DYNAMIC SELECTION OF HETEROGENEOUS ENSEMBLE TO IMPROVE BUG PREDICTION

A Report submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Information Technology

Submitted By

Akhil Attri	**2020UIN3356**
Pratik Koli	**2020UIN3357**
Hritik Meena	**2020UIN3358**

Under the supervision of

Dr. Ankita Bansal

Department of Information Technology

Netaji Subhas University of Technology

MAY 2024

CERTIFICATE

DEPARTMENT OF INFORMATION TECHNOLOGY

This is to certify that the work embodied in the Project-II Thesis titled, **Dynamic Selection of Heterogeneous Ensemble to Improve Bug Prediction** by *Akhil Attri (2020UIN3356)*, *Pratik Koli (2020UIN3357)* and *Hritik Meena (2020UIN3358)*, students of B.Tech., Department of Information Technology, under my guidance towards fulfilment of the requirements for the award of the degree of Bachelor of Technology. This work has not been submitted for any other diploma or degree of any University.

Dr. Ankita Bansal

Department of Information Technology

Netaji Subhas University of Technology

New Delhi

CANDIDATE(S) DECLARATION

DEPARTMENT OF INFORMATION TECHNOLOGY

We, *Akhil Attri (2020UIN3356)*, *Pratik Koli (2020UIN3357)* and *Hritik Meena (2020UIN3358)* of B. Tech., Department of Information Technology, hereby declare that the Project-II Thesis titled **Dynamic Selection of Heterogeneous Ensemble to Improve Bug Prediction** which is submitted by me/us to the Department of Computer Science and Engineering, Netaji Subhas University of Technology (NSUT) Dwarka, New Delhi in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology is original and not copied from the source without proper citation. The manuscript has been subjected to plagiarism check by Turnitin software. This work has not previously formed the basis for the award of any Degree.

DATE:

Akhil Attri	**Pratik Koli**	**Hritik Meena**
(2020UIN3356)	(2020UIN3357)	(2020UIN3358)

CERTIFICATE OF DECLARATION

DEPARTMENT OF INFORMATION TECHNOLOGY

This is to certify that the Project-II Thesis titled **Dynamic Selection of Heterogeneous Ensemble to Improve Bug Prediction** which is being submitted by *Akhil Attri (2020UIN3356)*, *Pratik Koli (2020UIN3357)* and *Hritik Meena (2020UIN3358)* to the Department of Information Technology, Netaji Subhas University of Technology (NSUT) Dwarka, New Delhi in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology, is a record of the thesis work carried out by the students under my supervision and guidance. The content of this thesis, in full or in parts, has not been submitted for any other degree or diploma.

DATE:

Dr. Ankita Bansal

Department of Information Technology

Netaji Subhas University of Technology

New Delhi

ACKNOWLEDGEMENT

We, the undersigned, would like to extend our heartfelt gratitude to the individuals who have contributed to the completion of this group project. Firstly, we are immensely thankful to each other as group members, Akhil Attri, Pratik Koli and Hritik Meena, for our collective effort, collaboration, and dedication throughout this project. Our teamwork and mutual support have been crucial in achieving our objectives and producing this work. We also extend our appreciation to our supervisor, Dr. Ankita Bansal, for their guidance, feedback, and support during the development of this project. Their expertise and mentorship have been instrumental in shaping our work and ensuring its academic rigor.

Furthermore, we acknowledge the faculty and staff at NSUT for providing the necessary resources, facilities, and academic environment that facilitated our group's research and learning process. Lastly, we want to express our gratitude to our friends and family for their encouragement, understanding, and patience during this project.

Together, we have successfully completed this group project, and we are thankful to everyone who has been a part of this collaborative endeavor.

Akhil Attri	**Pratik Koli**	**Hritik Meena**
(2020UIN3356)	(2020UIN3357)	(2020UIN3358)

ABSTRACT

In the pursuit of accurately predicting the defect proneness of source code files, various approaches have been explored within the research community. These approaches leverage diverse predictors, such as product or process metrics, and employ machine learning classifiers to categorize classes as either buggy or non-buggy, or to estimate the likelihood of encountering a fault in the near future. One notable contribution in this domain is the Adaptive Selection of Classifiers in Bug Prediction (ASCI) method, which dynamically selects the most suitable classifier from a pool of machine learning models based on class characteristics.

In this paper, we present a comprehensive analysis of the strategies employed in the ASCI methodology, focusing on the evaluation of single classifiers, ensemble techniques, and ASCI itself. Our study extends beyond a mere replication of ASCI's methodology; we critically assess its performance, identify limitations, and propose enhancements to further improve dynamic classifier selection in bug prediction tasks.

A key aspect of our research is the exploration of advanced machine learning algorithms for dynamic classifier selection. We depart from ASCI's use of Decision Tree as the core classifier selector and instead leverage Gradient Boost, a powerful ensemble technique known for its ability to handle complex data and capture nuanced relationships within datasets. By integrating Gradient Boost into our adaptive method, we aim to enhance bug prediction accuracy and robustness, addressing some of the shortcomings observed in traditional classifier selection approaches.

Our empirical evaluation, conducted on a diverse dataset comprising 30 software systems, demonstrates the efficacy of our enhanced approach compared to ASCI and standalone classifiers. The results showcase improved bug prediction performance, highlighting the potential of combining sophisticated machine learning algorithms with adaptive classifier selection strategies. Our findings contribute to the ongoing discourse on bug prediction methodologies and underscore the importance of continuous refinement and innovation in this domain.

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

In the intricate landscape of software development, practitioners encounter a myriad of challenges on a daily basis. From navigating the ever-evolving technological landscape to meeting stringent project deadlines, developers are constantly striving to ensure the integrity and functionality of the software they produce. However, amidst these challenges, one critical issue persists: the effective identification and mitigation of software bugs. The prevalence of bugs not only undermines the quality and reliability of software systems but also imposes significant costs in terms of time, resources, and user satisfaction.

To address this pressing concern, the research paper at hand delves into the realm of bug prediction models and ensemble techniques, which offer promising strategies for optimizing testing efforts in software development. The paper explores the intricate interplay between machine learning, data analysis, and software engineering practices to develop predictive models capable of anticipating the occurrence of bugs in software systems. By leveraging a diverse array of classifiers and methodologies, the paper aims to enhance the accuracy and efficiency of bug prediction, ultimately leading to improved software quality and reliability.

Central to the paper's exploration is the investigation of various bug prediction strategies, including within-project and cross-project approaches. Within-project strategies involve training bug prediction models using data specific to the project under analysis, while cross-project strategies leverage data from similar software projects. The paper scrutinizes the effectiveness of these strategies in different contexts, considering factors such as data availability, project specificity, and classifier performance.

Furthermore, the paper delves into the burgeoning field of ensemble techniques for bug prediction. Ensemble methods, which combine multiple classifiers to enhance prediction accuracy, have gained traction in recent years due to their potential to mitigate the limitations of individual classifiers. The paper investigates the efficacy of ensemble techniques such as the Validation and Voting (VV) strategy, as well as explores novel approaches to ensemble decision-making aimed at improving prediction accuracy and robustness.

In addition to traditional bug prediction methodologies, the paper explores innovative approaches, such as adaptive prediction models, which dynamically recommend the most suitable classifier based on the structural characteristics of software components. By harnessing the power of machine learning and data-driven insights, these adaptive models seek to address the inherent complexities of bug prediction in real-world software development scenarios.

Overall, the research paper aims to contribute to the ongoing discourse on bug prediction in software engineering by offering novel insights, methodologies, and empirical findings. By advancing our understanding of bug prediction techniques and their practical applications, the paper endeavors to empower software developers with the tools and knowledge needed to build more reliable and resilient software systems.

1.1 Motivation

Our dedication to advancing bug prediction methods stems from our commitment to enhancing software functionality and user experience. In the intricate realm of software development, identifying and rectifying bugs is paramount to ensure seamless user interaction. While traditional bug prediction approaches treat all projects uniformly, we acknowledge the inherent diversity among software projects, each characterized by unique structural, design, and contextual attributes. Hence, we embark on a novel approach to bug prediction known as dynamic classifier selection. This methodology recognizes the diversity of software projects and endeavors to devise intelligent algorithms capable of selecting the most suitable bug-predicting method for each project dynamically. Rather than adhering rigidly to a single approach, our objective is to remain adaptable, adjusting to the evolving needs of each project. The process of selecting the optimal bug-predicting method is akin to an ongoing journey. We continuously assess the performance of various methods and choose the most fitting one as the project evolves. Through this iterative process, our aim is to develop a bug prediction model that not only achieves accuracy but also aligns seamlessly with the dynamic nature of contemporary software development practices. In essence, our objective is to revolutionize bug prediction by introducing greater intelligence and flexibility into the process. We achieved this by exploring a new paradigm of selecting bug-predicting methods tailored to the unique characteristics of each project. By doing so, we aspire to enhance the reliability of software systems and provide developers with a tool that adapts to the ever-changing landscape of software development. Central to our research is a comprehensive examination of various machine learning classifiers. We seek to understand their strengths and weaknesses in bug prediction contexts, considering factors such as project size, programming language, and development methodology. By evaluating the adaptability of classifiers to the evolving characteristics of software projects, we aim to refine bug prediction methodologies. What sets our research apart is its potential to transform bug prediction models from mere academic exercises into practical tools for software developers. We endeavor to provide effective strategies that bolster the reliability and resilience of software systems, particularly considering their escalating complexity. In essence, our mission revolves around enhancing the intelligence and adaptability of bug prediction. By exploring dynamic classifier selection, we aim to tailor bug-predicting methods to the specific attributes of each software project. Our goal is to bolster the reliability of software systems and provide developers with a tool that aligns with the fluidity of modern software development practices. Central to our research is an exhaustive examination of diverse machine learning classifiers. We meticulously evaluate their strengths and weaknesses in bug prediction scenarios, considering factors such as project size, programming language, and development methodology. By assessing the adaptability of classifiers to the evolving characteristics of software projects, we seek to refine bug prediction methodologies. What sets our research apart is its potential to transform bug prediction models from mere academic concepts into practical tools for software developers. We envision furnishing effective strategies that enhance the resilience and dependability of software systems, especially in the face of escalating complexity. While conventional bug prediction methods remain prevalent, our research represents a pioneering endeavor into a new frontier of adaptability and responsiveness. Dynamic classifier selection holds the promise of significantly improving the accuracy and effectiveness of bug prediction—an aspect that has been relatively underexplored in current research endeavors. Our vision extends to the creation of a dynamic tool capable of continuously evolving and improving its predictions in real-time. This tool would possess the intelligence to discern the unique requirements of each software project, thereby revolutionizing bug prediction. The novelty of our approach lies in its adaptability, a facet that has yet to be extensively explored in the realm of bug prediction. As we venture into uncharted territories, our ultimate goal remains steadfast—to empower software developers with a toolkit that not only enhances the accuracy of bug predictions but also seamlessly adapts to the ever-changing landscape of software development. Our research serves as a beacon, illuminating the path towards a future where adaptability reigns supreme in bug prediction methodologies. While conventional bug prediction methods persist, our research represents a pioneering effort to introduce a more adaptable and effective approach. We envision developing a sophisticated tool capable of dynamically adjusting and improving predictions in real-time, thereby significantly enhancing bug prediction accuracy. As we chart this exciting course, we envision a future where dynamic classifier selection becomes integral to bug prediction. Our primary objective remains clear: to furnish software developers with a toolkit that not only enhances bug prediction accuracy but also seamlessly integrates with the evolving nature of software development projects.

In summary, our passion lies in reshaping the landscape of bug prediction through dynamic classifier selection. This journey represents an adventure into uncharted territory, guided by the goal of providing software developers with superior bug prediction methods that adapt and evolve alongside the dynamic nature of software development.

1.2 Problem Statement

In the realm of software development, practitioners face numerous challenges daily, balancing evolving requirements with strict deadlines while ensuring code integrity. However, limited resources often hinder thorough testing, necessitating efficient resource allocation to focus on bug-prone code segments. Bug prediction models, employing machine learning classifiers like Logistic Regression, offer a solution by forecasting class bug probabilities. These models, trained within or across projects, rely heavily on classifier choice, as different methods can significantly impact prediction accuracy.

Ensemble techniques, like the Validation and Voting (VV) strategy, aim to enhance prediction accuracy by combining classifier outputs. However, traditional ensembles may overlook individual classifier strengths. Local bug prediction clusters similar data for separate model creation, but its applicability within-project is limited by small cluster sizes.

An innovative solution, the dynamic classifier, recommends classifiers based on class structural characteristics, aiming to enhance accuracy. Exploring single classifiers, within-project vs. cross-project prediction, and ensemble techniques, we delve into bug prediction methodology nuances. Various classifiers, each with unique strengths, and limitations, offer diverse bug prediction strategies. Understanding within-project and cross-project distinctions is crucial, as each strategy presents its own challenges and opportunities.

Ensemble techniques, like Stacking and Adaptive Selection of Classifiers in bug prediction (ASCI), leverage multiple classifiers to improve prediction accuracy. ASCI, in particular, diverges from traditional ensemble methods by predicting the most suitable classifier for bug proneness without relying on base classifier predictions.

In summary, bug prediction methodologies encompass diverse classifiers, prediction strategies, and ensemble techniques. Understanding these methodologies is vital for devising effective bug prediction approaches to meet the evolving needs of software development projects.

Chapter 2

LITERATURE REVIEW

Bug prediction research entails crafting predictive models using diverse predictors like CK metrics, process metrics, and history-based metrics. Surveys in academic literature comprehensively outline these methodologies, cataloging various approaches. The discourse navigates through key methodologies, including single classifiers, within- and cross-project strategies, and ensemble techniques. Single classifiers, such as Logistic Regression and Decision Trees, are leveraged to discern patterns within datasets for bug prediction. However, the search for a superior classifier remains inconclusive, with performance often context-dependent.

Within-project prediction relies on project-specific historical data, allowing insights from past iterations to inform future bug predictions. On the other hand, cross-project prediction extends its scope across multiple projects, grappling with the challenge of heterogeneous datasets. Each strategy presents distinct advantages and challenges, with within-project prediction requiring robust project histories for accuracy, while cross-project prediction deals with dataset heterogeneity.

Ensemble techniques, like Stacking, emerge as potent strategies to enhance prediction accuracy across both within- and cross-project scenarios. These methods leverage the collective wisdom of diverse classifiers to mitigate the limitations of individual models and improve overall predictive performance.

Navigating the Spectrum: Within-Project vs. Cross-Project Bug Prediction
Bug prediction stands as a pivotal component of software development, aiming to anticipate and address potential software flaws before they manifest into critical issues. At the heart of bug prediction lie various methodologies, each with its unique approach and considerations. Among these methodologies, two prominent strategies dominate

the discourse: within-project bug prediction and cross-project bug prediction. This comprehensive exploration endeavors to delve into the intricacies of these strategies, unraveling their nuances, advantages, and limitations.

Understanding Within-Project Bug Prediction

Within-project bug prediction revolves around the notion of training classification models exclusively on historical data derived from the same software project. This strategy capitalizes on the wealth of project-specific information embedded within historical records, encompassing past faults, issue resolutions, and development trends. The fundamental premise underlying within-project bug prediction is the notion that historical data harbors valuable insights into recurring patterns and trends within the project's codebase.

The Significance of Historical Data

Central to within-project bug prediction is the availability of comprehensive historical data repositories. Mature projects, characterized by extensive development histories, are particularly conducive to within-project bug prediction methodologies. These projects boast rich repositories of historical data, spanning multiple iterations, releases, and bug fixes. By leveraging this wealth of information, classification models can discern intricate patterns indicative of potential bugs, thereby facilitating targeted bug identification and remediation.

Tailored Predictive Models

One of the primary advantages of within-project bug prediction lies in its ability to generate tailored predictive models finely attuned to the intricacies of the project at hand. Unlike generic bug prediction approaches, which adopt a one-size-fits-all mentality, within-project models are custom-tailored to reflect the idiosyncrasies of the project's codebase, architecture, and development practices. Consequently, these models exhibit heightened accuracy and efficacy in identifying and mitigating project-specific bugs.

Challenges and Considerations

Despite its merits, within-project bug prediction is not without its challenges. Chief among these challenges is the prerequisite for substantial historical data, which renders this approach less viable for nascent or newly initiated projects. Moreover, within-project models may suffer from overfitting, wherein the model becomes excessively attuned to historical data patterns, thereby limiting its generalizability to future instances.

Unveiling the Cross-Project Strategy

In stark contrast to within-project bug prediction, the cross-project strategy entails training classification models using data sourced from diverse software projects. This approach, born out of necessity in scenarios characterized by data scarcity within individual projects, seeks to capitalize on the collective wisdom gleaned from disparate sources. By aggregating insights from multiple projects, cross-project bug prediction endeavors to enhance the predictive capabilities of classification models, thereby compensating for the limitations imposed by data scarcity.

Mitigating Data Scarcity

One of the primary motivations behind the adoption of cross-project bug prediction is the mitigation of data scarcity inherent in certain project contexts. In instances where a single project lacks sufficient historical data to train robust bug prediction models, cross-project methodologies offer a pragmatic solution. By leveraging data from multiple projects, these methodologies expand the pool of available data, enriching the training dataset and bolstering the predictive accuracy of classification models.

Addressing Data Heterogeneity

However, the adoption of cross-project bug prediction introduces its own set of challenges, chief among them being data heterogeneity. Unlike within-project scenarios, where historical data shares a common context and lineage, cross-project datasets are characterized by inherent diversity. Each project contributes its unique coding practices, development methodologies, and architectural nuances, culminating in a heterogeneous amalgamation of data.

Balancing Generalizability and Specificity

Navigating the intricacies of data heterogeneity poses a formidable challenge in cross-project bug prediction. On one hand, classification models must exhibit robust generalizability, capable of extrapolating insights from diverse project contexts. On the other hand, these models must retain a degree of specificity, acknowledging and accommodating the unique characteristics of each contributing project. Achieving this delicate balance between generalizability and specificity constitutes a central tenet of cross-project bug prediction methodologies.

Comparative Analysis

Having explored the fundamental principles and considerations underlying within-project and cross-project bug prediction, it is imperative to conduct a comparative analysis to elucidate the respective strengths and limitations of each approach.

Within-Project Bug Prediction

Pros:

Tailored Predictive Models: Within-project bug prediction facilitates the development of bespoke predictive models finely attuned to the nuances of the project's codebase.

Enhanced Accuracy: By leveraging comprehensive historical data, within-project models exhibit heightened accuracy in identifying and mitigating project-specific bugs.

Cons:

Data Dependency: Within-project bug prediction necessitates access to substantial historical data, rendering it less viable for nascent or newly initiated projects.

Overfitting Risk: Models may succumb to overfitting, wherein they become excessively attuned to historical data patterns, limiting generalizability.

Cross-Project Bug Prediction

Pros:

Data Augmentation: Cross-project bug prediction enriches the training dataset by aggregating insights from diverse project contexts, mitigating data scarcity.

Knowledge Transfer: By leveraging collective wisdom from multiple projects, cross-project methodologies facilitate knowledge transfer and cross-pollination of ideas.

Cons:

Data Heterogeneity: Managing data heterogeneity poses a significant challenge, requiring sophisticated strategies to reconcile divergent project contexts.

Generalizability Concerns: Models must strike a delicate balance between generalizability and specificity, navigating the nuances of diverse project landscapes.

Research paper [8] investigates the application of classifier ensembles in software defect prediction through an industrial case study. By combining predictions from multiple classifiers, the research aims to enhance the accuracy of defect prediction models, thereby improving debugging and software quality assurance processes. The study offers insights into the practical challenges and opportunities associated with using classifier ensembles in real-world software development scenarios.

Similarly, research paper [9] focuses on software defect prediction using classifier ensembles to improve defect detection accuracy. Through various ensemble learning techniques and classifier combination strategies, the research aims to address limitations of individual classifiers and enhance overall prediction reliability. By integrating predictions from diverse classifiers, the study seeks to advance software quality assurance practices through empirical evaluations and comparative analyses.

Additionally, research paper [14] explores software defect prediction by examining the consistency among different classifiers in identifying defects across diverse datasets and projects. The study aims to assess the degree of consensus and variability in defect prediction outcomes, providing insights into the reliability and robustness of defect prediction methodologies. Through empirical analyses, the research sheds light on factors influencing classifier agreement and disagreement in defect prediction tasks.

Research paper [20] delves into enhancing the RBF-DDA algorithm's robustness for predicting fault-prone software modules. By incorporating neural networks, the study aims to augment the algorithm's predictive capabilities, improving its accuracy and reliability across diverse datasets and software domains. Through empirical evaluations, the research aims to demonstrate the effectiveness of these enhancements in bolstering predictive accuracy, contributing to advancements in software fault prediction methodologies.

In contrast, research paper [21] introduces a tailored RBF neural network for estimating defect probabilities in software modules. By focusing on defect prediction tasks, the study aims to enhance the accuracy and reliability of defect probability estimations using a novel neural network architecture. Through empirical evaluations, the research seeks to showcase the efficacy of this approach in defect prediction, potentially outperforming traditional methods and advancing software quality assurance practices.

These papers collectively highlight the potential of neural network-based approaches in improving the accuracy, reliability, and robustness of software defect prediction methodologies. Through empirical evaluations and experimentation, the research aims to provide insights into the effectiveness of these approaches in real-world software development scenarios, contributing to enhancements in software quality and reliability.

Research paper [5] explores cross-project defect prediction models, aiming to improve defect prediction accuracy and reliability by leveraging data from similar software projects. Through empirical evaluations, the study seeks to overcome limitations of within-project defect prediction models and enhance the generalizability of defect prediction methodologies across diverse software contexts. Similarly, research paper [28] focuses on classifier combination for cross-project defect prediction, investigating the effectiveness of ensemble learning techniques to improve prediction accuracy and reliability across various software projects. By identifying optimal classifier combinations, the research aims to mitigate biases and errors, thereby enhancing defect prediction models' robustness in software quality assurance practices.

Moreover, research paper [29] delves into multi-objective cross-project defect prediction methodologies, aiming to develop comprehensive defect prediction models optimizing multiple performance metrics simultaneously. By exploring trade-offs between different objectives, such as predictive accuracy and false positive rate, the study aims to identify Pareto-optimal solutions and inform the design of more versatile defect prediction models. Lastly, research paper [30] conducts a large-scale experiment on cross-project defect prediction, examining the interplay between data, domain, and process factors in model performance. Through rigorous empirical study, the research seeks to uncover key factors influencing model generalizability and reliability, providing insights into effective defect prediction methodologies for real-world software development scenarios.

Research paper [17] conducts a comparative analysis of change metrics and static code attributes for defect prediction, aiming to determine their effectiveness in software engineering tasks. The study evaluates the advantages and limitations of utilizing data from within a specific project (local) versus drawing insights from diverse projects (global) for predictive modeling. Through empirical analyses, the research aims to provide nuanced insights into the selection of data sources to enhance the accuracy and reliability of predictive models in software development practices. By elucidating the impact of data locality on predictive model performance, the study contributes to informed decision-making processes in software engineering.

Similarly, research paper [13] explores the construction of ensemble models for software defect prediction, focusing on diversity selection. The study investigates the process of building ensemble learning models that leverage the diversity of individual classifiers to improve the accuracy and robustness of defect prediction models. Through ensemble techniques, the research aims to enhance the generalization capabilities and predictive performance of defect prediction models. By identifying optimal configurations for ensemble models in software defect prediction, the study advances predictive modeling techniques for software quality assurance practices.

In contrast, research paper [15] introduces the PROMISE repository, an invaluable resource for empirical software engineering research. The repository serves as a comprehensive repository of datasets, tools, and resources pertinent to empirical software engineering endeavors. By offering access to a diverse array of empirical data, the PROMISE repository aims to facilitate transparency, reproducibility, and collaboration within the software engineering community. Through the curation and organization of empirical data from various sources, the repository enables researchers to conduct rigorous analyses, validate hypotheses, and derive meaningful insights into software development practices and methodologies.

Research paper [15] presents the PROMISE repository, a valuable asset for empirical software engineering research. This repository offers a wide range of datasets, tools, and resources crucial for conducting empirical studies in software engineering. By providing access to diverse empirical data, the PROMISE repository aims to promote transparency, reproducibility, and collaboration within the software engineering community. With curated and organized data from various sources, researchers can perform rigorous analyses, validate hypotheses, and gain meaningful insights into software development practices and methodologies.

Research paper [9] delves into software defect prediction using classifier ensembles, aiming to enhance predictive models' accuracy and reliability in software quality assurance. Through empirical evaluations, the study explores the construction and application of ensemble learning models for defect prediction tasks. By identifying optimal ensemble configurations and methodologies, the research contributes to advancements in predictive modeling techniques for software defect management.

In a similar vein, research paper [14] investigates the predictive capabilities of different classifiers in software defect prediction tasks. Through a comparative analysis, the study examines various classifiers to understand their effectiveness and overlap in defect identification. By exploring the consistency and discrepancies in defect prediction outcomes across different classifiers, the research sheds light on the diversity and complementarity of predictive models in software defect prediction, providing insights for selecting and utilizing classifiers in software quality assurance practices.

Research paper [25] focuses on benchmarking classification models for software defect prediction, aiming to establish a comprehensive framework for fair comparisons. The study's objective is to offer practitioners and researchers valuable insights into the effectiveness of various classification models across diverse datasets and experimental conditions. Through rigorous experimentation and analysis, the research aims to reveal new insights into the strengths and limitations of different classification methodologies.

The study systematically assesses various classification models, analyzing their predictive performance and investigating how dataset characteristics affect model efficacy. Through meticulous experiments and analyses, the research aims to pinpoint crucial factors influencing classification model performance and recommend standardized evaluation methods to improve result reliability and reproducibility. Ultimately, the goal is to provide stakeholders in software defect prediction with practical insights to guide their selection and use of classification models in practical settings.

In contrast, research paper [26] takes a critical stance towards the quality of data within the NASA software defect datasets. Building upon prior research utilizing these datasets for empirical analysis, the study conducts a detailed examination of potential biases, inconsistencies, and limitations inherent in the data. Through a meticulous critique, the research highlights the need for greater transparency and scrutiny in the utilization of datasets for empirical research in software defect prediction and empirical software engineering.

The study underscores the importance of robust data collection, curation, and reporting practices, advocating for heightened vigilance among researchers and practitioners when interpreting findings derived from datasets. By raising awareness of data quality concerns, the research aims to foster a culture of rigor and accountability within the empirical software engineering community. Ultimately, the goal is to enhance the credibility and reliability of research outcomes in the field of software defect prediction by addressing data quality issues head-on.

Research paper [16] introduces a comprehensive metrics suite tailored for object-oriented design, aiming to provide developers with effective tools for assessing and improving software quality. Through meticulous analysis and experimentation, the research identifies a set of key metrics that capture essential aspects of object-oriented design, including encapsulation, inheritance, coupling, and cohesion. By leveraging these metrics, developers can gain valuable insights into the quality and maintainability of their software systems, facilitating informed decision-making throughout the software development lifecycle.

In contrast, research paper [17] conducts a comparative analysis of change metrics and static code attributes to evaluate their efficacy for defect prediction. Through extensive experimentation and statistical analysis, the research examines the predictive power of these two types of metrics in identifying potential defects within software systems. By examining a range of metrics and their associations with defect incidents, the research scrutinizes the relative advantages and drawbacks of change metrics in comparison to static code attributes for predicting defects.

Research paper [18] introduces a developer-centered bug prediction model, emphasizing the importance of incorporating developer-centric factors into defect prediction frameworks. By considering developers' characteristics, such as experience, expertise, and coding habits, the research aims to enhance the accuracy and relevance of bug prediction models. Through empirical studies and modeling techniques, the research seeks to elucidate the impact of developer-centric factors on defect proneness and devise tailored bug prediction strategies that resonate with developers' needs and workflows.

Lastly, research paper [19] explores the concept of collective personalized change classification with multi-objective search, proposing a novel approach to change classification in software development. By harnessing multi-objective search algorithms, the research aims to optimize change classification processes by considering multiple criteria simultaneously, such as accuracy, recall, and developer preferences. Through empirical evaluations and algorithmic refinements, the study seeks to advance change classification techniques and empower developers with personalized and effective tools for managing software changes.

Chapter 3

RESEARCH METHODOLOGY

3.1 Dataset

The dataset that we have used has also been used by the paper titled "Dynamic Selection of Classifiers in Bug Prediction: An Adaptive Method" by Dario et al. This dataset contains various metrics of the classes of 30 real word Apache projects. It includes features like CK Metrics (Depth of Inheritance Tree, Function Cohesion), Lines of Codes, Buggy or not etc.

Merits of this dataset :

Diverse Project Representation: The dataset covers a range of software projects and systems, offering diversity in project types, technologies, and development contexts. This diversity enables researchers to explore various facets of software engineering and bug prediction across different domains.

Real-World Relevance: The data in the dataset is derived from actual software projects, making it highly relevant to real-world software development scenarios. Researchers can analyze real-world data patterns, bug occurrences, and development practices, enhancing the applicability of their findings.

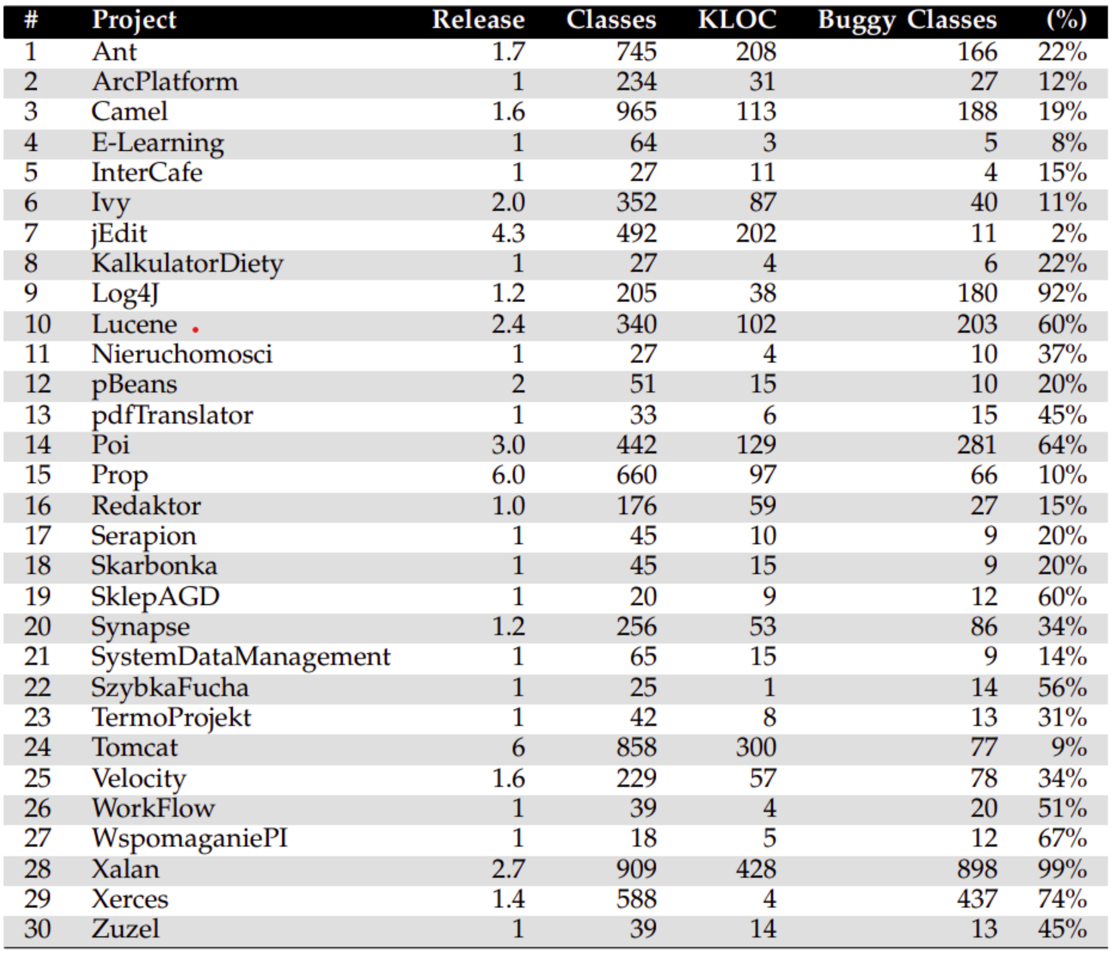
Bench-marking and Comparative Analysis: Researchers can use this dataset for bench-marking bug prediction models, evaluating algorithm performance, and conducting comparative analyses across multiple projects. This facilitates robust model evaluation and comparison against existing studies.

Model Training and Evaluation:

The structured nature of the dataset makes it suitable for training and evaluating bug prediction models, machine learning algorithms, and software quality assessment techniques. Researchers can develop and test predictive models using the dataset's comprehensive data attributes.

Community Collaboration:

The dataset fosters collaboration and knowledge sharing within the software engineering community. Researchers can contribute insights, share methodologies, and collaborate on research projects leveraging this dataset.



#	Project	Release	Classes	KLOC	Buggy Classes	(%)
1	Ant	1.7	745	208	166	22%
2	ArcPlatform	1	234	31	27	12%
3	Camel	1.6	965	113	188	19%
4	E-Learning	1	64	3	5	8%
5	InterCafe	1	27	11	4	15%
6	Ivy	2.0	352	87	40	11%
7	jEdit	4.3	492	202	11	2%
8	KalkulatorDiety	1	27	4	6	22%
9	Log4J	1.2	205	38	180	92%
10	Lucene	2.4	340	102	203	60%
11	Nieruchomosci	1	27	4	10	37%
12	pBeans	2	51	15	10	20%
13	pdfTranslator	1	33	6	15	45%
14	Poi	3.0	442	129	281	64%
15	Prop	6.0	660	97	66	10%
16	Redaktor	1.0	176	59	27	15%
17	Serapion	1	45	10	9	20%
18	Skarbonka	1	45	15	9	20%
19	SklepAGD	1	20	9	12	60%
20	Synapse	1.2	256	53	86	34%
21	SystemDataManagement	1	65	15	9	14%
22	SzybkaFucha	1	25	1	14	56%
23	TermoProjekt	1	42	8	13	31%
24	Tomcat	6	858	300	77	9%
25	Velocity	1.6	229	57	78	34%
26	WorkFlow	1	39	4	20	51%
27	WspomaganiePI	1	18	5	12	67%
28	Xalan	2.7	909	428	898	99%
29	Xerces	1.4	588	4	437	74%
30	Zuzel	1	39	14	13	45%

Figure 3.1: Dataset

3.2 ML Classifiers

3.2.1 Naive Bayes Classifier

The Naive Bayes classifier, rooted in Bayes' theorem, is a probabilistic machine learning algorithm commonly employed for classification tasks. Operating under the assumption of feature independence, it simplifies probability calculations, enabling efficient processing of large datasets. By evaluating the conditional probability of a data point belonging to each possible class, the classifier predicts the class label based on the highest probability.

Despite its simplistic nature and the unrealistic feature independence assumption, Naive Bayes demonstrates notable performance, particularly in text classification endeavors like sentiment analysis, document categorization, and spam filtering. Its efficacy stems from its adeptness in handling high-dimensional data efficiently and its resilience to data noise. Consequently, Naive Bayes classifiers find widespread application in diverse fields such as natural language processing, bioinformatics, and recommendation systems. Renowned for their computational efficiency, implementation simplicity, and scalability to large datasets, they remain a favored choice for practical applications.

3.2.2 RBF Classifier

The RBF (Radial Basis Function) classifier is adept at classification tasks, employing mathematical functions known as radial basis functions, which rely solely on distance from a central point to model and classify data effectively. By transforming input data into a high-dimensional space, the RBF classifier facilitates the application of linear classifiers, offering flexibility in handling complex datasets.

One of its standout features is its capacity to capture intricate nonlinear relationships within data, making it particularly suited for tasks where data lacks linear separability. This is made possible by radial basis functions, enabling the creation of decision boundaries beyond linear shapes, enhancing the classifier's versatility. Widely applied in domains like pattern recognition, image classification, and anomaly detection, the

RBF classifier demonstrates adaptability across various data types, exhibiting robustness against noise and outliers. Nonetheless, its sensitivity to hyperparameter choices, such as the number and placement of radial basis functions, underscores the importance of careful tuning to ensure optimal performance.

3.2.3 Logistic Regression

The Logistic Regression (LOG) classifier is a prevalent statistical model utilized primarily for binary classification tasks, despite its linear nature. Operating on the principle of predicting the probability of a binary outcome based on one or more predictor variables, it employs the logistic function, commonly known as the sigmoid function, to model the probability of the dependent variable belonging to a specific class. This approach calculates the log odds, representing the logarithm of the odds ratio of the event occurring, and transforms them using the logistic (sigmoid) function to produce probabilities ranging from 0 to 1, facilitating binary predictions.

Logistic regression stands out for its interpretability, with the model's coefficients offering valuable insights into the relationship between predictors and outcomes. Moreover, its versatility extends to handling both numerical and categorical input variables, making it applicable across a wide spectrum of tasks. With its ability to provide interpretable results and accommodate various types of input variables, logistic regression remains a popular choice for binary classification tasks in diverse fields.

3.2.4 Voting

A Voting Classifier is an ensemble learning method that integrates several individual classifiers to generate predictions. It consolidates the predictions made by each base classifier and selects the class label with the highest number of votes (mode) as the final prediction. Voting classifiers can adopt various strategies, including hard voting, where the class with the majority of votes is chosen, or soft voting, which considers weighted votes based on class probabilities. This approach tends to enhance prediction accuracy, particularly when the base classifiers exhibit diverse strengths and weaknesses or when confronted with noisy data. Decision trees, support vector machines, and logistic regression are among the commonly employed base classifiers for voting.

3.2.5 Decision Tree

The Decision Tree classifier is a model structured like a tree, dividing the input space recursively based on feature values to make predictions. Each node in the tree corresponds to a decision made using a feature's value, resulting in branches representing potential outcomes. This process repeats until reaching a leaf node, which signifies the final decision or class label. Decision trees are known for their intuitiveness, interpretability, and ability to process both numerical and categorical data. However, they are prone to overfitting, particularly with intricate datasets. To address this issue, techniques such as pruning and ensemble methods like Random Forests can enhance their performance and generalization ability.

3.2.6 Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) classifier is an artificial neural network designed for supervised learning tasks, featuring multiple interconnected layers of neurons, including input, hidden, and output layers. Neurons in each layer are connected to adjacent layers with weighted connections, facilitating information flow through the network.

MLPs utilize activation functions like sigmoid or ReLU to introduce nonlinearity, enabling them to learn intricate patterns and relationships in data. Through techniques such as backpropagation and gradient descent, the network adjusts connection weights during training based on input data and desired output.

A notable strength of MLPs lies in their capacity to model nonlinear relationships, rendering them suitable for tasks where linear classifiers may falter. They find widespread application across diverse domains, including image recognition, natural language processing, and financial prediction.

3.3 Performance Metrics

3.3.1 CK Metrics

CK metrics, also referred to as Chidamber and Kemerer metrics, constitute a suite of software complexity measures utilized to evaluate the quality of object-oriented design and code. Originating from the seminal work by Shyam R. Chidamber and Chris F. Kemerer in 1994, these metrics aim to quantitatively assess various facets of object-oriented software complexity, providing valuable insights into design quality, maintainability, and potential developmental challenges.

- **Weighted Methods per Class (WMC)** : WMC measures the complexity of a class by counting the number of methods it contains. It considers both the number of methods and their complexity, where more complex methods contribute more to the overall WMC value.
- **Depth of Inheritance Tree (DIT)** : DIT indicates the level of inheritance in a class hierarchy. It measures the length of the longest path from a class to the root of the inheritance tree. A high DIT value may indicate increased complexity and potential coupling issues.
- **Number of Children (NOC)** : NOC quantifies the number of immediate subclasses (children) of a class. A high NOC value may suggest that a class has many responsibilities or is highly specialized, potentially impacting maintainability and reusability.
- **Coupling Between Objects (CBO)** : CBO measures the level of coupling between classes in a system. It counts the number of classes that are directly coupled to a given class, either through method calls, attribute references, or parameter passing. High CBO values may indicate tight coupling and reduced modularity.
- **Response For a Class (RFC)** : RFC measures the number of methods that can be executed in response to a message sent to an object of the class. It includes both local methods and inherited methods. A high RFC value may indicate increased

complexity and potential maintenance challenges.

- **Lack of Cohesion in Methods (LCOM)** : LCOM assesses the cohesion within a class by examining the relationships between its methods and attributes. It quantifies the number of disjoint sets of methods that access different subsets of the class's attributes. Low LCOM values indicate better cohesion and encapsulation.

3.3.2 Process and History based Metrics

Process metrics in software engineering evaluate the efficiency and effectiveness of development processes, encompassing factors like productivity, resource utilization, and adherence to standards. On the other hand, history-based metrics analyze past data, including bug occurrences and code changes, to predict future outcomes and identify patterns that may indicate areas prone to issues like bugs or inefficiencies. Both types of metrics play crucial roles in optimizing software development practices, guiding decision-making, and enhancing overall code quality and reliability.

- **CA (Afferent Couplings)** : Afferent Couplings represent the number of classes outside the package that depend on classes within the package. High CA values indicate that many classes from other packages rely on classes within the package, potentially indicating higher package complexity or coupling.

- **CE (Efferent Couplings)** : Efferent Couplings represent the number of classes inside the package that depend on classes outside the package. High CE values suggest that many classes within the package depend on classes from other packages, which may indicate higher coupling or inter-package dependencies.

- **NPM (Number of Public Methods)** : NPM counts the total number of public methods in a class. It provides insights into the size and complexity of classes in terms of their externally accessible methods.

- **LCOM3 (Lack of Cohesion of Methods)** : LCOM3 measures the lack of cohesion among methods within a class. It quantifies the number of disjoint sets of methods that do not share instance variables, potentially indicating lower cohesion and increased complexity within the class.

- **LOC (Lines of Code)** : LOC counts the total number of lines of code in a class or method. It provides a basic measure of code size and complexity, although it does not account for differences in code structure or functionality.
- **DAM (Data Access Metric)** : DAM quantifies the number of unique attributes accessed by methods within a class. High DAM values suggest a higher level of data access complexity within the class.
- **MOA (Measure of Aggregation)** : MOA measures the number of classes directly aggregated by a class. It provides insights into the level of aggregation complexity within a class.
- **MFA (Method Fan-In)** : MFA calculates the number of methods from other classes that invoke methods within the class. High MFA values indicate that the class is frequently used by other classes, potentially indicating higher complexity or critical functionality.
- **CAM (Cohesion Among Methods)** : CAM measures the cohesion among methods within a class. It quantifies the number of method pairs that access the same instance variables, reflecting higher cohesion and better encapsulation within the class.
- **IC (Inheritance Coupling)** : IC measures the number of parent classes that a class inherits from. High IC values suggest a higher level of inheritance complexity within the class hierarchy.
- **CBM (Coupling Between Methods)** : CBM quantifies the number of method pairs in a class that share parameters or instance variables. It provides insights into the level of method-level coupling within the class.
- **AMC (Average Method Complexity)** : AMC calculates the average complexity of methods within a class, often based on metrics such as cyclomatic complexity or other complexity measures.

- **MAX CC (Maximum Cyclomatic Complexity)** : MAX CC represents the maximum cyclomatic complexity among methods within a class. Cyclomatic complexity measures the number of linearly independent paths through a method, reflecting its structural complexity.
- **AVG CC (Average Cyclomatic Complexity)** : AVG CC calculates the average cyclomatic complexity of methods within a class. It provides an overall measure of method complexity within the class.

Chapter 4

PROPOSED METHODOLOGY

In this section, we introduce Dynamic Classifier (DC), an innovative approach aimed at dynamically selecting classifiers tailored for bug prediction tasks. The detailed implementation guidelines and methodologies of DC are comprehensively elucidated in our supplementary materials [33].

DC orchestrates a multifaceted process, meticulously designed to optimize the intricate task of identifying the most suitable classifier for evaluating the bugginess of diverse software classes. Let us delve into the intricacies of each step, elaborating on its significance and methodology:

4.1 Classifier Configuration and Prediction:

At the outset, DC curates a diverse ensemble of classifiers, represented as C = c_1, . . .,c_n and constitutes a robust training set T = e_1 , . . . , e_m comprising an array of software classes. Each constituent classifier, denoted as c_i, undergoes an exhaustive experimentation phase against the training set T to discern and optimize its configuration parameters. For instance, in the case of employing Logistic Regression [3], this process entails meticulous parameter fine-tuning to optimize the logistic function. Concurrently, each classifier c_i furnishes predictions pertaining to the bug-proneness of individual classes $e_j \epsilon$ T. It's noteworthy that while DC remains impartial to the specific classifiers enlisted, the inherent synergy among the base classifiers is pivotal, advocating the desirability of a certain level of complementarity. The time efficiency of this phase hinges upon several factors, including the diversity and quantity of classifiers under consideration, as well as the scale of the training set. Nevertheless, empirical assessments conducted on systems akin to those delineated in our model evaluation indicate that this phase typically consumes only a nominal amount of computational resources, often just a few seconds.

4.2 Annotation of Training Set:

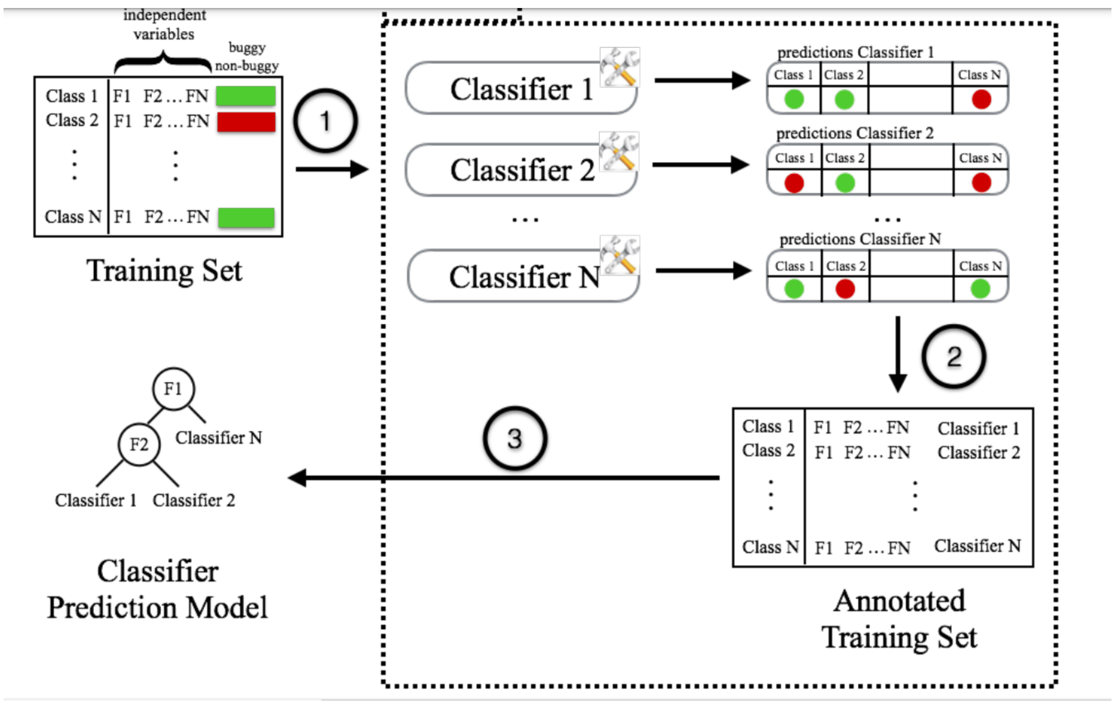


Figure 4.1: Classifier Prediction

Following the exhaustive classifier configuration phase, each constituent class ej ϵ T is meticulously annotated with pertinent information concerning the classifier ci ϵ C that exhibits optimal performance in accurately discerning its bugginess. In this annotation phase, two distinct scenarios are navigated: firstly, if a single machine learning classifier ci accurately predicts the probability of ej being buggy, then ej is unequivocally associated with ci. Conversely, if there are clashes then the class ej is pragmatically assigned the classifier ci that has the highest F-Measure across the entirety of the training set. Consequently, this phase results in a meticulously annotated training set denoted as T. It's worth mentioning that while alternative methodologies for constructing annotated training sets exist (e.g., leveraging multi-label classifiers), empirical analyses strongly support the efficacy of our proposed approach.

4.3 Construction of Classifier Prediction Model:

In the final phase of DC, an advanced classifier prediction model is developed using a decision tree (DT) approach. This phase aims to create a predictive model capable of identifying the most suitable classifier for evaluating a given class based on its structural attributes. The decision tree method is chosen for its ability to capture structural dynamics, recognizing that changes in a class's attributes may require different classifier evaluations. Random Forest, a widely respected classifier that combines multiple tree predictors, is proposed for implementing the decision tree.

Once the adaptive model is constructed, predicting the bugginess of new classes involves using the classifier identified by the decision tree as the most suitable option. This eliminates the need to utilize all base classifiers, as required by alternative ensemble techniques such as Validation and Voting (VV), Boosting, and Stacking.

In essence, Dynamic Classifier (DC) epitomizes a meticulously devised approach to dynamically selecting classifiers tailored explicitly for bug prediction tasks. This intricate process, characterized by the systematic phases of classifier configuration, training set annotation, and construction of a predictive model, is engineered to optimize performance metrics such as accuracy and efficiency. Through rigorous empirical evaluations and meticulous comparative analyses, DC stands poised to redefine the landscape of bug prediction methodologies, ushering in a new era of efficacy and precision.

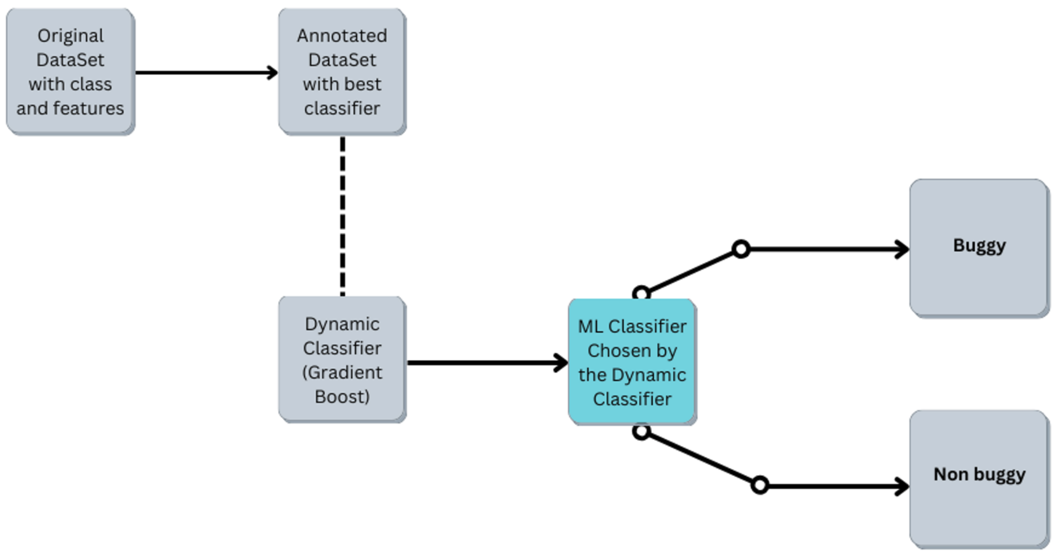

Original DataSet with class and features
Annotated DataSet with best classifier
Dynamic Classifier (Gradient Boost)
ML Classifier Chosen by the Dynamic Classifier
Buggy
Non buggy

Figure 4.2: Classifier Prediction

4.4 Study of complementarity

The empirical investigation aims to explore the synergy among various classifiers in predicting bugs within software classes, focusing on refining bug prediction methods within the same project. A dataset comprising 30 software systems from the Apache Software Foundation ecosystem is selected, each representing diverse projects with unique characteristics. Five machine learning classifiers - Binary Logistic Regression, Naive Bayes, Radial Basis Function Network, Multi-Layer Perceptron, and Decision Trees - are chosen based on their extensive usage in prior research and diverse learning characteristics.

The evaluation employs the 10-fold cross-validation approach, dividing the dataset into ten subsets for testing and training. The process iterates ten times to utilize all available observations, aligning with established bug prediction research practices. Metrics such as accuracy, precision, recall, F-measure, and Area Under the Curve are used to assess classifier performance. The study presents boxplots illustrating the distributions of these metrics across the 30 systems, with detailed results available in the online appendix. Rigorous statistical techniques like the Mann-Whitney U test and Cliff's Delta are applied to determine the significance and magnitude of observed differences, enhancing the interpretability of findings.

Finally, the study explores complementarity among classifiers by computing overlap metrics, aiming to elucidate the extent of agreement or disparity between sets of predictions derived from different classifiers, thereby shedding valuable light on potential synergies in bug prediction.

4.5 Result of complementarity study:

The analysis provides a thorough overview of the results obtained from applying standalone classifiers across diverse software systems. Upon examining the boxplots, it becomes clear that the findings from previous research, as referenced in [5] and [14], remain consistent. These results underscore the absence of a clear frontrunner among various classifiers in bug prediction. This lack of distinctiveness is further emphasized by the minor differences observed in accuracy, F-Measure, and AUC, as illustrated by the median values depicted in Figure 2. Despite these subtle variations, it is evident that the Multilayer Perceptron (MLP) classifier demonstrates a slightly superior average F-Measure of 0.56 compared to other classifiers such as Naive Bayes (NB), Logistic Regression (LOG), Radial Basis Function (RBF), and Decision Tree (DTree). Specifically, MLP outperforms NB by 3%, LOG by 4%, RBF by 2%, and DTree by 4%. However, it is important to acknowledge that while these differences are statistically significant, they are accompanied by negligible effect sizes.

The discussion explores the fascinating findings obtained from analyzing the Apache Xalan project. It is noted that all classifiers exhibit impressive precision and recall metrics. This trend can be explained by the significant proportion of buggy classes, comprising approximately 99% of the project's components. As a result, classifiers primarily trained on data related to buggy components may struggle to accurately differentiate non-affected components. Nonetheless, despite the limited number of non-buggy classes (only 11 in total), their influence on classifier performance is marginal, as the classifiers proficiently predict the majority of buggy classes..

The study offers insights into the performance of the Logistic Regression (LOG) classifier, particularly in within-project bug prediction scenarios. While previous literature suggests LOG's effectiveness in cross-project prediction, this study reveals a different perspective. It highlights the challenge posed by the inherent need for ample training data for LOG to produce reliable outcomes. Unlike cross-project approaches that leverage data from various projects, within-project strategies limit training to previous versions of the same system under examination, potentially impacting LOG's effectiveness.

Additionally, the study conducts a comprehensive analysis of classifier complementarity, summarizing results to highlight overlaps and unique predictions among pairs of classifiers. The findings indicate a substantial overlap in correctly classified instances across classifiers, suggesting accurate bug-proneness predictions irrespective of the classifier used. However, a minority of non-overlapping predictions, primarily concerning buggy classes, may lead to a notable decrease in model performance. These discrepancies underscore the importance of understanding classifier behavior in bug prediction tasks and the potential impact on overall model performance.

Moreover, the analysis highlights the potential for improving bug prediction performance by examining how classifiers perform in relation to classes with different structural characteristics. For example, in the Apache Velocity project, the Naive Bayes (NB) classifier demonstrates greater effectiveness in predicting buggy classes exceeding 500 lines of code (LOC), despite its relatively low F-Measure. In contrast, the Multilayer Perceptron (MLP) classifier achieves the highest F-Measure but performs well in predicting buggy classes with high coupling levels. This emphasizes the importance of adopting a nuanced approach to classifier selection based on class characteristics to enhance bug prediction efficacy.

Summary: This study reveals that although there are variations in performance, the five machine learning classifiers investigated produce similar results concerning accuracy. This implies that, at a fundamental level, they demonstrate comparable capabilities in predicting bugs within software systems. However, the analysis also points to an intriguing prospect: the potential synergies among these classifiers. Despite their individual performances being on par, their unique strengths and weaknesses suggest that combining them could lead to more robust bug prediction models. By leveraging the complementarity of different classifiers, there's an opportunity to create ensemble models or hybrid approaches that capitalize on the diverse perspectives offered by each classifier. This exploration of combined methodologies holds promise for achieving even better results in bug prediction tasks, potentially surpassing the performance of any single classifier in isolation.

Chapter 5

RESULTS

For addressing the problem of choosing the best strategy to classify classes as buggy or not based on their particular characteristics like CK metrics and so on, we tried three categories of approaches to tackle this problem : Standalone Classifiers, Ensemble techniques that combine multiple such single classifiers and Dynamic Classifier, which based on the characteristics of the class and based on the learning on the training dataset, recommends appropriate classifier to perform bug prediction, so as to provide maximum possible accuracy.

5.1 Standalone Classifiers

We have extensively compared standalone classifier models, such as Logistic Regression (LOG), Multi-Layer Perceptron (MLP), Naive-Bayes (NB), Radial Basis Function (RBF), and others, across multiple software projects within the PROMISE Dataset provided by NASA. Our analysis centers on evaluating the performance of these classifiers in bug prediction tasks. Essential metrics like Accuracy, Precision, Recall, Area Under Curve, and F-Measure were employed to evaluate classifier effectiveness. The findings are visually presented to aid in a clear interpretation and comparison of classifier performance across different metrics and projects.

Previous research has indicated that there is no clear leader among different classifiers in bug prediction. Discrepancies in accuracy, F-Measure, and AUC among these classifiers are relatively minor, as depicted by the median values. However, the Multi-Layer Perceptron (MLP) exhibits a slightly higher average F-Measure (0.56) compared to other classifiers (Naive Bayes (NB)=+3%, Logistic Regression (LOG)=+4%, Radial Basis Function Network (RBF)=+2%, Decision Tree (DTree)=+4%). While statisti-

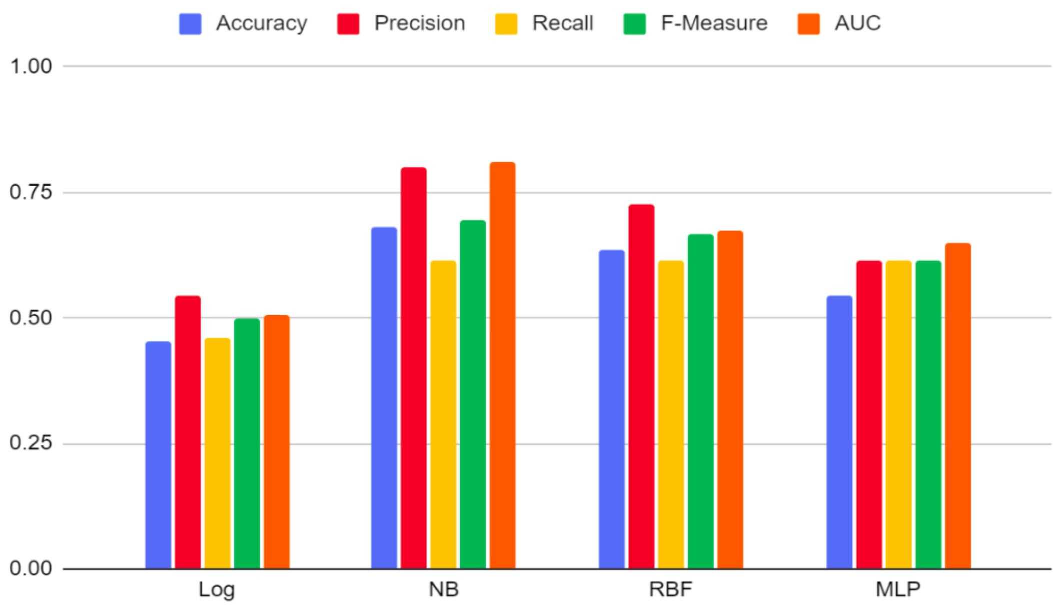


Figure 5.1: Szybkafucha.csv

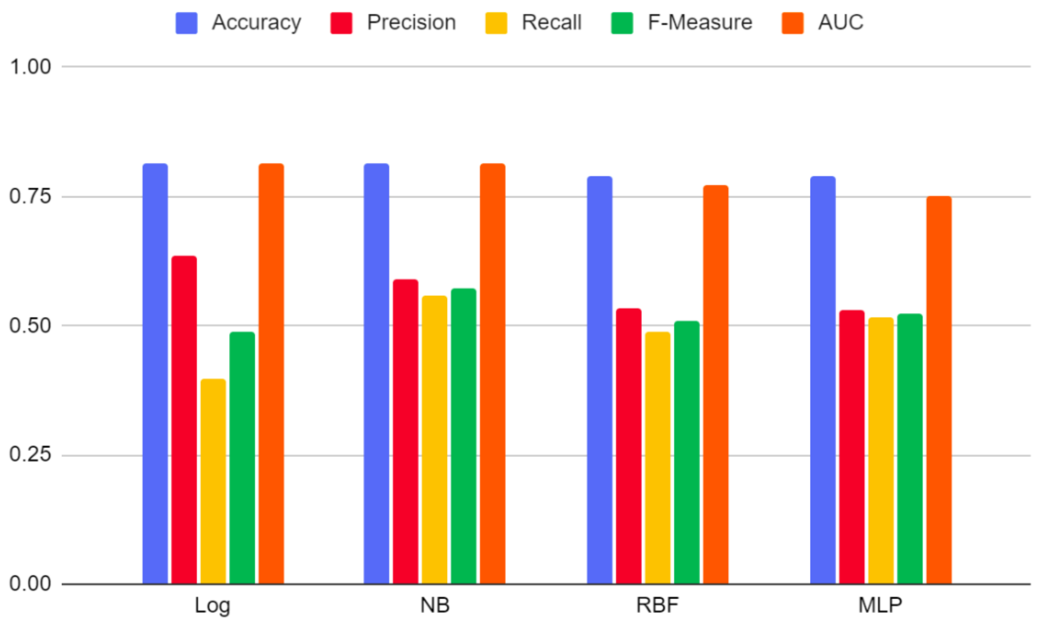


Figure 5.2: Ant-1.7.csv

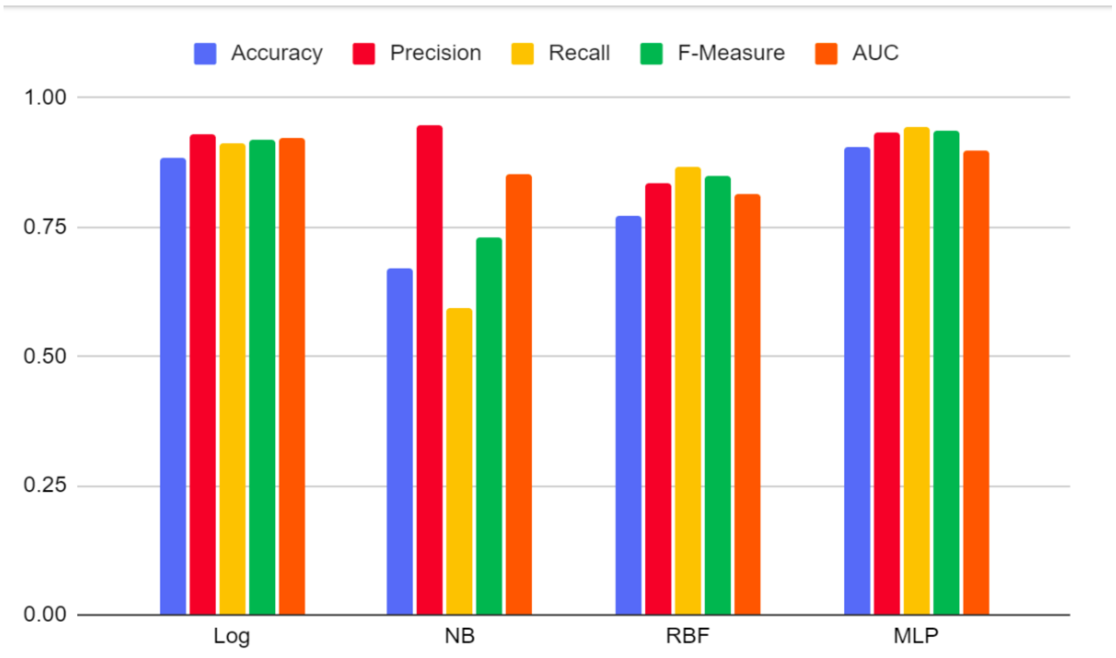


Figure 5.3: Xerces-1.4.csv

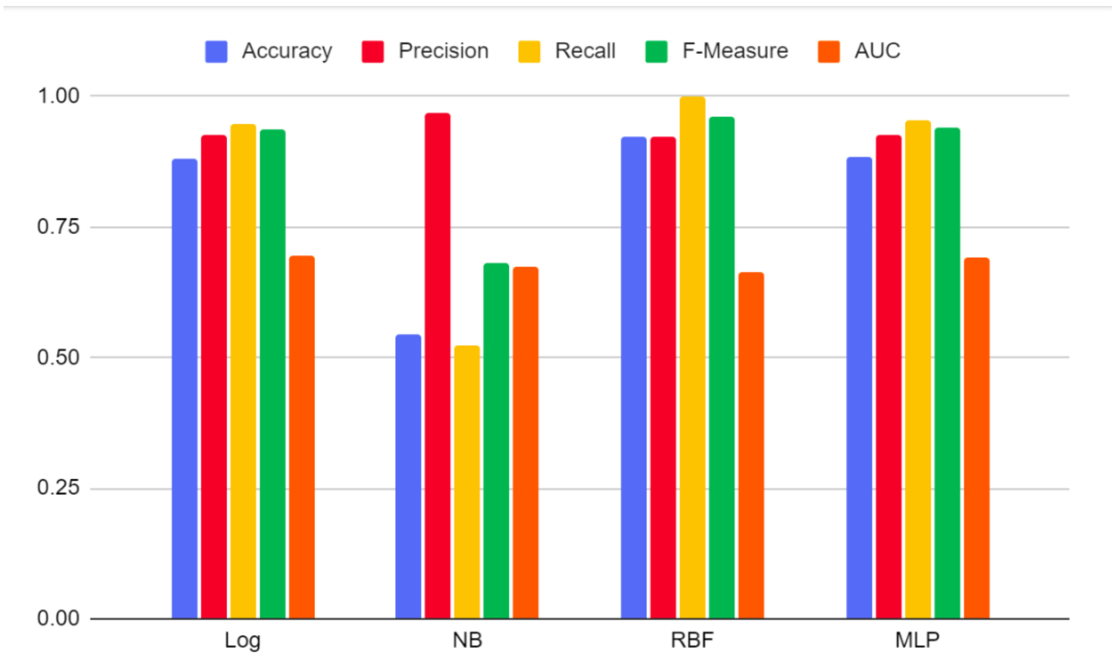


Figure 5.4: Log4j-1.2.csv

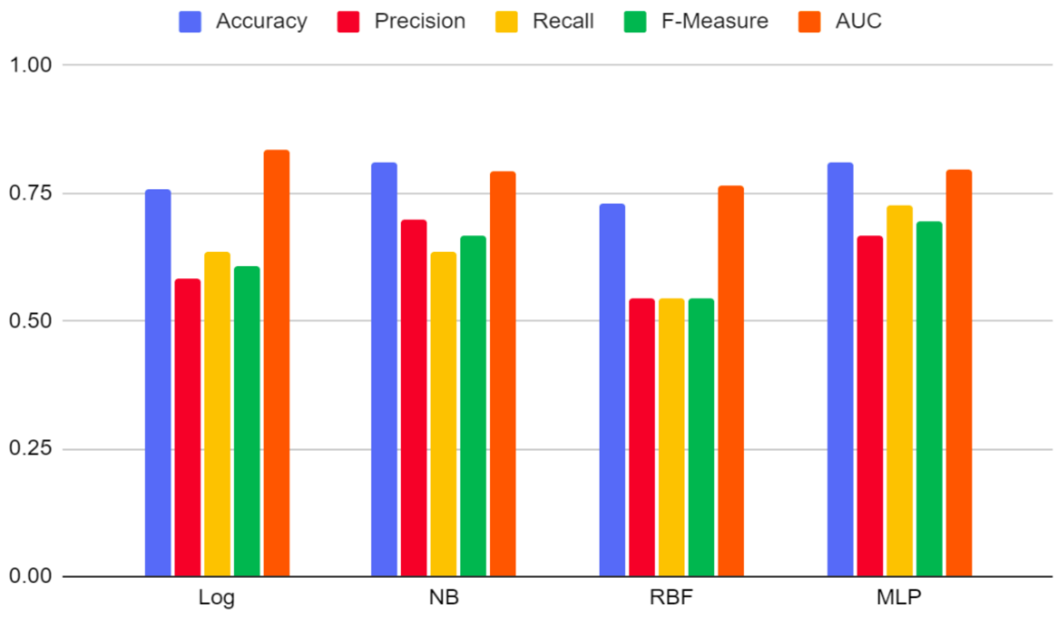


Figure 5.5: Termoproject.csv

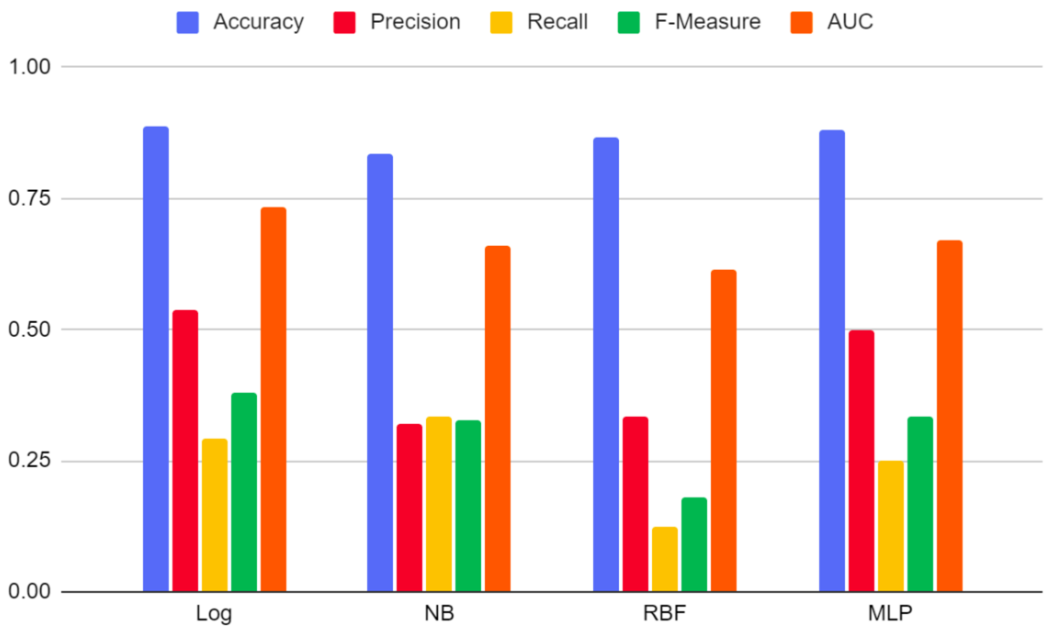


Figure 5.6: Arc.csv

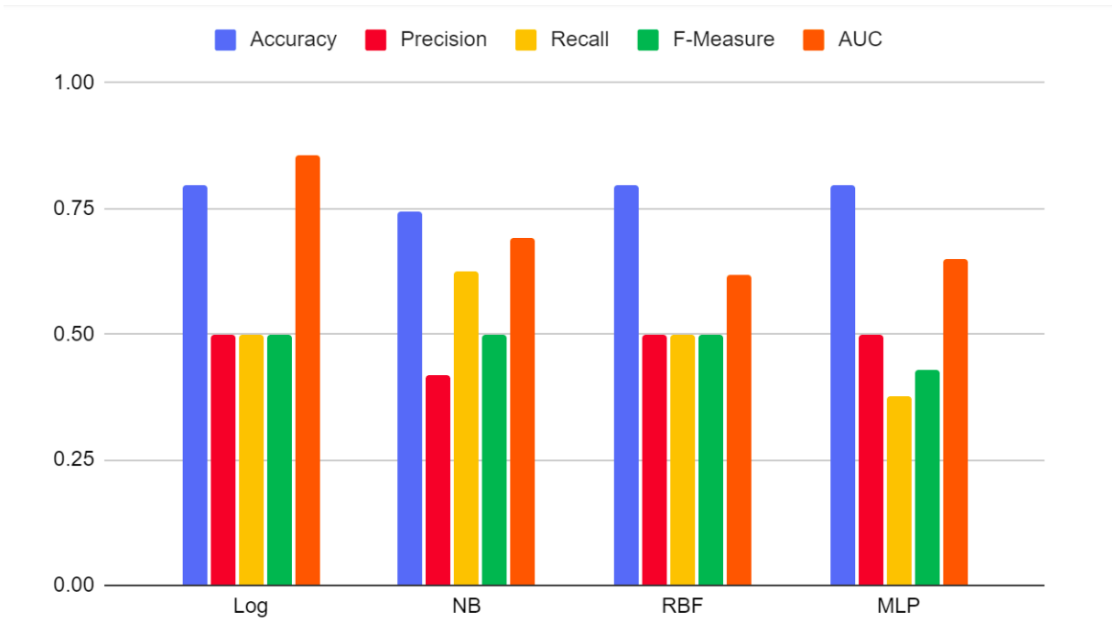


Figure 5.7: Serapion.csv

cally significant differences are observed between MLP and LOG as well as RBF, the effect size remains negligible.

While there are differences among the tested machine learning classifiers, they generally produce similar accuracy results. Nevertheless, their complementary nature suggests that combining these models could potentially lead to improved performance.

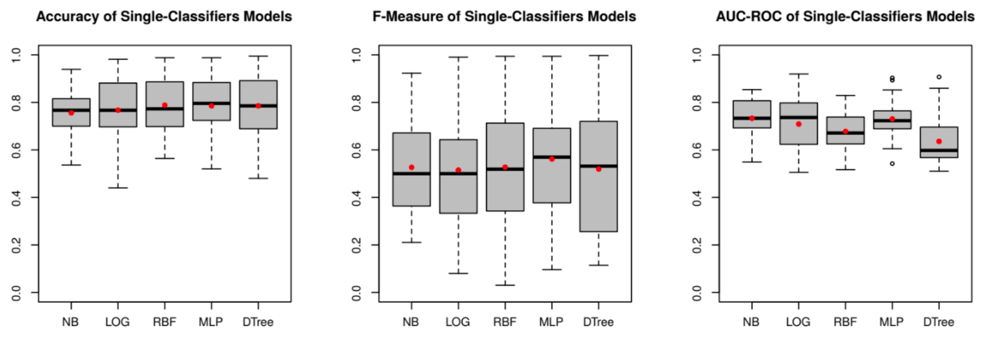


Figure 5.8: Single Classifier Analysis

5.2 Ensemble Techniques

Ensemble techniques in classification aim to bolster predictive accuracy by amalgamating multiple classifiers. Among these, the Validation and Voting (VV) ensemble method, akin to Boosting, has garnered attention for within-project bug prediction. VV predicts an entity as bug-prone if a majority of models, derived from different classifiers on the same training set, classify it as such; otherwise, it's deemed non-bug-prone. Studies by Misirli et al., Wang et al., and Liu et al. have consistently underscored VV's efficacy in bug prediction across projects. Other techniques explored include Bagging, HYDRA, and Stacking, each with its unique approach to combining classifier outputs for enhanced predictive performance.

Despite its merits, the VV technique exhibits some limitations. On average, VV shows a minor decrease in F-Measure, a modest increase in accuracy, and a noticeable decline in AUC-ROC compared to standalone classifiers. Particularly, its ability to distinguish between bug-affected and bug-free classes, as indicated by AUC-ROC results, is limited. Moreover, VV occasionally falls short of standalone models' performance, attributed to its reliance on a simple voting mechanism that doesn't consider individual classifier success rates. This poses challenges, especially when only a few classifiers make accurate predictions, undermining the reliability of VV's results. However, ongoing research continues to explore and refine ensemble techniques to address these challenges and further improve predictive capabilities in software defect prediction.

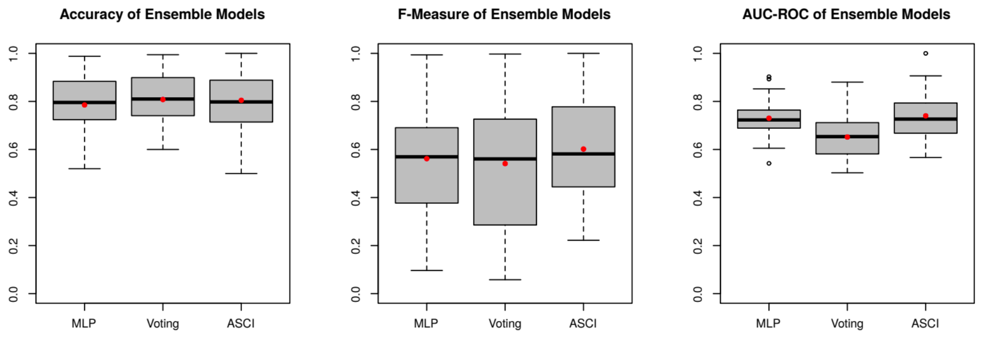


Figure 5.9: Analysis of Ensemble Methods

	MLP			VV			ASCI		
	$p-value$	d	$magn.$	$p-value$	d	$magn.$	$p-value$	d	$magn.$
MLP	-	-	-	0.91	-0.02	neg.	1.00	-0.17	small
VV	0.09	0.02	neg.	-	-	-	1.00	-0.12	neg.
ASCI	**<0.01**	**0.17**	**small**	**<0.01**	**0.12**	**neg.**	-	-	-

Figure 5.10: Single Classifier vs Ensemble vs ASCI

5.3 Dynamic Classifier

The adaptive approach proposed in this study exhibits variable F-Measure scores ranging from 18% to 100%, along with classification accuracies spanning from 44% to 100%. Additionally, the average AUC-ROC stands at 73%. These results suggest that the proposed method demonstrates notable prediction accuracy and remains resilient against inherent uncertainty in classifier prediction models. In comparison to the MLP model, the Dynamic Classifier outperforms it in 77% of cases (23 out of 30 systems), demonstrating F-Measure, accuracy, and AUC-ROC values 7%, 2%, and 1% higher than the baseline model, respectively. Statistical analysis (α ¡ 0.01) confirms the significant superiority of our approach, albeit with a small effect size (d = 0.17). The overall strong performance of MLP among tested classifiers reinforces the assertion that the Dynamic Classifier surpasses NB, LOG, RBF, and DTree on our dataset.

Our method exceeds the Validation and Voting ensemble technique in over 80% of cases, suggesting that selecting classifiers based on class characteristics may be more effective than aggregating results from various classifiers.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In conclusion, our investigation into bug prediction strategies using various classifier categories has yielded nuanced insights into their effectiveness and limitations. The thorough comparison of standalone classifiers such as Logistic Regression (LOG), Multi-Layer Perceptron (MLP), Naive-Bayes (NB), and Radial Basis Function (RBF) across multiple software projects within the PROMISE Dataset highlighted the marginally superior performance of MLP in terms of F-Measure, although differences among classifiers were generally minor. Ensemble techniques, particularly Validation and Voting (VV), showed promise in enhancing classification performance by combining classifier outputs, yet our analysis revealed limitations in accurately distinguishing bug-prone classes, especially when only a few classifiers make correct predictions. Our proposed Dynamic Classifier, which recommends an appropriate classifier based on class characteristics and learning from the training dataset, demonstrated commendable prediction accuracy and outperformed both MLP and VV in a significant majority of cases. This underscores the importance of considering class-specific attributes and dynamic classifier selection in optimizing bug prediction strategies, particularly in scenarios where ensemble techniques may exhibit limited efficacy in distinguishing between bug-prone and bug-free classes. Further research and experimentation with dynamic classifier selection strategies are warranted to explore their potential for enhancing bug prediction accuracy and reliability.

6.2 Future Work

6.2.1 Incorporating Additional Features

In the pursuit of enhancing bug prediction accuracy, a compelling avenue for future exploration involves the inclusion of a broader spectrum of features. While the current study predominantly focuses on code metrics and historical bug data, the incorporation of socio-technical factors could provide a more holistic understanding of bug occurrence dynamics. Features such as developer expertise, code complexity, team dynamics, and organizational factors could significantly contribute to refining bug prediction models. Investigating the impact of these additional features and developing robust techniques to seamlessly integrate them into the adaptive classifier selection framework constitutes an important future research direction.

6.2.2 Investigating Alternative Adaptive Methods

While the proposed adaptive method based on classifier accuracy shows promising results, exploring alternative adaptive strategies can offer valuable insights. For instance, investigating ensemble-based adaptive methods that leverage diversity or stability metrics could provide complementary perspectives. Comparative studies across various adaptive techniques, including ensemble diversity, performance stability over time, and other dynamic selection criteria, would contribute significantly to understanding the most effective approach for dynamic classifier selection in bug prediction scenarios.

6.2.3 Handling Imbalanced Data

The challenge posed by imbalanced datasets, where the number of bug instances is considerably lower than non-bug instances, remains a critical area for improvement in bug prediction models. Future research should focus on developing robust techniques to handle class imbalance within the adaptive classifier selection framework effectively. This may involve exploring advanced sampling methods, cost-sensitive learning algorithms, ensemble-based approaches tailored for imbalanced data, or hybrid techniques combining multiple strategies. Addressing class imbalance issues comprehensively will enhance the reliability and generalizability of bug prediction models across diverse software projects.

6.2.4 Exploring Transfer Learning Techniques

The integration of transfer learning techniques holds substantial promise for advancing bug prediction models, especially in scenarios with limited labeled bug data. Future research could delve into how transfer learning can be seamlessly incorporated into the adaptive classifier selection process. This exploration could include studying domain adaptation methods, pre-trained models, knowledge distillation, and transfer of feature representations to improve bug prediction performance across different projects, domains, or environments. Investigating the synergies between transfer learning and adaptive classifier selection would contribute significantly to model generalization and robustness.

6.2.5 Scalability and Real-time Adaptation

As software projects scale in complexity and size, ensuring the scalability and real-time adaptability of bug prediction systems becomes imperative. Future research should focus on developing scalable adaptive classifier selection mechanisms capable of handling large volumes of data efficiently. Additionally, exploring real-time adaptation strategies that dynamically adjust classifier selection based on evolving project characteristics or streaming data inputs would be crucial. These efforts will enable bug prediction models to remain relevant, accurate, and responsive to changing project dynamics and bug patterns over time.

6.2.6 Evaluation Metrics and Benchmarking

Establishing standardized evaluation metrics and benchmark datasets is essential for rigorously assessing the performance of adaptive bug prediction models. Future work should involve creating comprehensive benchmarking frameworks that encompass diverse projects, development environments, and industry domains. Comparative studies across multiple datasets, along with detailed performance analyses under varying conditions, will facilitate the reproducibility, comparability, and generalizability of adaptive classifier selection techniques. This approach will ensure robust model evaluation and foster advancements in bug prediction research.

6.2.7 User-Centric Design and Integration

The practical usability and seamless integration of adaptive bug prediction systems within software development workflows are paramount for real-world adoption. Collaborating closely with industry practitioners to understand their specific needs, preferences, and challenges related to bug prediction is essential. Future research should prioritize user-centric design principles and conduct usability studies to tailor adaptive classifier selection solutions effectively. This approach will ensure that bug prediction models meet the requirements of software development teams, enhance productivity, and facilitate informed decision-making processes.

6.2.8 Ethical Considerations and Bias Mitigation

Ethical considerations, including fairness, transparency, and bias mitigation, play a pivotal role in developing reliable and trustworthy bug prediction models. Future research efforts should focus on identifying and mitigating potential biases in data collection, feature engineering, and model training processes. Incorporating fairness-aware techniques, conducting comprehensive bias analyses, and promoting transparency in model decision-making will contribute to building equitable and unbiased adaptive classifier selection methods. These efforts are crucial for fostering trust, promoting inclusivity, and ensuring ethical AI practices in bug prediction research and application domains.

Chapter 7

APPENDIX

- **Accuracy**: measures correct predictions as a ratio of total predictions.
- **Precision**: assesses positive prediction accuracy, calculated as true positives divided by total positive predictions.
- **Recall**: measures a model's ability to find relevant instances, calculated as true positives divided by total actual positives.
- **Area under Curve**: assesses binary model performance by measuring the area under the ROC curve, indicating class discrimination.
- **F-Measure**: the F-score or F-measure is a measure of predictive performance.

Chapter 8

REFERENCES