is controlled. 's' stands for startprob (starting state distribution), 't' for transmat (transition matrix), and'm' and 'c' for means and covars, respectively, are all present in this acronym.
6. **params :** Determines which training process parameters are updated. It has the same letters as init_params and enables the updating of specific parameters.

Attributes :

1. **startprob_ :** Distribution of first state occupations (typically denoted by π)
2. **transmat_ :** Matrix of state transition probabilities.
3. **means_ :** Mean values for every state.
4. **covars_ :** Parameters for each state's covariance. Covariance_type determines the shape and significance of it.
5. **n_features :** Feature count in the data.
6. **monitor_ :** An instrument for diagnosing model convergence. Throughout training, it stores the log likelihood values.
7. **converged_ :** Whether the model has converged is indicated.

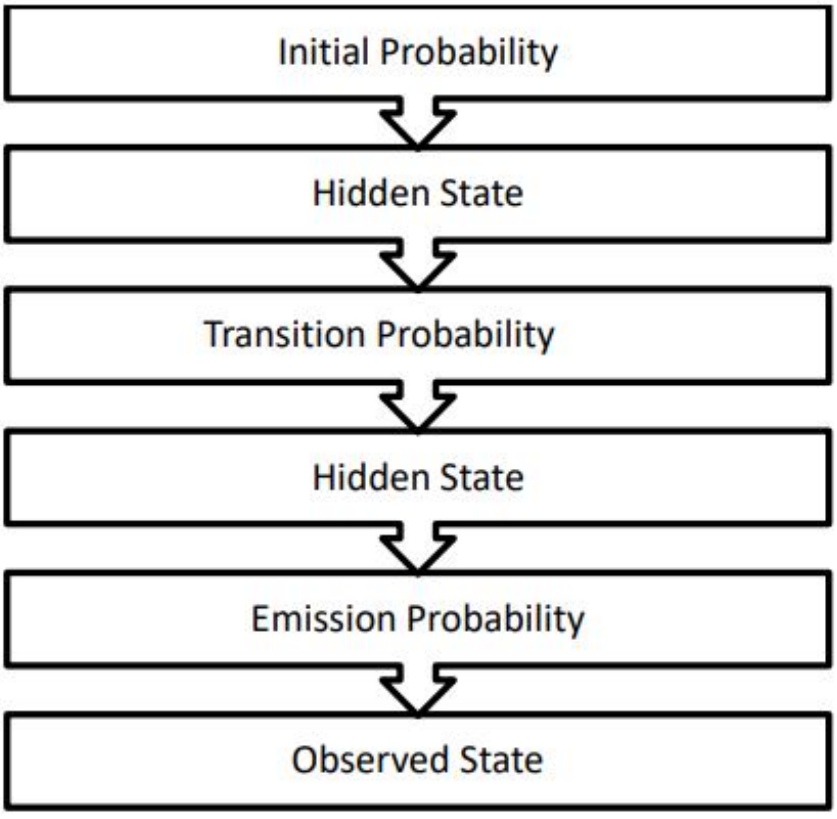

FIGURE 6.3: Block Diagram of HMM Model

6.2 Naive Bayes Multinomial

An improvement on the standard Naive Bayes classifier, the Multinomial Naive Bayes (MNB) classifier excels in classifying situations where data may be described as counts or frequency vectors, such as text categorization.

Principle : The Bayes theorem serves as the foundation for the Multinomial Naive Bayes classifier. The term "naive" refers to the classifier's presumption that, given the class name, the features are independent of one another. Despite being simplistic, this assumption makes the calculation simpler and, in practise, frequently performs quite well.

Bayes Theorem : Based on knowledge of conditions that might be associated to the event, it describes the likelihood of an event. The equation is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{6.5}$$

Where,

P(A | B) = posterior probability of class A given predictor B

P(A) = prior probability of class

P(B | A) = likelihood which is the probability of predictor given class

P(B) = prior probability of the predictor

Naive Assumption : This classifier takes the stance that the features are independent even though they might be interconnected. The name comes from how "naive" this assumption is.

Multinomial Distribution : The Multinomial Naive Bayes is used to classify features that can have several outcomes as opposed to the classic Naive Bayes, which only classifies features with binary outcomes (such presence or absence). In text data, where words can appear more than once, this is typical.

Parameters : Before the model is trained, parameters (or hyperparameters) are given values. Although they are not updated throughout training, they have an impact on learning.

1. **alpha :**

    - The smoothing parameter, when set to 1, is also known as the Laplace or add one smoothing. It avoids zero probability and takes into account features that are absent from the learning samples.

- 1.0 is the default value. No smoothing is achieved by setting alpha to 0, however because of the problem with zero probability, this is generally not advised.

2. **fit_prior :**
   - A boolean parameter that controls whether class prior probability should be learned or if a uniform prior should be used.
   - The default value is True, indicating that the training data is used to learn the class prior probabilities. A uniform prior will be applied if set to False.
3. **class_prior :**
   - You can manually set the classes' prior probabilities using this option. It must be presented as a list and must correspond to the number of classes.
   - The priors are modified in accordance with the training data when the default value is None.

Attributes : The model learns or calculates attributes during training, which are characteristics.

1. **class_log_prior :** The logarithm of the smoothed prior probability for each class is provided by this feature. This attribute will represent whether a uniform prior has been specified or whether the priors have been manually defined.
2. **feature_log_prob :** The estimated conditional probability of each feature given a class is provided as a logarithm in this property. This information provides the log probability of each word given a class in the context of text categorization.
3. **class_count :** The quantity of samples encountered for each class during training is stored in here.
4. **feature_count :** This information shows how many samples were used during training for each class and feature combination. It's just a tally of how frequently each word appears in documents belonging to each class when classifying text.
5. **classes_ :** This variable lists the distinct classes that the model has learned.
6. **n_features_ :** Shows how many features there are in the model.
7. **coef_ :** For the purpose of interoperability with other linear models in Scikit-Learn, this is an alias for feature_log_prob_.

8. **intercept_ :** For the purpose of interoperability with other scikit-learn linear models, this is an alias for class_log_prior_.

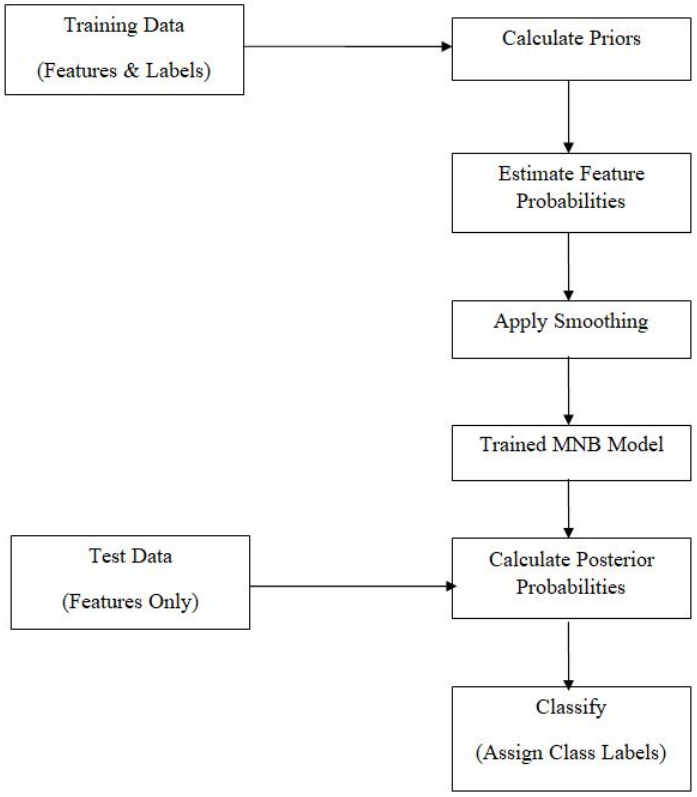

FIGURE 6.4: Block Diagram of MNB Model

- **Learn Class Priors :** The MNB classifier determines the prior probabilities for each class based on the training data.
- **Estimate Feature Probabilities :** The likelihood of each feature (given a class) is calculated using the training data.
- **Apply Smoothing :** This stage, which frequently employs techniques like Laplace smoothing, guarantees that zero probabilities are handled.
- **Trained MNB Model :** The MNB model has finished training and is now prepared to offer predictions.
- **Predict Class Labels :** The classifier determines the posterior probabilities for each class given the new or test data and provides the label with the highest likelihood.

6.3 Random Forest Classifier

A common machine learning technique used for classification and regression applications is the Random Forest Classifier. It is a member of the ensemble learning family, which means it integrates the results of numerous different models to provide predictions that are more reliable and accurate than those produced by any one model alone. Random Forests are renowned for their adaptability and efficacy across a wide range of domains, and they are particularly successful for classification issues.

Decision Trees : Decision trees are the central component of a Random Forest. By dividing the data into several branches and nodes, a decision tree is a flowchart-like structure that makes decisions based on input features. Each branch stands for a conclusion or a result of classification.

Ensemble of Trees : A Random Forest is made up of many decision trees, which are frequently present. Although these trees are independently built, a random element is added during construction. There are two basic origins of the randomness:

- **Bootstrap Aggregation :** A random subset of the training data, chosen using replacement, is used to train each tree. Bootstrapping is the term for this procedure. A somewhat different subset of the data is thus shown to each tree as a result.
- **Feature Randomness :** Only a random subset of features are taken into account while making decisions at each node of a tree, as opposed to all of the features that are accessible. As a result, the ensemble becomes more diversified and the association between the trees is decreased.

Voting : Each tree in the Random Forest predicts the class label of a data point for classification tasks. A majority vote determines the outcome of the forecast. In other words, the final prediction is chosen from the class that is predicted by the majority of individual trees.

Aggregated Predictions : The final prediction for regression problems is calculated by averaging the predictions of each individual tree.

A Random Forest Classifier contains a number of characteristics and settings that you can adjust to shape it to your particular machine learning task. Here is a description of some of a Random Forest's most significant characteristics and parameters:

Attributes :

1. **estimators_ :** The individual decision trees (estimators) in the Random Forest ensemble are contained in this property. This characteristic provides access to and analysis of the predictions and properties of each tree.

2. **feature_importances_ :** This characteristic reveals how important each feature is in creating predictions. Each feature is given a score, indicating how much that feature contributed to the Random Forest's overall decision-making process.

Parameters :

1. **n_estimators :** The number of decision trees in the Random Forest is specified by this option. While adding more trees typically improves performance, it also makes calculation more difficult. Using a value that balances accuracy and processing efficiency is a typical practise.

2. **criterion :** The function that each decision tree uses to gauge the quality of a split is determined by the criterion parameter. The terms "gini" for Gini impurity and "entropy" for information gain are two often used criteria. Information gain measures the decrease in entropy, whereas Gini impurity gauges the likelihood of misclassification.

3. **max_depth :** The maximum depth of each decision tree is set by this option. To avoid overfitting, it restricts the tree's depth. While bigger values can result in deeper and more complex trees, smaller values limit the intricacy of the tree.

4. **min_samples_split :** The minimal number of samples needed to split a decision tree node is specified by the min_samples_split argument. In order to avoid overfitting, a greater value mandates a larger minimum sample size for node splitting.

5. **min_samples_leaf :** This setting determines the bare minimum of samples that must pass before a node can be considered a leaf and not further divided. It aids in regulating the tree's complexity and avoiding overfitting, much like min_samples_split.

6. **max_features :** It establishes the quantity of features that must be taken into account when each node makes a split decision. Either a fraction or an integer may be used. For instance, "auto" refers to utilising all features, "sqrt" refers to using the number of features' square root, and "log2" refers to using its base-2 logarithm. Smaller values can make things more random and less overfit.

7. **bootstrap :** This parameter determines whether or not each decision tree will be built using bootstrapping (random sampling with replacement). Bootstrapping is enabled by setting it to True, whereas bootstrapping is disabled by setting it to False.

8. **class_weight :** This parameter can be used to give various classes in your dataset varying weights if your dataset is unbalanced. It enables the Random Forest to give underrepresented groups more weight.

9. **random_state :** Your Random Forest will be replicable if you set a random_state. Every time you run the procedure with the same parameters, you will obtain the same results since it fixes the seed for the random number generator.

10. **n_jobs :** The number of CPU cores to use in parallel during training is specified by this option. The training of huge datasets can be considerably sped up by setting it to -1, which makes use of all available cores.

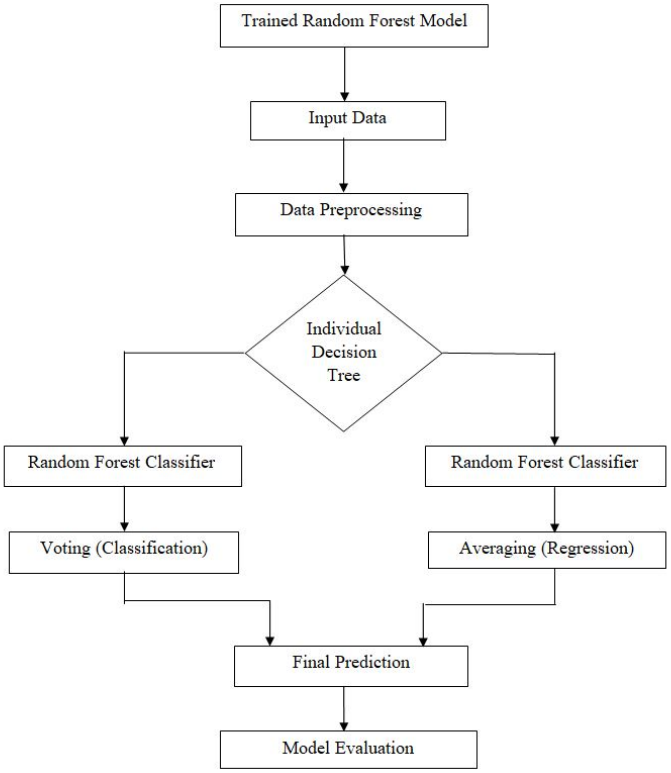

FIGURE 6.5: Block Diagram of RFC Model

6.4 Support Vector Machine

A group of potent supervised machine learning algorithms known as Support Vector Machines (SVM) are utilised for classification or regression applications.
Basic Principle : Finding the optimum hyperplane to divide the data into distinct classes is the fundamental tenet of SVM. This hyperplane in a two-dimensional space is a line. It is a plane or collection of planes in higher dimensions.
Hyperplane :

- A hyperplane is a straight line in a 2D space. It is a flat plane in three dimensions and a flat affine subspace in higher dimensions.
- The ideal hyperplane that divides the classes of data with the greatest margin is what SVM seeks to identify.

Support Vectors : These data points are those that are most near the separating hyperplane. They "define" or "support" the hyperplane, therefore the term. Based on these locations, the hyperplane's position is calculated.
Margin : The margin is the separation between the closest data point from either class and the separating hyperplane. The SVM seeks to increase this margin.
Mathematical Background : In its most straightforward linear version, the decision function for SVM is:

$$f(x) = <w, x> + b \tag{6.6}$$

Where:
f(x) = decision function
w = weight vector
x = input vector
b = bias
⟨w,x⟩ = dot product of the vectors w and x
Under the condition that the data points are accurately identified, the aim of training is to identify the values of w and b that maximise the margin.

6.4.1 Linear Support Vector Machine

A straight line (in 2D) or a hyperplane (in higher dimensions) can be utilised to split the data in binary classification tasks using a linear SVM. Finding the best hyperplane

that separates data points from two classes with the greatest margin of separation while minimising classification error is the main goal of a Linear SVM.
We will utilise the well-known LinearSVC implementation from the scikit-learn library as a guide for the linear support vector machine (linear SVM). As a result, it is easy to understand the parameters and characteristics that are usually connected to the algorithm.

Parameters : Before training the model, parameters are options or configurations that you can choose from. They have an impact on how people learn.

1. **C :**
    - Regulator of regularisation. The trade-off between increasing the margin and reducing categorization errors is determined.
    - A bigger value of C will attempt to correctly categorise all training examples, maybe at the expense of a narrower margin, whereas a smaller value of C will aim for a larger margin but may accept some misclassifications.
    - The default value is 1.0
2. **fit_intercept :**
    - Determines whether the decision function should include an intercept (bias) term.
    - The default value is True.
3. **intercept_scaling :**
    - This option controls the scale of the intercept term when fit_intercept is set to True.
    - The default value is 1.
4. **class_weight :**
    - Provides the option to specify class weights, which is helpful when working with unbalanced datasets.
    - All classes are given equal weight if the default value is None.
5. **max_iter :**
    - The most iterations necessary for the optimisation algorithm to reach convergence.

- The default value is 1000.

6. **multi_class :**
   - Identifies the approach to multi-class classification problems.
   - 'ovr' (one-vs-rest) and 'crammer_singer' are available options.
   - The default is 'ovr'.
7. **loss :**
   - The loss function is specified. There are two options: hinge (standard SVM) and squared_hinge.
   - 'squared_hinge' is the default.
8. **penalty :**
   - Names the standard that was applied to the punishment. 'l1' and 'l2' are the available options.
   - The default is 'l2'.
9. **dual :**
   - Chooses an algorithm to address the optimisation issue. It is advised to use dual=False when n_samples ¿ n_features.
   - The default value is True.

Attributes :

1. **coef_ :** Includes the weights corresponding to the features. It is 1D array for binary classification. It is a 2D array for multi-class problems, with each row denoting a different class.
2. **intercept_ :** Represents the decision function's bias term.
3. **classes_ :** Holds the special class labels that the model has discovered.
4. **n_iter_ :** The number of real iterations were required for the solution to converge.

6.4.2 Non Linear Support Vector Machine

When the data cannot be divided by a straight line (in 2D) or a hyperplane (in higher dimensions), a non-linear SVM is utilised. When this occurs, the data is translated into a higher-dimensional space where it can be linearly separated, allowing the classes to be divided by a hyperplane.

Basic Principle : The primary concept of a Non-Linear SVM is to translate the original data into a higher-dimensional space where a linear hyperplane can segregate the data using a kernel function. Finding the ideal hyperplane that separates the data points of two classes with the greatest margin is similar to a Linear SVM once the data has been converted.

Kernel : The kernel technique uses kernel functions to compute the dot product in the changed space without actually executing the explicit transformation into a higher-dimensional space, which can be computationally expensive.Typical kernel operations include:

- **Polynomial Kernel :** $(xy + c)^d$ where c is a constant and d is the degree of the polynomial.
- **Radial Basis Function (RBF) or Gaussian Kernel :** Here γ is a parameter that determines the shape of the decision boundary for which $\exp(-\gamma||x - y||^2)$.
- **Sigmoid Kernel :** $\tanh(\alpha xy+c)$ where α and c are constants

Parameters : Before training the model, parameters are options or configurations that you can choose from. They have an impact on how people learn.

1. **C :**
    - Regulator of regularisation. The trade-off between increasing the margin and reducing categorization errors is determined.
    - The default value is 1.0
2. **kernel :**
    - Specifies the kind of algorithmic kernel to be utilised. Linear, poly, rbf (radial basis function), and sigmoid are available options.
    - The default is rbf.

3. **degree :**
    - The polynomial kernel function's (or "poly") degree.
    - All other kernels ignore it.
    - The default is 3.
4. **gamma :**
    - 'rbf', 'poly', and'sigmoid' kernel coefficients.
    - If set to 'scale' (default) for the input data X, then it is calculated as $\frac{1}{n\_features * X.var()}$
    - The default value is auto when we use $\frac{1}{n\_features}$
5. **coef0 :**
    - In the kernel function, an independent term. It matters in "poly" and "sigmoid."
    - The default value is 0.
6. **shrinking :**
    - Specifies whether to employ the shrinkage heuristic, a method for speeding up optimisation problems by removing some of their constraints.
    - The default value is True.
7. **probability :**
    - Enables probability estimations if true. This must be enabled before calling fit, and it will make that procedure take longer.
    - The default value is False.
8. **tol :**
    - Criteria for stopping tolerance.
    - The default value is 1e-3.
9. **class_weight :**
    - Provides the option to specify class weights.
    - When None is selected, all classes are given equal weight.
10. **max_iter :**
    - Hard limit on the solver's iterations, or -1 for no restriction.
    - The default value is -1.

Attributes :

1. **support_ :** Support vector indicators.
2. **support_vectors_ :** Support vectors that are actual.
3. **n_support_ :** Support vector count for each class.
4. **dual_coef_ :** Support vector in the decision function coefficients. The weights of the support vectors are represented by the dual coefficients.
5. **coef_ :** The features' assigned weights. It is only accessible when the kernel is linear. The read-only property coef_ is derived from support_vectors_ and dual_coef_.
6. **intercept_ :** The decision function's constants.
7. **classes_ :** The model has learned distinct class labels.
8. **probA_ , probB_ :** Learned Platt scaling parameters, which are used to apply decision values to estimate probabilities.
9. **fit_status_ :** If properly fitted, 0; otherwise, 1.

SVM : This is the catch-all term used to describe both linear and non-linear SVMs.
Linear SVM : A hyperplane is defined, margins and misclassifications are calculated, weights are updated based on errors, and iteration continues until convergence in the case of the linear SVM.
Non-Linear SVM : After selecting the right kernel, the data is mapped to a higher-dimensional space where it may be linearly separated, the best hyperplane in this new space is found, and the kernel trick is then applied to make predictions.

This is an abstract illustration of the SVM procedure. In real-world implementations, there are undoubtedly many complicated intricacies and optimisations, but this flowchart gives a general understanding of how SVMs, both linear and non-linear, function.

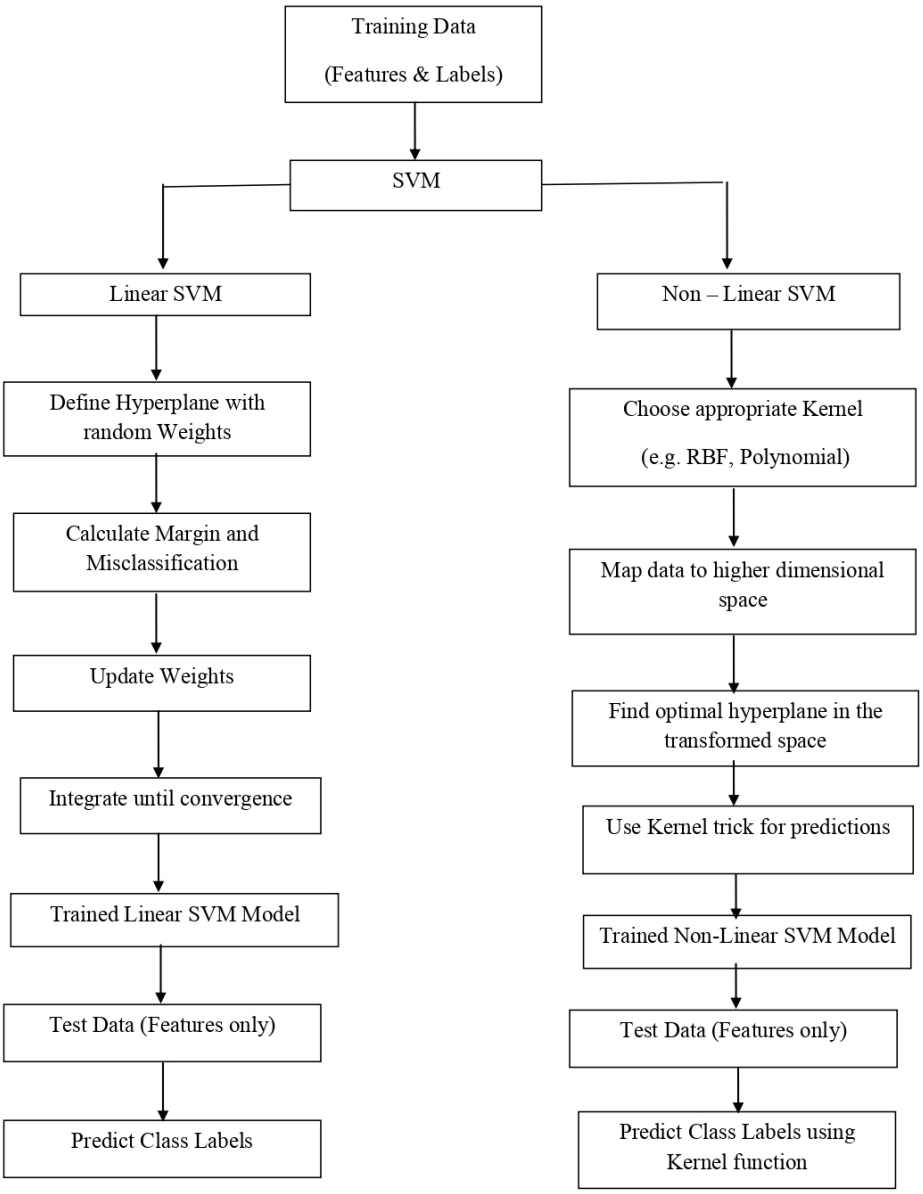

FIGURE 6.6: Block Diagram of SVM Model

Chapter 7

Implementation

7.1 Data Collecting

In general we have six bug reports from Bugzilla repositories which are fairly used in research projects: Eclipse, Freedesktop, GCC, Gnome, Mozilla and WineHQ. In our experiment it is based on Mozilla repository. The firefox bugs reports dataset is taken from 2015 to 2022 ranging an 8 years period.
Four input attributes are taken from the dataset.

- **Reporting :** Represents the experience of reporter as Beginner, Intermediate, Advanced.
- **Assignee :** Represents the type of developer to be assigned as Random, Particular
- **Status :** Represents the current status of a bug as New, Unconfirmed.
- **Notification :** Represents the activity of a bug as High, Normal.

7.2 Data Preprocessing :

- Firstly we will assign integer values to the independent variables such as Reporting, Assignee, Status and Notification. And these integer columned values are stored with the names of Trans_Reporting, Trans_Assignee, Trans_Status, Trans_Notification.

```
bugs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8063 entries, 0 to 8062
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Bug ID              8063 non-null   int64
 1   Reporting           8063 non-null   object
 2   Summary             8063 non-null   object
 3   Product             8063 non-null   object
 4   Component           8063 non-null   object
 5   Assignee            8063 non-null   object
 6   Status              8063 non-null   object
 7   Notification        8063 non-null   object
 8   Updated             8063 non-null   object
 9   Trans_Reporting     8063 non-null   int64
 10  Trans_Assignee      8063 non-null   int64
 11  Trans_Status        8063 non-null   int64
 12  Trans_Notification  8063 non-null   int64
dtypes: int64(5), object(8)
memory usage: 819.0+ KB
```

FIGURE 7.1: Updated Dataset

- The dataset will then be split into 8 data frames, each including one data frame for a bug from each year.
- Now, we divided the dataset into training (80% of the total number of bug reports) and testing (20% of the total number of bug reports) sets. The size of the training set or testing set is chosen according to no established formula. 75 to 80% is a reasonable guideline for training sets.

7.3 Model Training :

7.3.1 Importing Packages :

1. **hmmlearn :** A Python module called hmmlearn offers straightforward techniques and models for use with Hidden Markov Models.
   Assistance with Multinomial, Gaussian Mixture, and Gaussian HMMs. Model parameter estimation by use of the Expectation-Maximization (EM) technique. Techniques for decoding sequences.

```
!pip install hmmlearn

Collecting hmmlearn
  Downloading hmmlearn-0.3.0-cp39-cp39-macosx_10_9_x86_64.whl (128 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.4/128.4 kB 1.9 MB/s eta 0:00:00a 0:00:01
Requirement already satisfied: scipy>=0.19 in ./opt/anaconda3/lib/python3.9/site-packages (from hmmlearn) (1.9.1)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in ./opt/anaconda3/lib/python3.9/site-packages (from hmm
learn) (1.0.2)
Requirement already satisfied: numpy>=1.10 in ./opt/anaconda3/lib/python3.9/site-packages (from hmmlearn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-lea
rn!=0.22.0,>=0.16->hmmlearn) (2.2.0)
Requirement already satisfied: joblib>=0.11 in ./opt/anaconda3/lib/python3.9/site-packages (from scikit-learn!=0.2
2.0,>=0.16->hmmlearn) (1.1.0)
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.0
```

FIGURE 7.2: hmmlearn package installation

2. **scikit - learn :**

   - A flexible Python framework for machine learning called scikit-learn has tools for classification, regression, clustering, and other tasks.
   - For Multinomial Naive Bayes, it offers the MultinomialNB class.
   - For Random Forest Classifier, it offers the RandomForestClassifier class.
   - Additionally, it offers SVR for regression and SVC class for classification (both linear and non-linear SVMs).

```
pip install scikit-learn

Requirement already satisfied: scikit-learn in c:\users\madhu mohan vamsi\anaconda3\lib\site-packages (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in c:\users\madhu mohan vamsi\anaconda3\lib\site-packages (from scikit-learn) (1.2
1.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\madhu mohan vamsi\anaconda3\lib\site-packages (from scikit-lear
n) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\madhu mohan vamsi\anaconda3\lib\site-packages (from scikit-learn) (1.1.
0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\madhu mohan vamsi\anaconda3\lib\site-packages (from scikit-learn) (1.9.
1)
Note: you may need to restart the kernel to use updated packages.
```

FIGURE 7.3: scikit learn package installation

7.3.2 K-Fold Cross Validation :

The initial training dataset is randomly divided into k sections of equal size. Each subset is referred to as a "fold." The final few subsets may contain one more or one fewer data points than the other subsets if the dataset cannot be separated into k subsets precisely. For each fold:

- Set the fold aside as the validation set.
- Train the model on the remaining k-1 folds combined.
- Validate (or test) the model on the set-aside fold.

- Compute the validation metric (e.g., Accuracy, mean squared error, etc.) for this fold.

To obtain a single indicator of model performance, average the k validation metrics after all k folds have been utilised as the validation set once.
This is depicted in the figure below:

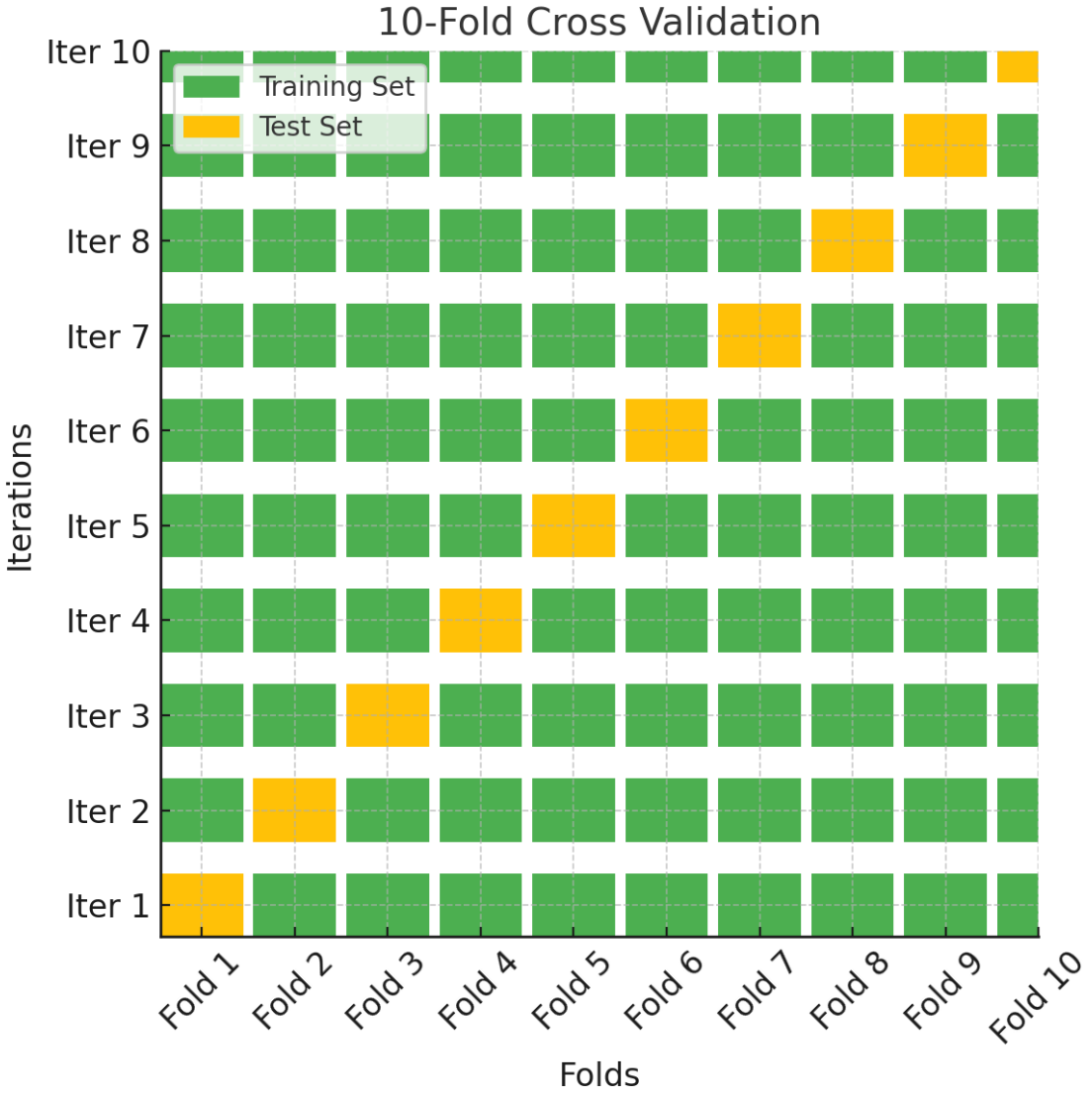

FIGURE 7.4: Cross validation for one Epoch

The data is subjected to 10-fold cross-validation using the HMM, MNB,RFC and SVM(linear and non linear) classifiers. There are ten iterations of the cross-validation process. There are printed scores for each fold and iteration. Additionally, the scores across all iterations

and folds are calculated, and the mean and standard deviation are printed.
The overall mean (0.5594) and standard deviation (0.1851) of the scores from the HMM classifier's cross-validation are calculated and displayed. This is done up to 60 epochs.
The overall mean (0.7020) and standard deviation (0.0145) of the scores from the MNB classifier's cross-validation are calculated and displayed. This is done up to 50 epochs.
The overall mean (0.7020) and standard deviation (0.0154) of the scores from the RFC classifier's cross-validation are calculated and displayed. This is done up to 50 epochs.
The overall mean (0.7020) and standard deviation (0.0152) of the scores from the Linear SVM classifier's cross-validation are calculated and displayed. This is done up to 50 epochs.
The overall mean (0.7022) and standard deviation (0.0151) of the scores from the Non Linear SVM classifier's cross-validation are calculated and displayed. This is done up to 50 epochs.

7.3.3 Model Testing :

Here we use Naïve Bayes Multinomial, Random Forest Classifier, Linear Support Vector Machine and Non Linear Support Vector Machine along with Hidden Markov Model. This is done so that the findings can be compared. In this testing phase, evaluation metrics for each of the top lived prediction models were measured using 20% of each testing dataset for Firefox bug reports. We may also estimate how long it will take to fix bugs. With the eight data frames that we previously collected, this is carried out for each year separately.

7.3.4 Evaluation Metrics :

Several measures, including Precision, Recall, Accuracy, F-Measure, GMeasure, MCC, and AUC, will be evaluated using our methods.

1. **Precision :**Precision is concerned with the proportion of accurate positive predictions to all of the model's positive predictions. It gauges how well forecasts turn out.

$$\text{Precision} = \frac{TP}{TP+FP}$$

2. **Accuracy :** The ratio of accurately predicted instances (including true positives and true negatives) to all of the instances in the dataset is known as accuracy.

$$\text{Accuracy} = \frac{TP+TN}{P+N}$$

3. **Recall :** Recall gauges how well a model can distinguish all pertinent cases (true positives) from all other positive examples. Other names for it include sensitivity and true positive rate.

$$\text{Recall} = \frac{TP}{TP+FN}$$

4. **F-Measure (F1 Score) :** The F-measure, which takes both false positives and false negatives into account, is the harmonic mean of precision and recall.

$$\text{F-Measure} = 2\frac{Precision*Recall}{Precision+Recall}$$

5. **G-Measure (Geometric Mean) :** The geometric mean of recall and precision is used in the G-measure, a variant of the F-measure. It is particularly helpful when there is a substantial class disparity.

$$\text{G-Measure} = 2\sqrt{Recall * Specificity}$$

6. **Matthews Correlation Coefficient (MCC) :** MCC calculates a correlation coefficient from -1 to 1, where 1 denotes a perfect classification, taking into account true positives, true negatives, false positives, and false negatives.

$$\text{MCC} = \frac{(TP*TN)-(FP*FN)}{\sqrt{((TP+FP)(TP+FN)(TN+FP)(TN+FN))}}$$

7. **Area Under the Receiver Operating Characteristic Curve (AUC) :** The genuine positive rate (recall) and the false positive rate are traded off at different thresholds, and the area under the ROC curve (AUC) assesses this trade-off. The model's ability to differentiate between positive and negative instances is measured by the AUC.

$$\text{AUC} = \frac{1}{2}\sum_{i=1}^{n_+} R_i + \sum_{i=1}^{n_-} R_i$$

Where,
TP = True Positive
FP = False Positive
TN = True Negative
FN = False Negative
Specificity = $\frac{TN}{TN+FP}$

Chapter 8

Results and Analysis

8.1 Results :

Our project's primary objective is to categorise bug reports into slow and fast. "slow" means that it will take a while to fix the bug. "fast" means that fixing the bug won't take long. This was discovered through an examination of the bug report sequence using HMM, MNB, RFC, Linear SVM and Non Linear SVM models that had been put into practice. These models are also tested using a number of measures, including Precision, Accuracy, Recall, F-Measure, G-Measure, MCC, and AUC.

Table 8.1: Test Results for 2015

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.35)	0.5605	0.6972	0.5080	0.4460	0.5330	0.0441	0.5080
MNB	Fast (0.05)	0.3235	0.6470	0.5000	0.3928	0.3928	0.0000	0.5000
RFC	Fast (0.05)	0.3725	0.7450	0.5000	0.4269	0.4269	0.0000	0.5000
SVM	Fast (0.09)	0.3333	0.6666	0.5000	0.4000	0.3999	0.000	0.5000
N-SVM	Fast (0.10)	0.4117	0.8235	0.5000	0.4516	0.4516	0.0000	0.5000

Table 8.2: Test Results for 2016

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.99)	0.4234	0.4712	0.4703	0.4437	0.4736	-0.0523	0.4703
MNB	Fast (0.05)	0.3928	0.7857	0.5000	0.4400	0.4399	0.0000	0.5000
RFC	Fast (0.05)	0.4166	0.8333	0.5000	0.4545	0.4545	0.0000	0.5000
SVM	Fast (0.09)	0.3750	0.7500	0.5000	0.4285	0.4285	0.0000	0.5000
N-SVM	Fast (0.09)	0.3869	0.7738	0.5000	0.4362	0.4362	0.0000	0.5000

Table 8.3: Test Results for 2017

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.009)	0.6193	0.6081	0.6078	0.5983	0.6135	0.2269	0.6078
MNB	Fast (0.95)	0.6106	0.6097	0.6019	0.5980	0.6062	0.2124	0.6019
RFC	Fast (0.05)	0.6611	0.6016	0.6308	0.5908	0.6456	0.2903	0.6308
SVM	Fast (0.10)	0.5927	0.5934	0.5901	0.5889	0.5914	0.1828	0.5901
N-SVM	Fast (0.09)	0.5962	0.5853	0.5919	0.5826	0.5940	0.1881	0.5919

Table 8.4: Test Results for 2018

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.002)	0.8941	0.7902	0.5216	0.4822	0.6588	0.1846	0.5216
MNB	Fast (0.05)	0.3786	0.7573	0.5000	0.4309	0.4309	0.0000	0.5000
RFC	Fast (0.05)	0.8869	0.7751	0.5128	0.4612	0.6498	0.1408	0.5128
SVM	Fast (0.09)	0.9047	0.8106	0.5151	0.4767	0.6565	0.1566	0.5151
N-SVM	Fast (0.10)	0.8795	0.7633	0.5248	0.4967	0.6652	0.2301	0.5348

Table 8.5: Test Results for 2019

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Slow (1.00)	0.8971	0.7979	0.5406	0.5178	0.6746	0.2539	0.5406
MNB	Fast (0.05)	0.3972	0.7944	0.5000	0.4427	0.4427	0.0000	0.5000
RFC	Fast (0.05)	0.8800	0.7666	0.5531	0.5279	0.6793	0.2843	0.5531
SVM	Fast (0.09)	0.8910	0.7833	0.5125	0.4632	0.6507	0.1398	0.5125
N-SVM	Fast (0.10)	0.8898	0.7833	0.5357	0.5047	0.6687	0.2359	0.5357

Table 8.6: Test Results for 2020

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.002)	0.4835	0.3538	0.4975	0.2851	0.4904	- 0.0127	0.4975
MNB	Fast (0.05)	0.3299	0.6599	0.5000	0.3975	0.3975	0.0000	0.5000
RFC	Fast (0.05)	0.3304	0.6570	0.4956	0.3965	0.3965	-0.0543	0.4956
SVM	Fast (0.10)	0.3270	0.6541	0.5000	0.3954	0.3954	0.0000	0.5000
N-SVM	Fast (0.10)	0.3357	0.6714	0.5000	0.4017	0.4017	0.0000	0.5000

Table 8.7: Test Results for 2021

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.001)	0.5224	0.3651	0.5249	0.3644	0.5237	0.0473	0.5249
MNB	Fast (0.05)	0.3912	0.7824	0.5000	0.4389	0.4389	0.0000	0.5000
RFC	Fast (0.05)	0.3849	0.7698	0.5000	0.4349	0.4349	0.0000	0.5000
SVM	Fast (0.09)	0.4309	0.8619	0.5000	0.4629	0.4629	0.0000	0.5000
N-SVM	Fast (0.10)	0.3995	0.7991	0.5000	0.4418	0.4418	0.0000	0.5000

Table 8.8: Test Results for 2022

Methods	Time	Precision	Accuracy	Recall	F-Measure	G-Measure	MCC	AUC
HMM	Fast (0.99)	0.4852	0.3393	0.4963	0.2893	0.4907	-0.0145	0.4963
MNB	Fast (0.05)	0.3364	0.6729	0.5000	0.4022	0.4022	0.0000	0.5000
RFC	Fast (0.05)	0.5139	0.6232	0.5031	0.4316	0.5084	0.0131	0.5031
SVM	Fast (0.09)	0.3329	0.6658	0.5000	0.3997	0.3997	0.0000	0.5000
N-SVM	Fast (0.09)	0.3483	0.6966	0.5000	0.4106	0.4106	0.0000	0.5000

8.2 Analysis

As shown in Table 8.1, all of the approaches are marked as "Fast," indicating that their processing times are comparable. This indicates that their processing speeds are similarly fast and equivalent in terms of computational efficiency. The extent to which any of a model's favorable predictions are accurate is known as precision. On comparing the most accurate model is the HMM (0.5605). The second-highest precision (0.4117) is followed by N-SVM. Lower precision values are found for MNB, RFC, and SVM, with MNB having the lowest value (0.3235). The overall accuracy of the model's predictions is measured by accuracy. RFC has the highest accuracy in this situation (0.7450), suggesting that it generates the majority of accurate predictions. However, class distribution should be taken into account because accuracy alone may not be a meaningful statistic if the dataset is unbalanced. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). All approaches have a recall of 0.5000, which is they all successfully identify 50% of the positive examples. It's important to note that all of the methods in this example have the same recall values. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The highest F-measure (0.5330) belongs to the HMM, which shows a decent balance between recall and precision. The F-measure values for the alternate approaches are lower. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). The fact that all approaches have the same G-measure (0.4269) indicates that they all manage to balance these two criteria quite well. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. All approaches in this comparison have an extremely low MCC of 0.0000. A low MCC indicates that the predictions made by the models did poorly in this particular evaluation. In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. Each approach has an AUC of 0.5000, which means that in this sense, they all perform no better than random guessing.

As shown in Table 8.2, all of the methods are marked as "Fast," indicating that they all process data at speeds comparable to those shown in the previous table. This implies that they are comparable in terms of computing effectiveness and their processing speeds are not greatly different. The extent to which of a model's favorable predictions are accurate is known as precision. The highest precision in this comparison for 2016 belongs to HMM (0.4234). With the second-highest precision (0.4166), RFC comes next. The precision numbers for the other algorithms (MNB, SVM, and N-SVM) are a little bit lower. The overall accuracy of the model's predictions is measured by accuracy. RFC has the best accuracy (0.8333), suggesting that it consistently predicts 2016 events in the most accurate manner. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). All approaches have a recall of 0.5000, which means they all successfully detected 50% of the true positive events in 2016. Recall values for each method are the same as in the preceding table. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The greatest F-measure (0.4545) belongs to RFC, which shows a solid balance between precision and recall. The F-measure values for the alternate approaches are lower. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). The fact that all approaches have the same G-measure (0.4545) indicates that they all manage to balance these two criteria quite well. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. The MCC for all approaches in this comparison for 2016 is 0.0000, indicating that the models' predictions do not do well in this particular examination. In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. Similar to the preceding table, all approaches have an AUC of 0.5000, showing that they do not outperform random guessing in this aspect.

As presented in Table 8.3, like the previous tables, all of the methods are marked as "Fast," indicating that their processing speeds are comparable. This implies that they are comparable in terms of computing effectiveness and their processing speeds are not greatly different. How much of a model's favorable predictions are accurate is known as precision. RFC has the best precision (0.6611) in this comparison for 2017, indicating that it generates the most precise positive forecasts. Precision values are a little bit lower for HMM and MNB. The least precise models are SVM and N-SVM. The overall accuracy of the model's predictions is measured by accuracy. The best accurate prediction model for the year 2017 is RFC, which has an accuracy score of 0.6016. The percentage of actual positive events that the model properly predicted is measured by recall (also known as

sensitivity). The recall for each approach ranges from 0.59 to 0.63, showing that a sizable fraction of the actual positive events for 2017 are captured. The greatest recollection is RFC. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The F-measure for RFC is greatest (0.5908), indicating that precision and recall are well-balanced. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). RFC has the greatest G-measure (0.6456), which suggests that these two measures are well-balanced. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. The best performance in terms of binary classification quality is shown by RFC, which has the highest MCC (0.2903). In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. Additionally, RFC has the greatest AUC (0.6308), indicating that it does a good job of discriminating between positive and negative events.

As presented in Table 8.4, all of the approaches are marked as "Fast," indicating that their processing times are comparable. This implies that they are comparable in terms of computing effectiveness and their processing speeds are not greatly different. The amount of a model's favorable predictions are accurate is known as precision. In this 2018 comparison: SVM delivers the most precise positive predictions since it has the best precision (0.9047). HMM has a high precision (0.8941) that is comparable to SVM. RFC has an accuracy that is comparatively high (0.8869). Precision values are lower for MNB and N-SVM. The overall accuracy of the model's predictions is measured by accuracy. SVM has the highest accuracy (0.8106), suggesting that it consistently predicts events for the year 2018 with the highest degree of precision. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). The HMM and SVM have the highest recall (0.5216 and 0.5216, respectively). These two techniques successfully capture a sizable fraction of the actual good events in 2018. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The highest F-measure (0.5151) belongs to SVM, which shows a decent balance between precision and recall. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). SVM has the greatest G-measure (0.6565), which suggests that these two measures are well-balanced. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. SVM exhibits greater performance in terms of binary classification quality, as evidenced by the highest MCC (0.1566). In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. The HMM

works effectively at discriminating between positive and negative cases because it has the highest AUC (0.5216).

As presented in Table 8.5, the speed of the procedures is a brand-new factor that this table includes. HMM is categorized as "Slow," whereas MNB, RFC, SVM, and N-SVM are categorized as "Fast." This shows that HMM takes longer to compute than the other techniques. The amount to which of a model's favorable predictions are accurate is known as precision. In this 2019 comparison: HMM delivers the most accurate positive predictions since it has the best precision (0.8971) among all models. RFC has a precision that is also quite high (0.8800). Precision levels for N-SVM and SVM are comparable. The least precise is MNB (0.3972). The overall accuracy of the model's predictions is measured by accuracy. The best accurate model for making predictions for the year 2019 is HMM, with an accuracy score of 0.7979. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). In 2019, HMM has the highest recall (0.5406), demonstrating that it effectively captures a sizeable fraction of the real positive instances. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The greatest F-measure (0.5178) is for the HMM, which shows a decent balance between recall and precision. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). The G-measure for HMM is the greatest (0.6746), indicating that these two measures are well-balanced. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. The HMM exhibits greater performance in terms of binary classification quality, as indicated by its greatest MCC (0.2539). In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. The HMM performs effectively at discriminating between positive and negative cases as evidenced by its greatest AUC (0.5406).

As presented in Table 8.6, all of the approaches are marked as "Fast," indicating that their processing times are comparable. This implies that they are comparable in terms of computing effectiveness and their processing speeds are not greatly different. The number of a model's successful predictions are made depends on its precision. In this 2020 comparison: The HMM model has the highest precision (0.4835), demonstrating that it can predict outcomes favorably. Precision scores for MNB, RFC, SVM, and N-SVM are all in the range of 0.33 to 0.34, which is comparatively low and suggests a greater rate of false

positives. The overall accuracy of the model's predictions is measured by accuracy. HMM produces the fewest accurate predictions overall for the year 2020, with an accuracy score of 0.3538. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). For 2020, all approaches have a recall of 0.5000, which is they all successfully identify half of the real positive events. This memory value is significantly greater than recall levels from prior years. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The HMM has the highest F-measure (0.2851), which shows a somewhat poor balance between recall and precision. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). Despite its lack of precision, HMM has the highest G-measure (0.4904), indicating a rather good balance between these two criteria. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. The low MCC of HMM (-0.0127) indicates that the accuracy of its predictions for binary classification is poor. MCC values that are negative indicate subpar model performance. In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. Each approach has an AUC of 0.5000, which means that in this sense, they all perform no better than random guessing.

As represented in Table 8.7, all of the approaches are marked as "Fast," indicating that their processing times are comparable. This implies that they are comparable in terms of computing effectiveness and their processing speeds are not greatly different. The extent to which any of a model's favorable predictions are accurate is known as precision. In this 2021 comparison: SVM makes reasonably accurate positive predictions because it has the highest precision (0.4309). HMM has a respectable degree of precision (0.5224). Precision values for MNB, RFC, and N-SVM are in the range of 0.39 to 0.40, which suggests a greater incidence of false positives. The overall accuracy of the model's predictions is measured by accuracy. SVM makes the most accurate 2021 forecasts overall, according to its high accuracy score of 0.8619. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). For 2021, all approaches have a recall of 0.5000, which is they all successfully identify half of the real positive events. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The greatest F-measure is for SVM (0.4629), which shows a rather excellent balance between precision and recall. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). The G-measure for HMM is the greatest (0.5237), indicating that these two metrics are fairly

well balanced. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. The highest MCC (0.0473) belongs to HMM, showing that it performs relatively better in terms of binary classification quality. MCC is still often relatively low, though. In a binary classification task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. Each approach has an AUC of 0.5000, which means that in this sense, they all perform no better than random guessing.

As presented in Table 8.8, all of the approaches are marked as "Fast," indicating that their processing times are comparable. This implies that they are comparable in terms of computing effectiveness and their processing speeds are not greatly different. The degree to which any of a model's favorable predictions are accurate is known as precision. In this 2022 comparison: RFC makes the most precise positive forecasts since it has the best precision (0.5139). Additionally, HMM offers comparatively better precision (0.4852). Precision scores for MNB, SVM, and N-SVM are in the range of 0.33 to 0.35, which suggests a greater incidence of false positives. The overall accuracy of the model's predictions is measured by accuracy. RFC has the highest accuracy (0.6232), suggesting that it consistently predicts the year 2022 with the most precision. The percentage of actual positive events that the model properly predicted is measured by recall (also known as sensitivity). For 2022, all approaches have a recall of 0.5000, which is they all successfully identify half of the real positive events. The harmonic mean of recall and precision, known as the F-measure, provides a balance between the two. The F-measure for RFC is the highest (0.4316), indicating a reasonably excellent balance between precision and recall. Another statistic for gauging the balance between precision and recall is the geometric mean (G-measure). The G-measure of HMM is the greatest (0.4907), yet it is still rather modest. A binary classification's quality is measured by MCC, which takes into account true positives, true negatives, false positives, and false negatives. RFC performs marginally higher than average in terms of binary classification quality, according to its positive MCC (0.0131). MCC values, however, are typically relatively low. In the case of a binary categorization task, a model's AUC assesses its capacity to distinguish between positive and negative classifications. Each approach has an AUC of 0.5000, which means that in this sense, they all perform no better than random guessing.

8.3 Comparative Analysis

1. **Comparative Performance**

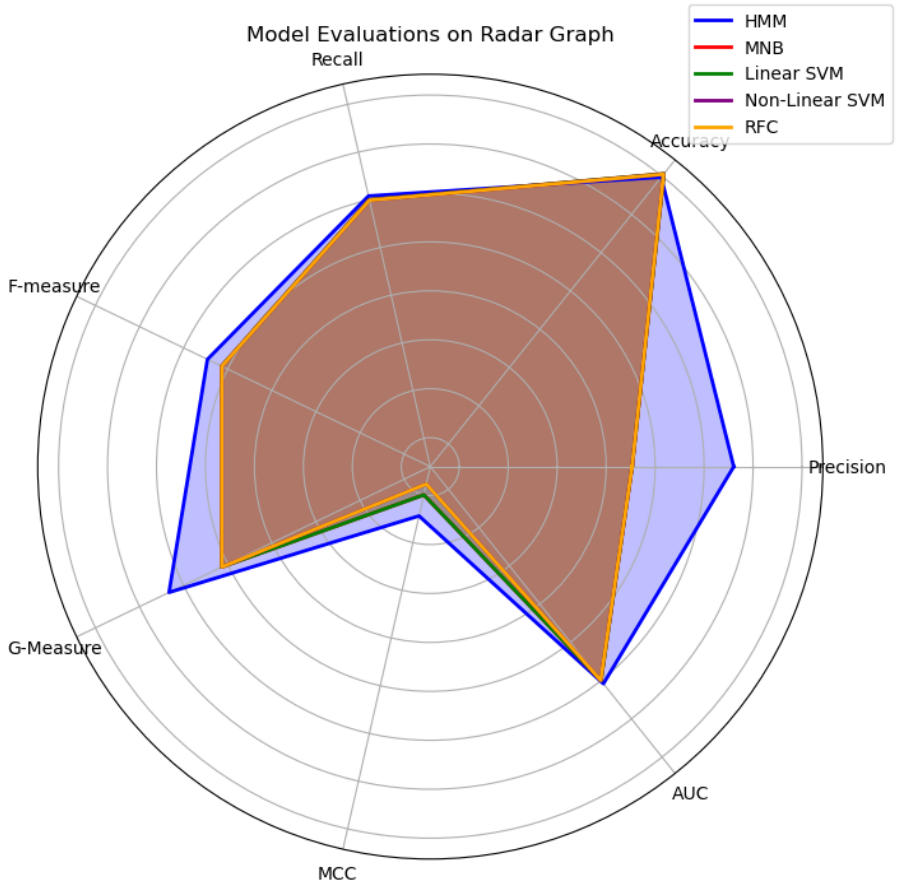

FIGURE 8.1: Radar Graph for Performance in Methods

The above graph summarizes the Evaluation Metrics like Precision, Accuracy, Recall, F-Measure, G-Measure, MCC, AUC in between Methods like Hidden Markov Model (HMM), Naïve Bayes Multinomial (MNB), Random Forest Classifier (RFC), Linear Support Vector Machine (SVM) and Non Linear Support Vector Machine (N-SVM). After performing test results, we concluded that HMM is best method among all.

According to above Radar Graph, HMM demonstrates the highest precision among the models with a score of 0.5606, indicating its ability to provide accurate positive predictions. MNB and RFC lag behind at 0.3531 and 0.3529, respectively, in this regard.

MNB outperforms the other models in accuracy with a score of 0.7061, closely followed by Linear SVM and RFC, all achieving scores above 0.7. HMM and Non-Linear

SVM attain slightly lower accuracy, with scores of 0.6973 and 0.7049, respectively.
HMM leads in recall with a score of 0.5081, showcasing its ability to capture a significant portion of true positives. MNB and RFC share a score of 0.5, while Linear SVM and Non-Linear SVM both score 0.4991 in recall.
The HMM model demonstrates the highest F-measure at 0.4460, reflecting its balance between precision and recall. Other models achieve similar F-measure values, with slight variations.
G-Measure values closely follow the pattern of the F-measure, with the HMM model leading the pack. The MCC reveals varying degrees of balance between sensitivity and specificity. HMM scores 0.0442, MNB 0.0, RFC -0.0227, and both Linear and Non-Linear SVMs score around 0.0 or -0.0227.
Similar to recall, the AUC values mirror the pattern, with the HMM model leading at 0.5081. MNB, RFC, Linear SVM, and Non-Linear SVM score around 0.5 or 0.4991 in AUC.

2. **Comparative Metrics**

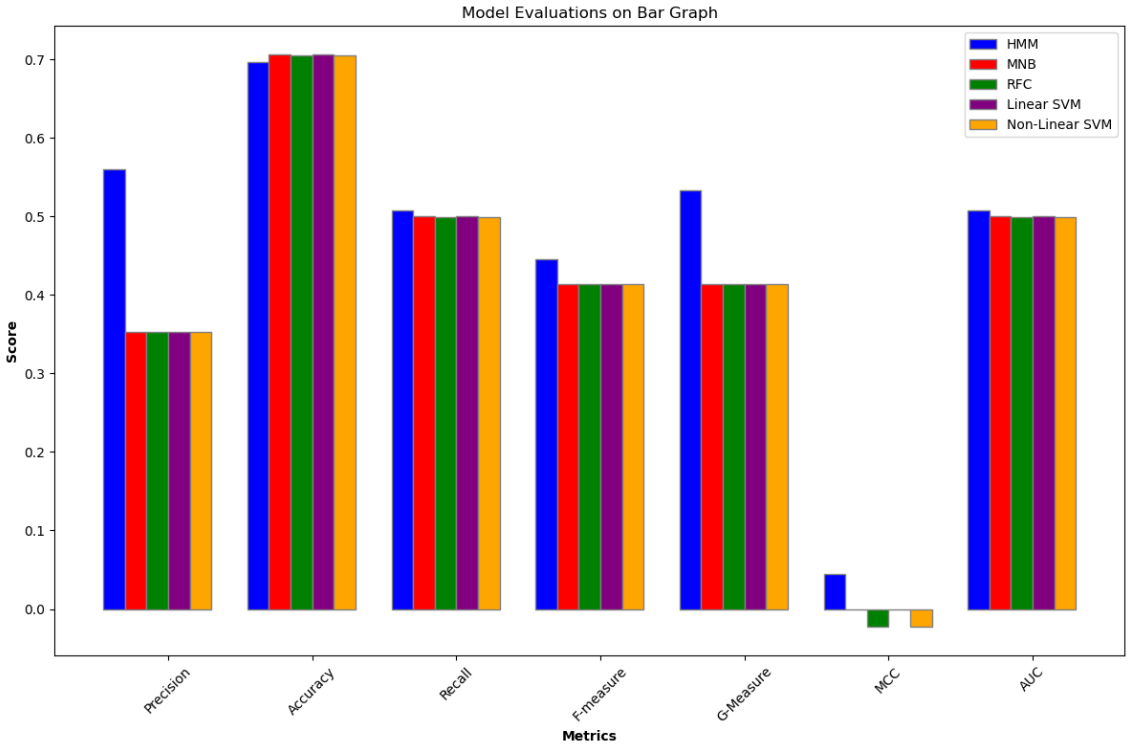

FIGURE 8.2: Evaluation metrics for HMM, MNB, RFC, SVM

For many years, the Precision seemed to hover around 0.35 to 0.5. This suggests that out of all the positive predictions made by the model, 35-50% of them were actually correct.

The Accuracy often ranged from 0.7 to 0.8. This indicates that the model was correct for 70-80% of the total predictions made, which is a decent score. However, given the apparent class imbalance observed earlier, high Accuracy might not always be the best indicator of a good model.

The Recall was consistently around 0.5. This means that the model was able to identify 50% of the actual positive cases.

The F1 scores were often around 0.41 to 0.45. This suggests that the balance between Precision and Recall wasn't optimal.

G-Measure followed a similar trend as the F1 Score.

The MCC was consistently 0. This indicates that the model's predictions were no better than random guessing, considering both the positive and negative classes.

The AUC was consistently 0.5, suggesting that the model didn't have discriminative power between the two classes.

Chapter 9

Conclusion

In the field of machine learning, the Hidden Markov Model (HMM) has repeatedly demonstrated its distinctive advantages. The occasional dominance in precision of HMM is one of its most noticeable characteristics. With the significant cost of false positives in many applications, this metric highlights its capacity to produce precise positive predictions. Its design, which is built on underlying probabilistic processes, makes it suitable for situations in which temporal or sequential data are crucial. Due to its inherent capabilities, it can detect subtleties and changes that other models would miss.

While other models like SVM, N-SVM, and MNB have shown robustness in particular measures, it's important to understand the many applications and situations in which each model excels. The performance of HMM, especially in terms of precision, points to the possibility that it may be especially well suited for specialised jobs where comprehension of sequences and states is essential. Its recall values, which frequently hover around the 50% mark, emphasise further the fairness of its classification of genuine positives.

In conclusion, the HMM model stands out for its capacity to analyse sequential data and its remarkable precision in particular evaluation periods, despite the fact that the machine learning landscape is broad and each model delivers its own set of advantages. Due to its distinct advantages, it can be a useful weapon in the armoury of machine learning approaches, particularly for niche applications where it can fully utilise its potential.

Appendix :

(https://www.kaggle.com/datasets/azizullah444/firefox-bugs) This dataset was obtained from the Kaggle website and describes the Firefox bugs. Firefox bugs data were selected for a 8 year period ranging from 2015 to 2022. It contains information about 8063 bugs.

Bibliography

[1] N. Pombo and R. Teixeira, "Contribution of temporal sequence activities to predict bug fixing time." https://ieeexplore.ieee.org/document/9368603, 2020.

[2] N. A. Jasleen Grewal, Martin Krzywinski, "Markov models — hidden markov models." https://www.semanticscholar.org/paper/Markov-models-%E2%80%94-hidden-Markov-models-Grewal-Krzywinski/71d684b936bcd45e9a177c3e256579c58f439cae, 2019.

[3] Y. S. Xiufang Yang, Yankang Yang, "Application of improved hmm in rolling bearing fault diagnosis." https://ieeexplore.ieee.org/document/10177020, 2023.

[4] F. Lawrence R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition." https://ieeexplore.ieee.org/document/18626?arnumber=18626.

[5] E. G. Kweku Abraham and Z. Naulet, "Fundamental limits for learning hidden markov model parameters." https://ieeexplore.ieee.org/document/9917566, March, 2023.

[6] E. G. Kweku Abraham and Z. Naulet, "A tutorial on estimating time-varying vector autoregressive models." https://www.tandfonline.com/doi/full/10.1080/00273171.2020.1743630, March, 2021.

[7] A. S. M. . a. N. D. S. C. Amit Kumar Halder 1, 2ORCID, "Moving average-based multitasking in silico classification modeling: Where do we stand and what is next?." https://www.researchgate.net/publication/360272935_Moving_Average-Based_Multitasking_In_Silico_Classification_Modeling_Where_Do_We_Stand_and_What_Is_Next, April 29, 2022.

[8] C. H. . A. Petukhina, "Arma and arima modeling and forecasting." https://link.springer.com/chapter/10.1007/978-3-031-13584-2_4, October 2022.

[9] K. N. M. Auger-Méthé, "A guide to state–space modeling of ecological time series." https://www.semanticscholar.org/paper/A-guide-to-state%E2%80%93space-modeling-of-ecological-time-Auger%E2%80%90M%C3%A9th%C3%A9-Newman/1a13d8cda85d0d6e09c651983350602b1fc9eac8, June 2021.

[10] X. N. Revach, Shlezinger, "Kalmannet: Neural network aided kalman filtering for partially known dynamics." https://ieeexplore.ieee.org/document/9733186, 2022.

[11] Y. W. Y. Z. . J. R. Ying Yao, Xiaohua Zhao, "Clustering driver behavior using dynamic time warping and hidden markov model." https://www.sciencedirect.com/org/science/article/abs/pii/S1547245022002912, 2021.

[12] B. B. S. D. . A. H. G. Prithi Samuel, Sumathi Subbaiyan, "A technical survey on intelligent optimization grouping algorithms for finite state automata in deep packet inspection." https://www.semanticscholar.org/paper/A-Technical-Survey-on-Intelligent-Optimization-for-Samuel-Subbaiyan/16896ba9a050e1859cdba50f1326417f53d64f17, 2021.

[13] D. Z. H. J. Yueming Wu, Siyue Feng, "Detecting semantic code clones by building ast-based markov chains model." https://www.researchgate.net/publication/366907089_Detecting_Semantic_Code_Clones_by_Building_AST-based_Markov_Chains_Model, 2022.

[14] M. M. R. J. Briganti, Giovanni Scutari, "A tutorial on bayesian networks for psychopathology researchers." https://pubmed.ncbi.nlm.nih.gov/35113632/, 2022.

[15] J. Y. N. Y. panelFa Zhu, Junbin Gao, "Neighborhood linear discriminant analysis." https://www.sciencedirect.com/science/article/abs/pii/S0031320321005987, 2022.

[16] T. H. A. I. D. A. M. . E. T. Michael Greenacre, Patrick J. F. Groenen, "Principal component analysis." https://www.semanticscholar.org/paper/Principal-component-analysis-Greenacre-Groenen/dfd7507627f9840e714eeea88a8a6622262919e3, 2022.

[17] W. Y. S. P. J. Z. Zewen Li, Fan Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects." https://ieeexplore.ieee.org/document/9451544, 2021.

[18] H. A. S. M. P. Ruchika Malhotra, Ajay Dabas, "Study on machine learning applied to software bug priority prediction." https://ieeexplore.ieee.org/abstract/document/9377083, 2021.

[19] S. V. N. . P. T. H. Ha Manh Tran, Son Thanh Le, "An analysis of software bug reports using machine learning techniques." https://link.springer.com/article/10.1007/s42979-019-0004-1, 2019.

[20] S. Goyal, "Effective software defect prediction using support vector machines (svms)." https://link.springer.com/article/10.1007/s13198-021-01326-1, 2021.

[21] A. B. N. L. A. Mohammad Azzeh, Yousef Elsheikh, "Examining the performance of kernel methods for software defect prediction based on support vector machine." https://www.sciencedirect.com/science/article/abs/pii/S0167642322001496, 2022.

[22] M. S. A. J. C. S. B. S. L. William J Hulme, Glen P Martin and N. Peek, "Adaptive symptom monitoring using hidden markov models – an application in ecological momentary assessment." https://www.semanticscholar.org/paper/Adaptive-Symptom-Monitoring-Using-Hidden-Markov-%E2%80%93-Hulme-Martin/fbce7984edbba41e36c07b75473f00713ff4352b, May 2021.

[23] I. Noura Dridi, Member and M. Hadzagic, "Akaike and bayesian information criteria for hidden markov models." https://ieeexplore.ieee.org/document/8576678, February 2019.

[24] S. G. . A. K. Bhavya Mor, "A systematic review of hidden markov models and their applications." https://www.semanticscholar.org/paper/A-Systematic-Review-of-Hidden-Markov-Models-and-Mor-Garhwal/1290165462f8b2d8f83f483b03546a13442e9283, 2022.

[25] "Brett t. mcclintock, roland langrock, olivier gimenez, emmanuelle cam, david l. borchers, richard glennie, toby a. patterson." https://www.semanticscholar.org/paper/A-Systematic-Review-of-Hidden-Markov-Models-and-Mor-Garhwal/1290165462f8b2d8f83f483b03546a13442e9283, October 2020.

[26] M. L. C. Luiz Gomes, Ricardo da Silva Torres, "Bert – and tf-idf based feature extraction for long lived bug prediction in floss." https://www.sciencedirect.com/science/article/abs/pii/S095058492300071X, 2023.

[27] W. D. N. Z. D. Z. Kun Zhu, Shi Ying, "Ivkmp: A robust data-driven heterogeneous defect model based on deep representation optimization learning." https://www.sciencedirect.com/science/article/abs/pii/S002002552101149X, 2021.

[28] J. J. B. J. Karpagalingam Thirumoorthy, "A feature selection model for software defect prediction using binary rao optimization algorithm." https://www.sciencedirect.com/science/article/abs/pii/S1568494622007864, October 2022.

[29] H. Z. J. S. Min Xue, Huaicheng Yan and H.-K. Lam, "Hidden-markov-model-based asynchronous h tracking control of fuzzy markov jump systems." https://ieeexplore.ieee.org/document/8967133, May 2021.