BuggyDe: A Machine Learning based Technique for Software Defect and Bug Severity Level Classification

by

Anik Paul Shuvo (201014051)
Nayla Muqim (201014100)
Ahammad Ullah (201014077)
Tasnima Yeasmin Tumpa (201014038)

Capstone project report (CSE 499) submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

Under the supervision of

Suravi Akhter
Lecturer

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY OF LIBERAL ARTS BANGLADESH

SPRING 2024

DECLARATION

Project Title	BuggyDe: A Machine Learning based Technique for Software Defect and Bug Severity Level Classification
Authors	*Anik Paul Shuvo, Nayla Muqim, Ahammad Ullah and Tasnima Yeasmin Tumpa*
Student IDs	201014051, 201014100, 201014077 and 201014038
Supervisor	Suravi Akhter

We declare that this capstone project report entitled *BuggyDe: A Machine Learning based Technique for Software Defect and Bug Severity Level Classification* is the result of our own work except as cited in the references. The capstone project report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Anik Paul Shuvo
201014051

Nayla Muqim
201014100

Ahammad Ullah
201014077

Tasnima Yeasmin Tumpa
201014038

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Date: June 3, 2024

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh
Mohammadpur, Dhaka - 1207

CERTIFICATE

This is to certify that the capstone project report entitled **BuggyDe: A Machine Learning based Technique for Software Defect and Bug Severity Level Classification**, submitted by **Anik Paul Shuvo** (ID: 201014051), **Nayla Muqim** (ID: 201014100), **Ahammad Ullah** (ID: 201014077) and **Tasnima Yeasmin Tumpa** (ID: 201014038) are undergraduate student of the **Department of Computer Science and Engineering** has been examined. Upon recommendation by the examination committee, we hereby accord our approval of it as the presented work and submitted report fulfill the requirements for its acceptance in partial fulfillment for the degree of *Bachelor of Science in Computer Science and Engineering*.

Suravi Akhter, Lecturer
Dept. of CSE, University of Liberal Arts Bangladesh

Muhammad Golam Kibria, Professor
Dept. of CSE, University of Liberal Arts Bangladesh

Mohammed Ashikur Rahman, Assistant Professor
Dept. of CSE, University of Liberal Arts Bangladesh

Shohag Barman, Assistant Professor
Dept. of CSE, University of Liberal Arts Bangladesh

Place: Dhaka
Date: June 3, 2024

ACKNOWLEDGEMENTS

We would like to express our deep and sincere gratitude to our research supervisor, *Suravi Akhter*, for giving us the opportunity to conduct research and providing invaluable guidance throughout this work. Her dynamism, vision, sincerity and motivation have deeply inspired us. She has taught us the methodology to carry out the work and to present the works as clearly as possible. It was a great privilege and honor to work and study under her guidance.

We are greatly indebted to our honorable teachers of the Department of Computer Science and Engineering at the University of Liberal Arts Bangladesh who taught us during the course of our study. Without any doubt, their teaching and guidance have completely transformed us to the persons that we are today.

We are extremely thankful to our parents for their unconditional love, endless prayers and caring, and immense sacrifices for educating and preparing us for our future. We would like to say thanks to our friends and relatives for their kind support and care.

Finally, we would like to thank all the people who have supported us to complete the project work directly or indirectly.

Anik Paul Shuvo, Nayla Muqim, Ahammad Ullah and Tasnima Yeasmin Tumpa
University of Liberal Arts Bangladesh
Date: June 3, 2024

Dedicated to
my loving Mother

– *Anik Paul Shuvo*

Dedicated to
my beloved parents

– *Nayla Muqim*

Dedicated to
my loving Mother

– *Ahammad Ullah*

Dedicated to
my beloved parents

– *Tasnima Yeasmin Tumpa*

ABSTRACT

Bugs and defects have been a usual presence which hampers the productivity of software systems. The reason is it leads to less secure software. One of the main reasons causing software defect is bugs. The software bug severity must be found out as early as possible so as to be able to upgrade the software reliability and quality. In the present time there is much research done on bugs. The project aims to use a machine learning based technique for software defect and bug severity level classification. The severity and priority levels of software bugs describe the effect these bugs will have on the software system and also determines which bug should be prioritized and solved at the earliest. To solve these problems, we have developed a multi-level classification system using machine learning techniques for bug severity and priority classification. For this, bug severity and priority classification are done on two popular open sources(e.g. Eclipse and netbeans). On the other hand, software defects prediction system, it provides prediction of defective code areas to contribute to industrial results and can quickly identify development faults. Nasa's datasets were collected from promise dataset repository for defect prediction. The datasets included CM1, JM1, KC1, KC2, and PC1. Cross-validation method was used to divide the data into training and testing sets. After preprocessing, we trained various machine learning models and discovered that each model's output varied. Therefore, we used ensemble learning to improve our model's accuracy . Our results showed that, for the bug severity classification, the ensemble model has attained a level of accuracy of 99.5for more research on software engineering, bugs, defects, and machine learning. The outcomes of this research are development of an automated Machine learning model for bug severity level classification and prioritization, early detection of the defect, and enhancement in software quality by cost and time saving.

Keywords: Machine Learning, Software, Software Engineering, Software Defect, Bug, Bug Severity Level

Contents

List of Figures

Chapter 1

Introduction

1.1 Background Literature

Software engineering is one of the most important parts of modern technology. Nowadays softwares is not only a part of computers but also a very important part of our daily life. The use of softwares increasing day by day because of the adaptation with the modern world. That is why the softwares needs to work properly otherwise it may cause problems. In addition, sometimes these softwares do not work based on the needs of the user's. These problems happen due to some reasons, which are known as bugs and defects. In the realm of software engineering, the terms ″bug″ and ″defect″ refer to errors or flaws that cause a software application to behave unexpectedly or incorrectly. These issues can arise at any stage of the Software Development Life Cycle (SDLC), from initial coding and design to testing and maintenance. Bugs are basically coding errors that happen because of inaccurate logic. On the other hand a defective software means, a software that fails to meet its specific requirements and work differently. Defects directly hamper a software's performance, functionality, security and usability. The defects appear for numerous reasons such as requirements mismatch, designing flaws or integration issues. A defective software leads to incorrect results, wrong data processing and huge amounts of security risks. As a result, users become dissatisfied. Defects decrease the quality and reliability of the software. Moreover, repairing any defective software is very time consuming and costly. It hampers development cycle time. In addition, it is hardly possible to make totally bug free software Samir et al. (2023). Though defects can be fixed, a research reveals that almost half of the developer's time is spent on fixing bugs Shatnawi & Alazzam (2022). So, fixing bugs is time consuming for developers. Additionally, it is costly too, because the developers charge based on per changes in the codes which increases the whole budget of the software. On the other hand, fixing these softwares again and again is a big hassle for developers. Another highlighted issue is predicting the bug severity levels. The software systems are distributed into various severity levels Ekanayake (2021). The bug severity level means how critical the bug is. If the level is medium then some major or minor parts do not work and if the level is low then some minor parts are not working properly but it may not stop the users from using the software. If the bug levels are not found then fixing the bugs becomes very difficult. Without understanding the levels of bugs the developers can not take appropriate decisions to fix the specific bugs. So, after recognizing the bug levels the developers can solve the bugs related issues easily. The software which is released with defects or bugs does not function properly. When the users report about the bug it helps the developers to solve it for the next release by this the users can get the next updated versions of that software Ekanayake (2021).

1.2 Problem Statement

Bug reports and defect prediction datasets may show data imbalance where the amount of data for some severity and priority levels may be present in minority whereas the others may be in majority. Such disparity in data may cause the effectiveness of ML models in identifying and prioritizing high-severity and high-priority bugs to be reduced and also fails to accurately identify defective code areas as these models may show biased results towards the majority classes, leading to poor performance in identifying critical bugs. So, to make sure that ML models work more effectively and to increase accuracy, data imbalance needs to be handled properly.

1.3 Objectives and Significance of the Study

1.3.1 Aims

The aims of this project are,

- **Enhancement of classification accuracy**: Improving the performance of classification models in terms of detecting software defects. Also, it will identify their severity by using effective feature extraction techniques.
- **Reduce Computational Complexity**: Minimizing or reducing the complexity related to processing large datasets by decreasing the number of features, thereby speeding up the model training and prediction phases.
- **Optimize Feature Selection**: Identifying only the most relevant features from the original dataset that significantly contribute to accurate defect and severity predictions. In addition, that will result in more accuracy in solving the problem.

1.3.2 Objective

- **Improving Software Quality**: Software defects and bugs can significantly impact the functionality and user experience of software products. By improving defect and bug severity identification, the overall quality and reliability of software can be enhanced, leading to higher user satisfaction.
- **Efficiency in Software Development**: Reducing the time and computational resources required for defect identification allows for more efficient software development cycles. This can lead to faster deployment and maintenance of software applications.

- **Handling Large and Complex Datasets**: Modern software development generates extensive and complex datasets. Effective feature extraction helps in managing this data, making it feasible to process and analyze for defect prediction.
- **Reducing Noise**: By eliminating irrelevant features, the noise in the dataset is reduced, allowing the model to focus on the most significant indicators of defects. This can lead to more accurate and reliable predictions.
- **Cost and Resource Savings**: Optimizing the feature set and reducing computational complexity can lead to significant cost savings in terms of computational resources and time, which is particularly important in large-scale software projects.
- **Advancing Research in Software Engineering**: This project contributes to the broader field of software engineering by providing insights and methodologies that can be applied to other areas of software quality assurance and maintenance.

1.3.3 Motivation

Software companies are trying to build fault free software modules. Defects have decreased the software's quality thus it cannot perform tasks with accuracy and efficiency. Therefore, identification of software bugs and defects is the most vital stage of the Software Development Life Cycle (SDLC). Any kinds of software defects must be identified and fixed at the earliest to enhance the utility and dependability of software. If bugs and defects are handled properly, it improves the quality of softwares.

1.3.4 Significance

1. **Enhanced Software Quality and Reliability**:

   - **Improved Defect Detection**: By accurately identifying software defects and their severity, this research directly contributes to improved software quality and reliability, reducing the likelihood of critical failures in production environments.
   - **User Satisfaction**: Enhanced defect identification leads to more stable software releases, which in turn boosts user satisfaction and trust in the software product.

2. **Optimization of Resources**:

   - **Reduced Computational Overhead**: The research aims to minimize the computational resources required for defect detection by reducing the feature set, thus making the process more efficient and cost-effective.
   - **Faster Development Cycles**: More efficient defect identification processes enable quicker development and deployment cycles, allowing development teams to deliver updates and new features more rapidly.

3. **Scalability to Large Datasets**:

   - **Handling Big Data**: As software systems grow increasingly complex, the ability to handle large datasets efficiently becomes crucial. This research addresses scalability issues by optimizing feature sets, making it feasible to process and analyze vast amounts of data.
   - **Future-Proofing**: By focusing on scalable solutions, the research ensures that methodologies remain relevant and effective as the size and complexity of software datasets continue to grow.

4. **Economic Impact**:

   - **Cost Savings**: Reducing the time and resources needed for defect identification can lead to significant cost savings for software companies, particularly in large-scale projects.

- **Risk Mitigation**: Early and accurate detection of defects reduces the risk of costly post-release fixes and damage to reputation, which can have substantial financial implications.

1.4 Scope and Limitation

This project has both scope and limitations like the other projects. This project opens more scope for future research on the topics like machine learning, software development, bugs, and defects. The limitations of this project are, difficulties in collecting bug reports,it will not define all the problems and solutions properly due to the time limit. It will not give 100 percent accurate results, and the accuracy of the results may differ.

1.5 Contribution

In our project, we have predicted whether there is any defect or not and also classified both severity and priority of bug reports, which were never done before. So, classifying both severity and priority of bug reports makes it a multi class and multi-label classification and defect prediction a binary class classification system. We worked with 11 different algorithms/models, among which 7 of them are ensemble learning algorithms. To make the system more robust, both sampling techniques(to solve class imbalance) and scaling(to reduce the impact of varying scales) together which have not been done in any other papers that we researched. In the other papers, they have either applied sampling or scaling techniques but not both of them. Also, we have applied adaboost with random forest which have not been used before for bug severity and priority classification and defects prediction. This model produces better accuracy than any other models.

Chapter 2

Literature Review

In this chapter, we conduct a comprehensive review of related works focusing on Software Defect Prediction and Bug Severity Classification. We aim to understand the existing approaches and methodologies, analyze their strengths and weaknesses, and identify gaps our project seeks to address.

2.1 Related Works and Performance Evaluation Criterion

2.1.1 Defect

In 2023, Iqra Mehmood published a study titled ”A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning.” This study addresses the essential issue of software flaws, which reduce software quality and incur large expenses. The study’s key contribution is to use feature selection techniques to improve the accuracy of several machine learning classifiers for defect prediction.The researchers explored many machine learning techniques, such as Random Forest, Logistic Regression, Multilayer Perceptron, Bayesian Net, and J48. These techniques were evaluated on five publicly available NASA datasets (CM1, JM1, KC2, KC1, and PC1) with the WEKA data preprocessing and analysis tool. The results showed that feature selection enhanced fault prediction accuracy when compared to models without it.When feature selection was used, the Random Forest model achieved accuracies of 90%, 88%, 87%, 91%, and 89% for the CM1, JM1, KC1, KC2, and PC1 datasets, respectively. Logistic Regression achieved even better accuracies of 92%, 90%, 89%, 93%, and 91% with feature selection, suggesting considerable gains. The Logistic Regression model outperformed the others, with 93% accuracy on the KC2 dataset. This extensive analysis emphasizes the importance of feature selection in constructing accurate machine learning models for software defect prediction, which improves software quality and reliability Mehmood et al. (2023) .

This study used cross-project defect prediction to reuse data from previous projects. This method is effective when training model data exceeds project requirements. The performance of Machine Learning methods depends on the dataset used for training the prediction model. Machine learning is used to forecast defect induction changes. This work employed five modules and repositories (CM1, JM1, KC1, KC2, and PC1) to model outcomes using the PROMISE dataset. They utilized four distinct classifiers to create the dataset: A Bayesian network,Random Forest, SVM, and Deep Learning based on F-measure improved resilience and outperformance of all other models. Random Forest and Deep Learning outperformed Bayes network and SVM in multiple studies (on all five datasets). The suggested defect prediction model requires some heterogeneous metric values to make effective predictions. This research conducts experiments on five different datasets using cutting-edge deep learning and random forest algorithms. Using 10-fold cross-validation, the suggested model finds errors with 90% accuracy. On all five datasets, the results reveal that Random Forest and Deep Learning outperform Bayes network and SVM. To improve fault identification accuracy, the author developed a Deep Learning classifier Mehmood et al. (2023).

In 2019,Ahmed Iqbal conducted a study Using NASA datasets that explored many supervised machine learning techniques for software defect prediction. A number of algorithms were used, including PART, Decision Tree (DT), Random Forest (RF), K Nearest Neighbor (KNN), kStar (K*), Naïve Bayes (NB), Multi-Layer Perceptron (MLP), Radial Basis Function (RBF), Support Vector Machine (SVM), and One Rule (OneR). This study showed that Random Forest had the highest accuracy, coming in at 95.6%, and Support Vector Machine came in second at 94.8%. The study used measures such as precision, recall, F-Measure, accuracy, MCC, and ROC Area to evaluate the efficacy of several classification approaches. Iqbal's studies provide a useful baseline for future research in software defect prediction, enabling comparative study and validation of new methodologies, models, or frameworks Iqbal et al. (2019).

A software defect prediction framework utilizing a Multi-filter Feature Selection Technique and Multi-Layer Perceptron (MLP) was presented by Ahmed Iqbal and Shabib Aftab in 2020. Ten popular supervised classifiers were used in the study to compare the outcomes: Naive Bayes (NB), Multi-Layer Perceptron (MLP), Radial Basis Function (RBF), Support Vector Machine (SVM), K Nearest Neighbor (KNN), kStar (K*), One Rule (OneR), PART, Decision Tree (DT), and Random Forest (RF).On the CM1 dataset, MLP-FS-ROS had the highest F-measure (0.800) and MCC (0.592), whereas RBF, SVM, and PART achieved 90.816% accuracy. For JM1, MLP-FS-ROS performed best in F-measure (0.558) and MCC (0.275), while MLP-FS had the highest accuracy (80.44%) and RF lead in ROC Area (0.738). On KC1, MLP-FS-ROS had the highest F-measure (0.641), RBF the highest accuracy (78.796%), while RF excelled in ROC Area (0.751) and MCC (0.346). For KC3, MLP-FS-ROS had the greatest F-measure (0.588) and MCC (0.358), while SVM, OneR, MLP, and MLP-FS tied for accuracy (82.758%) and RF led in ROC Area (0.807). On PC1, MLP-FS-ROS had the highest F-measure (0.900), ROC Area (0.955) Iqbal & Aftab (2020).

A study on software defect detection was conducted in the year 2021 by Mohd. Mustaqeem and Mohd. Saqib. They used a hybrid technique called Principal Component-based Support Vector Machine (PC-SVM), which combines PCA and SVM. Their study combined PCA for feature reduction and SVM for classification in an effort to overcome the drawbacks of current software defect prediction techniques, such as large dimensionality and computational complexity. The NASA PROMISE dataset was used in the study, and it was divided into training and testing sets for analysis. With 95.2% accuracy on the CM1 dataset and 86.6% accuracy on the KC1 dataset, the PC-SVM model showed considerable accuracy gains. These exceed other machine learning algorithms and standard methods, demonstrating the model's efficacy in software defect prediction Mustaqeem & Siddiqui (2022).

In 2022, Mutasem Shabeb Alkhasawneh published a methodology that combines feature selection and classification using radial basis function (RBF) neural networks to forecast software errors across fourteen NASA datasets. The datasets included CM1, JM1, KC1, and KC2. The data was divided into training and testing sets using K-cross-validation methods. The model's performance was assessed using precision, recall, the F-measure, and accuracy. The accuracy results were: CM1, 93.99%; JM1, 84.87%; KC1, 83.25%; and KC2, 79.11% . The suggested model outperformed existing techniques in the CM1 and PC1 datasets. However, it discovered limits in datasets KC1 and KC2, highlighting areas for future improvement Alkhasawneh et al. (2022).

In 2020, Mohammad Amimul Ihsan Aquil and Wan Hussain Wan Ishak did research on forecasting software flaws using machine learning techniques such as ensemble learning, clustering, and classification. According to the study, the stacking classifier (STC) outperformed other algorithms in terms of accuracy. The accuracy scores for major datasets were CM1 (95.12%), JM1 (90.28%), KC1 (89.74%), and KC2 (87.65%). Among the algorithms studied, Quadratic Discriminant Analysis (QDA) did particularly well among supervised learning algorithms, whereas K-Nearest Neighbors (KNN) and Gaussian Mixture Model (GMM) excelled in unsupervised learning algorithms. These findings demonstrate the effectiveness of ensemble approaches in predicting software defects, giving useful insights for enhancing software quality and maintenance Aquil & Ishak (2020).

In 2022, Mohd. Mustaqeem and Tamanna Siddiqui investigated software defect prediction using a hybrid technique known as Principal Component-based Multilayer Perceptron (PC-MLP). This methodology overcomes the limits of traditional software testing by employing machine learning for early defect discovery, resulting in improved performance and cost savings. The study compared numerous algorithms and discovered that the proposed PC-MLP model outperformed the others. The accuracy scores for the key datasets were CM1 at 94.4% and KC1 at 87.20%. The PC-MLP model outperformed other algorithms examined, including SVM, Decision Tree, and Random Forest. The study demonstrated that the PC-MLP model could handle big datasets effectively while maintaining good accuracy in the presence of noisy data and outliers Mustaqeem & Siddiqui (2022).

Satya Srinivas Maddipati and Malladi Srinivas focused their research on Software Defect Prediction using Kernel Principal Component Analysis (KPCA) and Conformalized Support Vector Fuzzy Inference System (CSANFIS) on identifying software bugs and defects in the early stages of software development. Within project defect prediction, it is necessary to create defect prediction models adapted to unique projects, taking into account factors such as lines of code, cyclomatic complexity measurements, and Halstead metrics. Cross-project defect prediction, on the other hand, uses features from previous projects to predict problem proneness in the present or developing project. The study emphasizes the use of machine learning models, such as decision trees, Bayes classifiers, Support Vector Machines (SVM), and Artificial Neural Networks (ANN), to construct effective software fault prediction models Maddipati et al. (2021).

2.1.2 Bugs

In 2023, Mina Samir conducted a study titled ”Improving Bug Assignment and Developer Allocation in Software Engineering through Interpretable Machine Learning Models.” The study addresses the ongoing challenge of software Bugs , which complicate the Software Development Lifecycle (SDLC) and result in large expenses. The researchers suggest an automated issue triage mechanism called Assign issue Based on Developer Recommendation (ABDR). This strategy is intended to reduce bug resolution time and costs while improving result explainability.The ABDR method utilizes supervised machine learning, analyzing past Bugs data and developer profiles to propose the best developers for new bug reports. Key features studied include developer similarity (DS), handling time (HT), and bug-fixing effectiveness (EB).Using these parameters, the model offers a bug prediction score to each developer.The study evaluates three machine learning models: Decision Tree, Random Forest, and XGBoost. XGBoost outperformed the others, earning 85% accuracy, 81% precision, 75% recall, and a 77% F1-score. The study underlines the significance of explainable AI (XAI) in improving transparency and confidence in machine learning models for bug triaging, offering insights into model decisions, and increasing system reliability Samir et al. (2023).

In the work done by Kakkar, they have discussed that to help the bug triage, a model for automatic bug classification is needed. So, for predicting bug severity and classifying bugs into multi-severity classes and to extract more relevant features, ant colony optimization based feature extraction techniques integrated with SVM, NB, F-SVM and DeepFM. The accuracy rates ranges from 85.73 and 89.38%, 78% to 80%, 73% to 76%, 92.67% to 97.27%, 71% to 77%,65% to 74%, 78.21% to 81.28% and 90.02% to 95.24% Kukkar et al. (2023).Kakkar et al.(2023). Pushpalatha,in 2020 explored the

priority prediction of bug reports using classification algorithms in order to improve open-source software development and maintenance. The study used multiple classification methods, such as Naïve Bayes, Random Tree, and Simple Logistic, to forecast bug report priorities. They examined Bugzilla reports for details such as the product name, component, assignee, status, severity, and summary text. Simple

Logistic performed better than Naïve Bayes and Random Tree, with an accuracy of 78% vs. 65% and 60%, respectively. Precision, recall, and F-measure were used to evaluate performance, with Simple Logistic outperforming other priority classes in terms of precision and recall.The study indicated that using classification algorithms, notably Simple Logistic, enhances the accuracy of bug report prioritizing, allowing developers to address bugs more efficiently and improving software quality and productivity Pushpalatha et al. (2020).

There are quite a number of studies devoted to research to find a proper model so as to be able to properly classify bug severity and priority. For example, Panda proposed topic modeling and IFS measure-based bug severity prediction. Here, to balance severity labels data by using SMOTE. IFS measure and topic modeling are used to calculate the relationship with existing severity labels for a new bug Panda & Nagwani (2023). Reference [14] improved bug reports for Mozilla, Eclipse,

and GCC by combining questions and answers from stack traces. They trained Naïve Bayesian, k-Nearest Neighbor (KNN), and Long Short-Term Memory (LSTM) models to categorize bug reports by severity level. The study found that the average F-measure for Mozilla, Eclipse, and GCC projects improved by 23.03%, 21.86%, and 20.59% compared to earlier studies. Furthermore, the investigation demonstrated that the Naïve Bayesian strategy beat all baseline approaches. Tan et al. (2020). assumed an equal distribution of bug reports by severity level, which may not always be accurate.

The main aim of the study done by Y.Bugayenko, is to review the recent situation and research on task prioritization in the software engineering domain. Here, it is found out that most of the task prioritization approaches involve bug prioritization. To measure the quality of a prioritization model, most used matrices are f-score, precision, recall, and accuracy Bugayenko et al. (2023). In the study done by

Tan Youshuai, to predict the severity of bug reports and to solve the problem of submission of a large number of bug reports, question-answer pairs from stack overflow of open source projects(e.g. Mozilla, Eclipse and GCC) are collected as dataset and logistic regression, naive bayes, K-Nearest Neighbour and long-short term are applied on them. The proposed method shows average F-measure increased by 23.03%, 21.86%, and 20.59% respectively Tan et al. (2020).

In another study by Kim , a feature selection algorithm was applied on the datasets of Eclipse and Mozilla open source. First, the topic of bug reports are classified. Then, the characteristics were learned from the CNN-LSTM algorithm to predict the severity of bug reports. The input of CNN is the severity specific features of each topic. Afterwards its output is taken as input of LSTM. Ultimately, severity is the output of LSTM. F-measure of Mozilla was 90.62% and 93.22% Kim & Yang (2022).

Kamna Vaid and Prof. Udayan Ghose presented a paper named "Machine Learning Based Predictive Analysis of Software Bug Severity and Priority" in 2024. The study aims to precisely predict the severity and priority of software bugs using machine learning techniques. Two machine learning algorithms, Random Forest (RF) Classifier and Support Vector Machine (SVM), were used to determine Bug severity and priority. The Random Forest Priority Model obtained a remarkable 87% accuracy, indicating its potential for effectively categorizing bug priorities. For severity prediction, the RF Severity Model performed differently across classes, with an overall accuracy of 60%. It received an F1-score of 0.79 for severity class 5, indicating its accuracy in forecasting high-severity issues.SVM models provided notable outcomes. The SVM Severity Model achieved 63% accuracy, with an impressive F1-score of 0.82 for severity class 5. Similarly, the SVM Priority Model achieved 87% accuracy, with particularly excellent performance in priority class 1.The study indicates that Random Forest models are highly successful for predictive analysis in software development, providing an extensive overview of model performance at different severity and priority levels. The highest accuracy achieved was 87 which was shared by both the Random Forest and SVM models for bug priority prediction Vaid & Ghose (2024) For a given textual bug report,Hirsch Thomas proposed an

ML based approach to predict its fault type and can help in data collection efforts. They created a dataset consisting of data from more than 70 projects to train and evaluate their approach. The macro average F1 score of their models on the bug classification problem reaches up to 0.69% Hirsch & Hofer (2022).

Jayalath Ekanayake conducted a study in 2021 called "Bug Severity Prediction using Keywords in Imbalanced Learning Environment," which addressed the challenge of predicting bug severity in software systems using machine learning. The study used Naïve Bayes, Decision Tree Learner, and Logistic Regression to categorize bugs as Blocking, High, Low, or Normal. Given the dataset's class imbalance, in which significant bugs were underrepresented, Ekanayake used Area Under the Receiver Operating Characteristic Curve (AUC) as an evaluation metric, believing it was more appropriate than traditional accuracy metrics.According to the study's findings, the Logistic Regression model outperformed all other models with an AUC of 0.65, notably in predicting Blocking and High severity problems. The Decision Tree and Naïve Bayes models were less effective, with the latter slightly outperforming random chance. This study emphasizes the significance of utilizing proper metrics in imbalanced learning environments and provides a solid foundation for future research in bug severity prediction Ekanayake (2021).

Chapter 3

Design and Development

3.1 Project Management and Financial Activities

Our project management activities are structured to guide the development of a web application for Software bug severity and priority classification and defect prediction. The initiation phase involves defining project objectives, forming a dedicated team under our supervisor, and establishing clear roles. Planning encompasses the creation of a detailed project plan, resource allocation, and risk management. Ongoing monitoring and controlling ensure project progress aligns with timelines, risks are mitigated, and data quality standards are maintained. The closing phase involves evaluating the model's performance and documenting key learnings.

Financial activities are streamlined through a comprehensive budget that covers personnel, technology, and miscellaneous costs. We prioritize resource allocation based on critical project phases, track expenses meticulously, and generate periodic financial reports for stakeholders. Vendor and tool management, contingency planning, and post-project financial assessments contribute to a transparent and accountable financial framework. Our approach ensures effective project management and financial stewardship, guiding the successful development of the Software bug severity and priority classification and defect prediction web application.

Project Budget :

Hardware	80,000
Software	10,000
GPU Cost	50,000

Functional Requirements : The functional requirements listed below outline the specific capabilities and features that the system must possess to achieve its intended purpose effectively:

Code	Requirement
REQ-1	The system shall accept various bug report-data as user input.
REQ-2	It shall validate and preprocess the input data, drop missing values and convert character data to numerical format.
REQ-3	The data shall be sourced from authentic datasets.
REQ-4	The system shall integrate defect data for Defect prediction.
REQ-5	The system shall implement multiple machine learning models for defect prediction. Supported models shall include 'Random Forest,' 'XGBoost,' 'Gradient Boosting,' 'Extra tree,' 'Ada Boost,' 'Decision Tree'.
REQ-6	The system shall perform ensemble learning using the top-performing machine learning models.
REQ-7	The system will be regularly maintained and updated ensuring its functionality.
REQ-8	The application should be designed to work seamlessly with popular web browsers.

Non Functional Requirements : The non-functional requirements presented here outline the qualities and constraints that the system must adhere to to ensure overall performance, usability, and security:

Code	Requirement
REQ-1	Specifying the max acceptability of the response time for Bug classification tasks for ensuring timely results.
REQ-2	Ensuring the system can handle noisy and incomplete data without significant loss of accuracy.
REQ-3	Ensuring that sensitive data used for training and classification tasks is adequately protected to prevent unauthorized access.
REQ-4	Specifying the requirement of computational resources (CPU, GPU, memory) for training and inference tasks.
REQ-5	Ensuring transparency in how the system makes decisions to build trust among users.

3.2 Timeline

The timeline of this project is carefully designed in order to get the results on time and more effectively. It started by selecting the project title on (date). After that first follow up was done. During the follow up session the main idea of the project, objectives, motivations, significance of the project were discussed. One of the most important parts of the project is the literature review. The reason is without the literature review it is not possible to proceed with the project. Also, it is impossible to predict the outcome of a particular project. Most of the time was allocated for coding. The reason is coding will show the accuracy rate of our project. The report writing process started after the literature review. In addition, during all the processes the report writing also continued. It helped to observe the progress of the project. Each presentation of the report helped to understand the problems or possible problems of the project. The poster presentation took place before the final presentation and it delivered the whole idea of the project.

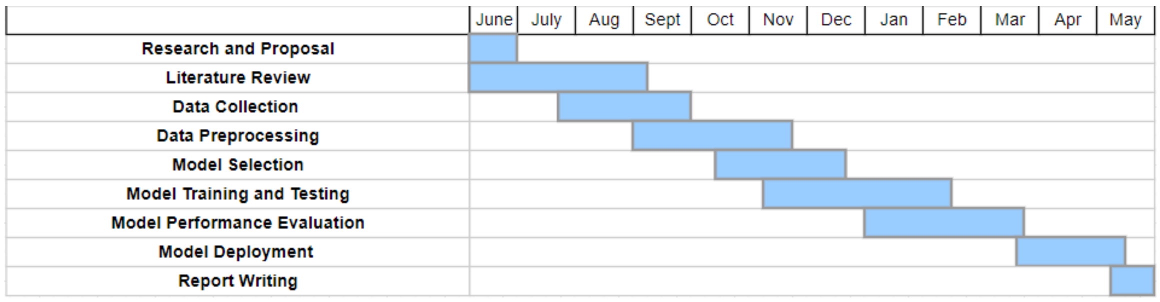


Figure 3.1: Gantt Chart

3.3 Adopted Tools and techniques

3.3.1 Hardware Requirements

The system demands the following hardware components to operate optimally:

- Processor: Dual-core or higher
- RAM: 8 GB or more
- Storage: Minimum 30 GB of free disk space
- Display: Monitor or screen for accessing the user interface
- Internet connectivity: High-speed internet for web application access
- Input devices: Keyboard and mouse or equivalent input methods

3.3.2 Software Requirements

The following software components are necessary for the proper functioning of the system:

- Python 3.x: The system requires Python version 3.x for development and execution.
- Streamlit: A web application framework used to create the system's user interface.

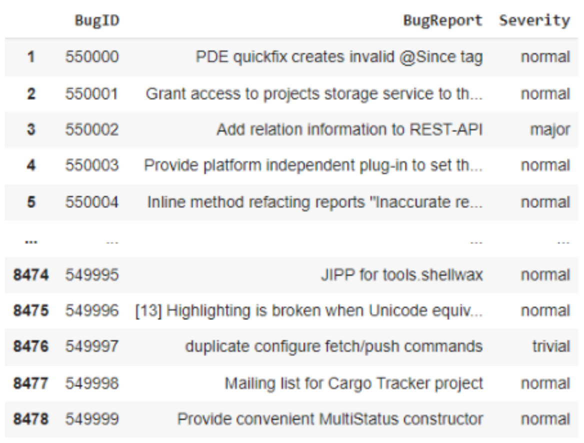
- Machine Learning Libraries: Including XGBoost, LightGBM for implementing machine learning models.
- Version Control System: Utilize Git to oversee the source code and promote collaboration within the development team.
- Integrated Development Environment (IDE): Utilize tools like PyCharm or Visual Studio Code to streamline coding and debugging processes for increased efficiency.

3.4 Design the solution and Implementation

3.4.1 Data Description

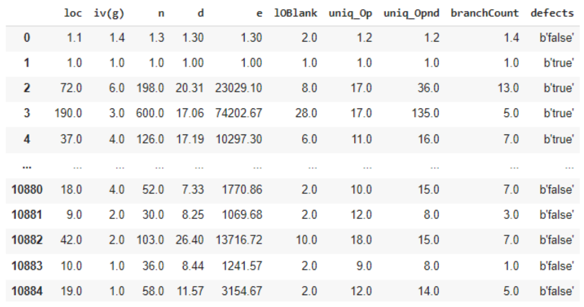
The datasets consist of bug reports of the open sources such as Eclipse and Netbeans which contain more than 12000 bug reports. Even though these are collected from kaggle, the main source of these datasets are from the article 'CNN Based Severity Prediction of Bug Reports' written by Ekanayake (2021). The preprocessed and cleaned dataset consists of features such as bugID, short description, severity level, priority level. The short description is considered as the main feature which is in textual format. This description part is considered as the most suitable for bug severity and priority classification.

The defect datasets CM1,JM1,KC1,KC2, PC1 are collected from the Promise Software Engineering Repository and the main source is from Nasa Metrics Data Program. The attributes present in these data sets are loc(line count of code), iv(g)(design complexity), n(total operators+operands),d(difficulty),e(effort),lOBlank(count of blank lines),uniqOp(unique operators),uniqOpnd(unique operands),branchCount(of the flow graph) and defects(has or has not one or more reported defects). These datasets are in numeric format.



	BugID	BugReport	Severity
1	550000	PDE quickfix creates invalid @Since tag	normal
2	550001	Grant access to projects storage service to th...	normal
3	550002	Add relation information to REST-API	major
4	550003	Provide platform independent plug-in to set th...	normal
5	550004	Inline method refacting reports "Inaccurate re...	normal
...	...	...	...
8474	549995	JIPP for tools.shellwax	normal
8475	549996	[13] Highlighting is broken when Unicode equiv...	normal
8476	549997	duplicate configure fetch/push commands	trivial
8477	549998	Mailing list for Cargo Tracker project	normal
8478	549999	Provide convenient MultiStatus constructor	normal

Figure 3.2: Bug Dataset Sample



	loc	iv(g)	n	d	e	lOBlank	uniq_Op	uniq_Opnd	branchCount	defects
0	1.1	1.4	1.3	1.30	1.30	2.0	1.2	1.2	1.4	b'false'
1	1.0	1.0	1.0	1.00	1.00	1.0	1.0	1.0	1.0	b'true'
2	72.0	6.0	198.0	20.31	23029.10	8.0	17.0	36.0	13.0	b'true'
3	190.0	3.0	600.0	17.06	74202.67	28.0	17.0	135.0	5.0	b'true'
4	37.0	4.0	126.0	17.19	10297.30	6.0	11.0	16.0	7.0	b'true'
...	...	...	...	...	...	...	...	...	...	...
10880	18.0	4.0	52.0	7.33	1770.86	2.0	10.0	15.0	7.0	b'false'
10881	9.0	2.0	30.0	8.25	1069.68	2.0	12.0	8.0	3.0	b'false'
10882	42.0	2.0	103.0	26.40	13716.72	10.0	18.0	15.0	7.0	b'false'
10883	10.0	1.0	36.0	8.44	1241.57	2.0	9.0	8.0	1.0	b'false'
10884	19.0	1.0	58.0	11.57	3154.67	2.0	12.0	14.0	5.0	b'false'

Figure 3.3: Defect Dataset Sample

3.4.2 Techniques used before Model Selection

For Bug Report:

- **Data Preprocessing**: Data preprocessing is essential for ensuring our dataset is clean, well organized, and ready for analysis. Several steps are required to deal with inaccurate data, dropping some unessential columns, removing punctuations,and getting the data prepared to feed into machine learning models. To remove or filter out common words, stopword removal has been used. Then to simplify words, stemming has been applied. Afterwards, lemmatization is applied to convert the words to their base form.

- **Vectorization for cleaned text and data split**: To vectorize and for information retrieval, we applied term frequency-inverse document frequency on the cleaned data. This is used for evaluating the importance of a word in a document. The dataset is then split into training and testing dataset.

For Defect:

- **Data Scaling and Normalization**: Data scaling and normalization are performed to mitigate the impact of varying scales and units across different features. This process brought all features to a standardized scale, ensuring that no single feature dominated the predictive model because of its magnitude. In this task, Robust Scaling, Standard Scaling, Min-Max scaling and Box-Cox Transformation are used.

- **Handling Imbalanced Data**: In software bug datasets, class imbalances are shared, where one outcome (e.g., the presence of a severity) is much rarer than the other. Appropriate tech-niques were applied such as oversampling, undersampling and SMOTE to address these imbalances and prevent bias in the project models.

3.4.3 Model Selection for both bug severity classification and defect prediction system

Following the compilation and preprocessing of the dataset, which involved addressing missing values and converting character data, we proceeded to choose the most appropriate classification models for our disease prediction project. In our quest to ensure the precision and dependability of our predictions, we delved into a broad array of machine learning algorithms, including:

- **Decision Tree**: A Decision Tree is a versatile and user-friendly machine learning technique that divides data into subsets based on the value of input attributes, resulting in a tree-like model of decisions. For textual data classification, Decision Trees offer various advantages. Interpretability, Handling Nonlinear Data, and Feature Importance

- **Logistic Regression**: Logistic Regression is a linear model for binary classification which predicts the likelihood of a binary result using one or greater than one predictor variables.

- **Naive Bayes**: Naive Bayes is a probabilistic classifier based on Bayes' theorem, which assumes that characteristics are independent within a class conditionally. It is particularly suitable for text categorization tasks.

- **Support Vector machine**: Support Vector Machine is an effective classifier which works by identifying the best hyperplane that separates various classes in the feature space. They are very useful for textual data classification for a number of reasons: High Dimensional Spaces, Margin Maximization.

3.4.3.1 Ensemble Learning

Ensemble Learning stands as a potent machine learning approach that unites multiple individual models, often termed base models or weak learners, to craft a better resilient and precise predictive model. The fundamental concept underlying ensemble learning is that a collection of diverse models can surpass the performance of any single model within the ensemble. There exist various methods to merge these models:

[1]Bagging: Bagging is a strategy for improving the stability and accuracy of machine learning algorithms. It involves training numerous models on various subsets of training data. The bagging(Bootstrap Aggregating) algorithms used here are:

- **Random Forest**: Random Forest is an ensemble learning method that creates many decision trees during training and returns the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

- **Extra Trees**: Extra Trees is an ensemble method similar to Random Forest that uses extra randomization in the tree-building process.

[2]Boosting: It is an ensemble learning technique which tries to improve the performance of a model by training several weak learners sequentially and independently by aiming to rectify the mistakes done by the previous ones. The concluding prediction is made by merging predictions of all the weak learners. Some of the boosting algorithms that are used in our project are:

- **Gradient Boosting**: Gradient Boosting is an effective ensemble technique that develops models successively, with each new model fixing mistakes in the preceding ones.

- **Ada Boost**: AdaBoost is an ensemble method that uses numerous weak classifiers to create a strong classifier by focusing on misclassified examples.

- **HistGradient Boosting**: HistGradient Boosting is a highly efficient Gradient Boosting solution that employs histograms to accelerate training.

- **XGBoost**: XGBoost is a more advanced version of gradient boosting that includes regularization and optimization approaches.

- **LGBOOST**: LightGBM is a highly efficient gradient boosting framework that prioritizes rapid training and reduced memory utilization.

Also, two models, AdaBoost with decision tree and Adaboost with Random Forest are applied for software defect prediction and bug severity and priority classification which takes advantage of both algorithm's strengths. Thus resulting in a powerful, accurate, and robust model. Random Forest provides a robust, dependable based foundation, while AdaBoost improves it by handling difficult-to-predict scenarios, yielding a model well-suited for these crucial tasks in machine learning applications. This ensemble models aid in the effective identification, classification, and prioritization of software faults, thereby enhancing software quality and maintenance procedures.

In bug severity and priority classification, we have combined feature extraction using tf-idf vectorization with a onevsrest classifier using ml models in a pipeline to ensure single workflow and to increase accuracy and to make it convenient to train and test across various models.

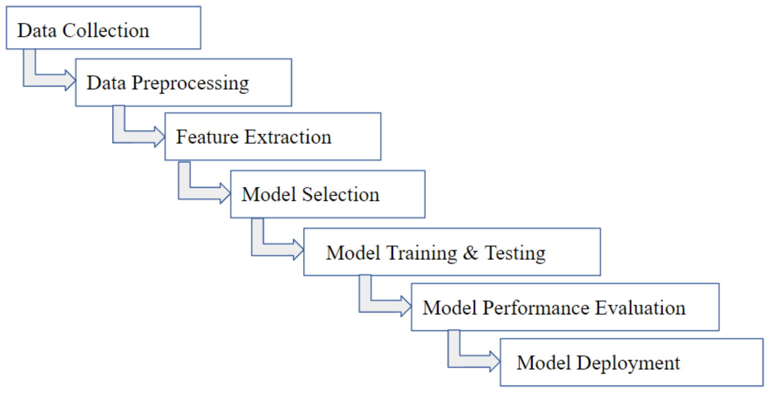
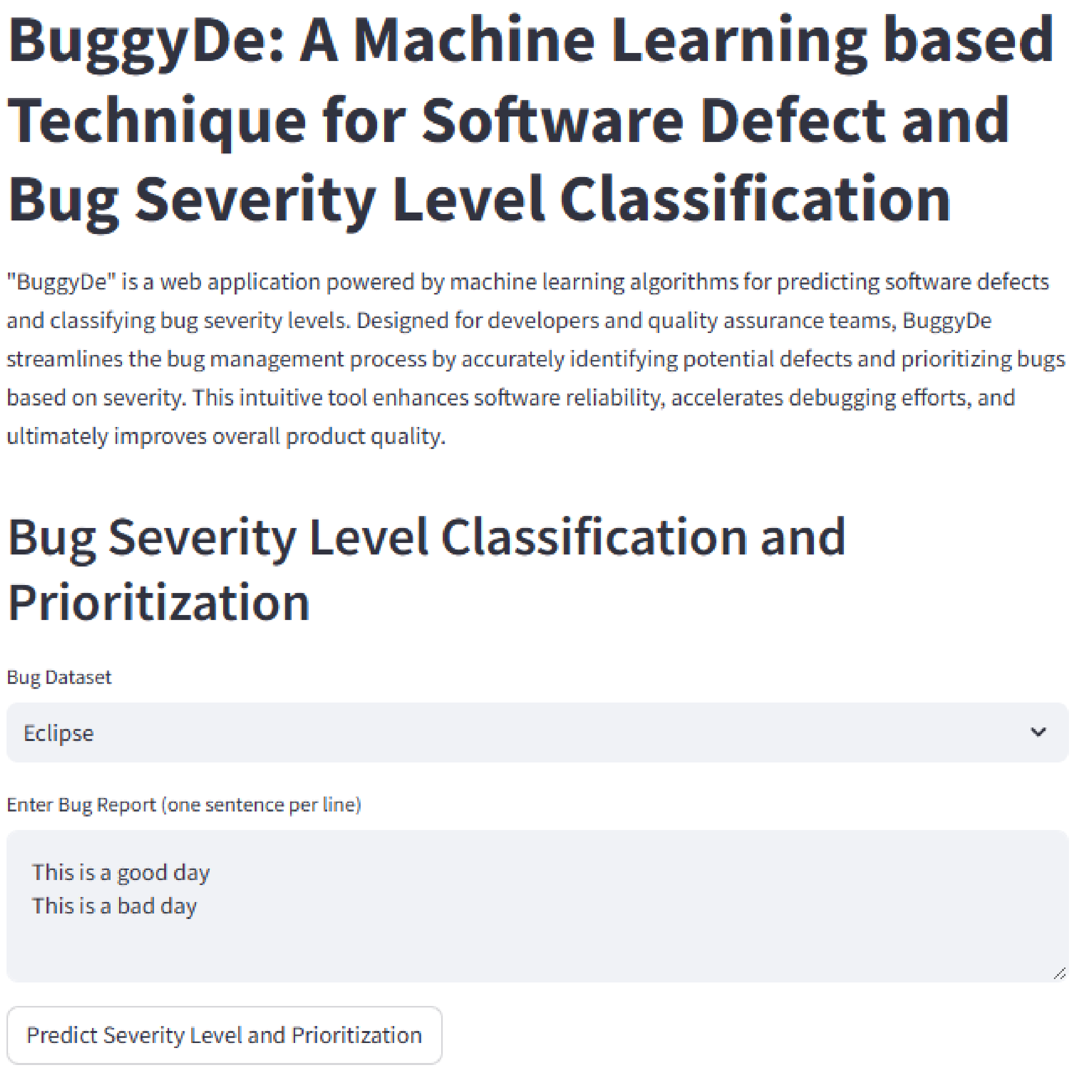


Figure 3.4: Methodology

3.4.4 WebApplication Development

In the final phase of our project, we turned our attention to crafting an accessible and user-friendly interface for our Software Defect and Bug Severity Classification models. To accomplish this, we opted for Streamlit, a versatile Python library renowned for simplifying the development of interactive web applications. Streamlit proved to be an excellent choice for several reasons. First and foremost, it seamlessly facilitated the integration of our machine learning models into a web-based platform, offering users the convenience of inputting their data and receiving instant predictions for both defects and bugs, all within a user-friendly interface.The development process was remarkably efficient and straightforward, thanks to Streamlit's intuitive design. With minimal effort, we created a pleasing web application, ensuring that users could easily navigate and interact with our predictive models. Our web application is a practical system for professionals and individuals keen on assessing software's bugs and defect prediction. By harnessing the power of Streamlit, we've made it incredibly convenient for users to access precise defect predictions.

Upon entering the application, users are prompted to provide data. For each type, specific questions guide users in providing the necessary information for accurate prediction. Once the data is inputted, users click the "Detect Defect" and " Predict Severity Level and Prioritization" button to obtain their results.


BuggyDe: A Machine Learning based Technique for Software Defect and Bug Severity Level Classification

"BuggyDe" is a web application powered by machine learning algorithms for predicting software defects and classifying bug severity levels. Designed for developers and quality assurance teams, BuggyDe streamlines the bug management process by accurately identifying potential defects and prioritizing bugs based on severity. This intuitive tool enhances software reliability, accelerates debugging efforts, and ultimately improves overall product quality.

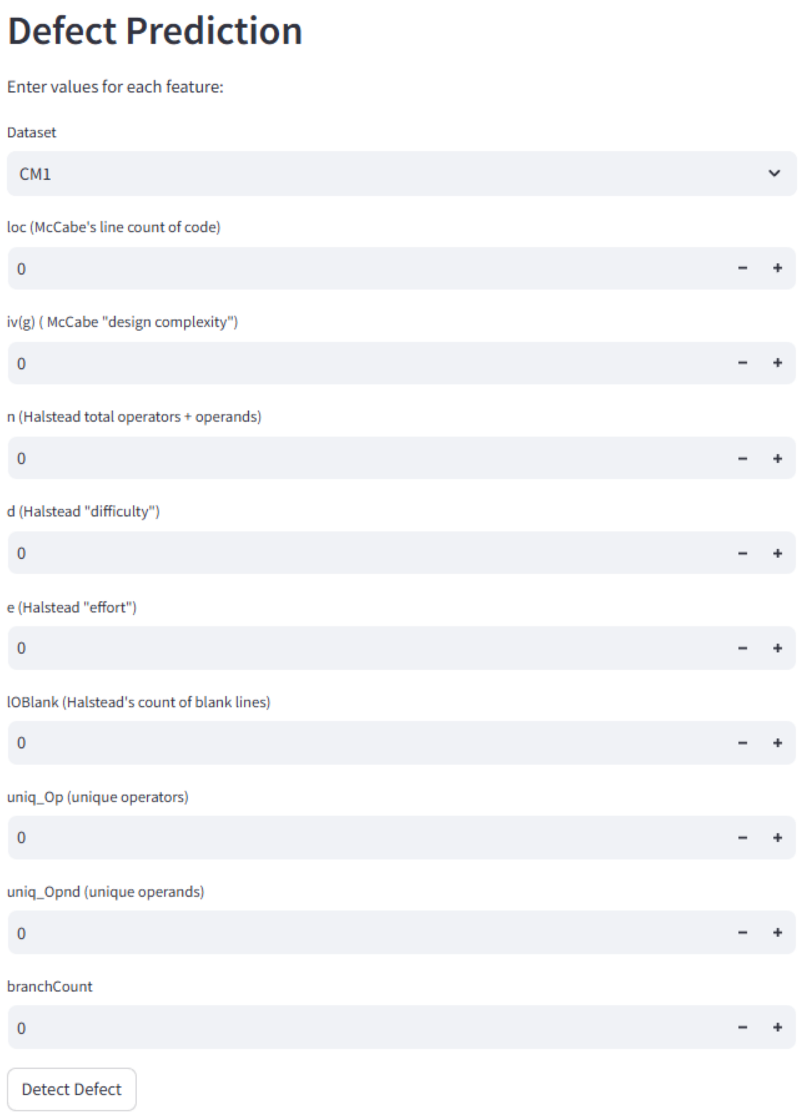
Bug Severity Level Classification and Prioritization

Bug Dataset: Eclipse

Enter Bug Report (one sentence per line)

This is a good day
This is a bad day

Figure 3.5: Web App UI for Bug Severity


Defect Prediction

Enter values for each feature:

Dataset

CM1

loc (McCabe's line count of code)

0

iv(g) (McCabe "design complexity")

0

n (Halstead total operators + operands)

0

d (Halstead "difficulty")

0

e (Halstead "effort")

0

lOBlank (Halstead's count of blank lines)

0

uniq_Op (unique operators)

0

uniq_Opnd (unique operands)

0

branchCount

0

Figure 3.6: Web App UI for Defect Prediction

3.5 Evaluate the solution

For Software Defect Prediction

- **Improved Accuracy:** Using both Random Forest and AdaBoost together can make predictions more accurate. For predicting software problems more accurately Random Forest acts as a strong base model and AdaBoost enhances it by focusing on the errors.

- **Handling Imbalanced Data:** In software defect datasets, there are usually fewer instances of defects compared to non-defective ones. AdaBoost can do this by giving more attention to the smaller group of defective items, which helps the model find defects better.

- **Robustness to Overfitting:** Random Forest helps prevent overfitting by using many decision trees, and AdaBoost can improve this by focusing on challenging cases, resulting in a more flexible model.

For Software Defect Prediction

- **Precision in Severity Classification:** Bug severity classification needs exact identification of the severity levels of bugs. AdaBoost's ability to focus on misclassified occurrences helps in fine-tuning the model to appropriately classify problem severity.

- **Prioritization:** Knowing which bugs are more serious is critical for prioritization. The combined technique can provide more dependable and precise severity forecasts, allowing for improved issue prioritization and fixing.

- **Feature Insights:** Feature importance of Random Forest can help to discover significant aspects influencing bug severity, providing valuable insights into the nature of software defects and assisting in the prioritization of critical issues.

Chapter 4

Result analysis

In this Chapter, a comprehensive analysis of our disease prediction models, examining their individual performance, the effectiveness of ensemble models, and the evaluation of our proposed models through various metrics.

4.1 Investigate the final result

Evaluation metrics serve as vital instruments in gauging the performance and efficiency of machine learning models. They offer insights into a model's performance quality, its precision in prediction, and its alignment with the particular objectives of a given task. The choice of evaluation metrics varies according to the specific machine learning problem, encompassing classification, regression, clustering, and more.

4.1.1 Accuracy Score

Accuracy stands as a foundational evaluation metric for gauging the performance of classification models, including those used in sentiment analysis. It quantifies the ratio of accurate predictions made by the model relative to the total number of predictions. The formula to calculate accuracy is:

Accuracy = Number Of correct predictions / Total number of predictions

A robust accuracy score signals that the model is delivering precise predictions, whereas a lower score implies potential performance issues. However, it's essential to look beyond accuracy to obtain a comprehensive assessment of a model's performance, particularly in scenarios with imbalanced datasets where one class predominates.

4.1.2 Precision Score

Precision plays a pivotal role in binary classification tasks like sentiment analysis, as it assesses the accuracy of optimistic predictions made by a model. It quantifies the ratio of correctly predicted positive samples (accurate optimistic predictions) to the total optimistic predictions generated by the model. The precision score can be calculated using the following formula:

Precision = TruePositives / TruePositives + FalsePositives

True Positives (TP) represent the count of samples accurately categorized as positive, whereas False Positives (FP) denote the number of samples erroneously classified as positive when they are, in fact, negative.

A heightened precision value signifies the model's dependability in predicting positive samples, consequently lowering the likelihood of false positive errors. In the context of sentiment analysis, a strong precision score indicates the model's adeptness at correctly identifying positive sentiments within reviews, thereby enhancing confidence in decision-making for tasks reliant on positive sentiment recognition.

4.1.3 Recall Score

Recall, alternatively referred to as Sensitivity or True Positive Rate, serves as a performance metric employed in binary classification tasks, such as sentiment analysis. Its purpose is to gauge a model's capacity to accurately pinpoint positive instances (true positives) out of the overall actual positive instances within the dataset. The formula for Recall is as follows:

Recall = TruePositives / TruePositives + FalseNegatives

True Positives (TP) signify the count of positive instances accurately predicted by the model, whereas False Negatives (FN) denote positive instances wrongly classified as negative.

A notable Recall value suggests the model's adeptness at comprehensively captur- ing positive instances. This attribute proves particularly significant in scenarios featuring imbalanced datasets or situations where the cost of overlooking positive instances are substantial. However, it's essential to note that a high Recall may coincide with a lower Precision (the ability to accurately predict true positives among all predicted positives), and in machine learning models, striking a balance between Recall and Precision often entails a trade-off.

4.1.4 F1 Score

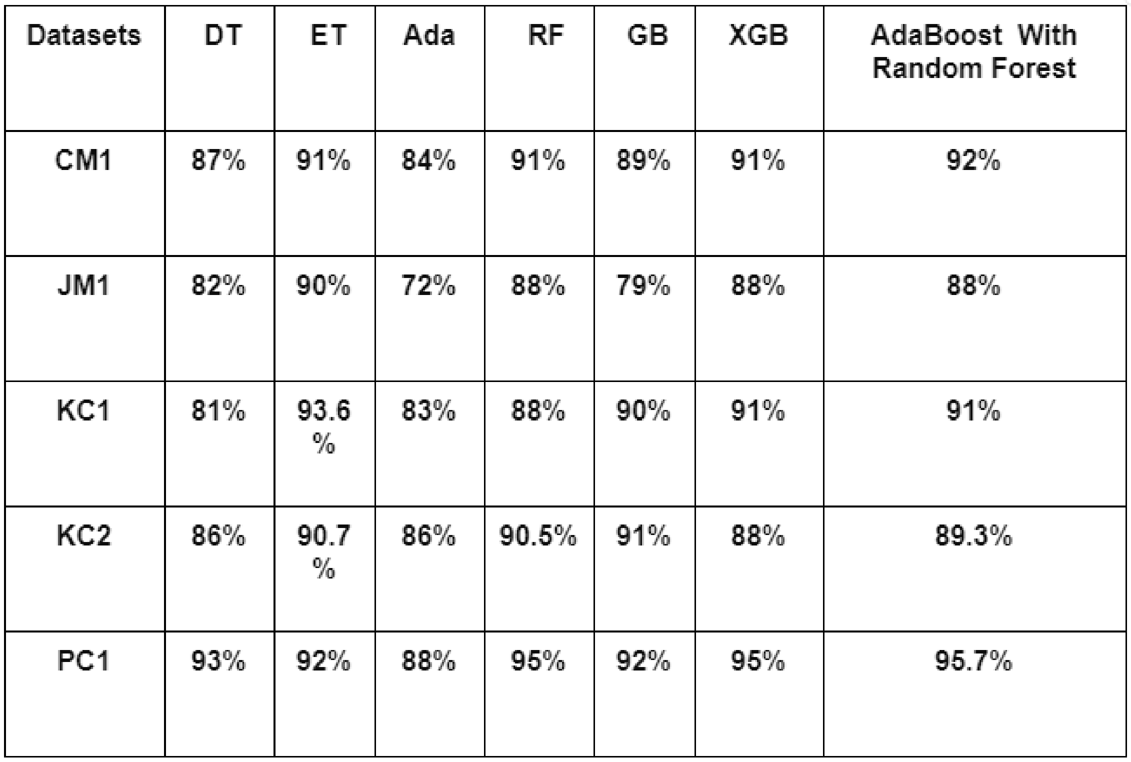
The F1 score stands as a prevalent metric for assessing the effectiveness of classification models, especially in situations featuring imbalanced datasets where one class might significantly outweigh the others. It amalgamates precision and recall into a unified measure, offering a well-rounded evaluation of a model's proficiency in recognizing positive samples while accurately mitigating false positives and false negatives. The formula for calculating the F1-score is as follows:

F1 = 2 · (Precision · Recall / Precision + Recall)

In this context, precision stands as the proportion of true positive predictions relative to the total predicted positive instances, signifying the accuracy of positive predictions. Meanwhile, Recall, also referred to as sensitivity or the true positive rate, quantifies the ratio of true positive predictions to the actual positive instances, portraying the model's capability to identify positive instances accurately. The F1-score, ranging between 0 and 1, serves as the amalgamation of precision and recall. A value of 1 indicates flawless precision and recall, while 0 suggests subpar performance. In sentiment analysis, where the accurate identification of positive and negative sentiments holds significant importance for meaningful analysis, the F1-score emerges as a crucial metric for model evaluation.

4.2 Individual Models Performance

4.2.1 Defect Prediction



Datasets	DT	ET	Ada	RF	GB	XGB	AdaBoost With Random Forest
CM1	87%	91%	84%	91%	89%	91%	92%
JM1	82%	90%	72%	88%	79%	88%	88%
KC1	81%	93.6 %	83%	88%	90%	91%	91%
KC2	86%	90.7 %	86%	90.5%	91%	88%	89.3%
PC1	93%	92%	88%	95%	92%	95%	95.7%

Figure 4.1: Defects Accuracy Result

For the Cm1 dataset, among all the models used, adaboost with random forest even though performs slightly better than ET,RF and XGB, it still shows the best accuracy. But for JM1 and KC1, Extra tree classifier performs the best. For KC2, Gradient Boosting and for PC1, adaboost with random forest show the best accuracy. Adaboost with Random forest works best for Pc1 among all the datasets. Among all the classifiers used, overall, the extra tree classifier shows the best accuracy when applied on the datasets.

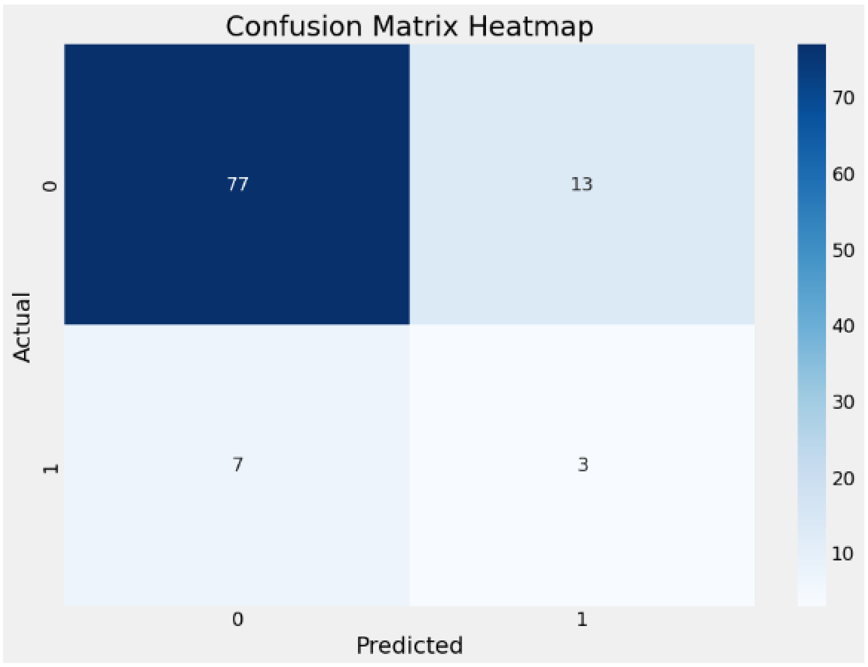
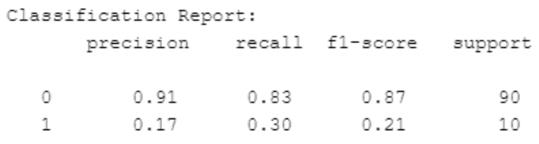


Figure 4.2: Confusion Matrix Defects



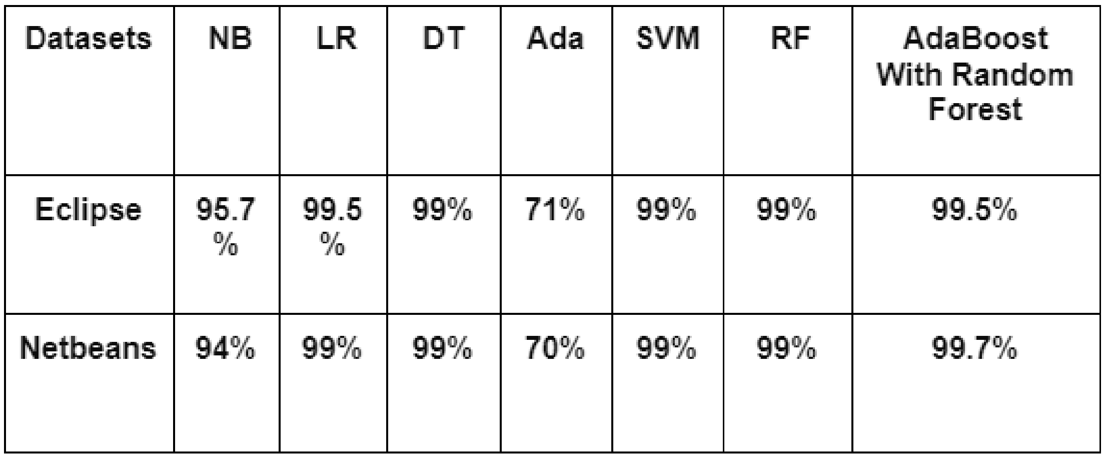
```
Classification Report:
              precision    recall  f1-score   support

   0       0.91      0.83      0.87        90
   1       0.17      0.30      0.21        10
```

Figure 4.3: Classification Report Defects

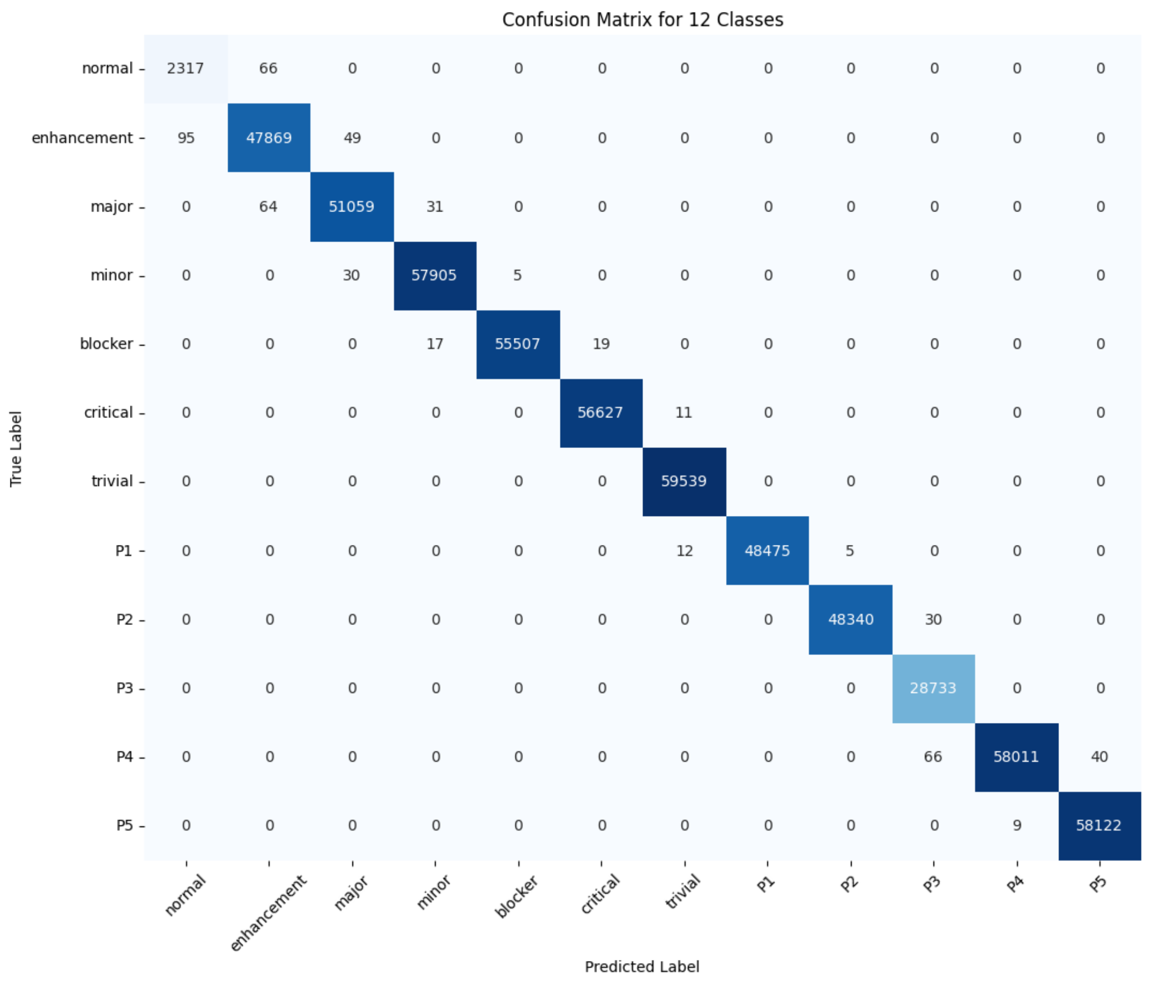
4.2.2 Bug Severity Level classification and Prioritization

The top-performing algorithms are logistic regression, decision tree, support vector machine, random forest, and AdaBoost with Random Forest, which routinely achieve accuracies of 99% or above. It looks to be the lowest performer in this collection of experiments, with accuracies around 70%. All algorithms perform equally on both datasets, with minor differences in accuracy. However, no major differences are identified between the Eclipse and Netbeans datasets. Random Forest and AdaBoost with Random Forest show that ensemble approaches can improve classification accuracy, with AdaBoost with Random Forest outperforming the others overall.



Datasets	NB	LR	DT	Ada	SVM	RF	AdaBoost With Random Forest
Eclipse	95.7 %	99.5 %	99%	71%	99%	99%	99.5%
Netbeans	94%	99%	99%	70%	99%	99%	99.7%

Figure 4.4: Bugs Accuracy Result


Confusion Matrix for 12 Classes

True Label \ Predicted Label	normal	enhancement	major	minor	blocker	critical	trivial	P1	P2	P3	P4	P5
normal	2317	66	0	0	0	0	0	0	0	0	0	0
enhancement	95	47869	49	0	0	0	0	0	0	0	0	0
major	0	64	51059	31	0	0	0	0	0	0	0	0
minor	0	0	30	57905	5	0	0	0	0	0	0	0
blocker	0	0	0	17	55507	19	0	0	0	0	0	0
critical	0	0	0	0	0	56627	11	0	0	0	0	0
trivial	0	0	0	0	0	0	59539	0	0	0	0	0
P1	0	0	0	0	0	0	12	48475	5	0	0	0
P2	0	0	0	0	0	0	0	0	48340	30	0	0
P3	0	0	0	0	0	0	0	0	0	28733	0	0
P4	0	0	0	0	0	0	0	0	0	66	58011	40
P5	0	0	0	0	0	0	0	0	0	0	9	58122

Figure 4.5: Confusion Matrix Bug

Chapter 5

Conclusions

In this chapter, the impact of the capstone project will be discussed. It has an impact on society and the environment. The ethical aspect and development of sustainability will be discussed while concluding the report and discussing the plan.

5.1 Societal, Health, Safety, Legal and Cultural Aspects

- **Improved Software Quality**: By predicting defects early, the quality of software released to the public improves, which enhances user satisfaction and trust in technology.
- **Economic Benefits**: High-quality software reduces downtime and maintenance costs, benefiting businesses and consumers economically.

5.1.1 Health and Safety

- **Reduction of Software Failures**: Critical systems, such as those in healthcare, transportation, and utilities, depend on reliable software. Predicting defects can prevent failures that might have severe health and safety consequences.
- **Stress Reduction for Developers**: Automating defect prediction reduces the pressure on developers, potentially improving their mental health and job satisfaction.

5.1.2 Legal Aspects

- **Compliance and Liability**: Automated tools must comply with software development regulations and standards. Predicting and mitigating defects can reduce the legal liabilities associated with software failures.
- **Data Privacy**: Ensuring that the tool respects data privacy laws, especially when handling sensitive or personal data within bug reports.

5.1.3 Cultural Aspects

- **Inclusive Development**: Incorporating diverse datasets can help create tools that are effective across different cultural contexts, ensuring broad applicability and fairness.
- **Global Collaboration**: Facilitating better software development practices can enhance collaboration between developers worldwide, promoting a more inclusive tech community.

5.2 Impact on Environment and Sustainability

5.2.1 Energy Efficiency

- **Resource Optimization**: Efficient defect prediction reduces the need for extensive debugging and testing processes, leading to lower energy consumption in development environments.
- **Sustainable Practices**: Promoting the use of green software engineering practices by identifying and fixing defects that may cause inefficient resource use.

5.2.2 Waste Reduction

- **Minimized Rework**: By identifying defects early, the need for significant rework is reduced, which in turn minimizes waste in terms of time and resources.

5.2.3 Sustainable Software Development

- **Long-term Reliability**: Developing reliable software contributes to sustainability by extending the software's life cycle and reducing the frequency of replacements.

5.3 Ethical and Professional Principles

5.3.1 Transparency and Accountability

- **Algorithmic Transparency**: Ensuring that the machine learning algorithms used in defect prediction are transparent and their decision-making processes are understandable.
- **Responsibility**: Developers and researchers must take responsibility for the accuracy and reliability of their predictions, addressing any biases or errors in the system.

5.3.2 Fairness and Non-Discrimination

- **Bias Mitigation**: Ensuring that the tool does not introduce or perpetuate biases, and that it works fairly across different demographic groups.
- **Equal Access**: Making the tool accessible to all developers, regardless of their resources or background.

5.3.3 Professional Integrity

- **Quality Assurance**: Adhering to high standards of software quality and to make sure that the tool contributes positively to the software development process.
- **Continuous Improvement**: Committing to ongoing research and improvement of the defect prediction models to keep up with evolving technology and software development practices.

By addressing these aspects, the project not only aims to improve the technical facets of software development but also ensures a positive impact on society, the environment, and upholds high ethical standards in the professional domain.

5.4 Brief Summary

In this project, we tried to build a sturdy defects prediction and bug severity and priority multi-label and multi class classification system by machine learning. Here, our target was to detect where there are any defects or not, and also to identify the bug severity and priority levels from bug reports. We obtained our datasets from Kaggle and Nasa promise repository. To predict bug severity and priority, firstly, we did data preprocessing since we used textual data. Next,any missing values are carefully handled, punctuations, URLs, HTML tags are removed to get clean data and data is converted to suitable format. Then multiple classification models such as Naive bayes, Logistic Regression , Support Vector ,Decision Tree , Random Forest pipeline, Extra Tree Classifier , bagging , adaboost p, gradient boosting , hist gradient boosting classifier are explored to figure out the most effective ones for this task. To initialize defect prediction tasks, firstly, since the data is numerical, scaling and normalization are applied in a data preprocessing pipeline. Then to reduce class imbalance, SMOTE, oversampling and undersampling are used on the scaled and split dataset. Then, various classification models are applied. However, individual models may show conflicting predictions for the same input. To solve this problem, a technique called ensemble learning is used to combine the strength of several models. The top performing models are merged through bagging and boosting so as to achieve more accurate results for example adaboost is used with random forest(which is used as the base estimator). A python library, streamlit, is used to build an interactive and well designed web application. Here, users can input the textual data of bug reports from open source such as Netbeans and Eclipse to predict the severity and priority. Also, users can predict whether the data contains any defect or not where '0' represents 'No' and '1' represents 'Yes'.

5.5 Future works

5.5.1 Enhancing Machine Learning Models

- **Algorithm Improvement**: Explore advanced machine learning and deep learning techniques to improve the accuracy and efficiency of defect prediction models.
- **Feature Engineering**: Develop new methods for extracting and selecting the most relevant features from software attributes and bug reports to enhance model performance.

5.5.2 Integration with Development Tools

- **Seamless Integration**: Work on integrating the defect prediction tool with popular development environments (IDEs) and version control systems (like GitHub, GitLab) for real-time predictions.
- **User Interface and Experience**: Improve the user interface to make it more intuitive and useful for developers, including features like visualizations of defect predictions and recommendations.

5.5.3 Scalability and Performance

- **Handling Large Datasets**: Enhance the tool's ability to process and analyze large-scale datasets efficiently, ensuring it remains responsive and useful in large projects.
- **Real-Time Analysis**: Develop capabilities for real-time defect prediction and analysis to assist developers during the coding process.

5.5.4 Cross-Project and Cross-Domain Applicability

- **Generalization**: Test and adapt the tool for use in different types of software projects (e.g., mobile apps, enterprise software, embedded systems) and different domains (e.g., healthcare, finance, automotive).

- **Transfer Learning**: Investigate the use of transfer learning techniques to apply models trained on one project to other similar projects, improving defect prediction accuracy across various contexts.

5.5.5 Addressing Bias and Fairness

- **Bias Detection and Mitigation**: Develop methods to identify and mitigate biases in the defect prediction models to ensure fair and unbiased predictions.

5.5.6 Collaborative Features

- **Team Collaboration**: Introduce features that facilitate better collaboration among development team members, such as shared dashboards, alerts, and reports on defect predictions.

- **Feedback Loops**: Implement mechanisms for developers to provide feedback on the tool's predictions, enabling continuous improvement and learning.

5.5.7 Predictive Maintenance and Evolution

- **Software Evolution**: Extend the tool to predict not only current defects but also potential future maintenance issues based on historical data and trends.

- **Long-Term Predictions**: Develop models that can make long-term predictions about software quality and reliability, helping in strategic planning and resource allocation.

5.5.8 Empirical Studies and Validation

- **Field Testing**: Conduct extensive empirical studies and field tests to validate the tool's effectiveness in real-world settings and gather insights for further refinement.

- **Benchmarking**:Compare the tool's performance with existing defect prediction models and tools to establish benchmarks and identify areas for improvement.

References

Alkhasawneh, M. S., et al. (2022). Software defect prediction through neural network and feature selections. *Applied Computational Intelligence and Soft Computing*, *2022*.

Aquil, M. A. I., & Ishak, W. H. W. (2020). Predicting software defects using machine learning techniques. *International Journal*, *9*(4), 6609–6616.

Bugayenko, Y., Bakare, A., Cheverda, A., Farina, M., Kruglov, A., Plaksin, Y., ... Succi, G. (2023). Prioritizing tasks in software development: A systematic literature review. *Plos one*, *18*(4), e0283838.

Ekanayake, J. (2021). Bug severity prediction using keywords in imbalanced learning environment. *Int. J. Inf. Technol. Comput. Sci.(IJITCS)*, *13*, 53–60.

Hirsch, T., & Hofer, B. (2022). Using textual bug reports to predict the fault category of software bugs. *Array*, *15*, 100189.

Iqbal, A., & Aftab, S. (2020). A classification framework for software defect prediction using multi-filter feature selection technique and mlp. *International Journal of Modern Education & Computer Science*, *12*(1).

Iqbal, A., Aftab, S., Ali, U., Nawaz, Z., Sana, L., Ahmad, M., & Husen, A. (2019). Performance analysis of machine learning techniques on software defect prediction using nasa datasets. *International Journal of Advanced Computer Science and Applications*, *10*(5).

Kim, J., & Yang, G. (2022). Bug severity prediction algorithm using topic-based feature selection and cnn-lstm algorithm. *IEEE Access*, *10*, 94643–94651.

Kukkar, A., Kumar, Y., Sharma, A., & Sandhu, J. K. (2023). Bug severity classification in software using ant colony optimization based feature weighting technique. *Expert Systems with Applications*, *230*, 120573.

Maddipati, S. S., et al. (2021). Software defect prediction using kpca & csanfis. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, *12*(9), 2429–2436.

Mehmood, I., Shahid, S., Hussain, H., Khan, I., Ahmad, S., Rahman, S., . . . Huda, S. (2023). A novel approach to improve software defect prediction accuracy using machine learning. *IEEE Access*.

Mustaqeem, M., & Siddiqui, T. (2022). An effective methodology for software defects prediction using principal component-based multi-layer perceptron (pc-mlp): A hybrid technique.

Panda, R. R., & Nagwani, N. K. (2023). An intuitionistic fuzzy representation based software bug severity prediction approach for imbalanced severity classes. *Engineering Applications of Artificial Intelligence*, *122*, 106110.

Pushpalatha, M., Mrunalini, M., & Sulav Raj, B. (2020). Predicting the priority of bug reports using classification algorithms. *Indian Journal of Computer Science and Engineering*, *11*(6), 811–818.

Samir, M., Sherief, N., & Abdelmoez, W. (2023). Improving bug assignment and developer allocation in software engineering through interpretable machine learning models. *Computers*, *12*(7), 128.

Shatnawi, M. Q., & Alazzam, B. (2022). An assessment of eclipse bugs' priority and severity prediction using machine learning. *International Journal of Communication Networks and Information Security*, *14*(1), 62–69.

Tan, Y., Xu, S., Wang, Z., Zhang, T., Xu, Z., & Luo, X. (2020). Bug severity prediction using question-and-answer pairs from stack overflow. *Journal of Systems and Software*, *165*, 110567.

Vaid, K., & Ghose, U. (2024). Machine learning based predictive analysis of software bug severity and priority. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(15s), 249–256.

Appendices

- **Defect**

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import re
```

From this snippet of code we can see that several libraries are imported for visualization and manipulation.

```
from matplotlib import pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score, accuracy_score,
roc_curve, confusion_matrix, multilabel_confusion_matrix,
classification_report
from sklearn.model_selection import GridSearchCV,
StratifiedKFold
```

Here, several libraries needed for data preprocessing are imported which also contains a collection of machine learning classifiers from sklearn library. The classifiers enclose a variety of algorithms/ML models such as SVC, LinearSVC,Logistic Regression, Multinomialnb. Also, a pipeline is reported which is required to merge multiple steps into a single workflow. Other libraries are OneVsRestClassifier and MultiOutputClassifier to handle multi-label Classification, LabelEncoder to target labels with values ranging between 0 and n-classes-1 .

```
correlation_matrix = df.corr()


# Plotting a heatmap for better visualization
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

In this snippet of code, correlation is calculated. Then a plot is set up so as to plot a heatmap. A title is added and the plot is displayed.

```
matplotlib.style.use('fivethirtyeight')
np.random.normal

scaler = preprocessing.RobustScaler()
robust_df = scaler.fit_transform(X)
robust_df = pd.DataFrame(robust_df, columns =columns)


scaler = preprocessing.StandardScaler()
standard_df = scaler.fit_transform(X)
standard_df = pd.DataFrame(standard_df, columns =columns)


MinMax_scaler = preprocessing.MinMaxScaler()
minmax_df = MinMax_scaler.fit_transform(X)
minmax_df = pd.DataFrame(minmax_df, columns =columns)


# Apply Box-Cox transformation to each column of the DataFrame
boxcox_df = X.apply(boxcox_transform)


fig, (ax1, ax2, ax3, ax4, ax5) =
plt.subplots(ncols = 5, figsize =(20, 5))

ax1.set_title('Before Scaling')
ploting(X, cols=columns, ax = ax1)
ax2.set_title('After Robust Scaling')
ploting(robust_df, columns, ax=ax2)
ax3.set_title('After Standard Scaling')
ploting(standard_df, columns, ax=ax3)
ax4.set_title('After Min-Max Scaling')
ploting(minmax_df, columns, ax = ax4)
ax5.set_title('After BoxCox normalization')
ploting(boxcox_df, columns, ax = ax5)
# sns.kdeplot(minmax_df['x1'], ax = ax4, color ='black')
# sns.kdeplot(minmax_df['x2'], ax = ax4, color ='g')
plt.show()


with open('minmax_scaler_cm1.pkl', 'wb') as f:
  pickle.dump(MinMax_scaler,f)
```

Here, in the provided code snippet, a comprehensive approach is demonstrated to scaling and normalizing a dataset in a data preprocessing pipeline. For that, Robust Scaling(to diminish the effect of outliers), Standard Scaling(to remove mean and to scale it to unit variance so as to standardize features), Min-max scaling(to scale each feature to given range so that the features can be transformed), Box Cox Transformation(applied to each column to stabilize variance). Then for the visualization of the scaling, 5-column subplots are created such as original data, after robust scaling, after standard scaling, after min max scaling and after box cox scaling.

```
from sklearn.model_selection import cross_val_score,
RepeatedStratifiedKFold,train_test_split


X_train, X_test, y_train, y_test = train_test_split(minmax_df,
y, test_size=0.2, random_state=42, stratify=y, shuffle=True)
```

Here, the scaled datasets are split into training and testing sets to ensure balanced evaluation of the model.

```
smote = SMOTE()
X_resampled_smote, y_resampled_smote =
smote.fit_resample(X_train, y_train)ros = RandomOverSampler()

X_resampled_ros, y_resampled_ros =
ros.fit_resample(X_train, y_train)rus = RandomUnderSampler()

X_resampled_rus, y_resampled_rus = rus.fit_resample(X_train,
y_train)

In this snippet, to address class imbalance,
SMOTE(to generate synthetic samples),
random oversampling(to duplicate minority class samples),
and random undersampling(to reduce majority class samples)

ada_clf = AdaBoostClassifier(RandomForestClassifier())

```

```
ada_clf.fit(X_resampled_smote, y_resampled_smote)

print_score(ada_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=True)

print_score(ada_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=False)

ada_clf = AdaBoostClassifier(base_estimator=
RandomForestClassifier())

ada_clf.fit(X_resampled_smote, y_resampled_smote)

print_score(ada_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=True)

print_score(ada_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=False)
```

In this code segment, RandomForestClassifier as the base estimator is trained with two adaboost classifiers and applied on resampled data. In the first one, the base estimator parameter is not specified explicitly, which means that a decision tree is used as default. Whereas, for the second one, the base estimator parameter is set to RandomForest explicitly.A 'print score' function is used to evaluate the model's performance. Whether using a RandomForestClassifier as the base estimator instead of the default decision tree improves the performance of the AdaBoost model can be assessed by comparing the results between these two classifiers.

```
bag_clf = BaggingClassifier(base_estimator=clf,
                            n_estimators=1000,
                            bootstrap=True, n_jobs=-1,
                            random_state=42)

bag_clf.fit(X_resampled_smote, y_resampled_smote)

print_score(bag_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=True)

print_score(bag_clf, X_resampled_smote, y_resampled_smote,
X_test , y_test , train=False)
```

Here, a Bagging Classifier is trained with a DecisionTreeClassifier as the base estimator which then creates an ensemble of 100 decision trees. The 'Bootstrap=True' parameter shows that to train each base estimator, bootstrap samples are used.

```
rf_clf = RandomForestClassifier(random_state=42)
params_grid = {"max_depth": [3, None],
               "min_samples_split": [2, 3, 10],
               "min_samples_leaf": [1, 3, 10],
               "bootstrap": [True, False],
               "criterion": ['gini', 'entropy']}
grid_search = GridSearchCV(rf_clf, params_grid,
                           n_jobs=-1, cv=5,
                           verbose=1, scoring='accuracy')
grid_search.fit(X_resampled_smote, y_resampled_smote)
```

In this code segment, a grid search is performed to search over the hyperparameter grid and takes the initialized RandomForestClassifier, number of jobs to run in parallel (n jobs=-1), the number of folds for cross-validation (cv=5), verbosity level (verbose=1), and the scoring metric (scoring='accuracy'). Then, the grid search is executed by fitting the RandomForestClassifier on the provided training data (X resampled smote, y resampled smote).

```
xt_clf = ExtraTreesClassifier(random_state=42)
xt_clf.fit(X_resampled_smote, y_resampled_smote)

print_score(xt_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=True)

print_score(xt_clf, X_resampled_smote, y_resampled_smote,
X_test, y_test, train=False)
```

In this code snippet, an ExtraTreeClassifier, which is an ensemble learning method, is trained. It is quite similar to random forest but has some differences in how some random splits are made. ExtraTreeClassifier is initialized with a fixed random state (random state=42). Then, it is trained on the resampled training data (X resampled smote, y resampled smote).

```
gbc_clf = GradientBoostingClassifier()
gbc_clf.fit(X_resampled_smote, y_resampled_smote)
```

Here, in this code snippet , GradientBoostingClassifier, which is an ensemble learning method, is fitted to resampled training data. A strong predictive model is built by joining the outputs of multiple weak learners, such as decision trees, in a sequential manner. Thus, gbc clf will be a trained GradientBoostingClassifier model ready to make predictions.

```
xgb_clf = xgb.XGBClassifier(max_depth=5, n_estimators=10000,
learning_rate=0.3,n_jobs=-1)

xgb_clf.fit(X_resampled_smote, y_resampled_smote)
```

Here, an XGBoost classifier, which is a popular implementation of gradient boosting, is trained. It is initialized with specified hyperparameters such as max depth, n estimators, and learning rate. It is known that its performance and efficiency in handling large datasets is quite high.

- **Bug**

```
data.sd = data.sd.str.lower()
data.head()
PUNCT_TO_REMOVE = string.punctuation
def remove_punctuation(text):
    """custom function to remove the punctuation"""
    return text.translate(str.maketrans('', '',
    PUNCT_TO_REMOVE))
from nltk.corpus import stopwords
import nltk
", ".join(stopwords.words('english'))
STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
    """custom function to remove the stopwords"""
return " ".join([word for word in str(text).split()
if word not in STOPWORDS])

from collections import Counter
cnt = Counter()
for text in data['sd'].values:
    for word in text.split():
        cnt[word] += 1
cnt.most_common(10)
FREQWORDS = set([w for (w, wc) in cnt.most_common(10)])
def remove_freqwords(text):
    """custom function to remove the frequent words"""
    return " ".join([word for word in str(text).split()
if word not in FREQWORDS])

from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
def stem_words(text):
    return " ".join([stemmer.stem(word)
    for word in text.split()])
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def lemmatize_words(text):
    return" ".join([lemmatizer.lemmatize(word)
for word in text.split()])
```

This code performs multiple text preprocessing steps on the ‘sd’ column of the dataset. All text in that column is converted to lowercase by using data.sd.str.lower(). Then punctuations are removed from a given text using remove punctuation. Then stopwords are imported from NLTK and common English stop words are removed from the text. Next, frequent words are identified. Word frequencies in the 'sd' column are counted using 'counter' and 10 most common words are stored in 'FREQ WORDS' . Afterwards, the most common words are removed by the function 'remove freqwords' . NLTK's 'PorterStemmer' is used to define the 'stem words' function which reduces words to their root form. Next, NLTK's 'WordNetLemmatizer' is used to define the 'lemmatize words' function which reduces words to their root form.

```
rom sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
tfidf_vectorizer = TfidfVectorizer(max_df=0.8,
max_features=10000)
xtrain_tfidf = tfidf_vectorizer.fit_transform(xtrain)
xval_tfidf = tfidf_vectorizer.transform(xval)
```

In this code segment , TF-IDF vectorization is imported, data is split into train-test. Then TF-IDF Vectorizer is initialized which is then fitted to transform the training data. The validated data is then transformed into a TF-IDF matrix using the previously fitted vectorizer, stored in xval tfidf.

```
logreg = LogisticRegression()
logreg_classifier = OneVsRestClassifier(logreg)
logreg_classifier.fit(xtrain_tfidf, ytrain)
predictions = logreg_classifier.predict(xval_tfidf)
from sklearn.metrics import accuracy_score
print("Accuracy score for Logistic Regression:")
print(accuracy_score(yval, predictions))
```

Here, to handle multi-class classification , a logistic regression classifier is wrapped in a One-vs-Rest(OvR) scheme which can train one classifier per class. This classifier can then distinguish the instances of one class from the rest. The OvR logistic regression classifier is then fitted on the TF-IDF transformed training data and prediction predictions are made on the validation set.

```
classifier = BinaryRelevance(GaussianNB())
classifier.fit(xtrain_tfidf, ytrain)
predictions = classifier.predict(xval_tfidf)
```

In this segment, binary relevance with Gaussian Naive Bayes is initialized. This is a strategy for multi-label classification where each label is treated as a single label classification problem.

```
confusion_matrix_flat = confusion_matrix.reshape(5, 4)
    plt.figure(figsize=(12, 8))
    sns.heatmap(confusion_matrix_flat, annot=True, fmt="d",
    cmap="YlGnBu", cbar=False)

    plt.xlabel('Component')
    plt.ylabel('Class')
    plt.title('Confusion Matrix Heatmap')
    tick_labels = ['TP', 'FP', 'FN', 'TN']
    plt.xticks([0.5, 1.5, 2.5, 3.5], tick_labels, rotation=0)
    y_labels = ['normal', 'enhancement', 'P1', 'P2',  'P4' ]
    plt.yticks(np.arange(0.5, len(y_labels)), y_labels,
    rotation=0)
    plt.show()
```

This code creates a confusion matrix heatmap of a reshaped confusion matrix. This is done using matplotlib and seaborn. The x-axis is labeled to ''Component' and the y-axis to 'Class'. This confusion matrix is visualized as a heatmap with a specified component and class labels so as to provide a clear graphical representation of the classification results.

```
NB_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('nb_model', OneVsRestClassifier(MultinomialNB(), n_jobs=-1))])
LR_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('lr_model', OneVsRestClassifier(LogisticRegression(),
n_jobs=-1))])
SVM_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('svm_model', OneVsRestClassifier(LinearSVC(), n_jobs=-1))])
DT_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('svm_model', OneVsRestClassifier(DecisionTreeClassifier(),
n_jobs=-1))])
RF_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('svm_model', OneVsRestClassifier(RandomForestClassifier(),
n_jobs=-1))])
ETC_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('ETC_model', OneVsRestClassifier(ExtraTreesClassifier(),
n_jobs=-1))])
BAGGING_pipeline = Pipeline([('tfidf', TfidfVectorize
(stop_words='english')),
('ETC_model', OneVsRestClassifier(BaggingClassifier(),
n_jobs=-1))])
ADA_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('ADA_model', OneVsRestClassifier(AdaBoostClassifier(),
n_jobs=-1))])
GBC_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('GBC_model', OneVsRestClassifier(GradientBoostingClassifier(),
n_jobs=-1))])
HGB_pipeline = Pipeline([('tfidf', TfidfVectorizer
(stop_words='english')),
('HGB_model', OneVsRestClassifier(HistGradientBoostingClassifier(),
n_jobs=-1))])
```

Here, several machine learning pipelines are created for text classification by using different classifiers, which are Naive bayes pipeline, Logistic Regression pipeline, Support Vector pipeline,Decision Tree pipeline, Random Forest pipeline, Extra Tree

Classifier pipeline, bagging pipeline, adaboost pipeline, gradient boosting pipeline, hist gradient boosting classifier pipeline with using OneVsRestClassifier scheme. These pipelines make it easy to switch between different models.

```
def run_ensemble_pipeline(pipeline, train_feats, train_lbls,
test_feats, test_lbls):
    print(type(train_feats))
    train_feats = np.array(train_feats) #train_feats.toarray()
    train_lbls = train_lbls.to_numpy()
    pipeline.fit(train_feats, train_lbls)
    predictions = pipeline.predict(test_feats)
    pred_proba = pipeline.predict_proba(test_feats)
```

In this snippet, firstly,trained features and labels are converted to NumPy Array. Then ensemble learning for the classification task is trained in the function run ensemble pipeline . Then, this trained pipeline is used to make predictions on the test features and then probabilities are predicted.

- **Web Application**

```
import streamlit as st
import numpy as np
import pickle
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import AdaBoostClassifier,
                      RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
import joblib
import sys
import importlib
```

From this snippet of code we can see that several libraries are imported for visualization and manipulation.

```
# Bug Data Input
st.header('Bug Severity Level Classification
and Prioritization')
text_options = ['Eclipse', 'Netbeans']
selected_text_dataset = st.selectbox
```

```
('Bug Dataset', text_options)
text_input = st.text_area('Enter Bug Report
(one sentence per line)',
'This is a good day\nThis is a bad day')

if st.button('Predict Severity Level and Prioritization '):
try:
if selected_text_dataset == 'Eclipse':
txt_pipeline = text_model.load_pipeline
("weights/bug_report/ET_pipeline_eclipse.pkl")

prediction, probability = text_model.predict_text
(txt_pipeline, text_input)

result_dict = {}

severity_labels = ['normal', 'enhancement', 'major', 'minor',
'blocker', 'critical', 'trivial']

priority_labels = ['P1', 'P2', 'P3', 'P4', 'P5']

for i, label in enumerate(severity_labels):
if prediction[0][i] == 1:
result_dict['bsr'] = label

for i, label in enumerate(priority_labels, start=7):
if prediction[0][i] == 1:
result_dict['pr'] = label

st.write("Prediction Results:")
st.write(f'- Severity Level: {result_dict.get("bsr",
"Unknown")}')
st.write(f'- Prioritization: {result_dict.get("pr",
"Unknown")}')

elif selected_text_dataset == 'Netbeans':
txt_pipeline = text_model.load_pipeline
("weights/bug_report/ET_pipeline_netbeans.pkl")
prediction, probability =text_model.predict_text(txt_pipeline,
text_input)

result_dict = {}

severity_labels = ['normal', 'enhancement']
priority_labels = ['P1', 'P2', 'P3', 'P4']
```

```

for i, label in enumerate(severity_labels):
 if prediction[0][i] == 1:
result_dict['bsr'] = label

for i, label in enumerate(priority_labels, start=2):
 if prediction[0][i] == 1:
result_dict['pr'] = label

st.write("Prediction Results:")
 st.write(f'- Severity Level: {result_dict.get("bsr",
 "Unknown")}')
st.write(f'- Prioritization: {result_dict.get("pr",
"Unknown")}')

else:
st.write('Please select a valid model option.')
except Exception as e:
 st.write("An error occurred:", e)
```

This code snippet is part of a Streamlit application that predicts the severity level and prioritizes bug reports based on user-supplied text input. The application begins by displaying a header and prompting the user to choose between two datasets, 'Eclipse' and 'Netbeans' via a dropdown menu. The user can then enter bug report text in a text field.When the user selects the prediction button, the program determines which dataset is selected and loads the pre-trained machine learning model pipeline. The text modelload pipeline function loads the model, whereas the text model predict text function generates predictions and probabilities based on the supplied text.For the 'Eclipse' dataset, the code compares the predictions to a set of predefined severity labels (such as 'normal', 'enhancement','major','minor', 'blocker', 'critical', 'trivial') and priority labels ('P1', 'P2', 'P3', 'P4', 'P5'). It populates a dictionary with the necessary labels based on the predictions. Similarly, the 'Netbeans' dataset compares to a different set of severity and priority labels.

```
#Defect Data Input
st.header('Defect Prediction')
st.markdown('Enter values for each feature:')

# Create select box for tabular dataset selection
tabular_options = ['CM1', 'JM1', 'KC1', 'KC2', 'PC1']
selected_tabular_dataset = st.selectbox('Dataset',
tabular_options)

# Create input fields for tabular data
feature_1 = st.number_input("loc(McCabe's line count of code)",
value=0)

feature_2 = st.number_input('iv(g)(McCabe "design complexity")'
,value=0)

feature_3 = st.number_input('n(Halstead total operators +
operands)',value=0)

feature_4 = st.number_input('d(Halstead "difficulty")',
value=0)

feature_5 = st.number_input('e(Halstead "effort")', value=0)
feature_6 = st.number_input("lOBlank
(Halstead's count of blank lines)",value=0)

feature_7 = st.number_input('uniq_Op (unique operators)',
value=0)

eature_8 = st.number_input('uniq_Opnd (unique operands)',
value=0)

feature_9 = st.number_input('branchCount', value=0)

tabular_data = [[feature_1, feature_2, feature_3, feature_4,
feature_5, feature_6, feature_7, feature_8, feature_9]]
```

This code snippet is part of Streamlit, to predict software Defect based on user-supplied feature values. The application begins by displaying a header labeled "Defect Prediction" and instructing users to enter values for each feature. The first interactive feature is a dropdown menu (selectbox) that allows users to select from five datasets: 'CM1', 'JM1', 'KC1', 'KC2', and 'PC1'. This option allows the prediction model to be tailored to the unique dataset used.

```
if st.button('Detect Defect'):
try:
 if selected_tabular_dataset == "CM1":
scaler = tabular_model.load_scaler
("weights/defect/scaler/minmax_scaler_cm1.pkl")

model_ = tabular_model.load_model
("weights/defect/model_cm1.pkl")

prediction = tabular_model.predict
(model=model_, scaler=scaler, data=tabular_data)

 st.write('Defect Prediction Result:', prediction[0])

elif selected_tabular_dataset == "JM1":
scaler = tabular_model.load_scaler
("weights/defect/scaler/minmax_scaler_jm1.pkl")

model_ = tabular_model.load_model
("weights/defect/model_jm1.pkl")

prediction = tabular_model.predict
(model=model_, scaler=scaler, data=tabular_data)

 st.write('Defect Prediction Result:', prediction[0])

elif selected_tabular_dataset == "KC1":
 scaler = tabular_model.load_scaler
 ("weights/defect/scaler/minmax_scaler_kc1.pkl")

model_ = tabular_model.load_model
("weights/defect/model_kc1.pkl")

prediction = tabular_model.predict
(model=model_, scaler=scaler, data=tabular_data)

st.write('Defect Prediction Result:', prediction[0])

elif selected_tabular_dataset == "KC2":
```

```
scaler = tabular_model.load_scaler
("weights/defect/scaler/minmax_scaler_kc2.pkl")

model_ = tabular_model.load_model(
"weights/defect/model_kc2.pkl")
prediction = tabular_model.predict
(model=model_, scaler=scaler, data=tabular_data)

st.write('Defect Prediction Result:', prediction[0])

elif selected_tabular_dataset == "PC1":
scaler = tabular_model.load_scaler
("weights/defect/scaler/minmax_scaler_pc1.pkl")

model_ = tabular_model.load_model
("weights/defect/model_pc1.pkl")

 prediction = tabular_model.predict
 (model=model_, scaler=scaler, data=tabular_data)

st.write('Defect Prediction Result:', prediction[0])
 else:
st.write("Please select a valid dataset option.")
```

When the user selects the ”Detect Defect” button, the application: Using bespoke tabular model functions, the program loads the corresponding Min-Max scaler and pre-trained model for the chosen dataset (CM1, JM1, KC1, KC2, or PC1). The user-provided feature values are scaled with the loaded scaler. The scaled data is fed into the model, which predicts whether a fault is present. The prediction results are displayed on the interface. If an invalid dataset is selected or an error occurs, the user is prompted to choose a valid dataset. This functionality allows users to predict potential software problems based on certain input data