ADVANCING SOFTWARE DEFECT PREDICTION THROUGH ENSEMBLE XAI METHODS: INSIGHTS AND PERFORMANCE EVALUATION

ENTEGRE XAI YÖNTEMLERİ İLE YAZILIM HATA TAHMİNİNİN GELİŞTİRİLMESİ: ANALİZLER VE PERFORMANS DEĞERLENDİRMESİ

BAHAR GEZİCİ GEÇER

DOÇ. DR. AYÇA KOLUKISA TARHAN

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of PhD of Science

in Computer Engineering

May 2024

ABSTRACT

ADVANCING SOFTWARE DEFECT PREDICTION THROUGH ENSEMBLE XAI METHODS: INSIGHTS AND PERFORMANCE EVALUATION

Bahar GEZİCİ GEÇER

PhD of Science, Computer Engineering

Supervisor: Doç. Dr. Ayça KOLUKISA TARHAN

May 2024, 194 pages

This doctoral thesis presents a comprehensive investigation into enhancing the interpretability and transparency of Machine Learning (ML) models in the domain of Software Defect Prediction (SDP) through Model-Agnostic eXplainable Artificial Intelligence (XAI) methods. The primary objective is to elucidate the decision-making processes of ML models, both at individual (local) and global levels, thus bridging the gap between predictive power and comprehensibility demanded by stakeholders in the SDP domain.

The methodological approach adopted involves an iterative and exploratory process, utilizing XAI techniques such as ELI5, SHAP, and LIME, among others. These techniques are systematically applied across multiple case studies, each focusing on specific aspects of model interpretability and transparency in SDP. Through iterative refinement and exploration, the research uncovers insights into the importance of features, contributions to individual predictions, and overall model decisions. Furthermore, ensemble modeling techniques are integrated to amalgamate feature importance scores obtained from diverse

XAI methods, thereby optimizing predictive accuracy while simultaneously preserving interpretability.

This research significantly contributes to the field of SDP by furnishing a thorough understanding of ML model decision-making processes. It enhances model interpretability and transparency, effectively addressing critical gaps in traditional feature selection and outlier detection methodologies. Moreover, it offers valuable insights into ensemble modeling approaches, elucidating their role in optimizing predictive accuracy while maintaining interpretability. Validation of the developed methodologies is conducted through rigorous empirical studies and comparative analyses, thus ensuring their effectiveness and usability in real-world SDP scenarios.

Keywords: explainable AI, XAI, defect prediction, artificial intelligence, machine learning

ÖZET

ENTEGRE XAI YÖNTEMLERİ İLE YAZILIM HATA TAHMİNİNİN GELİŞTİRİLMESİ: ANALİZLER VE PERFORMANS DEĞERLENDİRMESİ

Bahar GEZİCİ GEÇER

Doktora, **Bilgisayar Mühendisliği**

Danışman: Doç. Dr. Ayça KOLUKISA TARHAN

Mayıs 2024, 194 sayfa

Bu doktora tezi, Yazılım Hatası Tahmini (SDP) alanındaki Makine Öğrenimi (ML) modellerinin yorumlanabilirliğini ve şeffaflığını Modelden Bağımsız Açıklanabilir Yapay Zeka (XAI) yöntemleri aracılığıyla geliştirmeye yönelik kapsamlı bir araştırma sunmaktadır. Temel amaç, hem lokal (yerel) hem de küresel düzeyde makine öğrenmesi modellerinin karar verme süreçlerini aydınlatmak ve böylece SDP alanındaki paydaşların talep ettiği tahmin gücü ile anlaşılabilirlik arasındaki boşluğu doldurmaktır.

Benimsenen metodolojik yaklaşım, diğerlerinin yanı sıra ELI5, SHAP ve LIME gibi XAI tekniklerini kullanan yinelemeli ve keşifsel bir süreci içermektedir. Bu teknikler, her biri SDP'de model yorumlanabilirliği ve şeffaflığının belirli yönlerine odaklanan çok sayıda vaka çalışmasında sistematik olarak uygulanmaktadır. Yinelemeli iyileştirme ve keşif yoluyla araştırma, özelliklerin önemi, lokal tahminlere katkıları ve genel model kararlarına ilişkin içgörüleri ortaya çıkarmaktadır. Ayrıca, çeşitli XAI yöntemlerinden elde edilen özellik önem puanlarını birleştirmek için topluluk modelleme teknikleri entegre edilmiş, böylece tahmin doğruluğu optimize edilirken aynı zamanda yorumlanabilirlik de korunmuştur.

Bu araştırma, makine öğrenimi modeli karar verme süreçlerinin kapsamlı bir şekilde anlaşılmasını sağlayarak SDP alanına önemli ölçüde katkıda bulunmaktadır. Geleneksel özellik seçimi ve aykırı değer tespit metodolojilerindeki kritik boşlukları etkili bir şekilde ele alarak modelin yorumlanabilirliğini ve şeffaflığını artırmaktadır. Ayrıca, yorumlanabilirliği korurken tahmin doğruluğunu optimize etmedeki rollerini aydınlatarak, topluluk modelleme yaklaşımları hakkında değerli bilgiler sunmaktadır. Geliştirilen metodolojilerin doğrulanması, titiz ampirik çalışmalar ve karşılaştırmalı analizler yoluyla gerçekleştirilmekte, böylece gerçek dünyadaki SDP senaryolarında etkinlikleri ve kullanılabilirlikleri sağlanmaktadır.

Keywords: açıklanabilir yapay zeka, XAI, hata tahmini, yapay zeka, makine öğrenimi

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to everyone who has contributed to the completion of this doctoral thesis.

First and foremost, I am deeply grateful to my supervisor, Doç. Dr. Ayça Kolukısa Tarhan, for her invaluable guidance, support, and mentorship throughout this journey. Her expertise, encouragement, and unwavering commitment has been instrumental in shaping the direction and quality of this research.

I extend my appreciation to the members of my doctoral committee, Prof. Dr. Pınar Karagöz and Doç. Dr. Ali Seydi Keçeli, for their insightful feedback, constructive criticism, and valuable suggestions, which have significantly enriched the content and rigor of this thesis.

I am indebted to my colleagues, friends, and family for their unwavering support, understanding, and encouragement throughout this challenging yet rewarding journey. Their encouragement, patience, and belief in my abilities have been a constant source of motivation and inspiration.

I am profoundly grateful to my dear mother for her unwavering love, encouragement, and unwavering support throughout my doctoral journey. Mom! your sacrifices, guidance, and endless encouragement have shaped not only my academic endeavors but also my character and resilience. Your unwavering support and belief in my potential have been a guiding light, guiding me through the darkest moments and cheering me on in times of triumph. Your love and unwavering presence have been my anchor, grounding me in times of uncertainty and uplifting me to reach new heights. Thank you for your unconditional love, sacrifice, and unwavering belief in me. This achievement is as much yours as it is mine, and I am forever grateful for your boundless support and encouragement.

This thesis is a culmination of the collective efforts and support of numerous individuals, and I am deeply grateful for each and every contribution. Thank you.

Bahar GEZİCİ GEÇER

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

AI	:	**A**rtificial **I**ntelligence
DSR	:	**D**esign **S**cience **R**esearch
ELI5	:	**E**xplain **L**ike **I'**m 5
GB	:	**G**radient **B**oosting
LIME	:	**L**ocal **I**nterpretable **M**odel-agnostic **E**xplanation
ML	:	**M**achine **L**earning
MLP	:	**M**ulti **L**ayer **P**erceptron
NB	:	**N**aive **B**ayes
NN	:	**N**eural **N**etworks
PDP	:	**P**artial **D**ependence **P**lots
RF	:	**R**andom **F**orest
SDP	:	**S**oftware **D**efect **P**rediction
SHAP	:	**S**hapley **A**dditive **E**xplanations
SLR	:	**S**ystematic **L**iterature **R**eview
XAI	:	**E**xplainable **AI**
XAI4SE	:	**E**xplainable **AI** for **S**oftware **E**ngineering

1. INTRODUCTION

Before a software system is made public or put into use, a method called ”software defect prediction (SDP)” seeks to locate and rank any potential weak points. Enhancing software quality, cutting expenses, and wisely allocating testing resources are the objectives. Defect prediction encompasses a wide array of methodologies and metrics, ranging from traditional statistical models [2, 3] to advanced machine learning algorithms [4]. Studies in software defect prediction (SDP) cover various facets of the field, including data mining methods [5], early defect prediction [6], and decision-making processes [7]. These research efforts, documented across academic literature, provide valuable insights for both academics and industry professionals. In software engineering, predicting defects plays a vital role in increasing the quality of software systems, since finding and fixing those defects are one of the most expensive activities within the software development life cycle [6]. For example, before deploying a software product, it is very critical to predict whether there are remaining defects in it. When a customer observes the defects after the software product is deployed, the customer's trust in that product will decrease.

There is an increasing demand for AI-based software systems in industry and this is mostly because of the significant advances in machine learning (ML) models such as deep learning. Since some ML models are “black-box”, it is very difficult to understand or interpret the decision-making process or the key factors involved in the decision, or to get insights about the behavior of these models [8]. Although the prediction performances of these models are good, understanding the reasons behind the predictions is a very hard task. As a user understands a model or a prediction, trust in them will increase; and, if a user trusts a prediction or a model, s/he will be more motivated to use it. Therefore, getting insights about the model or predictions will make them more transparent, provide information about feature importance and human decision-making, and in turn, will build trust.

Explainability is one of the most studied quality attributes, and stakeholders adopting Artificial Intelligence (AI), particularly Machine Learning (ML) software, are becoming

more and more interested in it. Understanding the reasoning behind the predictions is crucial since AI-based techniques have a black-box character; and, their application in SDP is not an exception. Even, explaining ML models used for SDP is especially important due to the abstract and changeable nature of software product and therefore, the need of traceability in software development process. Interestingly, how AI explainability is addressed by design, how XAI methodologies might be applied for different user groups, and what opportunities can support the development of user-centered and explainable SDP have not yet been studied. Since complex software systems require diverse perspectives for understanding and different techniques capture different aspects of model behavior, a combination of techniques enhances the overall interpretability.

In addition to the gap that exists in literature, we may exemplify several practical scenarios to highlight the necessity of this investigation. For instance, consider a scenario where a XAI method indicates that a specific code module's complexity is a significant predictor of defects. Developers can then prioritize a thorough review and testing of this module. Simultaneously, project managers, armed with the knowledge that the model relies heavily on this factor, can allocate resources more efficiently. End-users, in turn, can better understand the potential areas of concern flagged by the model, fostering a collaborative and informed approach to software defect prevention. In essence, the application of the XAI method not only enhances the interpretability of SDP models but also empowers stakeholders with actionable insights to make informed decisions throughout the software development lifecycle. Another motivating scenario that vividly demonstrates the practical implication of an accurate SDP facilitated by XAI is where a software development team is working on a critical project with tight deadlines. Traditional ML models, while predicting software defects, might lack transparency, leaving developers in the dark about the specific reasons behind the predictions. By employing XAI methods in SDP, developers may not only receive accurate predictions but also gain insights into the influential factors driving those predictions. This transparency becomes particularly crucial when prioritizing bug fixes, allowing developers to focus on the most critical areas of the codebase, ultimately leading to more efficient and timely software development. Furthermore, consider the case of a project

manager responsible for overseeing multiple software development projects simultaneously. In this scenario, the ability to comprehend and trust the predictions of SDP models is paramount. With the use of XAI techniques, project managers can make well-informed decisions about resource allocation, project timelines, and risk management. The newfound transparency in the predictive process enables project managers to navigate complex projects with confidence and agility. These examples illustrate the practical relevance of this research by emphasizing how the incorporation of XAI in SDP directly impacts the efficiency, reliability, and overall success of software developments.

1.1. Research Problem

In recent years, the widespread adoption of ML techniques has revolutionized various fields, including SDP. Defect prediction plays a pivotal role in ensuring the reliability and quality of software systems by identifying potential defects early in the development process. However, as ML models increasingly find applications in SDP, there arises a critical need for understanding their decision-making processes, particularly concerning the interpretability and transparency of these models.

The general problem addressed in this thesis revolves around the need to enhance the interpretability and transparency of machine learning models applied to software defect prediction. Specifically, it focuses on developing methodologies and frameworks that enable stakeholders to understand the underlying logic of these models, interpret their predictions, and trust their outcomes.

This problem encompasses various challenges, including:

- Understanding the importance of features in datasets used in SDP and their impact on model predictions.
- Explaining the rationale behind individual predictions and model decisions to diverse user groups (e.g., Data Scientist, Domain Expert, End User).

- Facilitating the visualization and inspection of ML models to aid decision-making processes.
- Improving the transparency of feature selection and outlier detection techniques in SDP.
- Developing ensemble models that offer both local and global interpretability for enhanced decision support.

By addressing these challenges, this doctoral thesis delves into the realm of Model-Agnostic Explainability methods to illuminate the black box nature of ML models utilized in SDP. The primary objective is to unravel the intricacies of both local (individual) and global predictions, thus shedding light on the contributing factors behind model decisions. Through a series of research questions (RQs) and empirical explorations, this thesis endeavors to bridge the gap between the predictive power of ML models and the comprehensibility demanded by stakeholders in the SDP domain.

1.2. Proposed Solution

To address the overarching problem of enhancing interpretability and performance in machine learning for software defect prediction, this thesis proposes a multifaceted approach encompassing the following key strategies through a series of research questions and empirical explorations, guided by our research methodology adhering to an exploratory strategy based on Design Science Research (DSR) principles [9]. This methodology prioritizes iterative problem-solving cycles to devise inventive solutions for real-world issues.

1. Model-Agnostic Explainability Methods: Employing model-agnostic explainability techniques such as SHAP, LIME, and ELI5 to elucidate the underlying rationale of ML models used in SDP. These methods facilitate the identification of important features, understanding their contributions to individual predictions, and comprehending the overall decision-making process of the models.

2. XAI-Enabled SDP Framework: Introducing an XAI-enabled framework tailored specifically for SDP, integrating various ML explanations and evaluation measures. This framework aims to provide a systematic approach for interpreting different types of ML models, catering to diverse user types within the software development and quality assurance domain.

3. Transparency-enhancing Techniques: Developing methodologies to enhance transparency in feature selection and outlier detection processes employed in SDP. By leveraging XAI methods, the strengths and limitations of these techniques can be elucidated, fostering a deeper understanding of their impact on model interpretability and overall performance.

4. Ensemble Modeling for Interpretability: Proposing ensemble modeling techniques that prioritize both local and global interpretability in SDP. By aggregating insights from multiple interpretable models, ensemble approaches aim to enhance the comprehensibility of predictions while maintaining competitive accuracy levels.

5. Comparative Evaluation: Conducting a comprehensive comparative evaluation of proposed solutions against existing literature and benchmark datasets. This evaluation will provide insights into the effectiveness, reliability, and practical applicability of the proposed methodologies in real-world SDP scenarios.

Through the integration of these strategies, this thesis endeavors to advance the state-of-the-art in interpretable and transparent ML approaches for SDP, ultimately empowering stakeholders with actionable insights, fostering trust in ML-based decision-making processes, and facilitating the development of reliable software systems.

1.3. Scope of the Thesis

This thesis focuses on enhancing interpretability and transparency in ML models for SDP through a multi-case exploratory study. The scope of this thesis encompasses the following key aspects:

1. Interpretability Enhancement

    - Investigating methodologies and techniques to improve the interpretability of ML models used in SDP.
    - Exploring model-agnostic explainability methods, such as SHAP, LIME, and ELI5, to understand feature importance, contributions to individual predictions, and overall model decisions.

2. Transparency Enhancement

    - Addressing challenges related to transparency in feature selection, outlier detection, and model decision-making processes in SDP.
    - Developing and evaluating XAI-enabled SDP frameworks to enhance algorithm transparency, model visualization, and user trust in SDP systems.

3. Multi-Case Exploratory Study

    - Conducting a series of case studies on diverse SDP datasets (e.g., KC2, PC1, CM1) to explore different facets of interpretability and transparency enhancement.
    - Investigating local and global prediction analysis, model interpretability, feature selection, and ensemble modeling techniques across multiple case studies.

4. Comparative Analysis and Validation

    - Performing comparative analyses to evaluate the performance, usability, and effectiveness of the proposed methodologies against existing approaches and benchmarks.
    - Validating the developed solutions through empirical studies on different SDP datasets.

By addressing these key aspects within the defined scope, this thesis aims to contribute to the advancement of interpretable and transparent ML approaches for SDP, ultimately

enhancing decision-making processes and fostering trust in ML-based software quality assurance practices.

1.4. Contributions

This doctoral thesis contributes to the field of SDP by:

- Providing a comprehensive understanding of ML model decision-making processes through XAI methods.
- Enhancing model interpretability and transparency, thereby facilitating the deployment of ML models in SDP applications by providing a XAI-enabled framework that explores five different ML methods (RF, GB, NB, MLP, and NN) with well-known XAI methods (SHAP, LIME, ELI5, Anchor, Protodash) to give several user types (i.e. Data Scientist, Domain Expert, End User) the capacity to methodically explain local and global ML-based SDP results.
- Addressing critical gaps in traditional feature selection and outlier detection methods by leveraging XAI techniques.
- Offering insights into ensemble modeling approaches for optimizing predictive accuracy while maintaining interpretability.

Through these contributions, this research aims to empower SDP stakeholders with the knowledge and tools necessary to navigate the complexities of ML models effectively, ultimately improving the reliability and quality of software systems.

1.5. Organization

The organization of the thesis is as follows:

- Chapter 1 presents our motivation, contributions and the scope of the thesis.

- Chapter 2 provides bacjground information about SDP, ML models and XAI methods used in this study.
- Chapter3 provides related academic studies in the context of explainability.
- Chapter 4 gives a detail about the XAI-Enabled SDP framework adopted in this thesis.
- Chapter 5 introduces research methodology.
- Chapter 6 outlines the experimental studies conducted for software defect prediction utilizing XAI methods.
- Chapter 7 gives the results for Case Study 1.
- Chapter 8 gives the results for Case Study 2.
- Chapter 9 gives the results for Case Study 3.
- Chapter 10 gives the results for Case Study 4.
- Chapter 11 discusses our findings, emphasizing the efficacy of different XAI methods.
- Chapter 12 addresses validity threads.
- Chapter 13 concludes with implications and future work in defect prediction modeling.

2. BACKGROUND

2.1. Software Defect Prediction (SDP)

The prediction of software defects uses historical data from earlier software initiatives to create predictive models. To forecast the likelihood of defects in new or existing code, these models are trained using attributes or metrics collected from code and other software artifacts. Code complexity, code churn, code size, and developer experience are frequently utilized characteristics in software fault prediction. To create these models, statistical and machine learning methods are frequently used. Below, we summarize some attempts in the academic literature in the context of SDP. Since there are many scientific studies in this field from the past to the present and since the main focus of this study is not SDP but the explainability of the models, we summarize five SDP studies with the highest average number of citations.

Basili et al. [10] authored a paper, which is a landmark in the field, in 1996. The study addressed the limitations of code metrics as inputs for software defect prediction. It highlighted the requirement for a clearly defined measuring methodology and the fact that not all measures are equally valuable for predicting software defects. In order to assist developers in selecting the best tools for spotting potential defects, Juang et al. [11] compared and contrasted many static analysis techniques for bug detection. Machine learning methods were investigated by Menzies et al. [3] for predicting software defects. The authors showed how these methods could boost prediction accuracy and give information on the variables that affected defects most predictably. Moonen and Deursen [12] examined the relationship between software updates and defect prediction. The study highlighted how past change information could be a useful indicator of potential faults in the future. A study by Bavota et al. [13] covered a wider range of machine learning research in software engineering, including defect prediction. It offered a thorough summary of current research in the area.

These are few and well-recognized examples of significant publications in SDP. With improvements in ML, data mining, and software analytics, the field is still evolving, making

it essential for enhancing software quality and development procedures. To improve defect prediction accuracy and utility, researchers are always creating new strategies and improving already-existing ones.

2.2. Machine Learning Models

There are numerous ML models, each developed for a particular purpose or type of difficulty. In this study, we examine the ML models by categorizing them under three headings. The categorization of ML models into tree-based, neural network-based, and probabilistic-based approaches is not universally standardized across all studies or resources. However, these categorizations are common in various ML textbooks [14], [15], [16] and academic papers [17], [18], [19],[20], [21].

1. Tree-based ML model: is a class of algorithms that use decision trees as their fundamental building blocks. These models utilize tree structures to represent and make predictions based on the relationships between input features and target variables. Decision Trees, Random Forest, Gradient Boosting Machines (GBM), XGBoost are some commonly used tree-based models. Tree-based models have several advantages, including the ability to handle both numerical and categorical features, interpretability (especially with shallow decision trees), and capturing non-linear relationships and interactions between variables. They are robust to outliers and can handle missing values. However, they may be prone to overfitting, and the interpretability may decrease with deeper and more complex trees. These tree-based models find applications in various domains, including classification, regression, and feature selection. They are widely used for tasks such as risk analysis, fraud detection, customer segmentation, and recommendation systems, among others. In this study, we used GBM and RF that mix various decision trees to increase the classification's robustness and accuracy that build an ensemble of decision trees in a stage-wise manner [22].

2. Neural network-based ML model: A sort of technique called neural network-based ML models uses the error, or the difference between expected and actual values, to gauge how effectively the model is working [23], [24]. These models are frequently employed for tasks like regression analysis and classification. To perform better, all of these models rely on reducing the difference between expected and actual values. In this study, we used the Multilayer Perceptron (MLP) that can be applied to a variety of tasks, such as classification, regression analysis, and image recognition. Neural network-based ML models are particularly useful for complex problems where traditional rule-based systems may be inadequate.

3. Probabilistic ML model: ML models that are built on probabilistic notions and principles are known as probabilistic ML models [25], [26]. These models quantify the uncertainty and variability associated with the data by incorporating probability distributions and statistical methods into the predictions they make. ML models with a probabilistic basis can be used for many different classification issues. Instead of just labeling an input, these models often predict the likelihood that it belongs to a specific class. When there is doubt in the data and the decision is not definite, this can be quite helpful. Naive Bayes (NB), Logistic Regression (LR), Random Forests (RF) are some examples for this type of ML classification problems. In this study, we used NB classifier that is based on Bayes' theorem and assumes that features are conditionally independent given the class label.

2.3. Explainability and XAI methods

The demand to understand the systems we use is quite natural. Since AI-based software systems need to become more opaque due to their inherent complexity, sometimes people such as domain experts, system engineers, and customers struggle to understand particular parts of them. [27], [28]. Due to the difficulties involved in understanding AI-based software systems, there is an increasing interest about the explainability of these systems. Explanations offer numerous benefits, including enhancing comprehension of the system, justifying decisions, fostering trust, and improving usability. Conversely, inadequate

explanations can breed distrust [29], diminish user acceptance and satisfaction with the system [30], and hinder the adoption of new technologies. Consequently, ensuring explainability is vital for enhancing the quality of AI-based software systems and should be integrated into their development process.

According to Miller et al. [31], a commonly used definition of explainability is the extent to which an individual can understand the arguments for a decision or behavior. By (1) making every decision transparent and (2) directly describing the thinking behind each result, AI/ML models can be made more accessible [28]. As a consequence, there have been attempts to look into how to provide reasons for the decisions made by complex, black-box models. The public and academic communities are very interested in explainability, which is the ability to make AI algorithms understandable to people, as a result of the rapidly expanding use of AI and ML technologies, particularly those using opaque deep neural networks. This is a problem since lay users frequently request AI explanations. These users may not have a strong comprehension of AI technically, but they do have preconceived notions about what makes good explanations for judgments made in a similar area. As an illustration, among the most common methods for elucidating the prediction of an ML classifier, as numerous XAI algorithms aim to achieve, is displaying the features with the largest weights that contribute to the model prediction [32]. A model that predicts a patient has the flu, for example, may use the symptoms of sneezing and headache as supporting data [33]. It is debatable, however, if such an explanation considerably improves a medical decision-support tool or satisfies a doctor's requirement to comprehend the AI. To bridge the gap between user requirements and XAI algorithms for efficient transparency, user-centered methods and interdisciplinary cooperation to explainability have been urged by the HCI (Human Computer Interaction) field. This new area of study typically expands on frameworks for human explanations or carries out empirical research on how explanation elements affect human cooperation with AI.

Decision trees like those in white-box AI/ML models were commonly employed in earlier studies. We may exploit the transparency of such white-box AI/ML models to determine each feature's relative impact on the learned outcomes by directly examining the model

components. As opposed to sophisticated black-box AI/ML models, white-box AI/ML models usually produce generic explanations and are frequently less accurate. In terms of transparency and interpretability of the decision-making process, black-box and white-box AI are two distinct approaches to creating and developing artificial intelligence systems. A sort of machine learning algorithm known as ”black-box AI” bases its decision-making process on complicated, challenging-to-understand algorithms. The system’s decisions do not have a clear definition; instead, they develop through interactions between the data and the algorithm. Concerns regarding the possibility of bias and prejudice in the decision-making process have been raised by the lack of transparency and interpretability of the decision-making process in black-box AI systems. Contrarily, white-box AI describes a class of AI systems where the decision-making process is more open and understandable. With white-box AI, the system’s decision-making is based on clearly stated rules or logic that are accessible to people for review and comprehension. This strategy offers greater interpretability and transparency, which can be useful for finding any systemic biases or inaccuracies. white-box AI, also known as explainable AI (XAI), enables consumers and developers to comprehend how the system generates a specific conclusion or prediction. This can increase confidence in the system and guarantee that it is coming to just and moral conclusions.

An explanation of how an answer was obtained is essential for guaranteeing confidence and transparency in many applications. One such application is in medicine, where the doctors must be absolutely certain of their conclusions. For instance, they want to know how AI assessed a CT scan image to determine whether a person has an illness. AI-based systems aren’t entirely faultless. Understanding how a result was reached can, therefore, not only promote trustworthiness but also help prevent potentially fatal mistakes. Answers to additional ”wh*” questions (such as ”why,” ”when,” ”where,” etc.) may be necessary in some other applications (such as law and order). These ”wh*” inquiries are beyond the capabilities of conventional AI.

This necessity for explainability has given rise to Explainable AI (XAI), a new field of AI study. Figure 2.1 demonstrates how XAI can expand the capabilities of AI by addressing the

”wh” issues that were absent from conventional AI. Essential applications like health care, defense, law enforcement, etc. have shown a lot of interest in the XAI since in these fields, providing an explanation for how an answer was arrived at (i.e., responses to ”wh” queries) is just as crucial as providing the actual answer. XAI research has thus become a top focus in both academia and industry. Even if several studies have already been put forth, more and more work is still needed to fully utilize XAI.

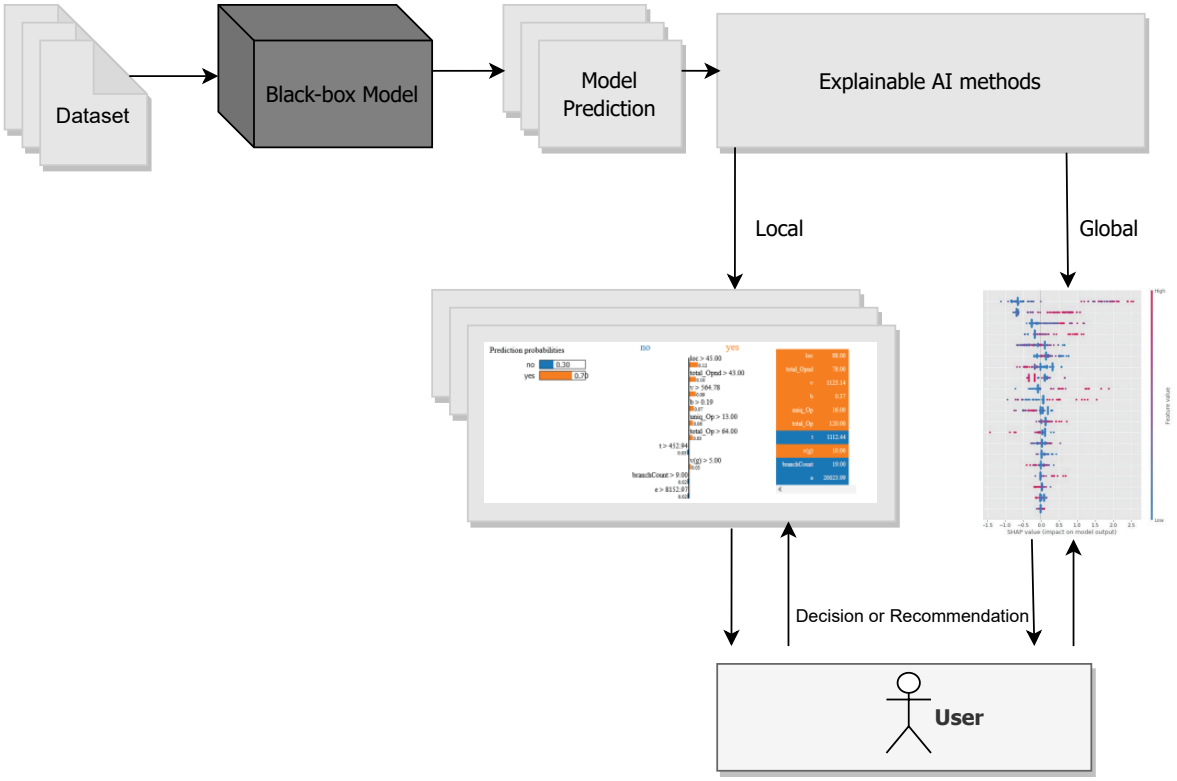


Figure 2.1 The process for both local and global explainability

Explainability methods seek to reveal the steps used by a model to arrive at a specific prediction, enabling people to comprehend and, if necessary, intervene or correct the decision-making process. Feature importance analysis, saliency maps, decision trees, and counterfactual justifications are a few common explainability methodologies. Recently, model-independent techniques have been used to explain the predictions of these sophisticated black-box AI/ML models at the instance and global levels (e.g., SHAP [34], LIME [33], etc.). Below, we list the XAI methods used in this study as described in Table 2.1. In the following paragraphs, we also overview these XAI methods.

Table 2.1 XAI methods used in this study

XAI Method	Scope		Intuition behind
	Global	Local	
SHAP	✓	✓	explain the contribution of each feature in a prediction.
LIME		✓	provide a local explanation for a specific prediction by fitting a simple model that approximates the behavior of the complex machine learning model in the vicinity of the prediction.
ANCHOR	✓		generate a set of rules that approximate the decision-making process of the machine learning model.
ELI5	✓	✓	provide a simple and easy-to-understand explanation of how the machine learning model is making its predictions.
PROTODASH	✓		find the most representative and diverse set of prototypes that capture the essence of the entire dataset.
PDP	✓	✓	understand the relationship between a target variable and a predictor variable while holding all other variables constant.

SHAP (SHapley Additive exPlanations.) A machine learning model's predictions are deciphered using the SHAP method, which is an explainability method [34]. It offers a framework for determining the significance of each feature for each prediction, enabling us to gauge the relative contributions of various features to the model's output. Any kind of ML model can be used with the SHAP because it is model-agnostic. It enables for the identification of key features in a model and offers an unifying framework for analyzing complex models, like deep learning models. Moreover, feature significance plots that depict the SHAP values can be used to help non-technical stakeholders understand the model's output. Overall, the SHAP method, which is well-known in the explainable AI community, provides a strong and adaptable approach for describing ML models. It offers a clear and understandable approach to comprehend how a model generates its predictions, which makes it simpler to trust and utilize in practical applications.

LIME (Local Interpretable Model-agnostic Explanations). In 2016, Ribeiro, Singh, and

Guestrin proposed the suggestion of this model-agnostic method [33]. An explainability method called LIME is used to decipher the predictions provided by an ML model. By building a local, interpretable model that roughly represents the behavior of the original model close to the prediction, it gives an explanation for a specific prediction. The LIME method can be combined with other explainability methods to acquire a more thorough knowledge of a model's behavior. Overall, it is a good method for describing the behavior of ML models.

ELI5 (Explain Like I'm Five) is an open-source Python library used for explaining ML models [35]. By emphasizing the key elements that contribute to a certain prediction, it offers a straightforward and natural approach to interpret a model's behavior. Model-agnostic, or adaptable to any kind of ML model, is how ELI5 is defined. In order to decide which features are most crucial for a certain prediction, the model's learning weights and input data features are examined. Moreover, ELI5 offers visualizations that can aid users in understanding the behavior of the model, like feature significance plots and decision tree visualizations.

Partial Dependence Plot (PDP) is an open-source Python library for model interpretation and explainability [36]. A partial dependence plot (PDP) is a useful tool for visualizing the relationship between a target variable and a set of predictor variables in a machine learning model.

Anchor is an explainability method that aims to provide interpretable and understandable explanations for the predictions made by machine learning models. It was introduced by Ribeiro et al. in 2018 as a way to generate rule-based explanations for individual instances [37]. The key idea behind Anchor is to identify a compact rule or condition that holds true for a specific instance and correlates with its predicted outcome. This rule is referred to as an "anchor." An anchor represents a simplified explanation that captures the most important factors leading to a particular prediction.

Protodash is an explainability technique used to interpret the behavior of ML models [38]. The prototype hypothesis, on which it is founded, contends that individuals classify objects according to prototypes, or representative examples, of each category. The Protodash method

finds a group of prototypes that most accurately reflect a specific class of data points. The behavior of an ML model is then explained using these prototypes by emphasizing the characteristics that are unique to each prototype. The Protodash method employs the prototypes after they have been found to describe the behavior of a ML model by emphasizing the attributes that are most crucial for each prototype. As it concentrates on representative samples rather than the full dataset, this strategy offers a more understandable and comprehensible explanation of the model's behavior. All things considered, the Protodash method is a potent tool for deciphering ML models, especially when the data is high-dimensional and challenging to display. By concentrating on typical cases, it offers a more understandable and interpretable description of the model's behavior, which can aid users in better comprehending the model's behavior and identifying areas for improvement.

The goal of explainable AI is to improve processes and make sense of the outcomes of ML and AI approaches. It strives the comprehensible, interpretable, and transparently the black-box character of the advanced predictive algorithms. To put it another way, we require XAI to support the predictions made by models, account for their shortcomings and misconceptions, enhance the models, and find new information, theories, and insights [39]. XAI methods can be categorized according to their scope (local vs. global). The scope of an interpretation reveals its localisation or globality. Local interpretation is centered on comprehending a single sample prediction, whereas global techniques aim to describe the behavior of the model as a whole [40]. Achieving global interpretability is frequently a challenge in practice due to the computational load of evaluating huge datasets with a rich range of factors, even if it is essential to recognize the overall factor importance, which produces "population-level" decisions [39]. In order to grasp the model globally, several interesting local techniques have recently been developed that combine local interpretations in various contexts [40], [33], [41], [35]. They include SHAP, LIME, and ELI5.

3. RELATED WORK

The ”related work” section of this doctoral thesis is divided into two subsections to provide a focused examination of distinct yet interrelated aspects of the research domain. This structured division allows for a comprehensive exploration of both general principles applicable to ML models and domain-specific considerations relevant to SDP, facilitating a deeper understanding of the research landscape.

3.1. Explaining ML models

This subsection delves into foundational studies and methodologies pertaining to the interpretability and transparency of machine learning models. This subsection encompasses discussions on various explainability techniques, including SHAP, LIME, and ELI5, aimed at elucidating the decision-making processes of ML models.

In the literature, there are some attempts to explain AI-based software, more specifically ML and deep learning models. Misheva et al. implemented SHAP and LIME methods over an ML-based model in the credit risk management domain [42]. Knapivc et al. analyzed the Explainable Artificial Intelligence (XAI) methods of SHAP and LIME for decision support in medical domain [43]. Another study that covers the explainability methods belongs to Islam et al. [44]. The authors defined explainability and interpretability from different perspectives (e.g., psychology and social science), and focused on the feature importance for explaining the ML model. Mane et al. conducted a study over a deep neural network model in cybersecurity domain [45], where they used different explanation methods such as SHAP, LIME, Contrastive Explanations Method (CEM), ProtoDash, and Boolean Decision Rules. Another study belongs to Bugaj et al. [46], which investigates the explainability of LightGBM modelling over a Home Credit dataset. In this study, the authors used SHAP method to provide a clear understanding for the local prediction, and proposed to improve the prediction results by considering the SHAP values.

Palacio et al. [47] establish a new conceptual framework to integrate research and techniques created in the discipline of explainable AI in order to tackle the increasing heterogeneity and lack of consensus on what defines an explainable or interpretable model (XAI). Two key definitions, ”explanation” and ”interpretation,” serve as the foundation for this framework. The authors start by pointing out two crucial requirements for such a framework: 1- Commensurable: commonly used measures must exist in order to accurately examine two distinct methods, 2- Universal: the context must be defined by a general process. The authors provide a real-world example where their framework can be used to compare several XAI techniques which are LIME, SHAP and MDNet, illuminating the degree to which they individually address various explainability pipeline aspects.

Ehsan et al. [48] focus on the issue of mapping the sociotechnical gap between societal needs and technical affordances in XAI systems. They empirically develop a framework to make it easier to map the sociotechnical gap in XAI using two case studies from two distinct fields (sales and mental health). The technological and social wings in this structure each had three construction pieces. The social wing comprises elements of trust, actionability, and values, while the technical wing includes data, model, and explanations. The authors offer a set of starter questions for each of these building pieces to help close the sociotechnical gap in XAI systems.

Hu et al. [49] propose the DARPA-sponsored Explainable AI Toolkit (XAITK), which relies on the achievements of the four-year DARPA XAI initiative. The toolkit has XAI-specific functionalities and provides a standard, searchable structure that offers technical and scientific direction for comprehending and utilizing AI technologies.

In order to increase transparency at each level of the ML process, Mane and Rao devise an explainable AI framework and deploy deep neural networks for network intrusion detection [45]. The authors apply SHAP, LIME, Contrastive Explanations Method (CEM), ProtoDash and Boolean Decision Rules via Column Generation (BRCG) methods to o NSL-KDD dataset to generate explanations. They evaluate explanations by considering user groups as Network analyst, data scientist, and end-user. As a result, the framework describes offered

explanations at each stage of the machine learning pipeline that are appropriate for various users in network intrusion detection systems.

Sanneman and Shah provide a three-level framework (perception, comprehension, and projection) to construct and assess explanations for the behavior of AI systems [50]. The authors suggest that XAI levels are based on what informational requirements human users have.

Wali and Khan [51] offer an intrusion detection system (IDS) capable of detecting all sorts of harmful content in network traffic using the global explanations created by the SHAP method. This IDS examines model justifications created during the creation and review phase for increasing user confidence and preserving operational integrity to provide the transparent decision-making strategy.

Heimerl et al. release NOVA, a cutting-edge annotation tool for emotional behavior analysis that employs an interactive approach that includes the ”human in the loop” [52]. NOVA uses cutting-edge eXplainable AI (XAI) algorithms to give users visual explanations in addition to a confidence value for automatically produced observations. By performing a user research with 53 participants, the authors look into how such approaches can help non-experts in terms of trust, as well as developing accurate mental models about the system. The findings show that, XAI visualisations assist users in developing more accurate mental models of the ML system. However, the authors argue that explanations in the field of AI should pay more attention to user needs in addition to the categorization problem and the model they seek to describe.

Kapcia et al. introduce ExMed, a framework that lets domain experts use XAI data analytics without explicitly needing programming knowledge [53]. The two real-world medical case studies—the first analyzing the effectiveness of the COVID19 control measure and the second calculating lung cancer patient life expectancy using the synthetic Simulacrum health dataset—illustrate its range of applications. Using XAI techniques, it may combine the adaptability of medical sub-domain transferability with an essence of trust through explainability.

Amini et al. examine all types of crash characteristics to identify the risk variables that lead to collisions with serious injuries [54]. To choose the machine learning algorithm with the highest prediction performance as the basis model, their approach first looks at a variety of machine learning models. Then, it applies two well-known state-of-the-art XAI approaches (namely, leave-one-covariate-out and TreeExplainer) from the literature. Lastly, their method establishes a unified ranking list of the most crucial factors causing serious auto accident injuries by using an information fusion strategy.

Rohlfing et al. provide a theoretical foundation that enabled them to investigate explainability as a social and interactive process [55]. Two processes that affect the course of contact on a microlevel are proposed by the authors to achieve the specific purpose of an explanation: Monitoring and scaffolding are first. Both are well-known from studies on interaction and development, with monitoring at the heart of a good interaction and scaffolding at the center of social learning.

Jin et al. work together to develop the end-user-centered XAI framework (EUCA) by fusing the knowledge of AI and HCI [56]. The authors begin by identifying twelve end-user-friendly explaining formats, such as feature-, example-, and rule-based explanations, that did not require expert understanding of the topic. To put EUCA into practice, XAI designers can make prototyping cards again for twelve explanatory forms using the available layouts and examples.

Houda et al. build an entirely novel XAI-based framework to provide explanations about any crucial deep learning-based decisions for IoT-related intrusion detection systems (IDSs) [57]. To find IoT-related intrusions, their approach use a novel IDS for IoT networks that they also designed by utilizing deep neural networks. On base of their DNN-based model, the system applies three primary XAI methods: RuleFit, Local Interpretable Model-Agnostic Explanations (LIME), and Shapley Additive Explanations (SHAP). To improve the understanding of DL-based decisions, their approach can offer either local and global explanations.

El-khawaga et al. propose a framework that enables comparing the explanations produced by a chosen number of XAI approaches that are now on the market in the context of Predictive Process Monitoring (PPM) [58]. Their study compare the XAI techniques at various degrees of granularity (global and local XAI).

Naz et al. propose an explainable AI-based framework to solve the issue of classification result explainability in the healthcare arena using medical picture CXRs [59]. This study provides a framework for explainability of lung disorders and first categorizes the pulmonary condition, and then uses the interpretable LIME model to further explain the categorization findings.

In Table 3.1, we categorize the related studies in the context of AI-based software by considering different levels of explainability.

3.2. Explaining software defect prediction models

In the recent years, particularly from 2020 onwards, there has been a growing effort to investigate the explainability and comprehensibility of defect prediction models.

Mohammadkhani et al. [60] performed a systematic literature review on the 24 most relevant published studies in eXplainable Artificial Intelligence for Software Engineering (XAI4SE), selected from a pool of 869 primary studies identified through keyword search. The study highlights the significance of defect prediction in software engineering, comprising 68% of the identified studies, emphasizing its prevalence in software maintenance. It notes a predominant application of XAI methods to classic machine learning models over more intricate ones. However, it underscores a notable absence of standard evaluation metrics for XAI methods in the literature, leading to confusion among researchers and a deficiency of benchmarks for comparisons.

Esteves and colleagues suggested examining the model constructed with the XGBoost algorithm across various projects within the Jureczko datasets [61]. Their emphasis was on determining feature significance through a model sampling technique, aiming to assess how

Study	User Type	Category of Methods	Explanation Method	Domain	ML Algorithm	Dataset
Sanneman et al. [50]	Engineer Scientist	Local Global Example-based	NA	NA	NA	NA
Naz et al. [59]	NA	Local Global	LIME	Healtcare	CNNs	Covid-ct Covid-net
Alikhademi et al. [**?**]	Developer	Local Global	LIME AI Explainability 360	Criminogenic risk	RF, LR, DL	Compas
Kapcia et al. [53]	Domain Expert	Local Global	LIME, SHAP	Healtcare	SVM, RF, MLP, XGBoost	Covid-19 Artificial Simulacrum
Rohlfing et al. [55]	NA	NA	NA	NA	NA	NA
Wang et al. [**?**]	NA	Local Global	Deep-SHAP	Power system	DL	Synthetic data
Mane & Rao [45]	Data scientist, Analyst, End User	Local Global Exemplar based	SHAP, LIME, Contrastive Explanations Method (CEM), ProtoDash Boolean Decision Rules	Cybersecurity	Deep Neural Network	NSL-KDD
Ehsan et al. [48]	Data Scientists, Data Analysts, Database Engineers, AI Operations Engineers, Product Managers, End users	Local, Global, Counterfactual	NA	Sales, Mentalhealth, Cybersecurity	NA	
Amini et al. [54]	NA	Local Global	Leave-one- covariate-out (LOCO) TreeExplainer (TE)	Motor vehicle crashes	DT, RF, GBT	Crash Report Sampling System (CRSS)
Palacio et al. [47]	NA	NA	LIME, SHAP, MDNet	NA	NA	NA
Jin et al.	End user	Feature-based Example-based Rule-based	NA	House, Health, Car, Bird	NA	Boston housing, Diabetes dataset, BDD100K, CUB-200
Houda et al. [57]	Cybersecurity experts	Local, Global	RuleFit, LIME, SHAP	Intrusion Detection System	DL	NSL-KDD, UNSW-NB15
Our study	Data scientist, Domain Expert, End User	Model-based, Attribution-based, Example-based	SHAP, LIME, Skater, ELI5, Protodash, CEM	Software Defect Prediction	Distance-based Rule-based Neural network-based Probabilistic	KC2

Table 3.1 Categorization of the context in related studies (from Section 3.1.) for the XAI

the number of features affects the performance of defect prediction models. They employed the SHAP method to comprehend the defect model, finding that the significance of feature

numbers could differ based on the project.

Shin et al. [62] investigated the reliability of two explainability methods, which are LIME and Breakdown, under different defect prediction models. Since they observed inconsistent results from these two model-agnostic techniques, they concluded that these methods are unrealiable for explaining the defect prediction models.

Santos and Figueiredo [63] conducted an exploratory study over five different Java projects by using XGBoost algortihm and, feature importance and SHAP methods. According to their results, they concluded that although it is difficult to generate a unique solution for defect prediction, it is possible to identify defects with different features' combination.

Jiarpakdee et al. [64] conducted an empirical study that focuses on local and global explainability of datasets consisting of 32 releases that span 9 open-source software systems. They used LIME and Breakdown methods, and concluded that there is a need for explainability of software defect prediction, and these methods are useful for both global and instance level.

Mori and Uchira [65] investigated the trade-off between model accuracy and interpretability for software defect prediction. They proposed a new "superposed naive Bayes (SNB)" classification model over 13 real world projects, and compared the accuracy and interpretability results with different classification algortihms such as ensemble learners, regression models, decision trees, support vector machines, Bayesian learners, neural networks, and rule-based learners. According to their results, their proposed SNB method gave an optimal accuracy and interpretability result by comparing to the other algorithms.

Al-Smadi et al. [66] introduce a novel framework employing eleven machine learning classifiers across twelve datasets for software defect prediction. To address feature selection, four nature-inspired search algorithms—particle swarm optimization, genetic algorithm, harmony algorithm, and ant colony optimization—are utilized. The synthetic minority oversampling technique (SMOTE) is adopted to tackle data imbalance issues. Moreover, the Shapley additive explanation model is employed to highlight the most influential

features. Results indicate that gradient boosting, stochastic gradient boosting, decision trees, and categorical boosting exhibit superior performance, achieving over 90% accuracy and ROC-AUC. Additionally, the ant colony optimization technique surpasses other feature extraction methods in this study.

Begum et al. [67] focus on the preprocessing and experimentation with machine learning models for software fault diagnosis using four real datasets provided by NASA, each containing twenty-one features. The preprocessing involved employing the Synthetic Minority Oversampling Technique and Label Encoding techniques to handle data imbalance and categorical variables, respectively. The study experimented with thirteen machine learning models for software fault diagnosis, including Random Forest Regression, Linear Regression, Naïve Bayes, Decision Tree Classifier, Logistic Regression, KNeighbors Classifier, AdaBoost, Gradient Boosting Classifier, Gradient Boosting Regression, XGB Regressor, XGBoost Classifier, Extra Trees Classifier, and Support Vectors Machine. After evaluation, the XGBR model demonstrated superior performance based on metrics like accuracy, mean square error, and R2 score. Moreover, the study employed XAI techniques such as LIME and SHAP to identify and interpret software fault-related features, providing insights into model predictions and feature importance.//

As seen in the previous sub-sections, there are studies that focus on the explainability of ML models in different domains as well as in SDP domain. However, there is no study that investigates the explainability of ML models for software defect prediction, which is very important to assure product quality in software engineering, by using several post-hoc model-agnostic methods (i.e., ELI5, SHAP, and LIME) over different NASA defect prediction datasets (i.e., KC2, PC1, CM1). The datasets and methods used in this study have not been investigated together in the literature. Therefore, in this study, we focus on understanding the RF, GB, NB, and MLP classifiers by using several post-hoc and model-agnostic methods over different well-known SDP datasets.

Our study represents a pioneering endeavor within the realm of Explainable AI, particularly focusing on SDP. Unlike previous research, which often superficially examines user

types, goals, and AI explainability, our work stands out by introducing a groundbreaking framework tailored specifically to the intricate nuances of SDP. As illustrated in Table 3.1, we systematically categorize related studies based on various levels of explainability, highlighting our study as the first to comprehensively address the distinct challenges within SDP.

What sets us apart is our proactive approach to not only synthesizing insights from existing perspectives but also integrating them into a cohesive framework. This framework not only identifies critical gaps within the SDP domain but also provides a structured roadmap to effectively bridge these gaps through the incorporation of diverse explanation types and evaluation measures. By adopting such a comprehensive strategy, our study significantly advances the practical implementation of interpretable AI in SDP, thereby enriching the broader landscape of AI-based software research.

Furthermore, we contribute to the field by developing ensemble models for both local and global interpretability in SDP. Leveraging techniques such as SHAP, ELI5, and LIME, we rank feature importance scores globally and locally, thus creating interpretable ensemble XAI models. These models not only enhance the interpretability of SDP outcomes but also offer insights into the underlying mechanisms driving the predictions, thereby improving trust and transparency.

In addressing a pervasive challenge within feature importance analysis, we shed light on the often-overlooked impact of explainability methods. Unlike conventional approaches that may introduce opacity to model predictions, our methodology ensures clarity and transparency, enhancing the overall understanding of SDP outcomes.

Moreover, we challenge the conventional wisdom surrounding feature importance analysis in SDP performance studies. Unlike previous works that lack a transparent rationale for data point removal, our methodology ensures replicability by providing a clear basis for feature selection. This approach enhances the validity and robustness of our findings, setting a new standard for future research in the field.

In summary, our study not only pushes the boundaries of Explainable AI within the context of SDP but also provides practical tools and insights to enhance the transparency, interpretability, and reliability of AI-based SDP systems.

4. METHODOLOGY

Our research methodology follows an exploratory approach grounded in Design Science Research (DSR) principles [9], which emphasizes iterative problem-solving cycles to develop innovative solutions for practical problems. We sequentially performed the following tasks across four case studies, each focused on enhancing interpretability and transparency in ML for software defect prediction.

The case studies adopted in this thesis centers around the utilization of various XAI techniques, such as ELI5, SHAP, and LIME, to enhance the transparency and interpretability of ML models in SDP. Unlike conventional approaches, which often lack transparency in feature selection and outlier detection, this research prioritizes clear explanations and insights into the decision-making process.

Furthermore, the thesis employs ensemble modeling techniques to leverage the strengths of various XAI methods, both locally and globally. By integrating feature importance scores from SHAP, ELI5, and LIME, and employing ML classifiers such as RF, GB, NB, and MLP, this research seeks to optimize predictive accuracy while maintaining interpretability.

This research follows a structured methodology, to address the challenges and objectives outlined in the study. The methodology is organized into five main phases as shown in Figure 4.1:

The structured methodology for addressing the problem of enhancing interpretability and transparency in ML models for SDP can be described in detail as follows:

1. Problem Identification and Motivation:

    - Initially, a Systematic Literature Review (SLR) [68] is conducted to explore existing research in the domain of software quality for AI-based software.
    - Through the SLR, a new software quality attribute, ”Explainability,” is identified, motivating further investigation into its role in ML models for SDP.

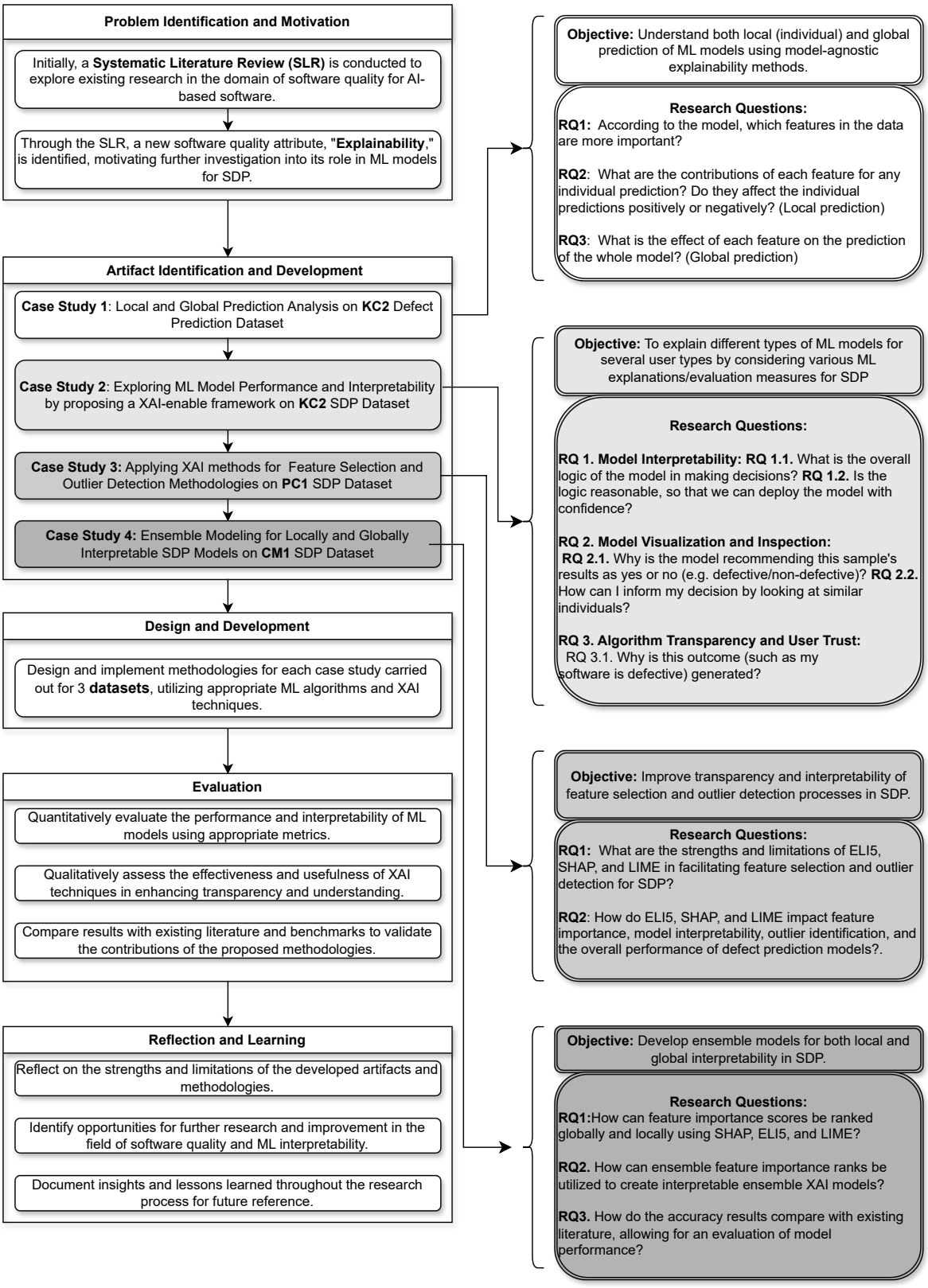


Figure 4.1 The steps of DSR method proposed in this study

- Identify the problem: The lack of interpretability in machine learning models for software defect prediction hinders stakeholders' understanding of decision-making processes.
- Justify the problem: Inadequate transparency can lead to mistrust in the predictions, potentially impacting software quality.

2. Artifact Identification and Development:

   - Four case studies are identified and developed to address different aspects of the problem:
     - Case Study 1: Local and Global Prediction Analysis on KC2 SDP Dataset
       * Objective: Understand both local (individual) and global prediction of ML models using model-agnostic explainability methods.
       * Research Questions:
         · RQ1: Identify important features according to the model.
         · RQ2: Analyze contributions of each feature for individual predictions.
         · RQ3: Investigate the effect of each feature on the prediction of the whole model.
     - Case Study 2: Exploring ML Model Performance and Interpretability by proposing an XAI-enabled framework on KC2 SDP Dataset
       * Objective: Increase performance of ML models for SDP while enhancing model interpretability.
       * Research Questions:
         · RQ1: Evaluate the overall logic of ML models in decision-making.
         · RQ2: Visualize and inspect model decisions for better understanding.
         · RQ3: Assess algorithm transparency and user trust in SDP models.
     - Case Study 3: Applying XAI methods for Feature Selection and Outlier Detection Methodologies on PC1 SDP Dataset
       * Objective: Improve transparency and interpretability of feature selection and outlier detection processes in SDP.

    - Research Questions:
        - RQ1: Examine limitations of traditional methods in feature selection and outlier detection.
        - RQ2: Prioritize transparency and interpretability through XAI techniques (ELI5, SHAP, LIME) and provide clear explanations for feature selection and outlier detection decisions.
  - Case Study 4: Ensemble Modeling for Locally and Globally Interpretable SDP Models on CM1 SDP Dataset
    - Objective: Develop ensemble models for both local and global interpretability in SDP.
    - Research Questions:
        - RQ1: Rank feature importance scores globally and locally using SHAP, ELI5, and LIME.
        - RQ2: Ensemble feature importance ranks to create interpretable ensemble XAI models.
        - RQ3: Compare accuracy results with existing literature to evaluate model performance.

3. Design and Development:
    - Methodologies for each case study are designed and implemented, utilizing appropriate ML algorithms and XAI techniques.

4. Evaluation:
    - The performance and interpretability of ML models are quantitatively evaluated using performance metrics such as accuracy, precision, recall, and F1-score.
    - The effectiveness and usefulness of XAI techniques in enhancing transparency and understanding are qualitatively assessed.
    - Results are compared with existing literature and benchmarks to validate the contributions of the proposed methodologies.

5. Reflection and Learning:

    - Strengths and limitations of the developed artifacts and methodologies are reflected upon.
    - Opportunities for further research and improvement in the field of software quality and ML interpretability are identified.
    - Insights and lessons learned throughout the research process are documented for future reference.

By following this structured methodology, the research aims to address the overarching objective of enhancing interpretability, transparency, and model performance in ML models in the domain of software quality assurance for SDP, with each component contributing to a comprehensive understanding and effective solution development.

5. IMPLEMENTATION OVERVIEW

We propose to answer the set of questions that were derived from the constituents of conceptualization of design science as a paradigm [9] as listed below:

1. *Problem instance*
    (a) What specific problem is the paper addressing?
2. *Problem Understanding Approach*
    (a) How did the authors gain an understanding of the problem they aimed to solve?
3. *Proposed Solution(s)*
    (a) How were the identified problems addressed through proposed interventions?
4. *Design Approach*
    (a) How did the authors formulate and reach their proposed solution?
5. *Validation Approach*
    (a) How did the authors implement the intervention/solution to validate its effectiveness in addressing the problem instance?
6. *The Technological Rule*
    (a) What outcomes do the authors aim to achieve through their research?
    (b) In what contexts or situations does their proposed solution apply?
7. *Relevance, Convincing the Target Stakeholder*
    (a) What types of problems and solutions are encompassed by the technological rule proposed in the paper?
    (b) Who are the relevant stakeholders for these problem-solution pairs?

(c) How do the authors persuade their readers regarding the relevance of the problem-solution pair to the stakeholders involved?

8. *Rigor*

   (a) What actions were implemented to ensure the comprehension of the problem instance is both reliable and accurate?

   (b) How were measures taken to guarantee that the proposed intervention effectively addresses the problem instance?

   (c) What methodologies were utilized to validate the design decisions made during the development of the proposed solution?

9. *Novelty*

   (a) What are the unique and innovative contributions of the paper to the field?

This doctoral thesis adopts an exploratory research method to investigate and address the challenges associated with enhancing interpretability and transparency in ML for SDP. The methodology employed in this research follows a systematic and iterative process aimed at exploring innovative solutions and gaining insights into the complex interplay between ML models, software datasets, and interpretability techniques. By adopting the DSR paradigm as a structured methodology as shown in Figure 5.1, we provide a comprehensive understanding of our research approach, intervention, and its implications for practitioners and stakeholders in the domain of SDP and ML interpretability.

5.1. Details of each step of the proposed methodology followed during implementation

In this subsection, we will give the details of 9 steps of the proposed methodology shown in Figure 5.1.

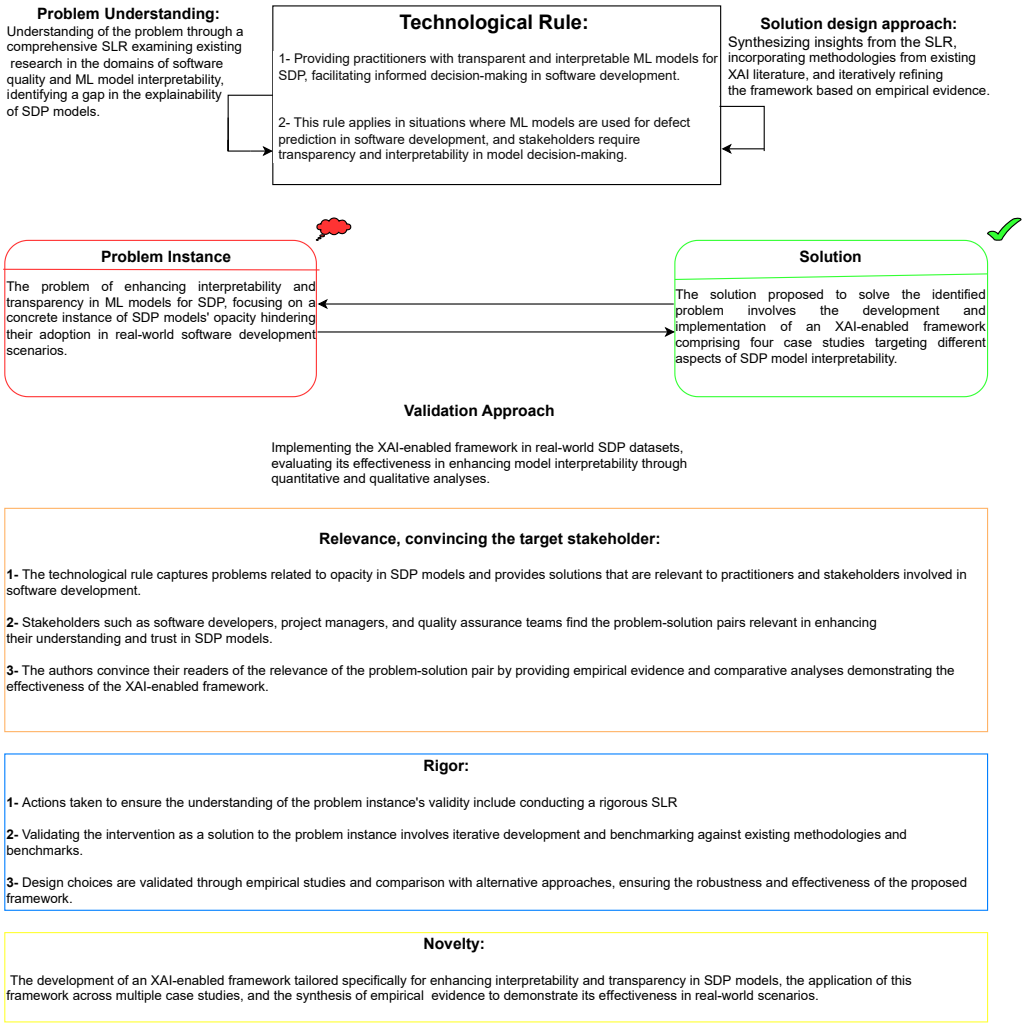


Figure 5.1 The visual abstract of proposed methodology in this study

1- Problem Instance. This doctoral thesis addresses the problem of enhancing interpretability and transparency in ML models for SDP, focusing on a concrete instance of SDP models' opacity hindering their adoption in real-world software development scenarios.

2- Problem understanding approach.

- We firstly gained an understanding of the problem through a comprehensive SLR [68] (Systematic Literature Review on Software Quality for AI-based Software) examining existing research in the domains of software quality for AI-based software, identifying a gap in the explainability of SDP models.

- Identified gaps, limitations, and emerging trends in the field to define the research problem and objectives.

One of the RQ in the SLR study we conducted during this thesis study is "*What are the experienced challenges of quality in AI-based software?*". By answering this RQ, we provided a detailed understanding of the popular research areas being investigated up to now in the scientific literature. As shown in Figure 5.2, we classified each challenge under a specific topic.

1. Software Quality. There are traditional quality models in the software engineering world. Given that AI-based software diverges from conventional software, there is no well-defined guideline, framework, road map, or model on measuring software quality for AI-based software. Therefore, inadequate process definitions to be followed, quality metrics, attributes, and assurance techniques cause new challenges in the academia or industry.

2. Software Development. AI-based software systems consistently display unique characteristics (e.g. being black-box) in engineering because models (components) are constructed by training with data in an inductive manner. Also, it is possible to encounter unexpected outcomes. Hence, it might cause new problems with development processes when trying to evaluate the quality of these systems.

3. Design. This category is about designing systems that imply ML models (components) by considering and performing the characteristics of "Change Anything Change Everything" (CACE) [69]. A slight change in training data may affect learning results, thus on the functional behavior of such systems.

4. Social Aspect. This category is about the communication between developers, development skills, the combination of data scientists, software engineers, and different kinds of branches when developing such systems.

5. Testing. Testing ML systems pose difficulties that appear from the different nature of ML systems, compared to relatively more deterministic conventional software systems [70]. Certain machine learning applications are designed to grasp the characteristics of datasets in

scenarios where human users lack knowledge of the correct answers. Testing and debugging such machine learning software is difficult because there is no reliable test oracle. [71].

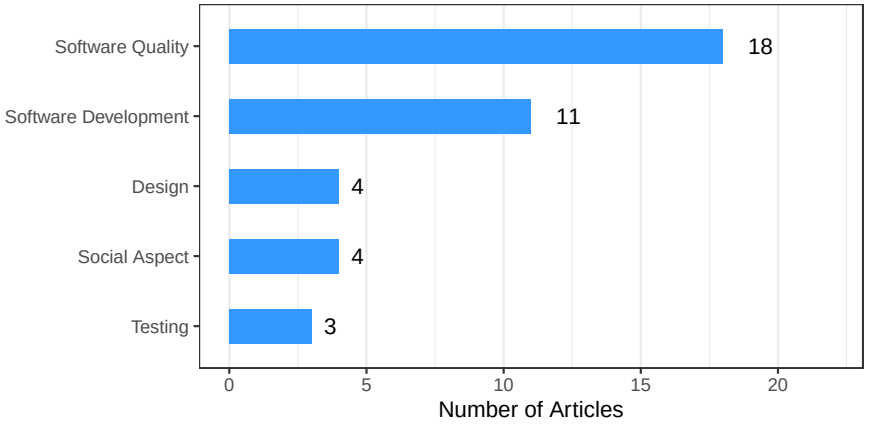


Figure 5.2 Distribution of experienced challenges in the primary studies

According to the results, the ”Software Quality” challenge is the most experienced challenge faced by the researchers with 18 studies, followed by 11 studies with the ”Software Development” challenge.

The list of challenges in the ”Software Development” category is about the difficulties due to the different nature of AI-based software from traditional ones. The necessity of new measurements has emerged to understand and explain the quality characteristics required for AI-based software since developing such software is inherently challenging and not transparent. Unlike traditional software, it solves more complex problems, its behavior depends on training data, and it is dependent on data and processes.

For the challenges identified in our SLR study, we then focused on the ”Software Quality” challenge and made an in depth analysis over this challenge. To comprehend and describe the current state of the art in this subject and examine its limitations and open issues that will guide future research, we did a comprehensive analysis of the literature from 1988 to 2021 and chose 280 primary articles.

To find and choose the pertinent papers in the literature for this study, we performed a string-based search. Before beginning the investigation, we consulted our earlier systematic

literature review (SLR) study [68] to gain knowledge and get more familiar with the context of quality for AI-based software, and we used its findings to construct search strings.

In order to improve search results for string-based searches, first we established search keys by adding synonyms and abbreviations as supplemental terms. To reach potentially relevant papers, we constructed and searched for 14 different sentences on Elsevier, ACM, IEEE, and Google Scholar to make a complementary search. After construction the search terms, the finals search string was composed of the terms that represent **Population**- *AI-based software* and **Intervention**- *Software Quality of AI-based software* by following the guidelines of Kitchenham [72]. We finalized our search terms as in Table 5.1.

Based on the investigation for all the quality attributes , we proposed to show the most investigated QAs by plotting their frequency. Figure 5.3 shows the frequency of all attributes in ISO 25010 and the newly founded ones out of ISO 25010.

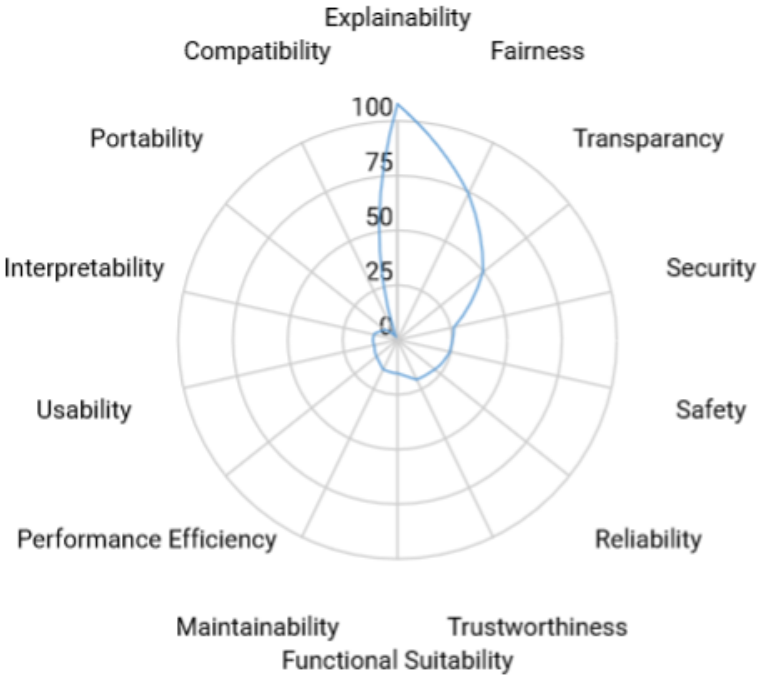


Figure 5.3 Radar chart for the frequency of each QA.

According to the results, by considering 280 studies we acquired after inclusion/exclusion criteria, we observed that the most investigated quality attributes (with the frequency of primary studies), which are not in ISO 25010, are Explainability (108), Fairness (74), and

Table 5.1 Search strings defined and number of studies returned on Google Scholar, Elsevier, ACM, and IEEE databases

Search String	**# returned studies (Elsevier)**	**# returned studies (ACM)**	**# returned studies (IEEE)**	**# returned studies (Scholar)**
1. (“Quality Attributes” OR “Non Functional Requirements” OR “Quality Characteristics” OR ”Quality Model” OR “ISO 25010”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems).	2,738	1,899	3,664	25
2. (“Functional Suitability” OR “Functional Completeness” OR “Functional Correctness” OR “Functional Appropriateness”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems).	2,663	623	142	9,190
3. (“Performance Efficiency” OR “Time Behavior” OR “Resource Utilization” OR “Capacity”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,692	33,563	28,931	2,170,000
4. (“Compatibility” OR “Interoperability”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,636	13,618	2,951	2,160,000
5. (“Usability” OR “Learnability” OR “Accessibility”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,699	23,035	2,683	3,640,000
6. (“Reliability” OR “Maturity” OR “Availability” OR “Recoverability” OR “Fault Tolerance”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,640	54,336	22,887	3,320,000
7. (“Security” OR “Privacy”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,639	42,516	30,243	5,640,000
8. (“Maintainability” OR “Modularity” OR “Testability” OR “Reusability” OR “Modifiability”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,642	11,139	1,756	85,700
9. (“Portability” OR “Adaptability”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,639	9,635	2,796	507,000
10. (“Explainability” OR “XAI”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,637	1,941	581	41,900
11. (“Fairness”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,637	7,275	733	359,000
12. (“Safety”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,649	19,229	14,067	4,340,000
13. (“Understandability” OR “Transparency”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,663	8,997	935	1,050,000
14. (“Trustworthiness”) AND (“Machine Learning” OR “ML” OR “Artificial Intelligence” OR “AI” OR “Deep Learning” OR “Computer Vision” OR “Neural Network”) AND (software OR systems)	2,636	2,681	372	150,000

Transparency (50). Also, among the quality attributes in the ISO 25010; Security (26), Reliability (22), and Functional Suitability (15) are the most investigated ones. Although it is observed that quality attributes in ISO 25010 are suitable for AI-based software, there are much more attempts at the newly added quality attributes specific to AI-based software.

Therefore, in our doctoral thesis, we delve into the findings derived from our SLR studies concerning the challenges encountered in the ”Software Quality” and ”Software Development” domain, particularly in the context of AI-based software. Our study highlights the inherent disparities between AI-based software and traditional ones, elucidating the unique difficulties stemming from their distinct nature. A prominent revelation from our research is the pressing need for novel metrics and methodologies to grasp and elucidate the essential quality attributes requisite for AI-based software. We underscore the complexity inherent in developing such software, compounded by its opaque and intricate nature. Unlike conventional software, AI-driven systems tackle more intricate problems, and their behavior is contingent upon the nuances of their training data and underlying processes. Through our doctoral thesis, we aim to shed light on the critical importance of explainability in AI-based software, addressing the challenges unearthed through our rigorous SLR study within the realm of software development.

3- Proposed solution(s). The intervention proposed to solve the identified problem involves the development and implementation of an XAI-enabled framework comprising four case studies targeting different aspects of SDP model interpretability.

Within this section, each case study can be introduced as a distinct intervention proposed to address the identified problem of enhancing interpretability and transparency in ML models for SDP. Each case study represents a unique approach or methodology designed to tackle different facets of the problem. Here’s how each case study can be introduced as a distinct intervention proposed to address the identified problem:

Case Study 1: Local and Global Prediction Analysis on KC2 SDP Dataset: In this case study, we propose an intervention aimed at understanding both local (individual) and global prediction of ML models using model-agnostic explainability methods. The objective is

to enhance the interpretability and transparency of ML models for SDP by analyzing the importance of features in the data, their contributions to individual predictions, and their effects on overall model predictions.

Case Study 2: Exploring ML Model Performance and Interpretability by proposing an XAI-enabled framework on KC2 SDP Dataset: This case study presents an intervention to improve the interpretability of ML models for SDP by proposing a XAI-enabled framework. By exploring different types of ML models and leveraging various XAI explanations and evaluation measures, we aim to enhance the understanding of model logic, visualization, and algorithm transparency, thus enabling confident deployment of the models.

Case Study 3: Applying XAI methods for Feature Selection and Outlier Detection Methodologies on PC1 SDP Dataset: In this case study, we introduce an intervention to enhance the transparency and interpretability of feature selection and outlier detection processes in SDP models. By applying XAI methods such as ELI5, SHAP, and LIME, we aim to improve the understanding of feature importance, model interpretability, and outlier identification, thus optimizing the overall performance of defect prediction models.

Case Study 4: Ensemble Modeling for Locally and Globally Interpretable SDP Models on CM1 SDP Dataset: This case study proposes an intervention to develop ensemble models for both local and global interpretability in SDP. By leveraging SHAP, ELI5, and LIME to rank feature importance scores globally and locally, we aim to create interpretable ensemble XAI models. Additionally, we compare the accuracy results with existing literature, allowing for an evaluation of model performance and demonstrating the effectiveness of the proposed approach.

Each of these case studies represents a unique intervention designed to address specific aspects of the problem of enhancing interpretability and transparency in ML models for SDP, thereby contributing to the development of a comprehensive XAI-enabled framework for SDP.

4- Design approach. We arrived at our proposed solution by synthesizing insights from the SLR, incorporating methodologies from existing XAI literature, and iteratively refining the framework based on empirical evidence and stakeholder feedback.

In the ”Design approach” section, we discuss each case study in more detail by explaining how the proposed solutions were developed and implemented. Here’s how we structure the discussion for each case study:

For **Case Study 1**, which focuses on Local and Global Prediction Analysis on the KC2 Defect Prediction Dataset, the following points discuss the rationale, methodology, analysis, and implementation details:

Rationale for Dataset Selection:

- The KC2 Defect Prediction Dataset was selected due to its widespread use as a benchmark dataset in software defect prediction research. It contains a diverse range of software metrics and defect labels, making it suitable for analyzing the interpretability and transparency of ML models in SDP.

Methodology for Local and Global Prediction Analysis:

In this part, we give details about methodology of Case Study 1, and Fig. 5.4 shows the taxonomy of explainability methods used in this study.

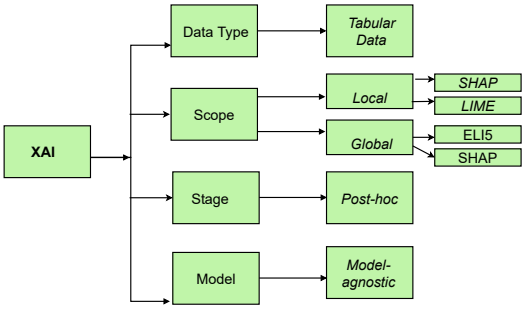


Figure 5.4 Taxonomy of methods applied in Case Study 1

- The methodology involves utilizing model-agnostic explainability methods such as ELI5, SHAP, and LIME to analyze local and global predictions of ML models.

- For local prediction analysis, individual instance-level explanations are generated to understand how each feature contributes to a particular prediction.
- For global prediction analysis, aggregate feature importance measures are calculated to assess the overall impact of each feature on model predictions.

Analysis of Features' Importance:

- Feature importance analysis is conducted to understand the relative importance of different features in the dataset according to the ML model.
- This analysis involves calculating SHAP values or LIME explanations for each feature, indicating their contributions to individual predictions.
- Additionally, global feature importance measures such as mean absolute SHAP values or feature permutation importance are computed to assess their impact on the overall model predictions.

Implementation Process:

- Preprocessing steps may include handling missing values to prepare the KC2 dataset for analysis.
- ML models are trained using various algorithms such as RF, GB, NB, and MLP.
- Model-agnostic explainability methods like SHAP or LIME are then applied to the trained models to generate explanations for individual predictions.
- Evaluation procedures involve assessing the fidelity of the explanations, measuring the model's predictive performance, and comparing the interpretability of different ML models.

By following these steps, the **Case Study 1** aims to provide insights into the interpretability and transparency of ML models for SDP, specifically focusing on understanding local and global prediction behaviors using the KC2 Defect Prediction Dataset.

In **Case Study 2**, we explore ML model performance and interpretability within the context of software defect prediction using the proposed XAI-based framework for SDP as shown İn Figure 5.5. This framework is designed to offer insights into the decision-making process behind the models used for defect prediction, leveraging Explainable AI (XAI) techniques to enhance interpretability and transparency.

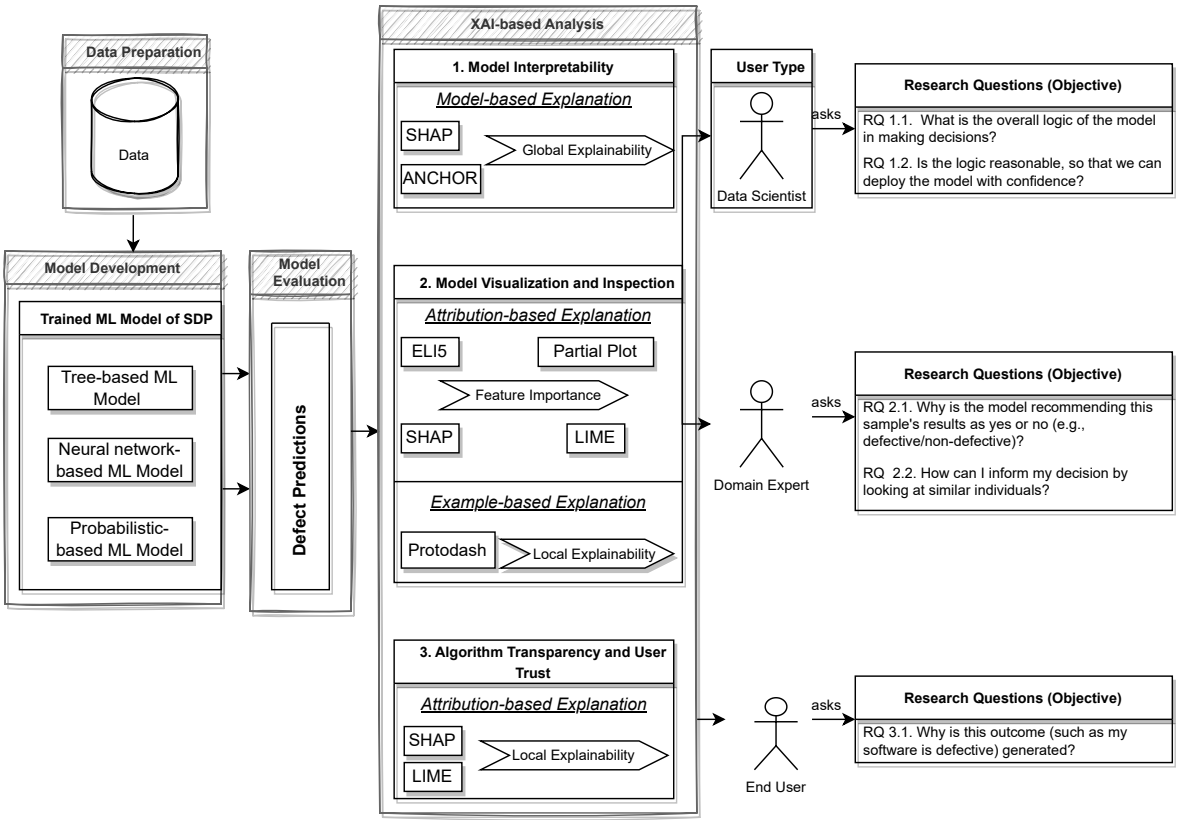


Figure 5.5 The proposed XAI framework for SDP in Case Study 2

Data preparation:

- Data collection and cleaning are essential steps in preparing the data needed for software defect prediction.

Model development:

- Creating an ML model for defect prediction involves using various methods, including decision trees, logistic regression, or deep neural networks. In this study, ML models are categorized into distance-based, neural network-based, and probabilistic-based models, and each is analyzed individually.

Model evaluation:

- The performance of the ML models is assessed using metrics such as accuracy, precision, recall, and F1-score. It's crucial to ensure that the model is reliable and precise enough to be applied for defect prediction.

Interpretation and visualization:

- The findings from the XAI analysis are interpreted and presented in an approachable manner. Visuals like feature importance plots, summary plots, or heatmaps are created to aid users in understanding how the model generates its predictions.

XAI-based analysis:

- XAI approaches are employed to examine the decision-making process of the models. This provides insights into the parameters crucial for defect prediction and the logic behind the model's predictions. The XAI analysis is categorized into "Model interpretability," "Model visualization and Inspection," and "Algorithm transparency and User trust," using methods such as SHAP, LIME, ELI5, Partial plot, Anchor, and Protodash.

By following this framework, **Case Study 2** aims to provide a comprehensive understanding of ML model performance and interpretability in software defect prediction, utilizing XAI techniques to enhance transparency and facilitate informed decision-making in software development.

In **Case Study 3**, we focus on applying XAI methods for Feature Selection and Outlier Detection Methodologies on the PC1 SDP Dataset. The following points discuss the dataset, methodology, utilization of XAI methods, and implementation details:

Rationale for Dataset Selection:

- The PC1 SDP Dataset is introduced as the dataset used in this case study, selected based on its relevance and suitability for exploring feature selection and outlier detection methodologies in software defect prediction. Justification for its selection may include factors such as dataset size, diversity of software metrics, and availability of labeled defect data.

Methodology for applying XAI methods:

- XAI methods, including ELI5, SHAP, and LIME, are employed for feature selection and outlier detection in the PC1 SDP Dataset.
- Feature selection involves using XAI methods to identify the most important features contributing to model predictions, thereby improving model interpretability.
- Outlier detection leverages XAI techniques to pinpoint instances deviating significantly from the norm, potentially indicating anomalous behavior or data quality issues.
- Our methodology involved meticulous execution of feature selection using ELI5 and SHAP across diverse models like RF, GB, NB, and MLP. This ensured a thorough examination of feature importance and model interpretability. Additionally, our pioneering use of LIME for outlier detection deepened our understanding of model behavior, which is vital for robust defect prediction.

The visual abstract template shown in Figure 5.6 highlights the technological rule proposed, the problem-solution instance addressed, and the assessment of the produced knowledge in terms of relevance, rigor, and novelty.

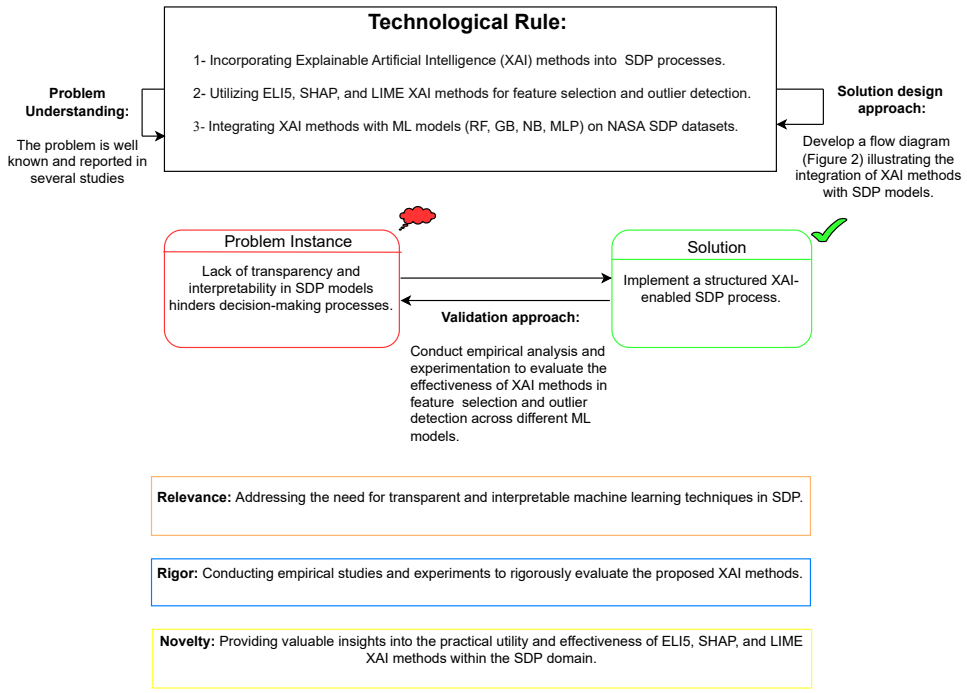


Figure 5.6 A visual abstract template capturing the key elements of Case Study 3

Utilization of XAI methods:

- XAI methods such as ELI5, SHAP, and LIME are utilized to enhance feature importance, model interpretability, and outlier identification.
- These methods provide insights into the relative importance of features, the impact of individual features on model predictions, and the rationale behind outlier classifications.

By following this methodology, **Case Study 3** aims to demonstrate the effectiveness of XAI methods for enhancing feature selection and outlier detection in software defect prediction, providing insights into model interpretability in the PC1 SDP Dataset.

Case Study 4 delves into Ensemble Modeling for Locally and Globally Interpretable SDP Models on the CM1 SDP Dataset. Here's a discussion of the key aspects:

Rationale for Dataset Selection:

- The CM1 SDP Dataset is introduced as the dataset under scrutiny in this case study. Its selection is justified by its relevance to the investigation of ensemble modeling techniques for locally and globally interpretable SDP models. Factors such as dataset characteristics, defect prediction context, and availability of labeled data contribute to its suitability for this study.

Methodology for Ensemble Modeling with XAI methods:

- Ensemble modeling methodology revolves around achieving both local and global interpretability using SHAP, ELI5, and LIME. Ensembling XAI methods involves aggregating the insights from multiple XAI techniques to provide a more comprehensive view of model behavior. This approach combines the strengths of individual XAI methods and offers enhanced interpretability and reliability in ML models.

Ensemble XAI-enabled SDP Models

- Denote SDP models that ensemble different XAI methods for improved interpretability and performance in defect prediction tasks.

Utilization of Feature Importance Scores:

- Feature importance scores, obtained through SHAP, ELI5, and LIME, are ranked globally and locally to create interpretable ensemble XAI models. This process involves identifying the most influential features in predicting defects both at the individual instance level and across the entire dataset. By incorporating feature importance rankings from multiple XAI methods, the ensemble models aim to enhance interpretability while maintaining predictive accuracy.

Figure 5.7 outlines how we have adopted and applied the DSR methodology suggested by Engström et al. [9] to our study, in order to address the challenges faced in SDP through the exploration of ensemble XAI methods.

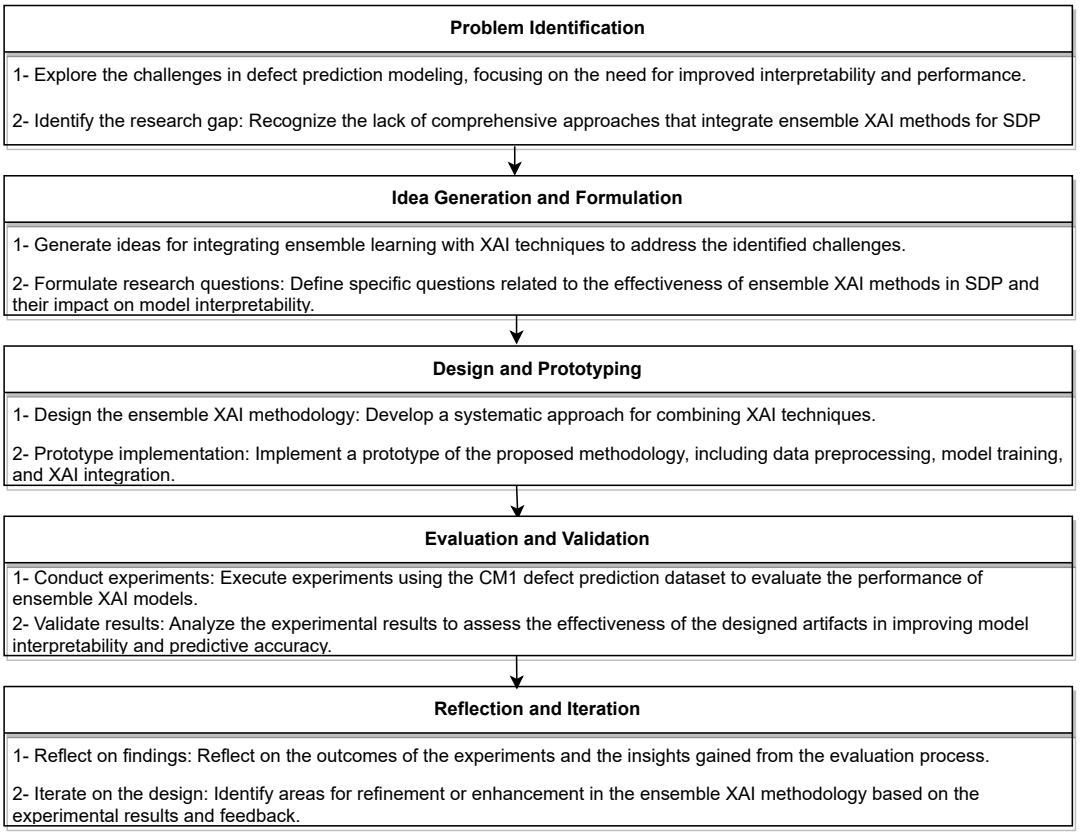


Figure 5.7 DSR methodology adopted and followed in Case Study 4

Implementation Process:

- The implementation process entails ensemble model training, evaluation, and comparison with existing literature.
- We combine and create ensemble feature importance scores by ranking features based on importance in each method (ELI5, SHAP and LIME). We calculate ensemble importance scores by averaging the rank values of each XAI method for each feature.
- We select the top k features based on ensemble importance scores obtained from the previous steps and use the selected features.
- Then, we retrain the ML models and calculate various performance metrics including accuracy, precision, recall, F1-score, and AUC for the retrained model using the test dataset.

- Evaluation metrics such as accuracy, precision, recall, F1-score and AUC are utilized to assess the performance of ensemble models.
- Comparisons with existing literature provide insights into the effectiveness of the proposed approach in enhancing both local and global interpretability of SDP models.

By following this methodology, Case Study 4 aims to demonstrate the effectiveness of ensemble modeling techniques in enhancing the interpretability of SDP models while maintaining predictive performance. The CM1 SDP Dataset serves as a suitable platform for exploring these techniques and contributing to the advancement of interpretable defect prediction methodologies.

To summarize, by discussing each case study in the ”Design approach” section, we provide readers with a comprehensive understanding of how the proposed solutions were developed and implemented to address the identified problem of enhancing interpretability and transparency in ML models for SDP.

5- Validation approach.

- We iterate on the design and implementation of the proposed solutions based on the feedback received from professors and preliminary evaluations.
- We applied the intervention/solution to the problem instance by implementing the XAI-enabled framework in real-world SDP datasets (KC2, PC1, CM1), evaluating its effectiveness in enhancing model interpretability through quantitative and qualitative analyses, and validated the effectiveness and generalizability of the refined methodologies through rigorous experimentation, comparative analysis, and case studies.

Here, the iterative nature of the implementation process is highlighted. Each case study is applied to the problem instance (SDP models' opacity) to validate its effectiveness in enhancing interpretability and transparency. Through iterative experimentation and

evaluation, the performance of each case study is assessed, and adjustments may be made based on the insights gained.

6- The Technological Rule.

- The effect we wish to achieve through our research is to provide practitioners with transparent and interpretable ML models for SDP, facilitating informed decision-making in software development.
- This rule applies in situations where ML models are used for defect prediction in software development, and stakeholders require transparency and interpretability in model decision-making.
- In summary, the proposed solution in the thesis is the development and implementation of an XAI-enabled framework for enhancing interpretability and transparency in SDP models.

7- Relevance, convincing the target stakeholder.

- The technological rule captures problems related to opacity in SDP models and provides solutions that are relevant to practitioners and stakeholders involved in software development.
- We convince our readers of the relevance of the problem-solution pair by providing empirical evidence and comparative analyses demonstrating the effectiveness of the XAI-enabled framework.

8- Rigor.

- Actions taken to ensure the understanding of the problem instance's validity include conducting a rigorous SLR.
- Validating the intervention as a solution to the problem instance involves iterative development and benchmarking against existing methodologies and benchmarks.

- Design choices are validated through empirical studies and comparison with alternative approaches, ensuring the robustness and effectiveness of the proposed framework.

In this section, the actions taken to ensure the validity of the intervention and the design choices also underscore the iterative process. The refinement and validation of each case study's methodologies, as well as the ongoing assessment of their effectiveness, demonstrate the rigor of the research approach.

9- Novelty. The novel contributions in the paper include the development of an XAI-enabled framework tailored specifically for enhancing interpretability and transparency in SDP models, the application of this framework across multiple case studies, and the synthesis of empirical evidence to demonstrate its effectiveness in real-world scenarios.

By following this iterative and exploratory approach, our research aims to advance the understanding and practical application of interpretable and transparent ML techniques in the context of SDP. Through systematic exploration and iterative refinement, we seek to contribute valuable insights and innovative solutions to address the challenges faced in enhancing ML model interpretability and transparency for SDP.

6. OVERVIEW OF CASE STUDIES

In this section, we outline the case studies conducted for software defect prediction utilizing XAI methods. Both individual and hybrid approaches of XAI methods have been employed as locally and globally for the prediction of software defects. The experiments were implemented using the Python programming language and the KAGGLE tool. The NASA software defect dataset repository was utilized as the primary dataset for this study.

6.1. Dataset Details

In this study, three datasets are available for software defect prediction: KC2, PC1, and CM1. These datasets are utilized to apply software defect prediction techniques. Due to the prevalence of studies utilizing these datasets, a comparative analysis is presented to demonstrate the significance of the proposed hybrid approach.

Table 6.1 offer insights into the datasets utilized in the study, encompassing module counts, programming languages, defect percentages, lines of code, and total defects within each module.

Table 6.1 Description of NASA Software Defect Datasets

Dataset	**Programming Language**	**Defect Percentage**	**Lines of Code**
KC2	C++	20.50	522
PC1	C	6.94	1109
CM1	C	9.83	498

Additionally, Table 6.2 outlines the attributes present in each dataset, which play a crucial role in feature selection and subsequent model development for software defect prediction.

Table 6.2 Attributes Details of NASA Software Defect Datasets Used in This Study

Feature	**Description**	**Data Type**
loc	McCabe's "line count of code"	numeric
v(g)	McCabe "cyclomatic complexity"	numeric
ev(g)	McCabe "essential compelxity"	numeric
iv(g)	McCabe "design complexity"	numeric
n	Halstead "total operators + operands"	numeric
v	Halstead "volume"	numeric
l	Halstead "program length"	numeric
d	Halstead "difficulty"	numeric
i	Halstead "intelligence"	numeric
e	Halstead "effort"	numeric
b	Halstead "number of delivered bugs"	numeric
t	Halstead's "time estimator"	numeric
IOCode	Halstead's "line count"	numeric
IOComment	Halstead's "count of lines of comments"	numeric
IOBlank	Halstead's "count of blank lines"	numeric
IOCodeandComment	Halstead's "count of lines of code and comments"	numeric
uniq_Op	"unique operators"	numeric
uniq_Opnd	"unique operands"	numeric
total_Op	"total operators"	numeric
total_Opnd	"total operands"	numeric
branchCount	"of the flow graph"	numeric
problems	module has/has not one or more reported defects	categorical

6.2. Evaluation Metrics

The selection of evaluation metrics in this doctoral thesis is underpinned by the fundamental objective of rigorously assessing the performance of the proposed methodologies in the context of software defect prediction. Each chosen metric offers unique insights into different facets of model performance, collectively providing a comprehensive understanding of the models' efficacy in achieving the study's objectives.

Accuracy: Accuracy serves as a fundamental metric quantifying the overall correctness of predictions made by the models. It delineates the ratio of correctly predicted instances to the total number of instances in the dataset. Accuracy holds paramount significance in this thesis as it provides a holistic evaluation of the predictive capabilities of the models under scrutiny.

Precision and Recall: Precision and recall offer nuanced perspectives on the models' performance in handling positive instances (defective software components). Precision emphasizes the ability of the models to minimize false positives, ensuring that identified defects are genuine and actionable. On the other hand, recall focuses on capturing all positive instances, thereby minimizing the risk of false negatives, which could lead to undetected defects. These metrics directly relate to the study's objective of enhancing the accuracy and reliability of defect prediction models.

F1-score: The F1-score provides a balanced assessment by considering both precision and recall. It is particularly valuable in scenarios where achieving a balance between false positives and false negatives is critical. By striving for a high F1-score, the study aims to develop models that achieve optimal performance across both precision and recall, thereby maximizing the overall effectiveness of defect prediction.

Area Under the Receiver Operating Characteristic Curve (AUC-ROC): The AUC-ROC metric evaluates the models' ability to discriminate between positive and negative instances across various threshold settings. It directly relates to the objective of developing models with robust discriminatory power, ensuring reliable differentiation between defective and non-defective software components. Overall, the selection of these evaluation metrics is

carefully aligned with the objectives of the study, aiming to develop and assess defect prediction models that exhibit high accuracy, reliability, and interpretability in real-world software development contexts.

7. RESULTS FOR CASE STUDY 1: Explainable AI for Software Defect Prediction with Gradient Boosting classifier

In this case study, Model-agnostic explainability methods, including SHAP, ELI5, and LIME, were employed to analyze feature importance for both local and global predictions. We seek to understand both local (individual) prediction and global prediction (i.e., model as a whole). Accordingly, we aim to answer the research questions (RQ) below by using the model-agnostic explainability methods:

- **RQ1.** According to the model, which features in the data are more important?
- **RQ2.** What are the contributions of each feature for any individual prediction? Do they affect the individual predictions positively or negatively? (Local prediction)
- **RQ3.** What is the effect of each feature on the prediction of the whole model? (Global prediction)

with RQ1, we focus on feature importance that the analysis revealed the importance of features in predicting software defects according to the model. With RQ2, we focus on contributions to individual predictions. For individual predictions, the contributions of each feature were examined to determine their impact, either positive or negative, on the classification outcome. Lastly, with RQ3, we focus on the effect on the whole model. The effect of each feature on the overall prediction of the model was investigated, shedding light on their collective influence on model predictions. The results from the feature importance analysis were interpreted and visualized using various techniques such as feature importance plots and summary plots. These visualizations facilitated a deeper understanding of the model's decision-making process and provided actionable insights for stakeholders.

7.1. Results for RQ1. According to the model, which features in the data are more important?

If we decide that an ML classifier is untrustworthy, a widespread task is to modify the feature sets and retrain the classifier in order to improve generalization, which is known as ”feature engineering”. Explanations can help in this task by introducing the important features, especially to remove features which were not generalized. Therefore, with this RQ, we propose to understand the most important features in the dataset by considering their weights. To achieve this, we focused on the feature importance and used the concept of ”permutation importance” within ELI5 library. In doing so, we calculated importance weights after fitting the model with the GB classifier. By this way, we guaranteed not to change the model or the prediction of the model. The advantage of using ”permutation importance” concept is that it is model-agnostic and can be calculated many times with different permutations of the features. Firstly, we trained the model, and shuffled an individual column randomly by leaving the target and the other columns in place, and made a prediction. Then, by checking these prediction and target values, we obtained permutation feature importance weights within the decrease in a model score. A drop in the model score is indicative of how much the model depends on the feature. According to the whole process, we observed the results shown in Fig. 7.1. While the features at the top indicates the most important ones, the features at the bottom shows the least important ones across the model. According to the results, the most important features are ”total_Opnd” and ”uniq_Opnd” features, sequentially. The first numerical values in the weight column show how much the relevant feature degrades the performance of the model, while the numerical values after ”+/-” show the change in the performance of the model after reshuffle.

After analyzing each feature according to their impact on model prediction by using permutation importance with ELI5, we also investigate for how a feature affect the model prediction. For this purpose, we showed the partial plot for the feature ”total_Opnd”, ”uniq_Opnd”, and ”b” as shown in Fig. 8.2, Fig. 7.3 and Fig. 7.4, respectively. The y-axis in the figures is interpreted as the change in model prediction performance.

Weight	Feature
0.2544 ± 0.3415	total_Opnd
0.1996 ± 0.3061	uniq_Opnd
0.0713 ± 0.1776	ev(g)
0.0650 ± 0.3353	n
0.0533 ± 0.2616	loc
0.0518 ± 0.2267	IOBlank
0.0434 ± 0.1516	IOComment
0.0393 ± 0.1996	total_Op
0.0372 ± 0.1328	uniq_Op
0.0264 ± 0.2166	IOCode
0.0246 ± 0.1299	b
0.0238 ± 0.1960	d
0.0199 ± 0.1776	i
0.0198 ± 0.1922	v
0.0187 ± 0.1759	v(g)
0.0148 ± 0.1398	iv(g)
0.0115 ± 0.1799	e
0.0111 ± 0.2345	t
0.0098 ± 0.0994	branchCount
0.0028 ± 0.0836	l
... 1 more ...	

Figure 7.1 Permutation importance weights for features in KC2 dataset

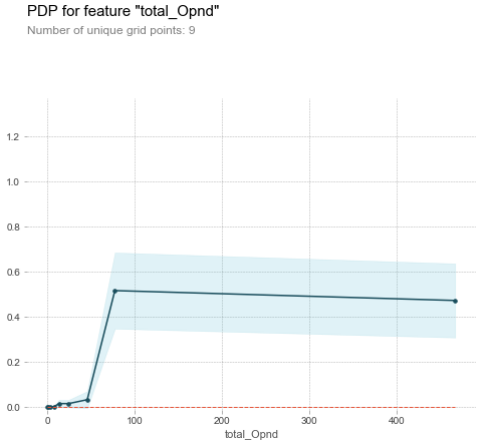


Figure 7.2 Partial dependence plot for the feature ”total_Opnd”

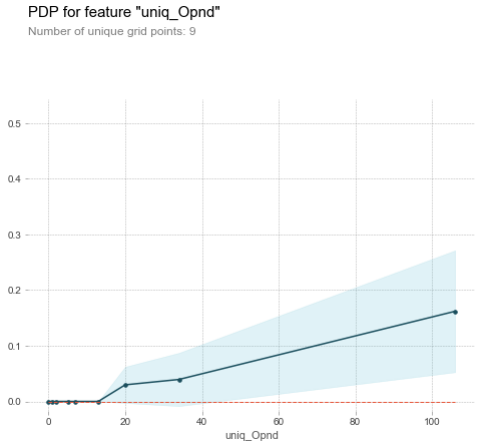


Figure 7.3 Partial dependence plot for the feature ”uniq_Opnd”

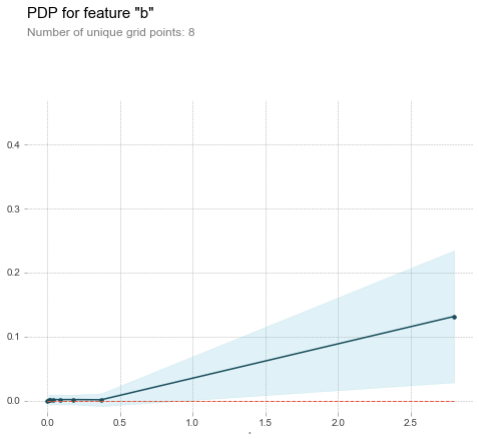


Figure 7.4 Partial dependence plot for the feature ”b”

In case of Fig. 8.2, we observe that increasing the total number of operands increases the chances of predicting the defects' existence as ”yes”, but increasing the number has a small effect on the model prediction performance. However, when we look at the Fig. 7.3 and Fig. 7.4, we observe that increasing the number of unique operands and number of bugs increases the chances of predicting the defects' existence as ”yes”.

Lastly, we depict 2D Partial Dependence Plots for the features of ”total_Opnd” and ”b” in order to observe the interaction between these features. In Fig. 7.5, we see the model predictions for any combination of the two features. While the results in the graphs of Fig. 8.2 and Fig. 7.4 are supportive with the Fig. 7.5, the model has its highest prediction result with the increase in the number of errors at a certain level of ”total_Opnd”, no matter how much the value of ”total_Opnd” increases.

Although there is no study that investigates the interaction between these two metrics, it is inevitable to say that increasing in the number of bugs makes the software more defective [73]. However, while increasing the total number of operands up to the value of 78 increases the prediction performance, increasing the value of this metric after 78 has only a small affect on the prediction results.

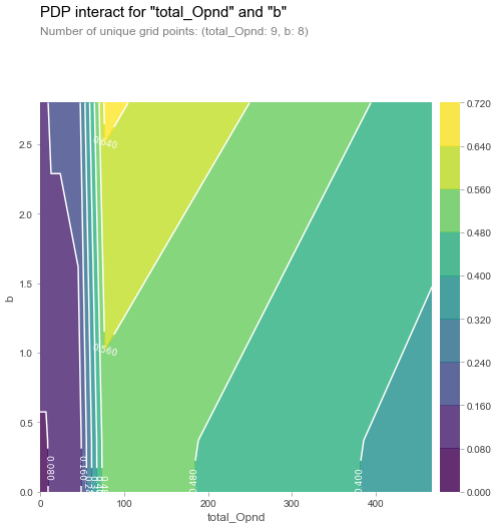


Figure 7.5 2D Partial Dependence Plots for the features ”total_Opnd” and ”b”

7.2. Results for RQ2. What are the contributions of each feature for any individual prediction? Do they affect the individual predictions positively or negatively? (Local prediction)

Up to here, we have observed general insights from a machine learning model. From now on, we focus on the local prediction, break down the model for a single prediction, and seek to understand the contribution of each feature on an individual prediction. In order to classify a single instance as ”yes” or ”no”, and explain a single prediction by the ML model; we used both SHAP and LIME methods, and then compared the results obtained with these two methods. We selected an instance from KC2 dataset (row 12) randomly and analyzed the SHAP and LIME results. We show the respective model outputs for that instance in Fig.7.6 and Fig.7.7.

According to the results, the model predicts the defects' existence as ”yes” with a change of 70% by using SHAP and LIME methods. This indicates that prediction probabilities of these two methods are the same.

We describe Fig. 7.6 in detail as the following:

- Output value: indicates the model prediction performance for a single instance (i.e., row 12 of KC2 dataset).
- Base value: indicates the model prediction performance if the model does not know any training data. It is defined as ”the value that would be predicted if we did not know any features for the current output”.
- Colors (red or blue): if the features increase the prediction performance, then they are shown as red; otherwise, they are shown as blue.

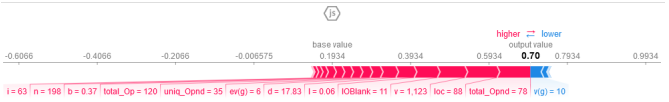


Figure 7.6 SHAP results for row 12 of KC2 dataset

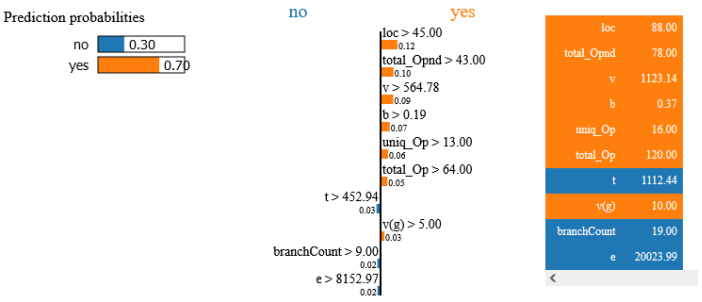


Figure 7.7 LIME results for row 12 of KC2 dataset

Each feature contributes to the model’s prediction performance with a specific SHAP value which may be positive or negative. Sum of the base prediction and all the features’ SHAP values constitutes the final model prediction performance which is called as ”output value”. The values near a feature mean that the feature has maximum contribution with these values. For example, when the value of ”total_Opnd” is equal to 78, the model has its best prediction performance.

We describe Fig. 7.7 in detail as follows:

- Prediction probability: refers to the probability of model prediction for an instance in case of ”yes” or ”no”. *”Yes/0.7”* means with a change of %70, the instance will be classified as ”yes”.

- Colors (orange or blue): features having positive correlations with target are shown in orange; otherwise, in blue.
- loc>45.00: High values of lines of code positively correlate with the existence of defects.
- t>452.92: High values of total t negatively correlate with the existence of defects.

The central plot in Fig. 7.7 shows the contributions of the top features to the prediction results corresponding to the weights of linear model which is called as coefficient values (e.g. coefficient value for ”loc” is 0.12; since it has the max weight, it is shown at the top of the plot). The leftmost plot shows each feature and its optimal value for contributing to model prediction result at the maximum level. For example, while the value of ”loc” is 88, the model has its best prediction performance. Based on these findings, we observed very close results for both SHAP and LIME methods, demonstrating the availability of these methods to interpret ML models with confidence.

Although the values are extracted using LIME methods for a single instance, for human understandability we write the rules using the original scales of the data.

Rule: *If ”line count of code” is GREATER THAN 45 AND ”total operands” GREATER THAN 43 AND ”halstead volume” is GREATER THAN 564.78 AND ”number of delivered bugs” is GREATER THAN 0.19 AND ”unique operators” is GREATER THAN 13.00 AND ”total operators” is GREATER THAN 64.00 AND ”McCabe cyclomatic complexity” is GREATER THAN 64.00 THEN the code is defectiveness.*

7.3. Results for RQ3. What is the effect of each feature on the prediction of the whole model? (Global prediction)

We firstly have begun by observing about permutation feature importance and partial dependence plots by RQ1, in order to have an overview of what the model has learned. After that, we have investigated for SHAP and LIME values with RQ2, in order to break

down the components of the individual predictions. Finally, with RQ3, we have widened SHAP values and proposed to see how aggregating many SHAP values could give more detailed alternatives to permutation feature importance and partial dependence plots. With permutation feature importance, we have observed which features are most important for the model. However, we could not get any insights about how a feature matters for the model with this approach. To enable this, we used SHAP summary plots as shown in Fig. 7.8.

From Fig. 7.8, we get much more detail about each feature's effect over the model prediction performance. While color shows each feature value as high or low, horizontal location shows the SHAP values about whether the effect of that value has caused a higher or lower prediction performance. According to the plot, we observe that higher values of total_Opnd and uniq_Opnd increase the model's prediction performance.

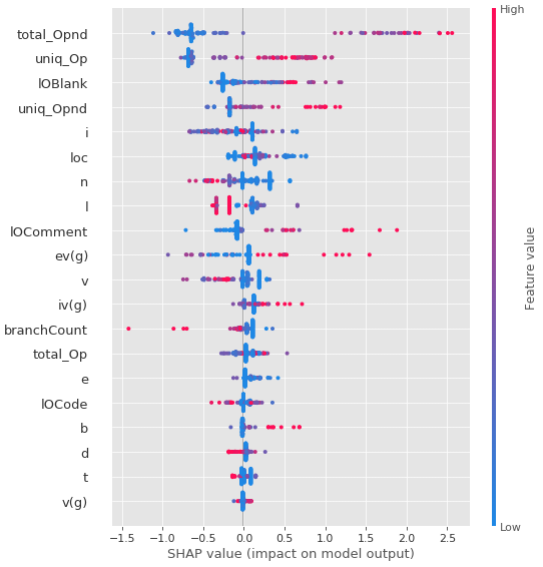


Figure 7.8 SHAP summary plot for the model

7.4. Comparing the GB feature important results with XAI methods used in this study

Since GB ML classifier has its own feature importance algorithm, we compare these feature importance results with XAI results. Fig. 7.9 represent the feature importance results for

GB ML classifier. If we go into detail with the Fig. 7.1, Fig. 7.8, and Fig. 7.9, we observe that the most important feature for all methods is total_Opnd. In addition, while the most important feature is the same for all methods, the other features' importance (ev(g), loc, etc.) are not in the same order, although their importances are very close with the results of GB classifier.

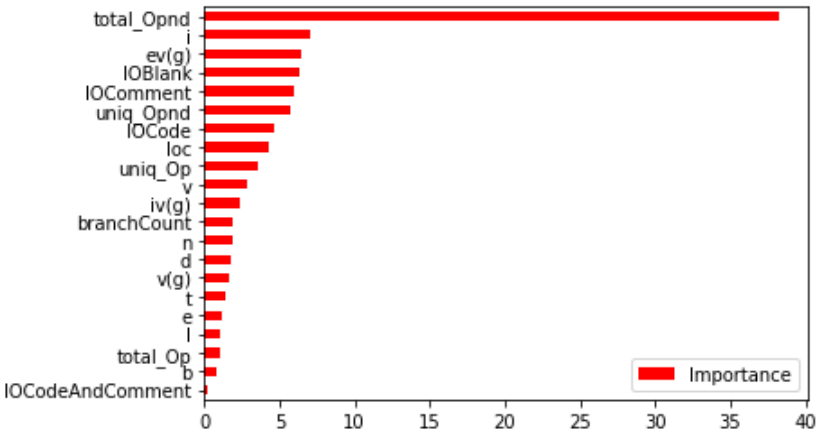


Figure 7.9 Feature importance for the GB model

8. RESULTS FOR CASE STUDY 2:

Explainable AI Framework For Software Defect Prediction

The proposed XAI-enabled framework was analyzed to enhance model interpretability and transparency. Different ML models were explored, including RF, GB, NB, and MLP. By offering insights into the decision-making process behind the models used for defect prediction, XAI can play a significant role in the prediction of software defects. In Figure 5.5, we present our proposed XAI-based framework for SDP. The actions that can be included in a XAI-based system for SDP are explained in the following paragraphs.

Data preparation: Data collection and cleaning are the steps in the preparation of the data needed to anticipate software defects. The data should contain details about the software, such as the amount of code, any prior flaws, and other pertinent information.

Model development: Creating an ML model to predict software defects is the task at hand in this step. Different methods, including decision trees, logistic regression, or deep neural networks, might be used to create the model. In this study, we categorize the ML models into 3 parts as distance-based [20], [19], neural network-based [16], and probabilistic-based ML models [15], [18], and analyze each of them individually.

Model evaluation: In this step, the model's performance is assessed using indicators including accuracy, precision, recall, and F1-score. It's crucial to make sure the model is reliable and precise enough to be applied for defect prediction.

XAI-based analysis: In this step, the model's decision-making process is examined using XAI approaches. This can give insight into the parameters that are most crucial for predicting defects as well as the logic behind the predictions made by the model. We analyze the XAI for SDP in three parts: "Model interpretability", "Model visualization and Inspection", and "Algorithm transparency and User trust". We used 6 different XAI methods to make analyses on the explainability of SDP which are SHAP, LIME, ELI5, Partial plot, Anchor, and Protodash.

Interpretation and visualization: In this step, the XAI analyses findings are interpreted and presented in an approachable manner. To aid users in understanding how the model generates its predictions, visuals like feature importance plots, summary plots, or heatmaps are created.

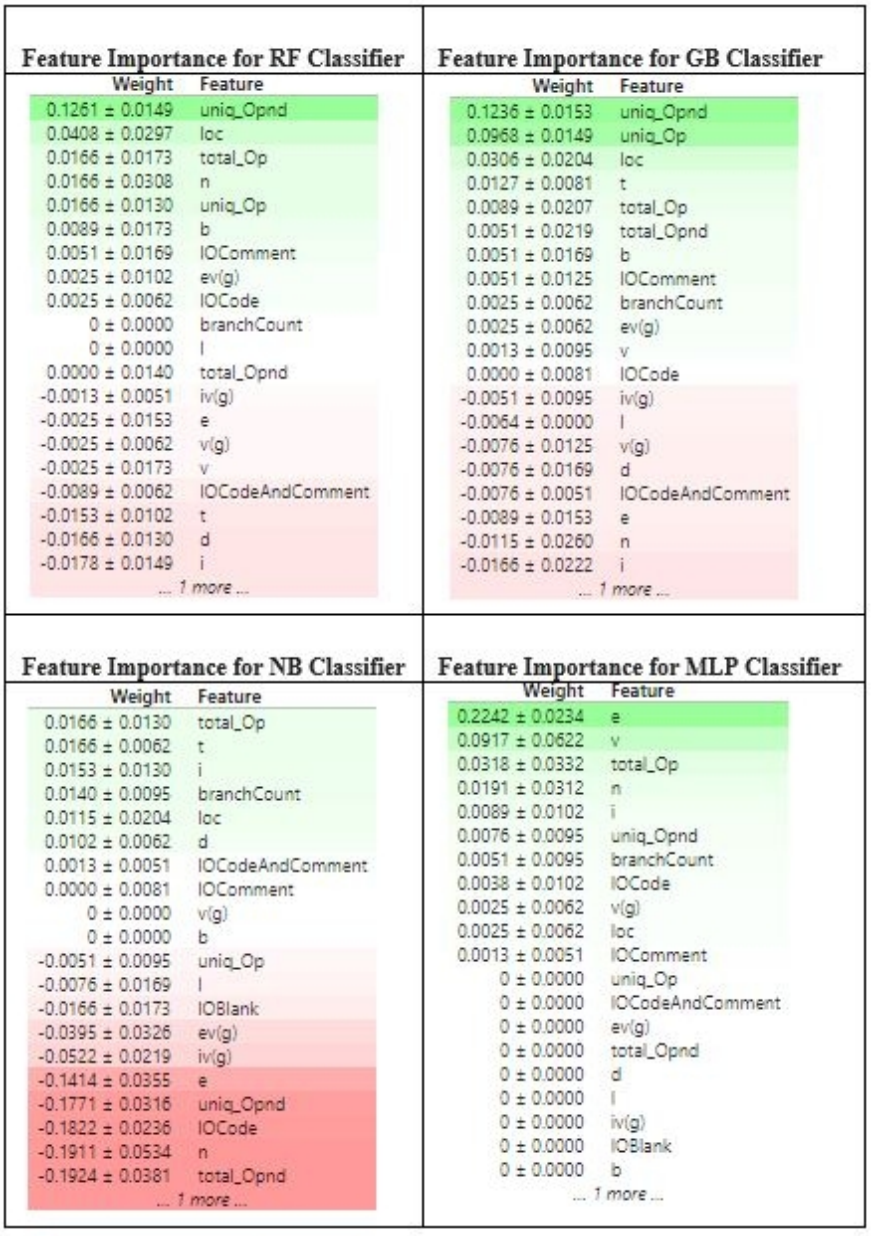
8.1. Global Explainability

Global explainability in XAI refers to the ability to provide an overarching and comprehensive explanation or understanding of how an AI or machine learning model makes decisions across its entire scope. It aims to answer questions about the model's behavior at a high level and reveal the key factors, features, or patterns that influence its decisions on a global scale.

By analyzing the global explainability of different ML models, we can identify the most influential features in the KC2 defect prediction dataset according to the RF, GB, NB, MLP, and NN models. Features with higher importance are more influential in the model's decision-making process and contribute more to the model's predictions.

- ***Feature Importance with ELI5:*** ELI5 provides a convenient way to visualize feature importances indicating the relative importance of each feature in the RF classifier's prediction. To explain and interpret feature importances using ELI5 on the KC2 defect prediction dataset with an ML classifier, we first need to load the dataset, train the ML model, and then use ELI5 to calculate and visualize the feature importances. According to Figure 9.1., we obtained the importance scores for each feature in the dataset of all 4 models. The higher the score, the more important the feature is for predicting the target variable (problems). The visualization displays a representation of the feature importances, allowing us to easily interpret and compare the importance of different features. The interpretation of feature importances depends on the specific dataset and context. It is essential to consider domain knowledge and the specific problem we are trying to solve when interpreting the feature importances. For KC2 dataset, we observed that while the "uniq_Opnd" is the most important feature for

both RF and GB classifiers, ”total_Opnd” is the most important one for NB and ”e” is the most important one for MLP classifier.



Feature Importance for RF Classifier

Weight	Feature
0.1261 ± 0.0149	uniq_Opnd
0.0408 ± 0.0297	loc
0.0166 ± 0.0173	total_Op
0.0166 ± 0.0308	n
0.0166 ± 0.0130	uniq_Op
0.0089 ± 0.0173	b
0.0051 ± 0.0169	IOComment
0.0025 ± 0.0102	ev(g)
0.0025 ± 0.0062	IOCode
0 ± 0.0000	branchCount
0 ± 0.0000	l
0.0000 ± 0.0140	total_Opnd
-0.0013 ± 0.0051	iv(g)
-0.0025 ± 0.0153	e
-0.0025 ± 0.0062	v(g)
-0.0025 ± 0.0173	v
-0.0089 ± 0.0062	IOCodeAndComment
-0.0153 ± 0.0102	t
-0.0166 ± 0.0130	d
-0.0178 ± 0.0149	i
… 1 more …	

Feature Importance for GB Classifier

Weight	Feature
0.1236 ± 0.0153	uniq_Opnd
0.0968 ± 0.0149	uniq_Op
0.0306 ± 0.0204	loc
0.0127 ± 0.0081	t
0.0089 ± 0.0207	total_Op
0.0051 ± 0.0219	total_Opnd
0.0051 ± 0.0169	b
0.0051 ± 0.0125	IOComment
0.0025 ± 0.0062	branchCount
0.0025 ± 0.0062	ev(g)
0.0013 ± 0.0095	v
0.0000 ± 0.0081	IOCode
-0.0051 ± 0.0095	iv(g)
-0.0064 ± 0.0000	l
-0.0076 ± 0.0125	v(g)
-0.0076 ± 0.0169	d
-0.0076 ± 0.0051	IOCodeAndComment
-0.0089 ± 0.0153	e
-0.0115 ± 0.0260	n
-0.0166 ± 0.0222	i
… 1 more …	

Feature Importance for NB Classifier

Weight	Feature
0.0166 ± 0.0130	total_Op
0.0166 ± 0.0062	t
0.0153 ± 0.0130	i
0.0140 ± 0.0095	branchCount
0.0115 ± 0.0204	loc
0.0102 ± 0.0062	d
0.0013 ± 0.0051	IOCodeAndComment
0.0000 ± 0.0081	IOComment
0 ± 0.0000	v(g)
0 ± 0.0000	b
-0.0051 ± 0.0095	uniq_Op
-0.0076 ± 0.0169	l
-0.0166 ± 0.0173	IOBlank
-0.0395 ± 0.0326	ev(g)
-0.0522 ± 0.0219	iv(g)
-0.1414 ± 0.0355	e
-0.1771 ± 0.0316	uniq_Opnd
-0.1822 ± 0.0236	IOCode
-0.1911 ± 0.0534	n
-0.1924 ± 0.0381	total_Opnd
… 1 more …	

Feature Importance for MLP Classifier

Weight	Feature
0.2242 ± 0.0234	e
0.0917 ± 0.0622	v
0.0318 ± 0.0332	total_Op
0.0191 ± 0.0312	n
0.0089 ± 0.0102	i
0.0076 ± 0.0095	uniq_Opnd
0.0051 ± 0.0095	branchCount
0.0038 ± 0.0102	IOCode
0.0025 ± 0.0062	v(g)
0.0025 ± 0.0062	loc
0.0013 ± 0.0051	IOComment
0 ± 0.0000	uniq_Op
0 ± 0.0000	IOCodeAndComment
0 ± 0.0000	ev(g)
0 ± 0.0000	total_Opnd
0 ± 0.0000	d
0 ± 0.0000	l
0 ± 0.0000	iv(g)
0 ± 0.0000	IOBlank
0 ± 0.0000	b
… 1 more …	

Figure 8.1 Feature importance weights’ for each ML model with ELI5

- ***Partial Dependence Plot:*** is used to visualize the relationship between a target variable and one or more predictor variables while holding other variables constant. In the context of the KC2 software defect prediction dataset, a PDP can help us understand

how the predicted defect rate is affected by different features or variables in the dataset. Interpreting a PDP involves examining the shape of the plot and the trends it reveals. Here are some key points to consider when interpreting a PDP for the KC2 dataset:

1. X-axis: The x-axis represents the values of the selected feature. It is important to note the range and distribution of the feature values.
2. Y-axis: The y-axis represents the partial dependence, which indicates the effect of the feature on the predicted defect rate. It represents the change in the predicted defect rate when the feature value varies while keeping other variables constant.
3. Main trend: It looks for any overall patterns or trends in the PDP. For example, if the PDP has an increasing or decreasing trend, it suggests that the feature has a significant influence on the defect rate.
4. Saturation points: It allows to observe if there are any saturation points where changing the feature value has little to no effect on the defect rate. This could suggest that the feature has a limited impact on the prediction once it reaches a certain threshold.

According to Figure 8.2, while there is an increasing trend for the ”uniq_Opnd” and ”b” features, which means that these features have an important effect on the model prediction performance, there is a saturation point for the ”total_Opnd” feature which means increasing the number after a certain threshold value has a small influence on the predicted defect rate. This imply that beyond a certain threshold, the ”total_Opnd” feature becomes less relevant in predicting defects. To conclude, by analyzing these aspects of the PDP, we can gain insights into how different features in the KC2 dataset influence the predicted defect rate. This information can help us identify important predictors and understand their impact on the software defect prediction task.

- ***Shap Summary Plot:*** To visualize and interpret a Shap summary plot for an ML model on the KC2 defect prediction dataset, we first loaded the KC2 defect prediction dataset, preprocessed the dataset by separating the features and the target variable (problems), splitted the dataset into training and testing sets, trained a ML classifier

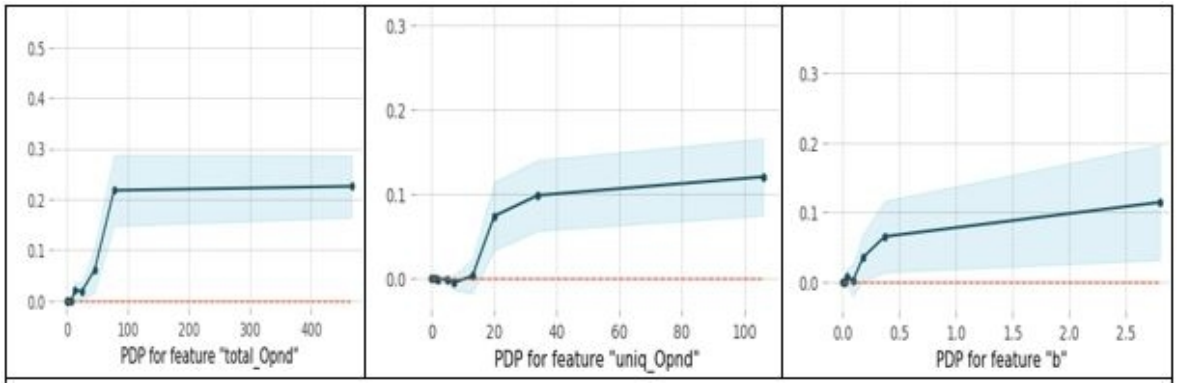


Figure 8.2 Partial dependence plot for ”total_Opnd”, ”uniq_Opnd”, and ”b” features, sequentially

on the training set, generated Shap values for the test set using the trained model, and finally created and interpreted the Shap summary plot.

The Shap summary plot will demonstrate how each feature affects the output of the model. The features will be arranged in ascending order of importance, starting with the most crucial ones. The impact of each feature is shown as a horizontal bar. Whether the feature has a high (red) or low (blue) value is indicated by the color of the bar. The position of the bar indicates whether the feature will have a favorable or negative influence.

We may determine which features have the biggest impact on the model's predictions by examining the Shap summary plot. Positive contributions are represented by red bars, while negative contributions are represented by blue bars. The influence of the associated feature increases as the bar length increases.

According to Fig. 8.3, while ”uniq_Opnd” feature has the highest contribution for the RF and GB models, total_Opnd and ”e” features are the most important features for the NB models and MLP models, sequentially. If we look at each plot, it is possible to say that RF and GB models give similar results. Since NB classifiers are probabilistic models and do not have a direct way to calculate feature importances, using SHAP values may not be straightforward. However, we can still generate

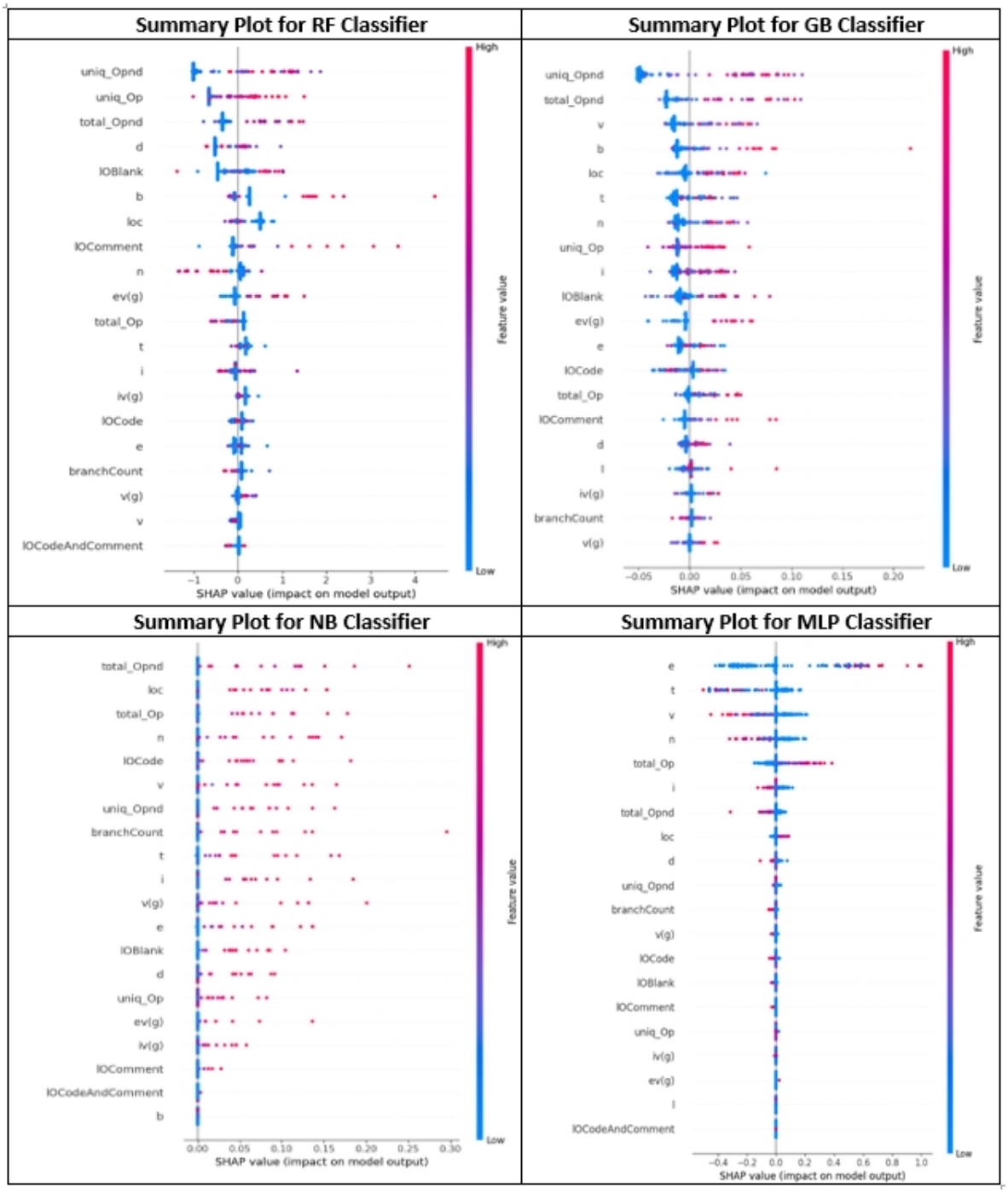


Figure 8.3 SHAP summary plot representation for RF, GB, NB, and MLP models

an approximate Shap summary plot by treating the NB model as a black box and utilizing Kernel SHAP. In this case, the Shap summary plot provides an overview of the estimated feature importances based on the approximate Shap values derived

from the Kernel SHAP explainer. The interpretation of feature importances from Kernel SHAP for an NB classifier may not be as accurate as for other models, it is vital to mention. The feature independence assumption made by NB can make it difficult to capture complicated relationships between features. As a result, the Shap values and accompanying summary plot might not accurately reflect the features' true underlying significance to the NB Classifier. In addition, Shap values for MLP Classifier models may not be as easily interpreted as with other models either. The interpretation of feature importance is more difficult in MLP models since they can contain a lot of hidden layers and are hence complex. Although more caution is advised when evaluating the data, the Shap summary plot can still shed light on the relative contributions of the features. Lastly, by considering the most important features for the ELI5 and SHAP summary plots, it is possible to say that both methods gave consistent result with each other by comparing different ML models' result.

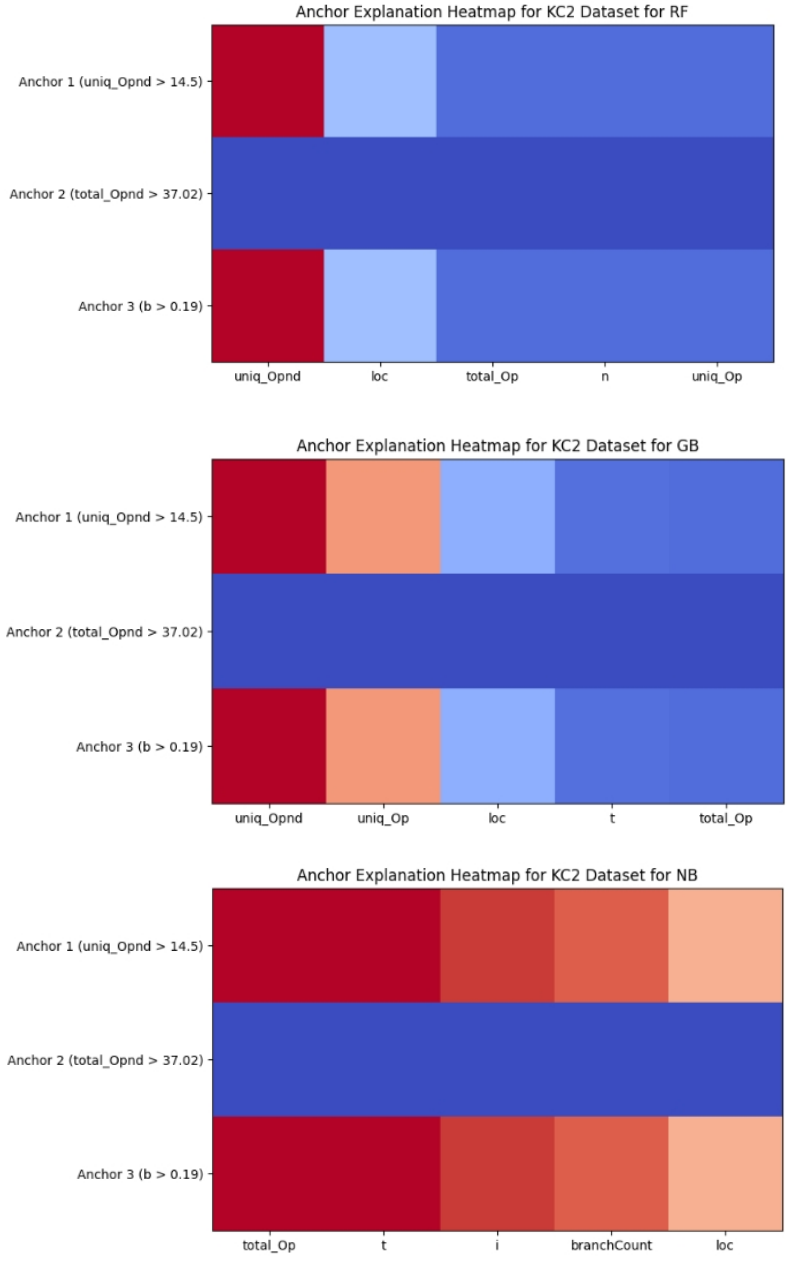
- ***Anchor Explanation Heatmap:*** The anchor explanation heatmap is a technique used for model interpretation and explanation in machine learning. It is a heatmap that shows the importance of features or input variables for a given prediction. In case of the KC2 defect prediction dataset, which contains software metrics and defect data for 522 software modules, an anchor explanation heatmap could be used to visualize which software metrics are most important for predicting software defects across the entire dataset. To generate an Anchor Explanation Heatmap for the KC2 dataset, we loaded the KC2 defect prediction dataset, separated the features and target variable (defective units), and defined three anchor (feature) conditions based on the mean values of the features. To generate an anchor explanation heatmap for the KC2 dataset, we used an interpretable machine learning algorithm or a model-agnostic interpretation technique, permutance importance with ELI5, to extract explanations for each prediction in the dataset. The importance scores for each software metric then were aggregated across the entire dataset to generate an anchor explanation heatmap. We also assigned importance values to each feature based on their relative importance in the anchor. We then created a matrix to represent the heatmap, filled it in with the

importance values, and plot the heatmap using matplotlib library [**?**].

Choosing the optimal anchor conditions for the KC2 defect prediction dataset depends on the specific features and characteristics of the dataset, as well as the goals of the interpretation. There are some considerations to help guide the selection of anchor conditions such as analyzing feature importance, considering domain knowledge or expert insights about the KC2 dataset (”Are there any specific features or conditions that are known to be strongly associated with defects in software projects?”), conducting exploratory data analysis on the KC2 dataset to understand the distribution and relationships between features, etc. In this study, we identified and used the three features that contribute significantly to the prediction of defects (total_Opnd, umiq_Opnd, b) as shown in Figure 8.2. These features are candidates for anchor conditions as they capture the key factors driving the predictions.

Interpreting the results of an Anchor Explanation Heatmap for the KC2 dataset requires understanding the heatmap visualization and the insights it provides. Here are some key points to consider when interpreting the results:

1. Heatmap Colors: The heatmap consists of colored cells, where each cell represents a feature-value combination. The color intensity indicates the importance of that feature-value combination in determining the prediction. Darker colors usually indicate higher importance.
2. Importance of Features: The heatmap allows you to assess the relative importance of different features in the prediction. Features with darker cells in the heatmap indicate their stronger influence on the prediction. Focus on these features to gain insights into the underlying patterns and relationships in the dataset.



In an anchor explanation heatmap visualization, different anchors represent different subsets of the data where the prediction is most reliable. Each anchor can be thought of as a ”sufficient condition” that must be true for the prediction to be valid. The importance of different anchors depends on the context and purpose of the analysis. In some cases, a single anchor might be the most important for understanding a particular prediction, while in other cases, multiple anchors might be necessary to fully understand the prediction across different subsets of the data. For example, in a

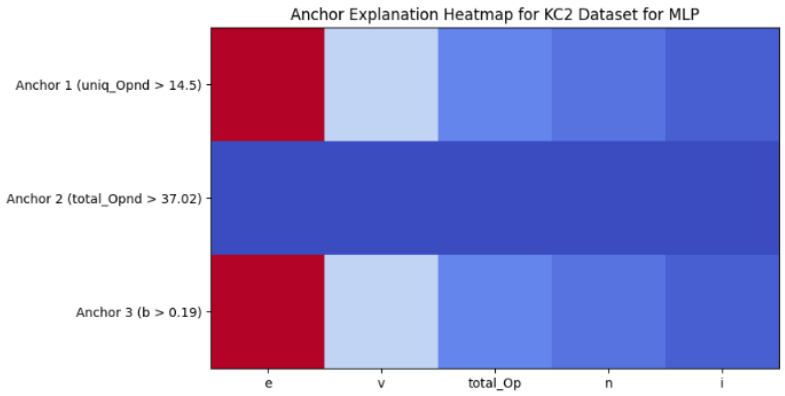


Figure 8.4 Anchor explanation heatmap for KC2 dataset

defect prediction model for software development, different anchors might represent different code quality issues or metrics that are associated with defects. One anchor might represent a particular combination of high code complexity and low code coverage, while another anchor might represent high coupling between different parts of the code.

The resulting heatmap will show the importance of each feature in the three anchors (uniq_Opnd>14.5, total_Opnd>37.02, b>0.19), with higher importance values shown in warmer colors (e.g., red) and lower importance values shown in cooler colors (e.g., blue). This can help users understand which features are most important in predicting defects in the KC2 dataset and can be used to identify areas where the model may need improvement. That is, by examining the heatmap and analyzing the patterns, decision rules, and importance of features, we can gain a deeper understanding of the KC2 dataset and the factors that contribute to the prediction of defects. These insights can help identify key features, potential risk factors, and areas for further investigation or improvement in software defect prediction. Interpreting different anchors in an anchor explanation heatmap involves understanding which features or input variables are most important for each anchor and how they are related to the prediction. For example, for anchor 1 (uniq_Opnd>14.5), the most important features are ”uniq_Opnd”, ”total_Opnd”, ”n”, ”uniq_Op”, and ”loc”, sequentially in case of RF classifier. For anchor 2, each feature is of equal importance. For anchor 3 (b>0.19),

the most important features show similar pattern with anchor 1 condition. This can help developers prioritize areas for improvement in their code or identify potential causes of defects across different subsets of the data. In fact, these results support the findings we obtained with the PDP shown in Figure 8.2. Because as seen in the PDP graph, the increase in the number of bugs (b) increased the defect prediction rate, and as seen in the anchor3 case, an increase in the importance value of the features was observed when the number of bugs (b) was greater than the threshold value of 0.19. Vice versa, as seen in the PDP plot, it was concluded that increasing the total_Opnd feature above a certain threshold value had no effect on the defect prediction of the model. Similarly, in the case of anchor2, we observed that all features are of equal importance, i.e. they have same and small effect on the defect prediction result.

Overall, the interpretation of different anchors in an anchor explanation heatmap depends on the specific context and purpose of the analysis, and requires an understanding of the features or input variables that are most important for each anchor.

- ***Feature clustering with SHAP:*** Up to now, we analyzed the importance of each feature on KC2 dataset using different explainability techniques. In this part, we propose to understand the importance and impact of different groups of features on the defect prediction by using feature clustering with SHAP method. The purpose of this method on the KC2 defect prediction dataset is to identify groups or clusters of features that exhibit similar patterns of importance in predicting defects. The goal is to gain insights into the relationships and interactions among the features, understand their relative importance, and potentially identify feature subsets that are more informative for defect prediction.

  By applying the shap feature clustering method, which combines hierarchical clustering with SHAP values, we clustered the features based on their individual

contributions to the model predictions. This allowed us to group features together that have similar effects on the prediction outcomes. The resulting feature clusters and their characteristics then be visualized and analyzed to identify meaningful patterns and relationships. We looked for similarities or differences among the features within each cluster, identified any patterns or relationships that emerged from the cluster analysis. Figure 8.5 shows the feature clusters using visualization techniques as a bar plot (Figure 8.5(a)) and, for clarity, as a dendrogram (Figure 8.5(b)). To draw meaningful conclusions about the clusters and their characteristics, we combined the bar plot and dendrogram interpretation. This provided us a visual representation of the feature clusters and their relationships.

According to Figure 8.5(a), ("total_Opnd" + 5 other features) are collectively most important for defect prediction. By looking at Figure 8.5(b), we can see these five features which are uniq_Opnd, b, total_Op, n, v in detail. By examining the clusters as shown in Figure 8.5(b), we can identify which features tend to have higher or lower importance values, providing insights into the relative significance of different sets of features. The vertical axis of the dendrogram represents the measure of dissimilarity or distance between the clusters being merged. The height at which two clusters merge indicates how dissimilar or similar they are. Lower merge heights suggest higher similarity, while higher merge heights indicate greater dissimilarity. Clusters that merge at lower heights on the dendrogram are more similar to each other than those that merge at higher heights. Closer proximity of branches indicates a higher level of similarity, while greater distance indicates a larger dissimilarity. For example, "l"+"d" features are much more similar with each other comparing with "IOComment" feature. By using feature clustering, we also can perform subset identification of features that exhibit similar patterns of importance. For example, ("total_Opnd"+ 5 other features) is one subset, ("loc"+"e"+"t"+"IOCode") is another subset. This can be useful for feature selection, as it allows you to potentially identify smaller subsets of features that retain the predictive power of the entire feature set. These subsets can

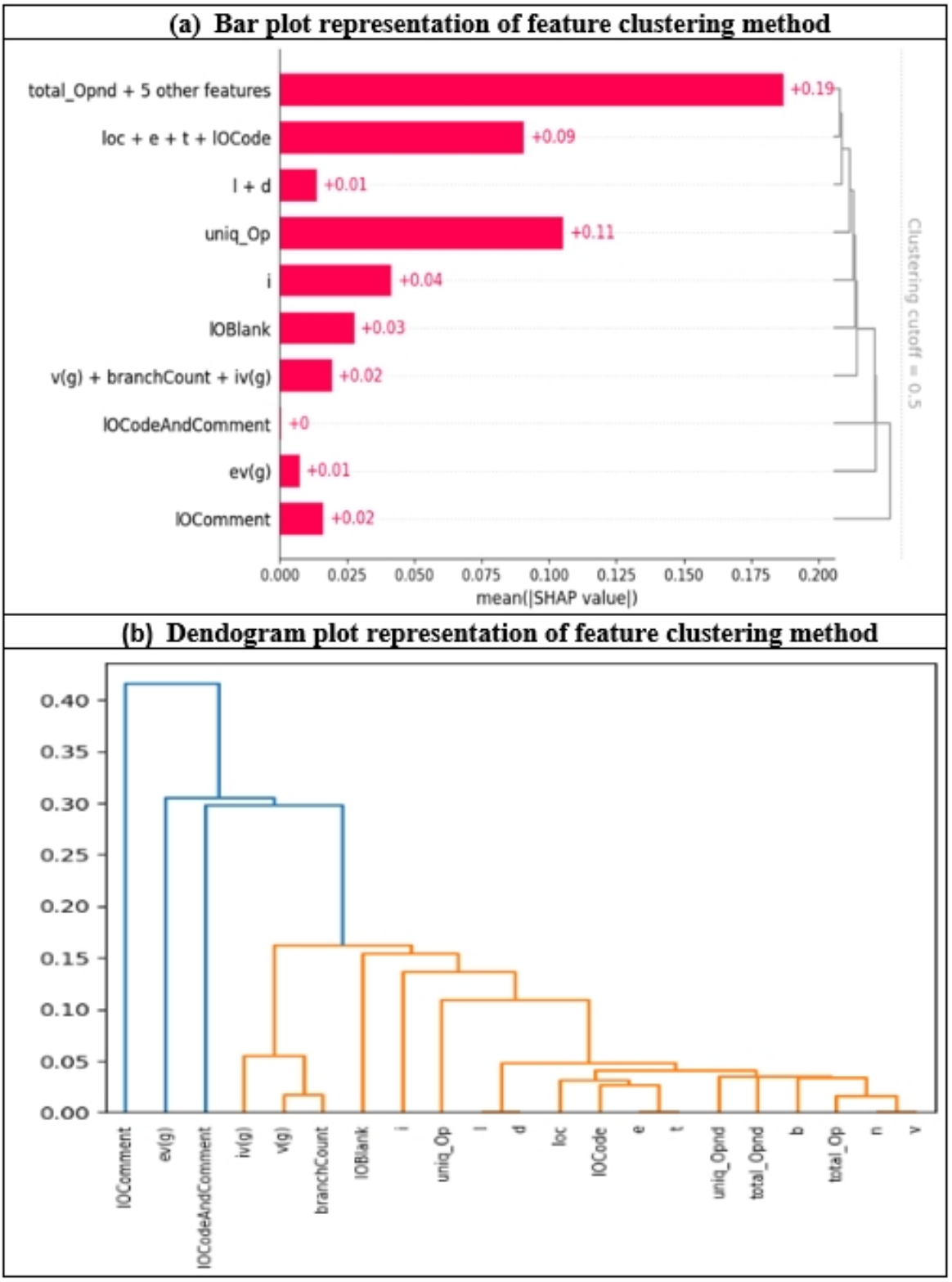


Figure 8.5 Feature clustering representation as bar plot and dendogram plot for KC2 dataset

help simplify the model and improve interpretability without sacrificing performance. In addition, understanding feature clusters can provide guidance for decision-making in defect prediction tasks. For example, if certain clusters consistently exhibit high importance, they can be prioritized for further investigation or targeted for specific interventions to mitigate defects.

Overall, the purpose of feature clustering with SHAP method on the KC2 defect prediction dataset was to gain insights into the structure and relationships among the features and their importance in predicting defects. This analysis can aid in feature selection, model interpretation, and decision support for defect prediction tasks. This interpretation can help understand the importance and impact of different groups of features on the defect prediction.

8.2. Local Explainability

Local explainability in XAI refers to the ability to provide an explanation or insight into how a specific AI or machine learning model arrived at a particular decision or prediction for an individual data point or instance. It aims to answer questions like ”Why did the model make this prediction for this specific case?” by focusing on the factors, features, or patterns that influenced that particular decision.

- ***Shap Waterfall Plot:*** To communicate the reasoning behind the prediction to stakeholders, clients, or end-users who may not have a technical background, the waterfall plot provides a visual and intuitive explanation of the model's decision for a specific instance. The plot provides a clear narrative of the decision-making process, allowing you to follow the sequence of feature contributions. It helps answer questions like ”Why did the model make this prediction?” and ”Which features were the most influential in the decision?”, etc.

Figure 8.6 represent the waterfall visualization for a specific instance (instance 0) on KC2 dataset. Interpreting a SHAP waterfall plot for local explainability involves understanding the contribution of each feature and how they collectively influence the model's prediction for a specific instance. Here's a step-by-step guide to interpreting a SHAP waterfall plot:

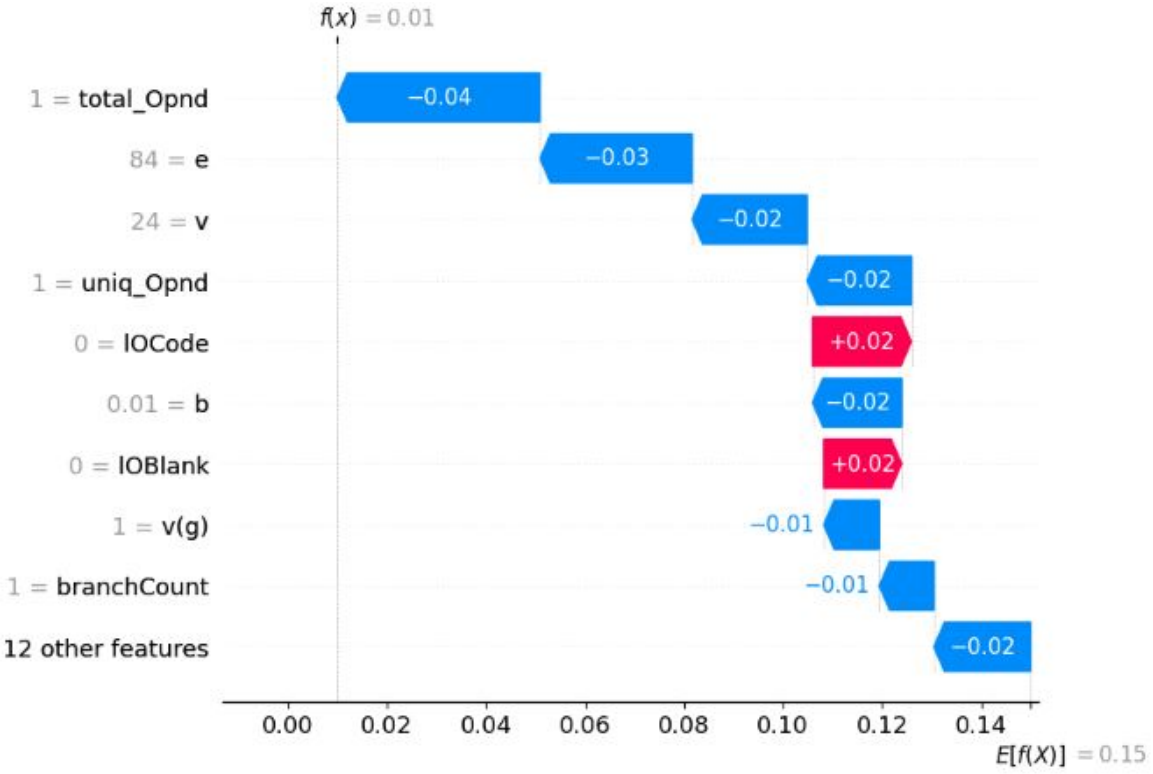


Figure 8.6 Waterfall plot for the first instance of KC2 dataset with RF classifier

1. Start with the base value: The SHAP waterfall plot typically begins with the model's base value, representing the average prediction for the entire dataset. This serves as the starting point for the explanation. The base value represents the expected model output when no specific features are observed or known. It serves as a reference point for understanding the contributions of individual features to the prediction. For this instance, base value is E[f(x)]=0.15. This indicates that the expected model output, on average, is 0.15 when no features are considered. While x-axis represent the prediction effect value, y-axis represents the value of each feature for a specific instance (instance 0) on KC2 dataset.

2. Observe the first feature contribution: The first horizontal bar in the plot represents the contribution of the first feature. It shows the impact of that feature on the prediction compared to the base value. The height and direction of the bar indicates the magnitude and direction of the feature's effect. The first feature in the plot is the "total_Opnd" and it affects the model prediction negatively (-0.04) as blue bars indicate the negative effect and red bars indicate the positive effect on the model prediction. Positive contributions indicate that the feature increases the prediction, while negative contributions indicate a decrease in the prediction.
3. Follow the subsequent feature contributions: Move along the plot, observing each subsequent horizontal bar. Each bar represents the contribution of a specific feature and shows how it influences the prediction relative to the previous step. The height and direction of the bars indicate the individual feature effects. While "total_Opnd" is the first feature that contributes to the prediction most (0.04), "e"=0.03, "v"=0.02, "uniq_Opnd"=0.02, etc. are the other features that contributes to prediction, sequentially.
4. Cumulative effect: As we progress through the plot, the cumulative effect of all the feature contributions becomes apparent. The cumulative sum of the horizontal bars represents the final prediction for the instance. By adding up the contributions (0.15 - 0.02-0.01-0.01+0.02-0.02+0.02-0.02-0.02-0.03-0.04), we can see how the model arrived at the specific prediction for that instance f(x)=0.01 (There may be small biases due to the rounding).

To conclude, the plot visually represents how each feature affects the prediction. It shows the direction and magnitude of the effect, indicating whether a feature increases or decreases the prediction. This information helps in understanding the specific role of each feature in the model's decision for that instance. For complex models with numerous features and intricate interactions, the waterfall plot simplifies the explanation process. It breaks down the prediction into manageable steps, making it easier to comprehend the contribution of each feature.

- ***Shap Force Plot:*** This visualization technique serves as a tool for local explainability in machine learning models. It aids in comprehending the contribution of individual features to determine the output or prediction of the model, specifically for a given instance, denoted as instance 0.

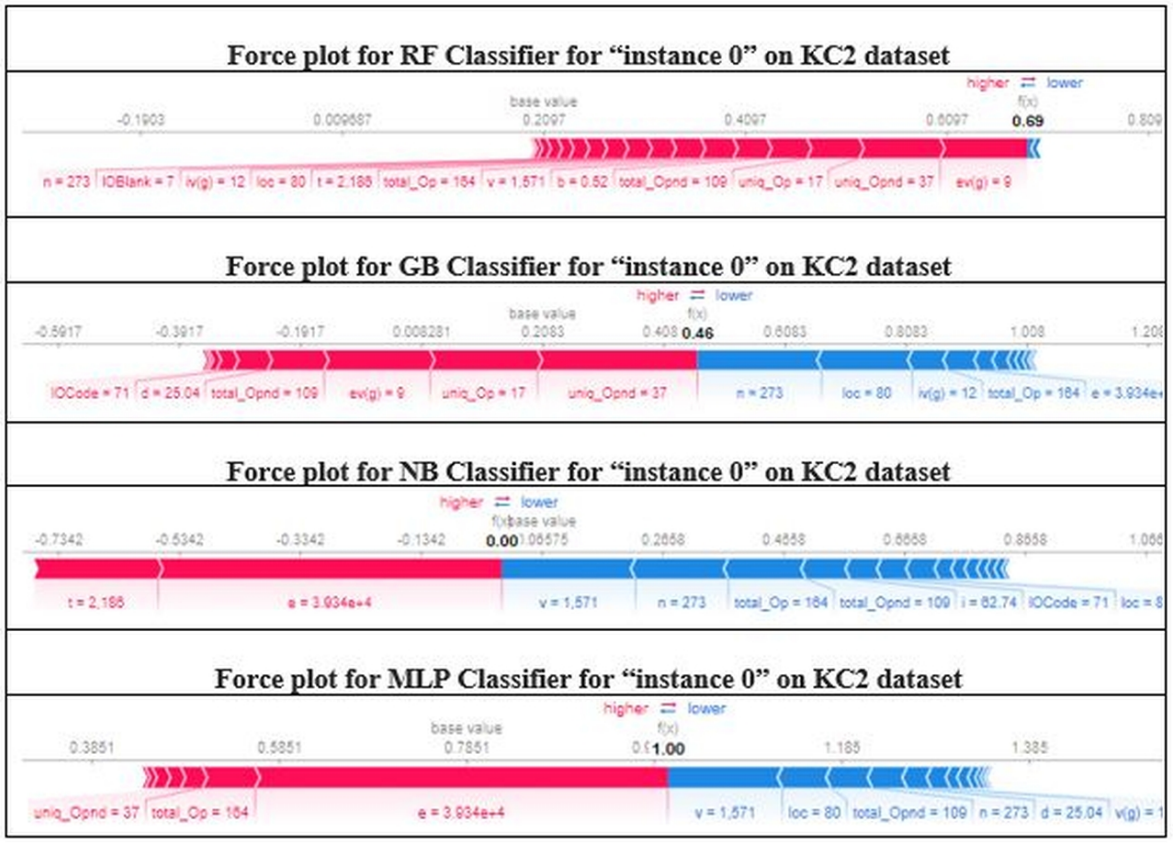


Figure 8.7 Force plot for a single instance (instance 0) on KC2 dataset

To interpret a SHAP force plot visualization on the KC2 defect prediction dataset, we first need to understand the dataset and the purpose of the defect prediction task. Figure 8.7 represents SHAP force plot that provides insights into the contribution of individual features (metrics in this case) to the prediction of whether a module is defective or not. Each feature is represented by a horizontal bar in the plot, and the length and color of the bar indicate its impact on the prediction. Here's how we can interpret the plot:

1. Positive and Negative Contributions: In a SHAP force plot, the bars can extend both to the left (negative contribution-blue bar) and to the right (positive

contribution-red bar) of the plot's center line. The direction indicates whether the feature value is pushing the prediction towards a defective or non-defective outcome. Positive values contribute to a higher likelihood of being defective, while negative values contribute to a lower likelihood. For example, for the force plot for GB classifier, while "uniq_Opnd" feature has positive values and contributes to the prediction as non-defective, "loc" feature contributes negatively and pushes the prediction towards defective prediction.

2. Length of the Bars: The length of the bars represents the magnitude of the feature's contribution. Longer bars indicate greater importance or influence of that feature on the prediction. By comparing the lengths of the bars, we can identify the most influential features in the defect prediction task. These are the metrics that have the most significant impact on determining whether a module is defective or not. If we look at Figure 8.7 for the GB classifier, we can say that "uniq_Opnd" feature contributes the most to the model prediction.

3. Overall Prediction: The SHAP force plot also displays an overall base value, often represented by a horizontal dashed line. This base value indicates the average prediction of the model without considering any specific feature values. The cumulative effect of the feature contributions determines the final prediction, which can be calculated by summing up the lengths of the bars. For example, let us examine the force plot for RF classifier. While base value is 0.21, the final prediction that means the cumulative effect of the all features' contribution for the specific instance (instance 0) is f(x)=0.69.

To conclude, SHAP force plots offer transparent explanations by showing how each feature contributes to the final prediction. This transparency is crucial, especially when the model's predictions have high stakes or need to be justified to end users. By displaying the contributions visually, end users can gain a deeper understanding of how their input affects the model's output. In addition, SHAP force plots can assist in error analysis by highlighting the features that contributed the most to incorrect predictions. When a model makes a mistake, understanding the reasons behind

it becomes crucial for diagnosing and rectifying potential issues. End users can leverage SHAP force plots to identify which features might be causing errors and take appropriate actions to address those shortcomings.

- ***LIME:*** It can be used to provide interpretability to machine learning models trained on the KC2 defect prediction dataset. When applying to the KC2 defect prediction dataset, LIME can provide local explanations for individual predictions made by the model. By using LIME on the KC2 dataset, we can gain insights into the important features that contribute to the predictions of defects. LIME assigns importance weights to different software features, indicating their relative influence on predictions. By visualizing these feature importances, we can identify which metrics have a significant impact on defect predictions.

  When using the ”show_in_notebook” method of the LIME explanation object to visualize explanations in a notebook environment for the KC2 defect prediction dataset, it generates a visual representation of the local explanation for a specific instance as shown in Figure 8.8. The visualization provides insights into the features that contribute to the model's prediction for that instance.

  The ”show_in_notebook” method typically generates a table (right side of the figure) and a bar chart (left side of the figure) to represent the feature importances. The table displays the features along with their corresponding contributions to the prediction, while the bar chart visually represents the magnitudes of these contributions. Interpreting the LIME visualization on the KC2 defect prediction dataset involves understanding the information presented in the table and the bar chart. Here's how we can interpret them:

  1. Table
     - Features: The table lists the features (software metrics) used in the dataset.

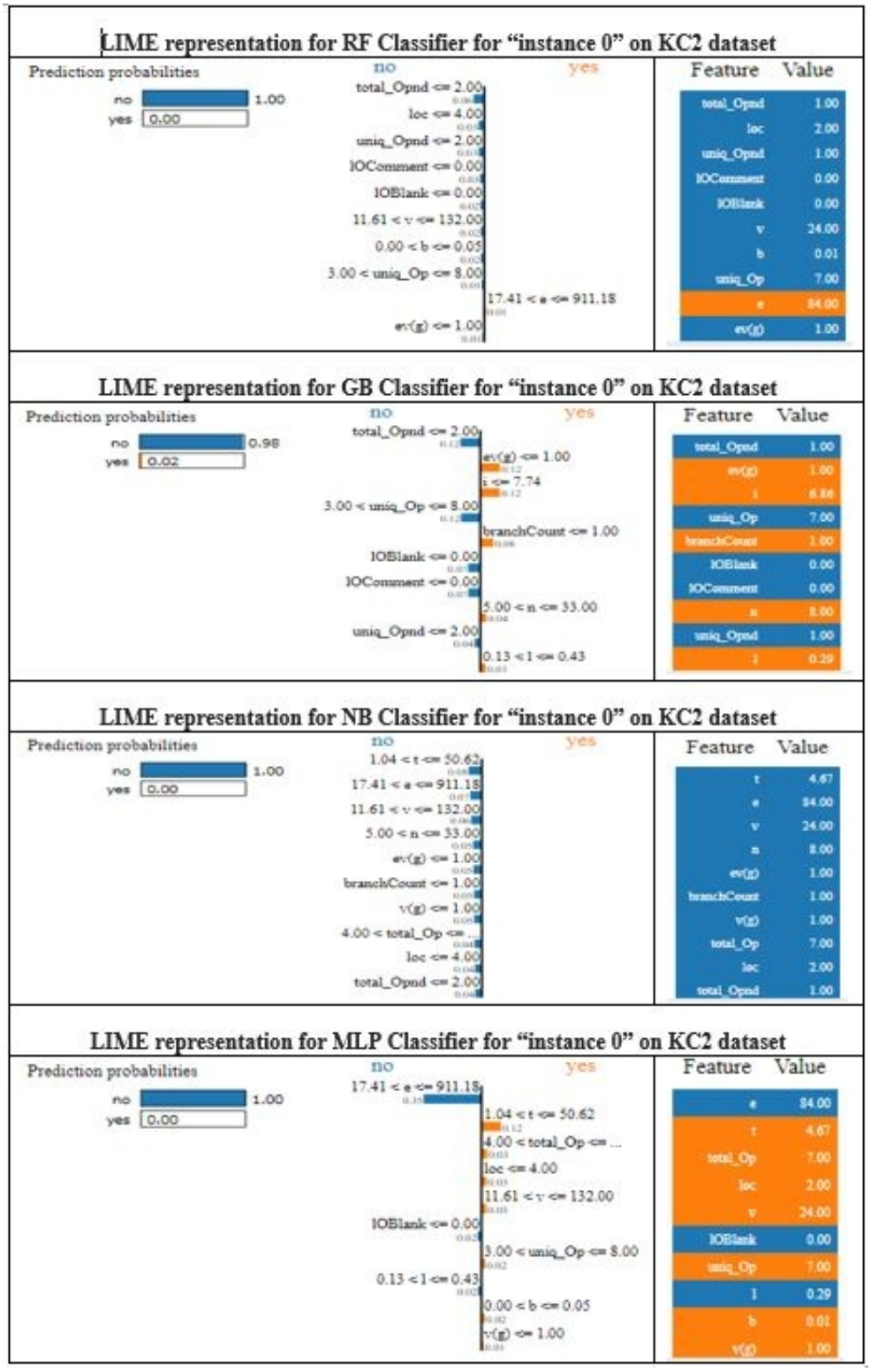


Figure 8.8 LIME representation for a single instance (instance 0) on KC2 dataset

- Contributions: Each feature has an associated contribution value, indicating its influence on the prediction. Positive contributions (orange ones) suggest that higher values of the feature contribute to the prediction of a defective module, while negative contributions (blue ones) suggest the opposite. For example, if we look at Figure 8.8 by considering RF classifier, while higher value of total_Opnd, loc, uniq_Opnd, and the other features shown as blue contribute to the model prediction as non-defective, higher value of feature ”e” contributes to the model prediction as defective.
- Importance: The table include additional information, such as feature importance degree, which can help you understand the relative significance of each feature. The most important feature is given at top, and from top to bottom, feature importance degree decreases. If we look at Figure 8.8 for the RF classifier (at the top of the Figure), we can say that the most important features are ”total_Opnd”, ”loc”, and ”uniq_Opnd” features, sequentially. From the table, we also can see the specific value for the specific instance that means the value of ”total_Opnd” for that instance (instance 0) is 1.00 and, increasing the value of this feature contributes to the prediction of a defective module.

2. Bar Chart

- Feature Importance: The bar chart visually represents the feature importance’s. Each bar corresponds to a feature, and its length indicates the magnitude of its contribution. Longer bars represent more influential features. Near the bars, there are coefficient values for each corresponding feature, which represent the importance level the features. If we look at Figure 8.8 for the RF classifier (at the top of the figure), we can say that the most important features are ”total_Opnd”, ”loc”, and ”uniq_Opnd” by considering the coefficient values as 0.06, 0.03, and 0.03, sequentially.
- Positive and Negative Contributions: The bars can be colored differently to represent positive and negative contributions. Positive contributions may be

displayed in one color (e.g., orange) while negative contributions in another color (e.g., blue). This coloring helps differentiate features that increase or decrease the likelihood of a defect.

By analyzing the table and bar chart, we can identify the specific features that have a significant impact on the prediction of defects in the KC2 dataset. The feature contributions highlight the direction and magnitude of their influence, providing insights into why the model made a particular prediction for the instance being explained. It's also important to note that interpretation may vary depending on the specific context of the dataset and the model used. According to the result of RF and GB, while "total_Opnd" is found as the most influential feature for the model prediction, "e" and "t" features are most influential features for NB and MLP models.

Overall, interpreting the LIME visualization allows end-users to gain a better understanding of the factors influencing the model's predictions, identify important software metrics for defect occurrence, and make informed decisions regarding software quality and defect prevention efforts.

- ***Protodash method:*** In the specific case of applying the Protodash method to the KC2 defect prediction dataset, the purpose would be to gain insights and understanding about the characteristics of software modules that are prone to defects. The Protodash method, when combined with domain expertise, can help identify the most representative modules and uncover patterns or features that contribute to defect-proneness. We now demonstrate how to produce explanations by choosing similar or prototypical modules to a developer in question that a domain expert could find interesting. This could make it easier for the domain expert to comprehend whether a software module is defective or not in light of other similar software modules. One should be aware that the chosen prototype modules are a part of the training set that was used to train a ML model that predicts whether these modules are

defective or not. The method also computes weights for each prototype illustrating how similar it is to the relevant software module(s).

In this part, we will examine one instance of getting prototypes—one for a case where the software module was defective (problems=1). In that situation, we present the top five prototypes from the training data and the similarity of their feature values. To apply the Protodash method to the KC2 defect prediction dataset using a neural network (NN), we firstly preprocessed the dataset. This involves handling missing values, normalizing numerical features, encoding categorical variables, and splitting the dataset into training and testing sets. Then, we build and train an NN model using the training set of the KC2 dataset.

Case 1- Obtain feature representations as explanations for a software module predicted as "1": We used the trained NN model to obtain similar features for each module in the dataset. We considered a sofware module (id=8) whose label was defective (1) as shown in Figure 8.9.

Case 2- Find top five similar prototypes: After selecting the module, we found similar modules predicted as "1" using the protodash explainer. Then, in Figure 8.10, we showed related software modules and the degree to which they have similarities with the selected module, as shown in the final row of the table below named as "Weight".

Once the prototypes are selected, it is better to analyze and interpret them in collaboration with domain experts. Examining the commonalities and differences among the prototypes, identifying the critical features that contribute to defect-proneness, and gaining insights into the characteristics of the modules prone to defects are the steps to be followed. This step requires domain expertise and qualitative analysis to provide meaningful interpretations of the selected prototypes.

loc	33.00
v(g)	4.00
ev(g)	3.00
iv(g)	4.00
n	84.00
v	408.07
l	0.11
d	8.86
i	46.05
e	3615.96
b	0.14
t	200.89
lOCode	24.00
lOComment	3.00
lOBlank	3.00
lOCodeAndComment	1.00
uniq_Op	11.00
uniq_Opnd	18.00
total_Op	55.00
total_Opnd	29.00
branchCount	7.00
problems	1.00

Figure 8.9 Feature−value information for the specified instance (id=8) on KC2 dataset

Case 3: Calculate how closely a feature of a prototypical sample looks similar to that of the selected sample. The closer a prototype sample's weight is to 1, the more closely it resembles the chosen sample in that attribute. In Figure 8.11, we can examine how many features of prototypes resemble those of the selected sample. Following that, a clear explanation is given.

The five prototypical samples that are most similar to the chosen sample are shown in Figure 8.10. According to the importances of the weights that the method outputs, the prototype listed under column 0 is by far the most typical sample (weight=0.59). This is verified intuitively by the feature similarity table in Figure 8.11, which shows that 17 out of the 21 features in this prototype are similar with %80 degree to those of the

	0	1	2	3	4
loc	40.000000	64.00000	5.000000	94.000000	13.0000
v(g)	4.000000	4.00000	1.000000	20.000000	2.0000
ev(g)	1.000000	1.00000	1.000000	17.000000	1.0000
iv(g)	4.000000	3.00000	1.000000	15.000000	2.0000
n	102.000000	92.00000	4.000000	279.000000	33.0000
v	495.510000	464.08000	6.340000	1613.000000	134.8900
l	0.090000	0.14000	1.330000	0.050000	0.1000
d	11.610000	7.13000	0.750000	20.720000	10.0000
i	42.680000	65.12000	8.450000	77.860000	13.4900
e	5753.470000	3306.60000	4.750000	33418.040000	1348.8600
b	0.170000	0.15000	0.000000	0.540000	0.0400
t	319.640000	183.70000	0.260000	1856.560000	74.9400
lOCode	29.000000	28.00000	3.000000	75.000000	7.0000
lOComment	1.000000	28.00000	0.000000	7.000000	1.0000
lOBlank	7.000000	5.00000	0.000000	8.000000	1.0000
lOCodeAndComment	1.000000	0.00000	0.000000	0.000000	2.0000
uniq_Op	11.000000	9.00000	1.000000	16.000000	10.0000
uniq_Opnd	18.000000	24.00000	2.000000	39.000000	7.0000
total_Op	64.000000	54.00000	1.000000	178.000000	19.0000
total_Opnd	38.000000	38.00000	3.000000	101.000000	14.0000
branchCount	7.000000	7.00000	1.000000	39.000000	3.0000
22	1.000000	1.00000	1.000000	1.000000	1.0000
Weight	0.597128	0.06991	0.022418	0.111145	0.1994

Figure 8.10 Feature−value information for the top five similar samples and similarity weights with the chosen sample on KC2 dataset

	0	1	2	3	4
loc	0.81	0.39	0.43	0.16	0.54
v(g)	1.00	1.00	0.65	0.10	0.75
ev(g)	0.73	0.73	0.73	0.11	0.73
iv(g)	1.00	0.82	0.56	0.12	0.68
n	0.83	0.92	0.43	0.13	0.59
v	0.86	0.91	0.49	0.12	0.62
l	0.96	0.94	0.08	0.89	0.98
d	0.66	0.77	0.29	0.16	0.84
i	0.88	0.50	0.25	0.31	0.30
e	0.84	0.98	0.75	0.09	0.83
b	0.85	0.95	0.48	0.12	0.59
t	0.84	0.98	0.75	0.09	0.83
lOCode	0.82	0.86	0.44	0.14	0.51
lOComment	0.83	0.09	0.75	0.69	0.83
lOBlank	0.29	0.53	0.39	0.21	0.53
lOCodeAndComment	1.00	0.29	0.29	0.29	0.29
uniq_Op	1.00	0.66	0.13	0.36	0.81
uniq_Opnd	1.00	0.63	0.29	0.20	0.43
total_Op	0.86	0.98	0.42	0.14	0.56
total_Opnd	0.77	0.77	0.47	0.12	0.64
branchCount	1.00	1.00	0.65	0.10	0.75

Figure 8.11 Feature weights similarity degree of top five similar samples with the chosen sample on KC2 dataset

chosen sample for which we are attempting to explain the prediction. Additionally, the domain expert believes that the selected module is a member of a group of defective modules who have virtually number of bugs (feature ”b”) after examining the prototypical samples and their features. The domain expert might feel more comfortable labeling the software modules as defective after hearing this explanation.

To conclude, the findings from the XAI-enabled framework analysis were interpreted and visualized using techniques such as feature importance plots, SHAP summary plots, and LIME explanations. These visualizations enhanced the understanding of the models' decision-making processes and facilitated informed decision-making by stakeholders.

9. RESULTS FOR CASE STUDY 3:

Applying XAI Methods for Feature Selection and Outlier Detection in Software Defect Prediction

In this case study, the effectiveness of ELI5, SHAP, and LIME in facilitating feature selection was assessed. The feature importance scores generated by each method were compared, and their impact on model interpretability and predictive performance was analyzed. In addition, the ability of the XAI methods to identify outliers in the dataset was evaluated. Metrics such as precision, recall, and F1-score were computed to assess the accuracy of outlier detection and the robustness of the models against anomalies. ELI5 and SHAP provided valuable insights into the relative importance of features and their contributions to model predictions. These explanations enhanced the interpretability of the models and facilitated a deeper understanding of the underlying data patterns. LIME was utilized for outlier detection, allowing for the identification of instances that deviated significantly from the norm. The explanations provided by LIME helped understand the reasons behind outlier classifications and highlighted potential data quality issues. Here is the flow diagram of the whole process applied in this study as shown in Figure 9.1. For this purpose, we used PC1 defect prediction dataset to demonstrate the execution of the process flow.

9.1. ELI5 for Global Feature Selection

ELI5 is a XAI method that facilitates feature selection and provides explanations for machine learning models. While ELI5 does not offer a direct feature selection method, it provides insights into feature importance at a global level. It can be used to interpret models and calculate feature importances based on coefficients (for linear models), permutation importance (by shuffling feature values and observing the impact on model performance), and tree-based model feature importance. We firstly propose to find the most important features in PC1 dataset by using ELI5 as shown in Figure 9.2.

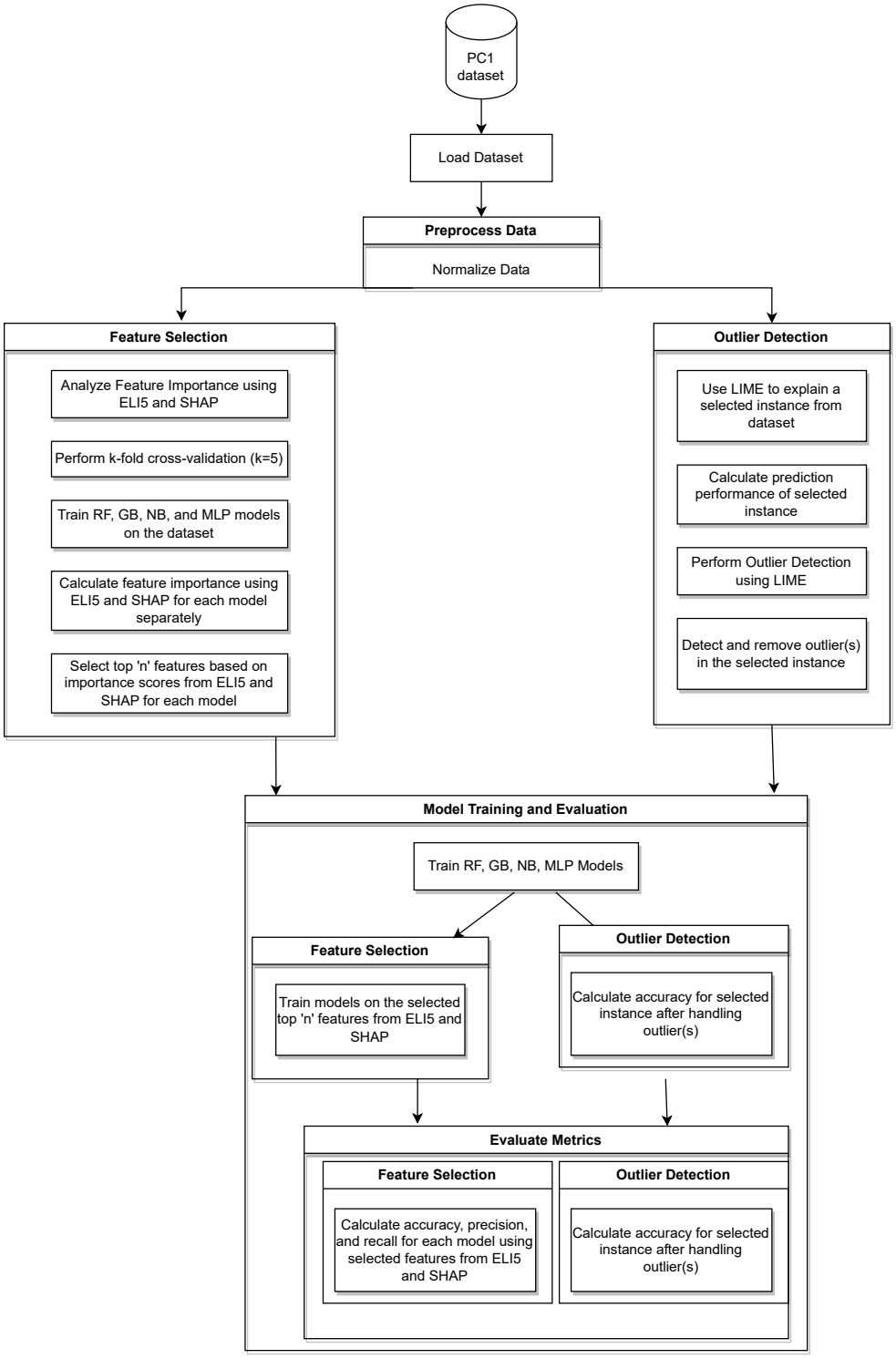


Figure 9.1 Flow diagram of the XAI-enabled SDP process followed in this study

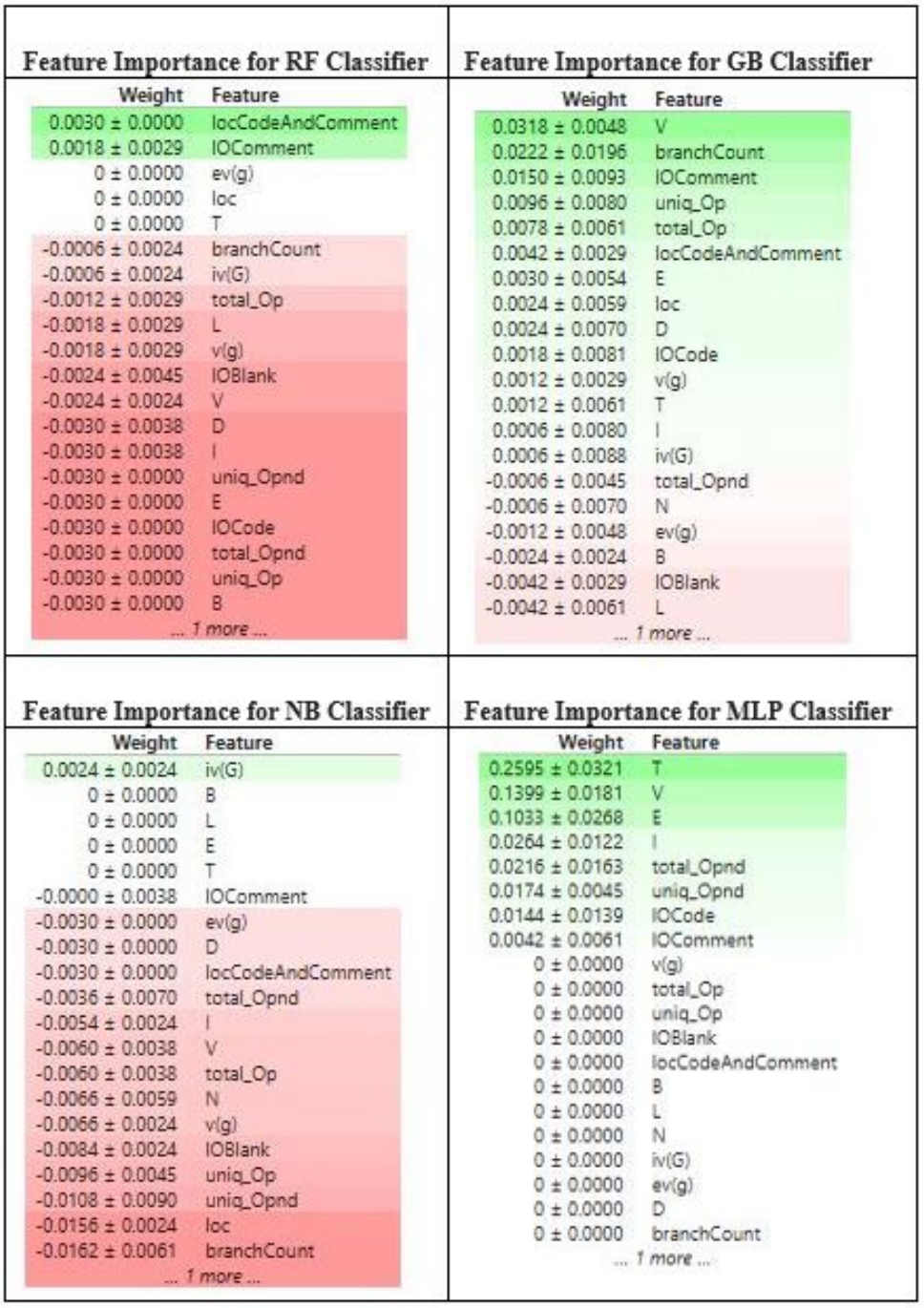


Figure 9.2 Feature Importance results using ELI5 for ML models

By considering findings from Figure 9.2, we selected the most important features as shown in Table 9.1 and calculated performance metrics (accuracy, precision, recall, f-score) for each model separately.

Table 9.1 Selected features with ELI5 for each model

ML model	Selected Features
RF	'lOComment', 'locCodeAndComment'
GB	'V', 'branchCount', 'lOComment', 'uniq_Op', 'total_Op', 'lOCode', 'D', 'loc', 'E', 'locCodeAndComment', 'total_Opnd'
NB	'iv(G)', 'lOComment'
MLP	'T','V','E','I','total_Opnd','uniq_Opnd','lOCode','lOComment'

Table 9.2 Performance metric results with ELI5 for all ML models

ML model	Accuracy	Recall	Precision
Cetiner et al. (RF) [1]	0.837	0.60	0.24
Our Study (RF with ELI5)	0.922	0.92	0.90
Cetiner et al. (GB)	0.831	0.57	0.17
Our Study (GB with ELI5)	0.916	0.91	0.88
Cetiner et al. (NB)	0.810	0.43	0.24
Our Study (NB with ELI5)	0.868	0.91	0.88
Cetiner et al. (MLP)	0.823	0.00	0.00
Our Study (MLP with ELI5)	0.919	0.91	0.88

We firstly loaded the PC1 dataset and selected the specific attributes (Table 9.1) for each model, splitted the data into training and testing sets, normalized the data using "StandardScaler". Then, we initialized each model classifier, separately, conducted k-fold cross-validation to evaluate the model's performance (Since the study we selected to compare the our result use k=5, we also selected k=5). Finally, we fit each model on the training set and evaluated their performance metrics on the test set.

In Table 9.2, we showed each performance metric based on their average value for each model and compared the result with the study of Cetiner et al. [1]. Since they used average value of accuracy, recall, and precision metrics, in order to reduce bias while comparing the studies with each other, we also used average values.

The accuracy results of the algorithms used for the PC1 dataset with feature selection with ELI5 are shown in Figure 9.3. Cetiner et al. [1] used Principal Component Analysis (PCA) method for feature selection, and after performing this method, the number of features were reduced to 15 from 21 features. Then, they compared the results (accuracy, recall, f-score) for ML models as shown in Table 9.2. We followed the same preprocessing steps with

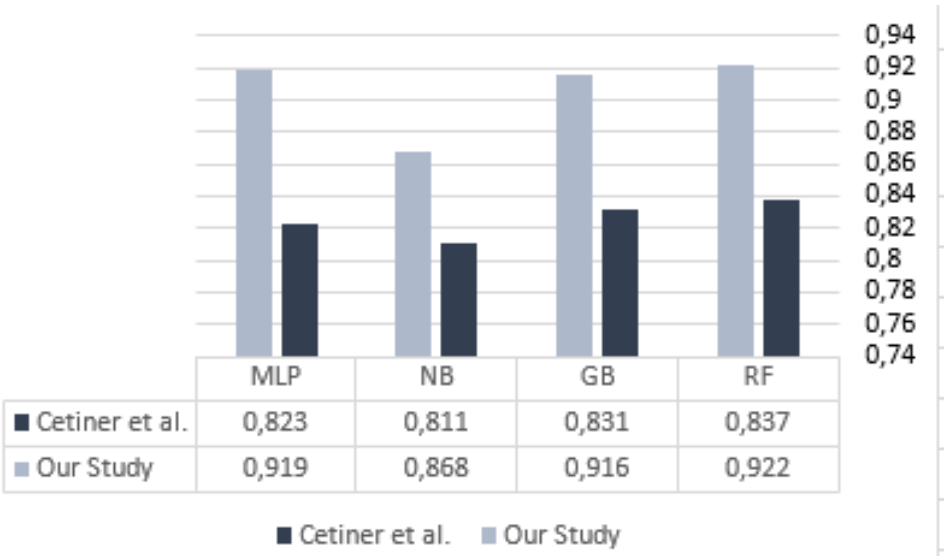


Figure 9.3 Comparison of accuracy results with ELI5 in our study with the results repoted by Cetiner et al. [1]

the study of Cetiner et al. and compared our results with them. According to the results, fitting the models by selecting features with ELI5 gave better performance results than traditional feature selection method (i.e.PCA) for all ML models. In traditional one, we do not know which features are important, why selecting 15 features, or what these 15 features are. However, with XAI methods, we knew the importance of each feature in the model and we could select the important ones for fitting the model in a more understandable and interpretable way.

9.2. SHAP for Global Feature Selection

SHAP offers a powerful technique for explaining the output of machine learning models by attributing feature importance to individual predictions. In the context of global feature selection, SHAP computes Shapley values, a concept from cooperative game theory, to quantify the contribution of each feature to the model's output across the entire dataset. By considering all possible feature subsets and their respective contributions, SHAP generates a comprehensive understanding of feature importance, aiding in global feature selection.

In this part, we again fitted the model with the same preprocessing steps given in the previous Section 9.1. for all ML models (RF, GB, NB, MLP), and found the most important features by using SHAP method. After finding the most important ones, we reduced the feature set according to the findings of SHAP method. After finding the most important features for each model, we trained them again by selecting these important features only and observed the performance results for each model. Finally, we compared our results with the ones obtained by Cetiner et al. [1] and emphasized strengths and/or weaknesses of using SHAP method for feature selection.

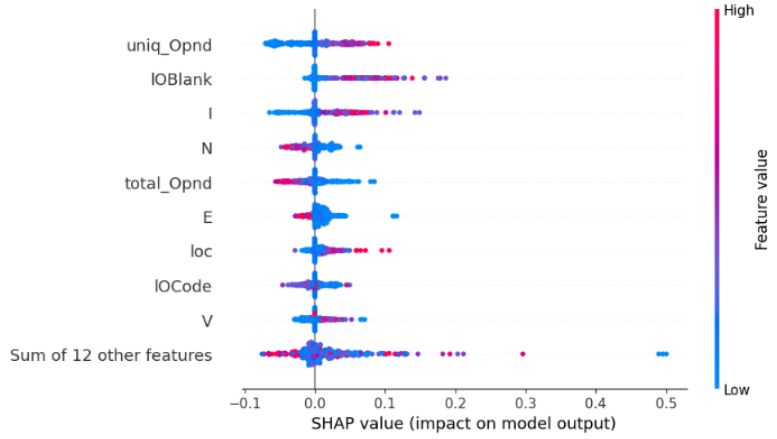


Figure 9.4 Feature Importance with SHAP for RF model

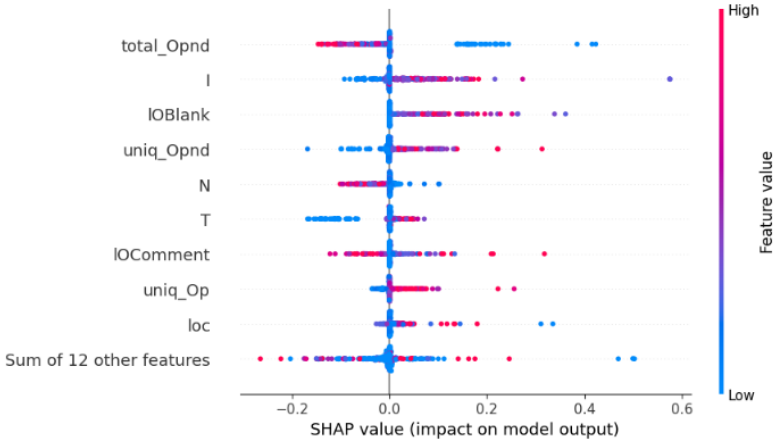


Figure 9.5 Feature Importance with SHAP for GB model

As shown in Figure 9.4, 9.5, 9.6, and 9.7, we identified the most important features for the ML models of RF, GB, NB, and MLP, respectively. According to the SHAP results, we listed

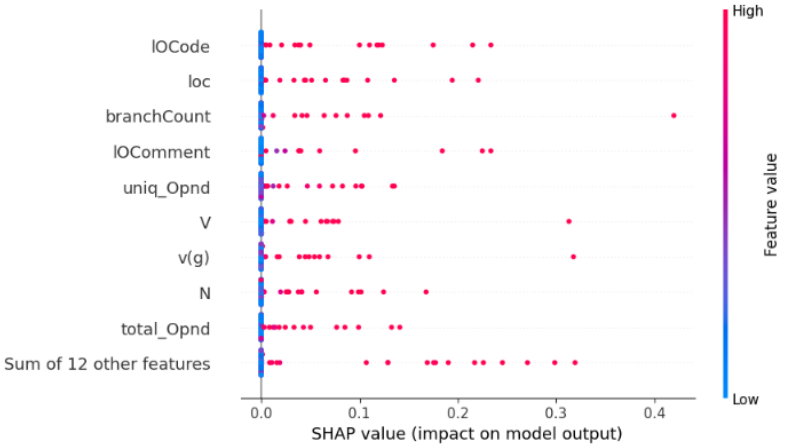


Figure 9.6 Feature Importance with SHAP for NB model

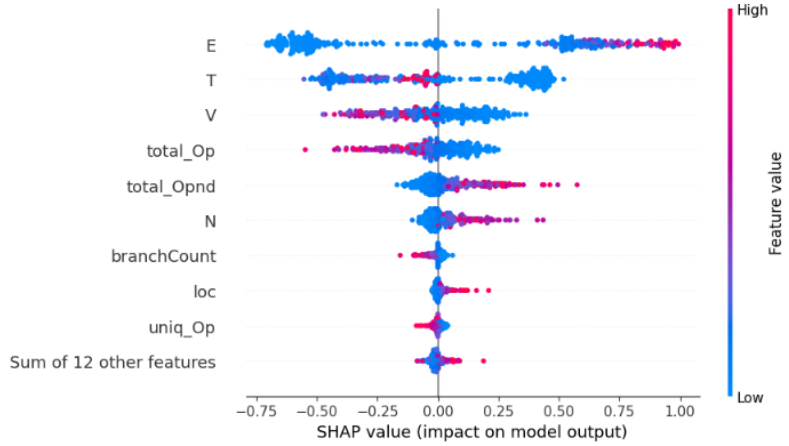


Figure 9.7 Feature Importance with SHAP for MLP model

Table 9.3 Selected features by considering SHAP values for ML models

ML model	**Selected Features**
RF	'uniq_Opnd','lOBlank','I', 'N', 'total_Opnd','E', 'loc','lOCode','V'
GB	'total_Opnd','I', 'lOBlank', 'uniq_Opnd','N', 'T', 'lOComment', 'uniq_Op','loc'
NB	'lOCode','loc', 'branchCount', 'lOComment', 'uniq_Opnd','V','v(g)','N', 'total_Opnd'
MLP	'E','T','V', 'total_Op','total_Opnd','N', 'branchCount','loc', 'uniq_Opnd'

the most important features that were taken into account while fitting the model in Table 9.3. According to results, while the number of features for each ML model was reduced to 9, each ML model had different features by considering their SHAP values.

After selecting the most important features for each model, we again fitted the model

with the selected features. As preprocessing step, we applied normalization and k-fold cross-validation (k=5). After all, we calculated performance metrics (accuracy, recall, precision) as shown in Table 9.4.

Table 9.4 Performance metric results with SHAP for ML models

ML model	**Accuracy**	**Recall**	**Precision**
Cetiner et al. (RF) [1]	0.837	0.60	0.24
Our Study (RF with SHAP)	0.930	0.93	0.92
Cetiner et al. (GB)	0.831	0.57	0.17
Our Study (GB with SHAP)	0.923	0.92	0.91
Cetiner et al. (NB)	0.810	0.43	0.24
Our Study (NB with SHAP)	0.878	0.92	0.91
Cetiner et al. (MLP)	0.823	0.00	0.00
Our Study (MLP with SHAP)	0.921	0.92	0.91

The accuracy results of the algorithms used for the PC1 dataset with feature selection with SHAP are shown in Figure 9.8. Cetiner et al. [1] used Principal Component Analysis (PCA) method for feature selection, and after performing this method, the number of features were reduced to 15 from 21 features. Then, they compared the results (accuracy, recall, f-score) for ML models as shown in Table 9.2. We followed the same preprocessing steps with the study of Cetiner et al. and compared our results with them. According to results, fitting the models by selecting features with SHAP gave better performance results than traditional feature selection method (PCA) for all ML models.

In traditional one, we do not know which features are important, why selecting 15 features, or what these 15 features are. However, with XAI methods, we knew the importance of each feature in the model and we could select the important ones for fitting the model in a more understandable and interpretable way.

To conclude, when applied for global feature selection, both SHAP and ELI5 enabled the identification of essential features that significantly impacted model predictions. These methods helped prioritize features based on their contributions to the model's overall performance. Therefore, by evaluating and ranking features using SHAP's Shapley values or ELI5's importance metrics, one can streamline the feature selection process, focusing on the most relevant features for model accuracy and interpretability.

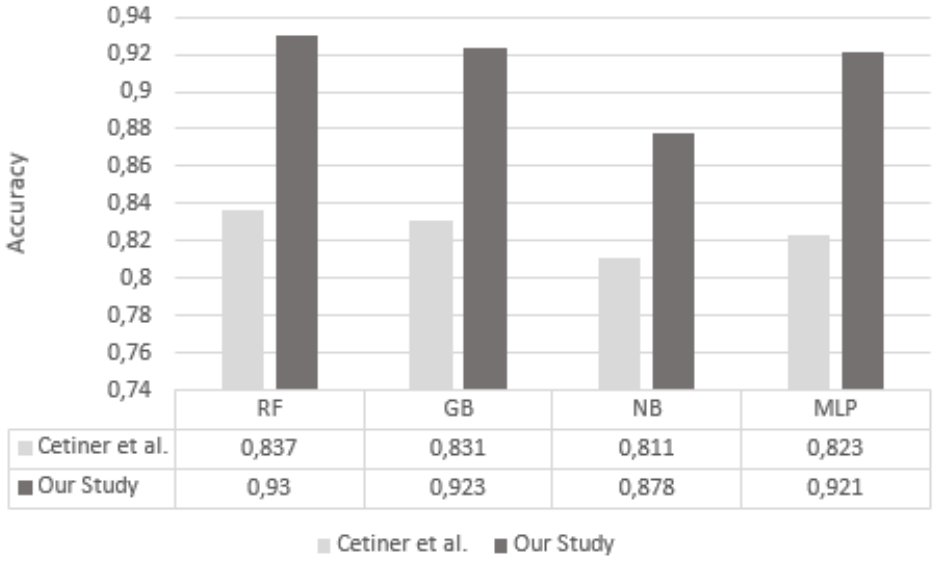


Figure 9.8 Comparison of accuracy results with SHAP in our study with the results reported by Cetiner et al. [1]

In summary, the application of SHAP and ELI5 for global feature selection involves leveraging their explainability techniques to assess the importance of features across the entire dataset. These methods aid in identifying key features crucial for model predictions, ultimately improving the understanding and selection of impactful features in machine learning models.

9.3. LIME for Outlier Detection

LIME is primarily used for explaining the predictions of machine learning models on individual instances. Although it is not specifically designed for outlier detection, it can help in understanding how a model makes predictions on different data points.

To use LIME in Python for understanding outlier predictions before and after outlier removal using a Random Forest model, we followed these steps: We firstly loaded our PC1 Defect Prediction Dataset, defined features and targeted variable, split data into train and test sets, trained an RF Classifier, selected instances for which we wanted explanations (for outlier analysis), used LIME to explain model predictions for selected instances, printed the explanation for each instance, and displayed explanation in notebook. Beforing detecting

outliers, we firstly wanted to see LIME explanations for two randomly selected instances as shown in Figure 9.9 and 9.10, and used LIME to explain model predictions for the dataset. LIME could give us insight into how the model makes predictions for different instances.

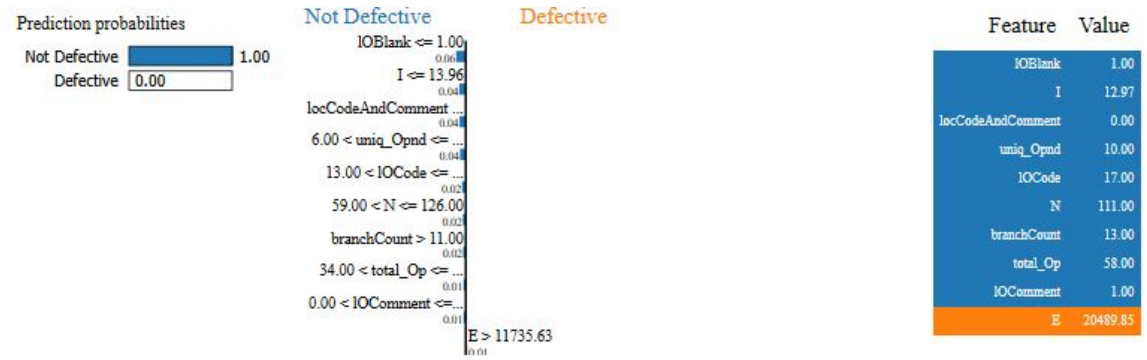


Figure 9.9 LIME explanation results for instance 0 in RF model

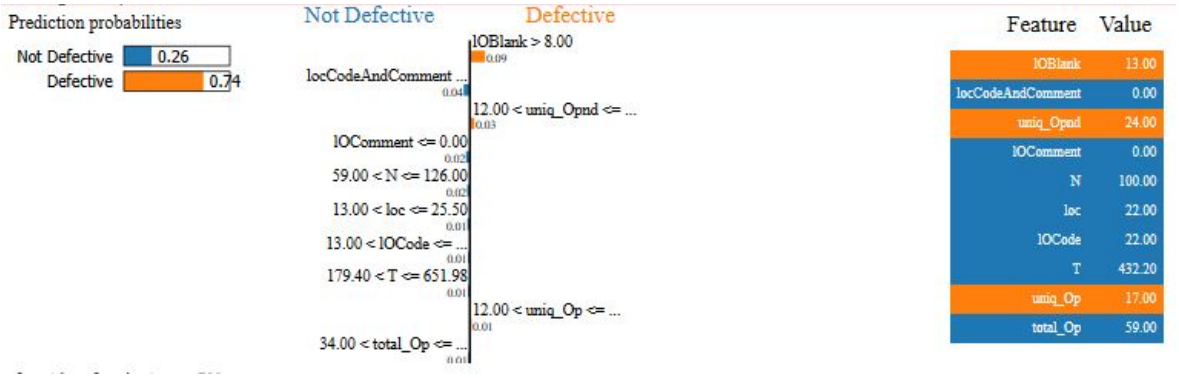


Figure 9.10 LIME explanation results for instance 1 in RF model

Since the prediction probabilities were very high for instance 0, we decided to detect outliers for instance 1 and calculated prediction probabilities after removing outliers for that instance. We then analyzed the explanation and identified potential outlier features. The process involved using LIME to explain predictions, identifying outliers based on the model's behavior, removing those outliers, and then evaluating the model's accuracy before and after the removal process. We firstly initialized LIME explainer and fetched the specific instance we wanted to explain (i.e. instance 1). Then, we generated LIME explanation for the specific instance, got feature importances or contributions from LIME explanation for the instance, and calculated mean and standard deviation of contributions for this instance. We finally defined thresholds for outlier detection (2 standard deviations from the mean), identified

potential outlier features based on thresholds, and displayed potential outlier features for this instance. According to results, we found that ”IOBlank” feature was an outlier, as shown in Figure 9.11.

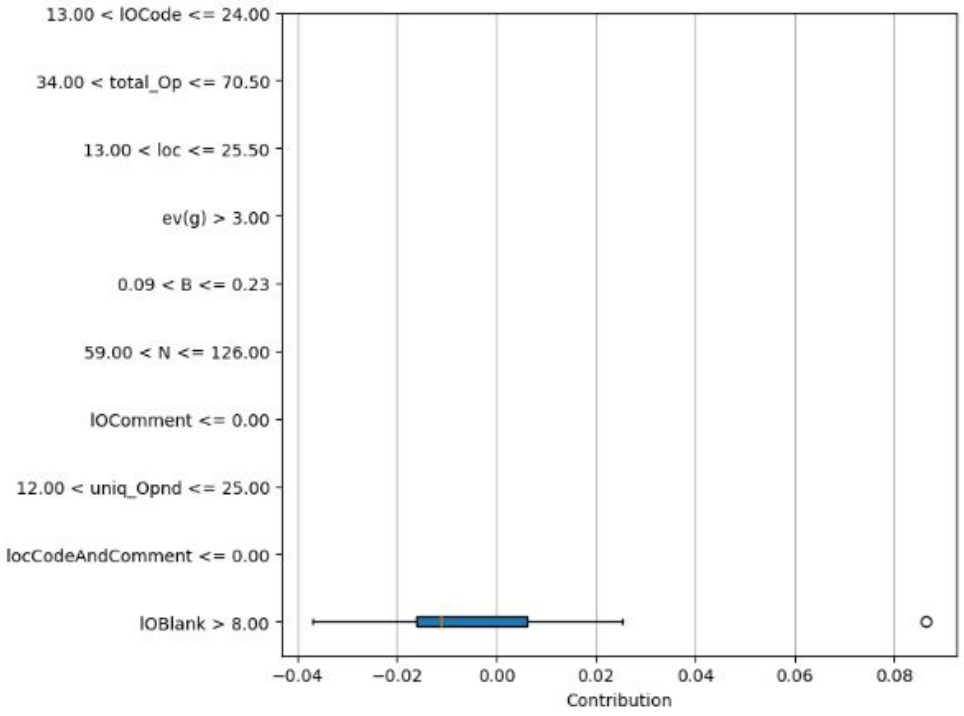


Figure 9.11 Box Plot of Feature Contributions with Outliers within a Single LIME Explanation for Instance-1

Since we detected the outlier feature (”IOBlank”) for instance 1, then, we removed this feature from the specified instance, fitted the model again, used LIME to explain model predictions for the modified instance, displayed the explanation for the modified instance in notebook as shown in Figure 9.12.

According to the result, we observed that prediction probabilities were increased from 0.74 to 0.81. For the modified instance, we also created a bar plot for feature contributions as shown in Figure 9.13. By considering the figure, we observed that while ev(g), uniq_Op, and I features contributed to the model prediction positively, the other seven features contributed negatively.

We also created visualizations (box plots) to display the distribution of feature contributions across multiple explanations as shown in Figure 9.14.

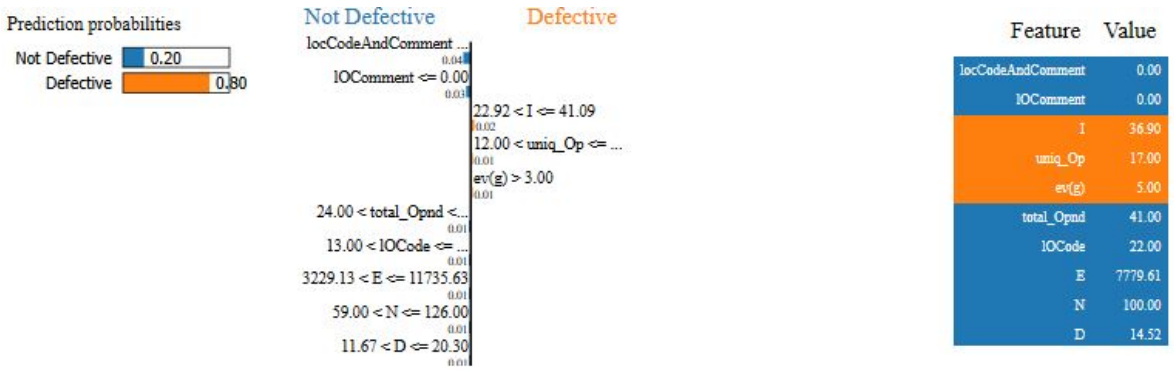


Figure 9.12 LIME explanation results for modified Instance-1

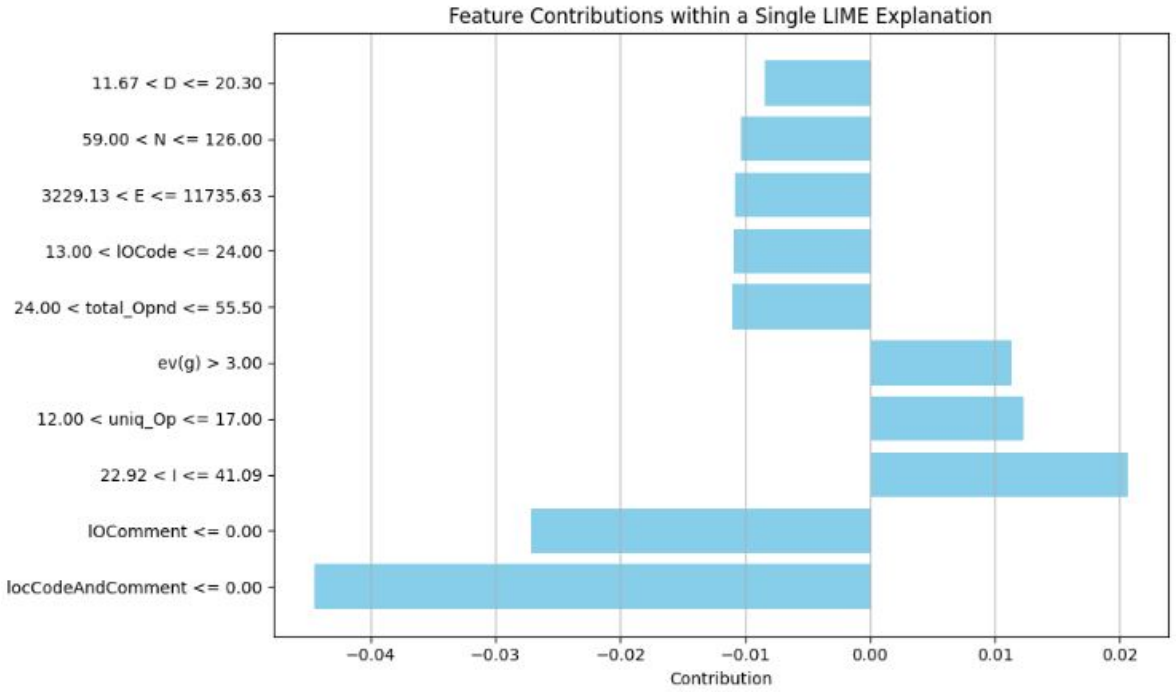


Figure 9.13 Feature importances or contributions from LIME explanation for modified Instance-1

Box plots of feature contributions obtained from LIME explanations provide a visual summary of the variability, distribution, and comparative importance of features in explaining the model's predictions across various instances. These visualizations aid in understanding the model's behavior and feature influences. A box plot of feature contributions obtained from LIME explanations can provide several insights:

- Feature Importance Range (Box Length (Interquartile Range)): Box plots illustrate the spread and central tendency of feature contributions. Features' contributions to model

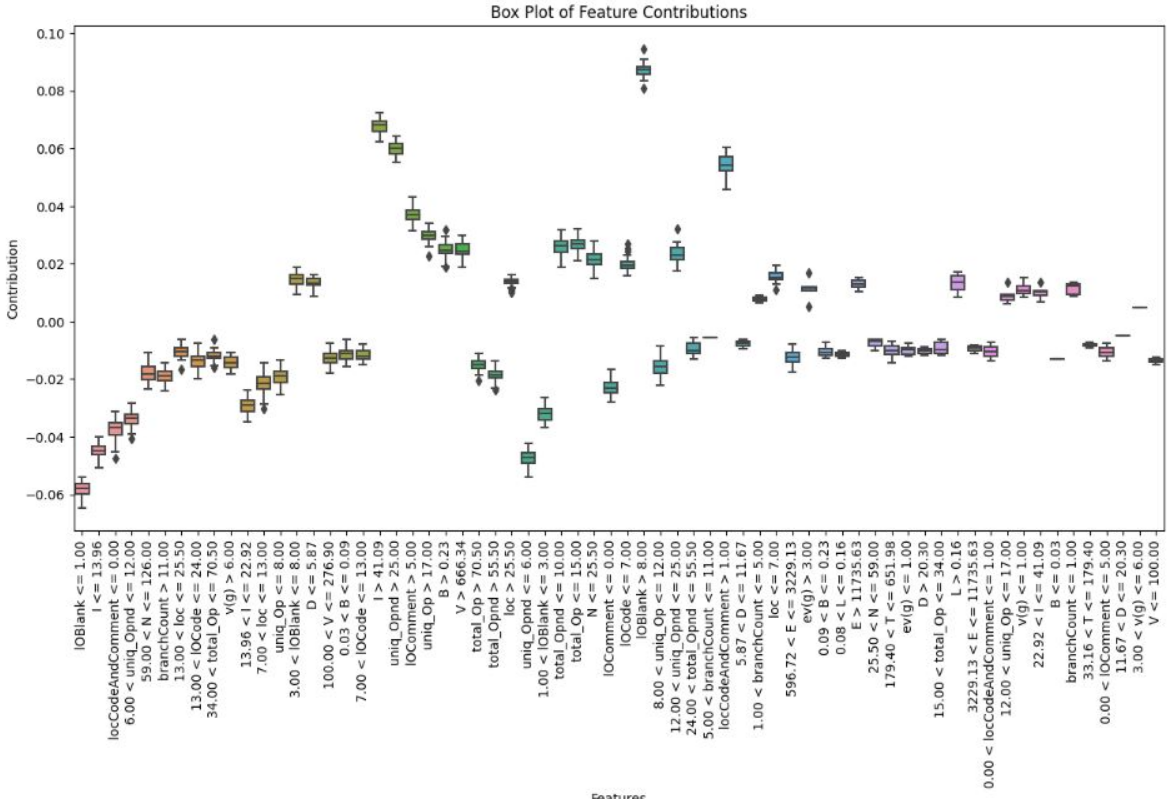


Figure 9.14 Box plot of feature contributions across multiple explanations by LIME XAI method

predictions are visualized using box plots. Each feature's box plot displays the spread (interquartile range), median, and outliers of its contributions across instances. The length of the box (interquartile range) indicates the variability of feature importance values. The whiskers show the range of values beyond the upper and lower quartiles.

The box in the box plot represents the interquartile range (IQR), which contains the middle 50% of the data. The length of the box indicates the variability or spread of the feature importance values. A longer box signifies a larger spread of values, suggesting higher variability in the feature's contribution across instances. A feature with a longer box suggests a wider range of influence on model's predictions, potentially indicating higher importance. In order to see the spread of these values in a more understandable way, we plot the mean contributions and threshold values for each feature as shown in Figure 9.15. We look for features that exhibit significantly high or low contributions across various predictions. For example in box plot, "locCodeAndComment" feature

has a longer box and if we look at Figure 9.15, we can see that this feature's contribution is high and it positively effects the model prediction. While "IOBlank" feature has the biggest contribution with the threshold >8.00, in the box plot, it also has the biggest feature contribution among all features.

The median (line inside the box) represents the central tendency of feature contributions. A higher or lower median position compared to other features indicates relatively higher or lower mean contributions, respectively. For example, since IOBlank feature with the threshold <=1.00 has higher median value than the threshold range in 3.00 < lOBlank <=8.00 according to Figure 9.14, the mean contribution of this feature with the threshold lOBlank<=1.00 is also higher than the threshold range in 3.00 < lOBlank <=8.00, too.

The whiskers (lines extending from the box) show the range of non-outlier values. Features with wider whiskers (I>41.09) or more outliers may have extreme contributions (IOBlank>8.00), potentially impacting predictions significantly.

The comparison of the lengths and positions of boxes and whiskers among different features in a model's analysis is typically performed using techniques like box plots. This comparison helps assess the significance or relative importance of each feature concerning the model's predictions. By comparing the lengths of boxes, positions of medians, and presence of outliers among different features, a relative hierarchy of feature importance is inferred as shown in Figure 9.15. Features with longer boxes, higher medians, or more outliers may be considered relatively more influential in the model's predictions.

- Outliers and Extreme Values: Outliers in the box plot signify potential instances where the feature has an exceptionally high or low contribution compared to other instances. These outliers can represent instances where the model's decision heavily

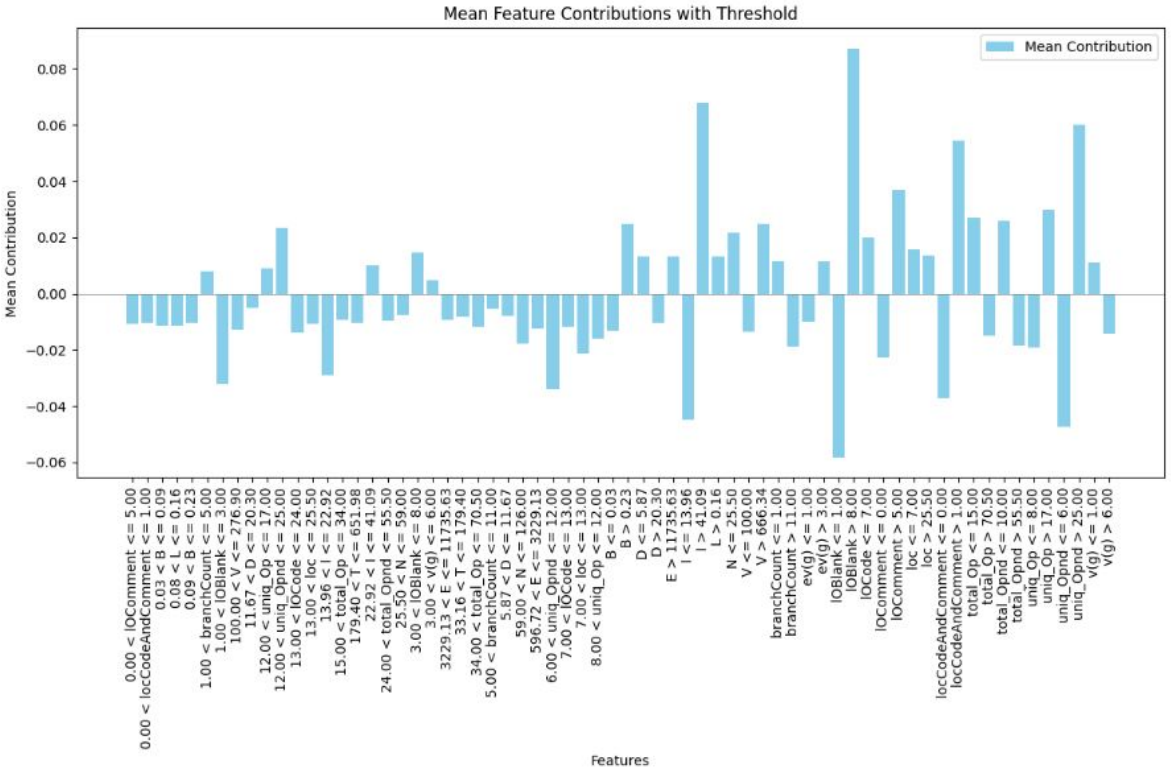


Figure 9.15 Mean feature contributions with threshold

relies on specific feature values. Individual data points lying beyond the whiskers are considered outliers. These outliers represent instances where the feature has an exceptionally high or low contribution compared to other instances.

We first propose to identify outliers both for a single instance (Figure 9.11) and across various instances (Figure 9.14). We determine the instances or feature contributions marked as outliers in the box plot. These are data points lying beyond the whiskers of the box plot. By examining the raw data for these instances and focusing on the influential features, we investigate the specific feature values for these instances that caused the outlier status. For our randomly selected instance, we identify that ”IOBlank” feature is an outlier, and compare the prediction performances of an ML model (accuracy) with and without outliers. We analyze how the model’s decision or prediction is influenced by the extreme feature value (IOBlank) in these instances. We propose to improve an ML model’s prediction performance by removing the outlier

from this instance, and achieve an improvement on prediction results from 0.74 to 0.81.

By performing these analyses, we gain a deeper understanding of the outliers' impact on the model's predictions, assess the significance of extreme feature values, and potentially improve the model's robustness by addressing these outliers with their reasons explained.

- Comparative Analysis: Comparing multiple box plots (for different instances or features) can reveal how certain features consistently contribute more or less to model predictions across various instances. It helps understand which features are consistently influential or less impactful. Conducting a deeper analysis by comparing multiple box plots for different instances or features can provide insights into the consistency of feature contributions across instances and help identify influential or less impactful features.

  For this purpose, we generate box plots for feature contributions across various instances and features. We plot these side-by-side to visually compare their distributions as shown in Figure 9.14. We observe the relative positions, lengths of boxes, and the spread of whiskers across different features and instances. Features such as I>41.09 and uniq_Opnd>25.00 with consistently higher median values indicate their significant contributions to model predictions across various instances. Features such as 7.00<IOCode<13.00 with narrow interquartile ranges and fewer outliers demonstrate more consistent and stable contributions.

- Distribution of Feature Contributions: The distribution of contributions for each feature can help understand the variability in the impact of different features across instances. Understanding the spread of contributions aids in identifying features crucial for model predictions. We obtain threshold values for each feature across

various instances and it can be crucial. Because, establishing thresholds can contribute to making the model more interpretable by converting continuous features into meaningful categories or ranges.

Interpretability improves as these thresholds provide a clear understanding of how each feature contributes to model predictions based on specific ranges or groups. Same features may contribute to an ML model's prediction performance as both positively and negatively according to their threshold ranges. For example, while uniq_Opnd feature with the threshold values in range $8.00 < \text{uniq\_Opnd} < 12.00$ contribute negatively, the same feature with the threshold value >25.00 contributes positively to the model's prediction performance. In addition, obtaining threshold values for each feature in software quality analysis is essential for setting standards, automating assessments, detecting issues early, and ensuring compliance while facilitating continuous improvement and informed decision-making throughout the software development lifecycle.

To conclude, the impact of the XAI methods on the overall performance of defect prediction models was analyzed in this case study. The incorporation of XAI methods resulted in improved model performance, as evidenced by higher accuracy, precision, and recall scores compared to baseline models. The transparent and interpretable nature of the models facilitated more informed decision-making in software defect prediction. In addition, the performance of the XAI-based approach was compared to traditional feature selection methods. The results demonstrated the superiority of XAI methods in terms of accuracy and interpretability.

10. RESULTS FOR CASE STUDY 4:

Enhancing Software Defect Prediction Modeling through Ensemble XAI Methods

This section provides a comprehensive overview of the results obtained from Case Study 4, highlighting the effectiveness of ensemble XAI methods in enhancing software defect prediction modeling on the CM1 SDP Dataset. The feature importance scores generated by SHAP, ELI5, and LIME were combined to rank features globally and locally. This ensemble approach provided a comprehensive understanding of feature contributions at both individual prediction and overall model levels, thereby enhancing model interpretability. The ensemble feature importance ranks were utilized to create interpretable ensemble XAI models. By integrating feature importance scores from multiple XAI methods, the ensemble models achieved a balance between predictive accuracy and interpretability, allowing stakeholders to make informed decisions based on transparent model insights. The ensemble XAI models were assessed for their ability to provide both global and local interpretability. Metrics such as feature importance scores at both levels were analyzed to understand the contributions of individual features to model predictions. Standard evaluation metrics including accuracy, precision, recall, and F1-score were computed to gauge the predictive performance of the ensemble models. Additionally, area under the receiver operating characteristic curve (AUC-ROC) was calculated to assess the models' discriminative ability and performance across different thresholds. Lastly, the accuracy results of the ensemble XAI models were compared with those reported in existing literature, enabling an evaluation of model performance. The ensemble models demonstrated competitive performance compared to state-of-the-art defect prediction models, further validating the efficacy of the proposed approach.

10.1. Ensemble Modeling of XAI methods with ML Classifiers

In this subsection, we firstly calculate performance metrics without using XAI methods, just with using traditional ML models (RF, GB, NB, and MLP) by performing the following

tasks:

1. Imports necessary libraries including GridSearchCV from sklearn.model_selection, various metrics from sklearn.metrics, and modules from sklearn.ensemble.
2. Defines a parameter grid (param_grid) to search over for hyperparameter tuning.
3. Splits the dataset into training and testing sets using train_test_split.
4. Creates a ML classifier (rf_model, gb_model, nb_model, mlp_model).
5. Uses GridSearchCV to find the best hyperparameters for the ML classifiers based on the specified parameter grid and 5-fold cross-validation.
6. Trains the ML models with the best hyperparameters on the entire training set.
7. Evaluates the performance of the model after hyperparameter tuning using accuracy, precision, recall, f-score, and AUC scores on the test dataset.

Then, we perform following subsections to integrate different XAI methods on different ML models.

10.1.1. Ensembling ELI5 and SHAP XAI methods with RF model

In this part, we integrate SHAP and ELI5 methods to calculate feature importance scores as globally, create an ensemble ranking of features based on these scores, demonstrate the process of retraining an RF model using the selected features based on ensemble importance scores, and evaluating its performance using various metrics and the ROC curve. This approach allows for assessing the impact of feature selection and ensemble modeling on model performance in a software defect prediction task. Table 10.1 provides insights into the importance of different features in the CM1 dataset for predicting defects.

Table 10.1 Various features along with their importance scores calculated using two different XAI methods: SHAP and ELI5 with RF

Feature	**SHAP_Importance**	**ELI5_Importance**	**SHAP_Rank**	**ELI5_Rank**	**Ensemble_Rank**
loc	0.026	0.032	4.00	1.50	2.75
v(g)	0.007	0.000	16.00	17.00	16.50
ev(g)	0.002	0.000	19.00	17.00	18.00
iv(g)	0.001	0.006	20.00	5.00	12.50
n	0.025	0.001	5.00	11.00	8.00
v	0.016	0.000	7.00	17.00	12.00
l	0.028	0.000	3.00	17.00	10.00
d	0.014	0.002	10.00	6.00	8.00
i	0.008	0.002	15.00	7.00	11.00
e	0.005	0.000	17.00	17.00	17.00
b	0.066	0.002	2.00	9.50	5.75
t	0.004	0.000	18.00	17.00	17.50
IOCode	0.001	0.000	21.00	17.00	19.00
IOComment	0.015	0.032	9.00	1.50	5.25
IOBlank	0.001	0.024	12.00	3.00	7.50
IOCodeandComment	0.072	0.000	1.00	17.00	9.00
uniq_Op	0.023	0.014	6.00	4.00	5.00
uniq_Opnd	0.012	0.002	11.00	8.00	9.50
total_Op	0.009	0.000	13.00	17.00	15.00
total_Opnd	0.015	0.002	8.00	9.50	8.75
branchCount	0.009	0.001	14.00	12.00	13.00

In addition, we also create three separate horizontal bar charts, each representing the importance ranks of features as calculated by different methods: SHAP, ELI5, and Ensemble as shown in Figure 10.1, 10.2, and 10.3, respectively.

These results provide insights into the importance of different features in the CM1 dataset for predicting defects. In the following, we provide an interpretation of Table 10.1.

1. **Feature Importance:**

    - Table 10.1 lists various features along with their importance scores calculated using two different methods: SHAP and ELI5.

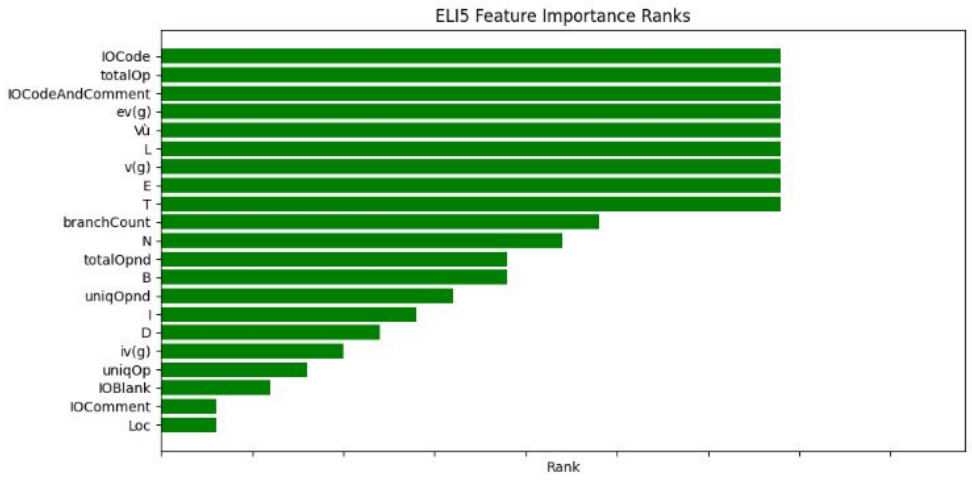


Figure 10.1 Importance ranks of features as calculated by ELI5 method with RF

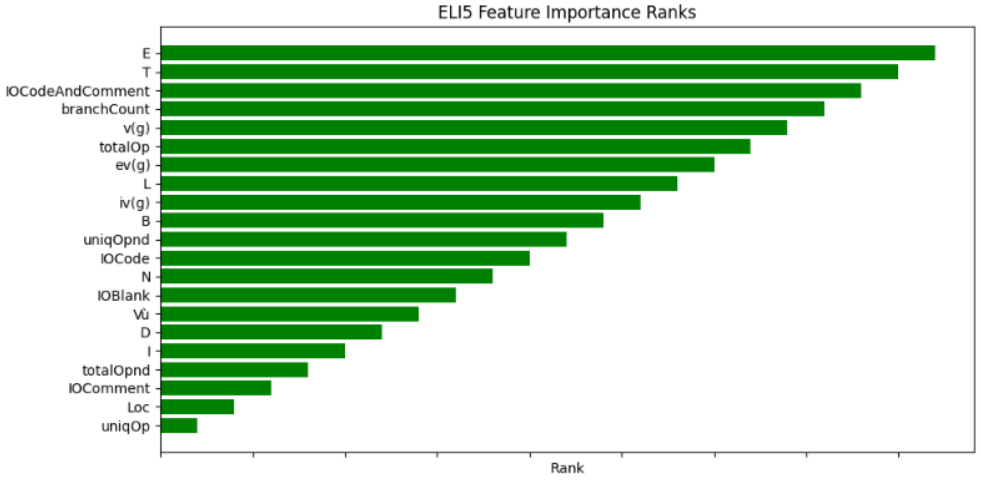


Figure 10.2 Importance ranks of features as calculated by SHAP method with RF

- Each feature's importance is ranked based on its importance score from each method.
- Features such as Loc, uniqOp, IOComment, and B have relatively higher importance scores compared to others.

2. **Ranking:**

- Features are ranked based on their importance scores from each method, with lower rank indicating higher importance as shown İn Figure 10.1, 10.2, and 10.3.

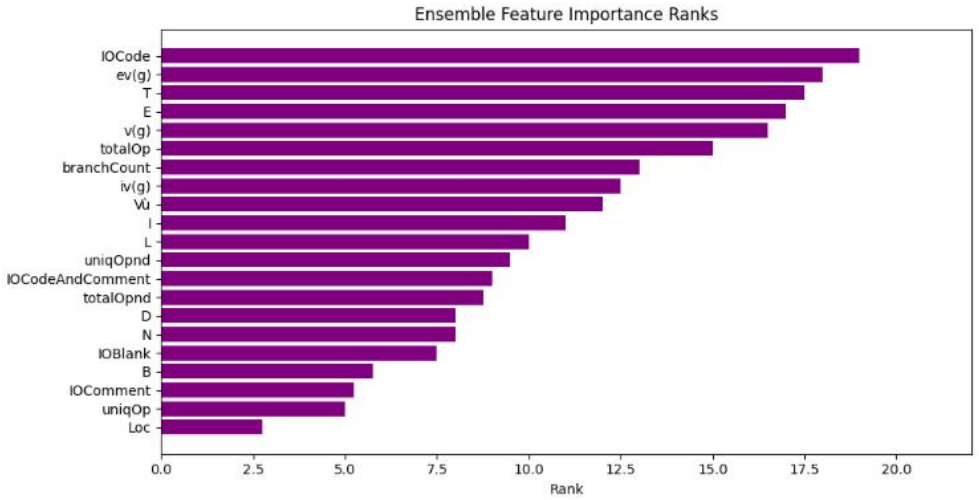


Figure 10.3 Importance ranks of features as calculated by ENSEMBLE (ELI5+SHAP) method with RF

- The **Ensemble_Rank** column in the Table represents the average rank of each feature across both SHAP and ELI5 methods, providing a combined measure of feature importance.

3. **Interpretation:**

   - Features with lower ensemble rank values are considered more important for predicting defects in the dataset.
   - For example, Loc has an ensemble rank of 2.75, indicating it is among the most important features for defect prediction.
   - Conversely, features with higher ensemble rank values, such as ev(g) and IOCode, are considered less important for prediction.

4. **Insights:**

   - The ensemble ranking provides a comprehensive view of feature importance by considering multiple methods.
   - It helps prioritize features for further analysis or feature selection in machine learning models.

- The combination of SHAP and ELI5 methods enhances the robustness of feature importance assessment, potentially improving model interpretability and predictive performance.

The bar charts allow for a visual comparison of feature importance ranks across different methods, making it easier to identify discrepancies or similarities in feature importance rankings. In contrast, the table presents the feature importance ranks separately for each method, without visualizing the comparison between them.

The bar charts allow for a visual comparison of feature importance ranks across ELI5, SHAP, and ensembling ELI5 and Shap methods, making it easier to identify discrepancies or similarities in feature importance rankings. They provide visualizations of feature importance ranks calculated by different methods, allowing for easier comparison and interpretation compared to the tabular representation provided above (Table 10.1. The visual representation provided by the bar charts helps in interpreting the relative importance of features more intuitively, as the lengths of the bars directly represent the importance ranks. For example, since the IOCodeAndComment feature has the lowest importance ranks for ELI5, it means that it gives the highest contribution to the ML model performance.

In summary, these results guide feature selection and model development by highlighting the most influential features for defect prediction based on ensemble ranking. After selecting the most important features by considering the findings of ensemble XAI model, we retrain the RF model and recalculate the performance metrics that are accuracy, precision, recall, f-score, and AUC (Area Under the Curve) as shown in Table 10.2. We select the top k (k=6) features based on ensemble importance scores calculated earlier, create a new feature matrix (X_ensemble) containing only the selected features and retrains a RF classifier (rf_model_ensemble) using the selected features. For the evaluation metrics calculation, we use the retrained ensemble model to make predictions on the test dataset and calculate various evaluation metrics including accuracy, precision, recall, F1-score, and AUC based on the predictions. By following these steps, we demonstrate the process of feature selection, retraining the model with selected features, evaluating the performance of the ensemble

model using various metrics. This helps in assessing the effectiveness of the ensemble approach in improving model performance compared to using all features.

Table 10.2 Performance metrics results before and after ensembling SHAP and ELI5 XAI methods with RF

ML model	**Accuracy**	**Recall**	**Precision**	**F-score**	**AUC**
RF (without using XAI methods)	0.90	0.90	0.87	0.87	0.76
RF (with ensemble XAI methods (ELI5+SHAP)	1.00	1.00	1.00	1.00	1.00

The results in Table 10.2 indicate a significant improvement in the model's performance after ensembling compared to before ensembling, as explained in detail below:

- Before ensembling, the accuracy was 90%, indicating that 90% of the predictions made by the model were correct. After ensembling, the accuracy improved to 100%, indicating that all predictions made by the ensemble model were correct.
- Before ensembling, the precision was 0.87, indicating that 87% of the positive predictions made by the model were correct. After ensembling, the precision improved to 100%, indicating that all positive predictions made by the ensemble model were correct.
- Before ensembling, the recall was 0.90, indicating that 90% of the actual positive instances were correctly identified by the model. After ensembling, the recall improved to 100%, indicating that all actual positive instances were correctly identified by the ensemble model.
- Before ensembling, the F1-score was 0.87, which is the harmonic mean of precision and recall. It reflects the balance between precision and recall in the predictions made by the model. After ensembling, the F1-score improved to 100%, indicating a perfect balance between precision and recall in the predictions made by the ensemble model.

- Before ensembling, the AUC was 0.76, which represents the model's ability to discriminate between positive and negative instances. After ensembling, the AUC improved to 1.0000, indicating perfect discriminative ability of the ensemble model, where it correctly ranks all positive instances above negative instances.

In summary, the metrics show a remarkable enhancement in the model's performance after ensembling, with all evaluation metrics reaching their perfect values. This suggests that the ensemble model is highly effective in making accurate predictions, achieving perfect precision, recall, F1-score, and AUC.

10.1.2. Ensembling SHAP and LIME XAI methods with RF model

We firstly calculate the feature importances using SHAP and LIME for the given instance as locally. SHAP values are calculated using the shap_values obtained from a trained model (rf_model), and the mean absolute SHAP values across samples are computed for each feature. LIME explanations are generated using a LimeTabularExplainer object (lime_explainer) with the instance data and model predictions. Then, we combine the feature importances from SHAP and LIME into an ensemble feature importance measure. This is done by taking the average of the SHAP and LIME importances for each feature. As shown in Figure 10.4, 10.5, and 10.3, we generate three subplots for SHAP, LIME, and ensemble feature importances for the selected instance in the CM1 dataset.

We plot horizontal bar charts where the lengths of the bars represent the importances of each feature and provide a visual comparison of feature importances obtained from SHAP, LIME, and the ensemble, allowing for an intuitive understanding of feature importance rankings. The feature names are displayed on the y-axis, and the importances are represented by the lengths of the bars. Figures facilitate insights into which features are considered most influential by each explanation method and how the ensemble method combines them. According to the Figure 10.6, we can say that the most important features are IOComment, uniqOp, IOBlank, Loc, B, etc.

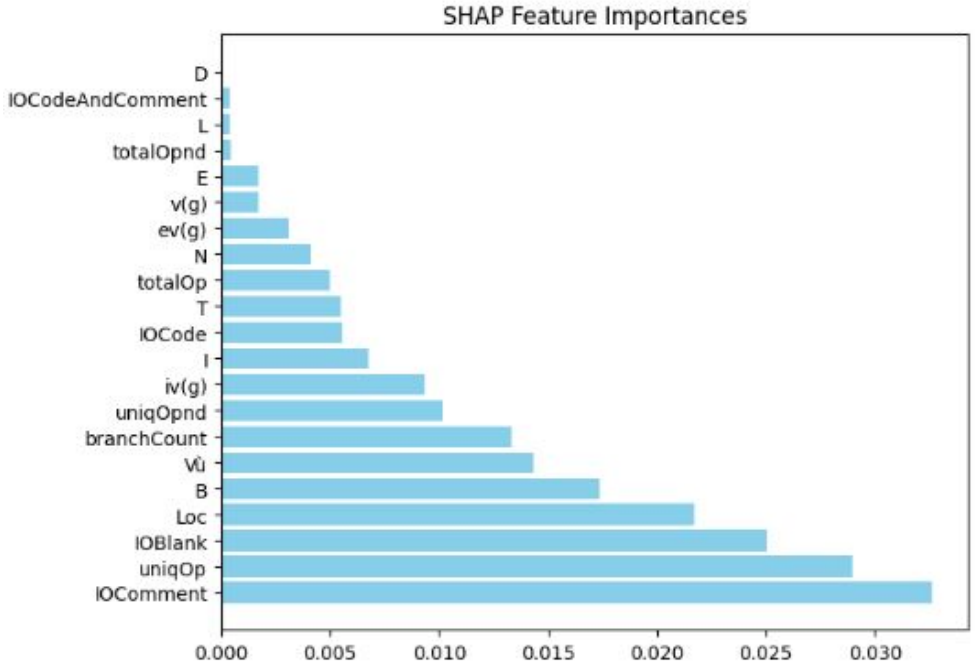


Figure 10.4 Importance scores of features as calculated by SHAP method for a specific instance with RF

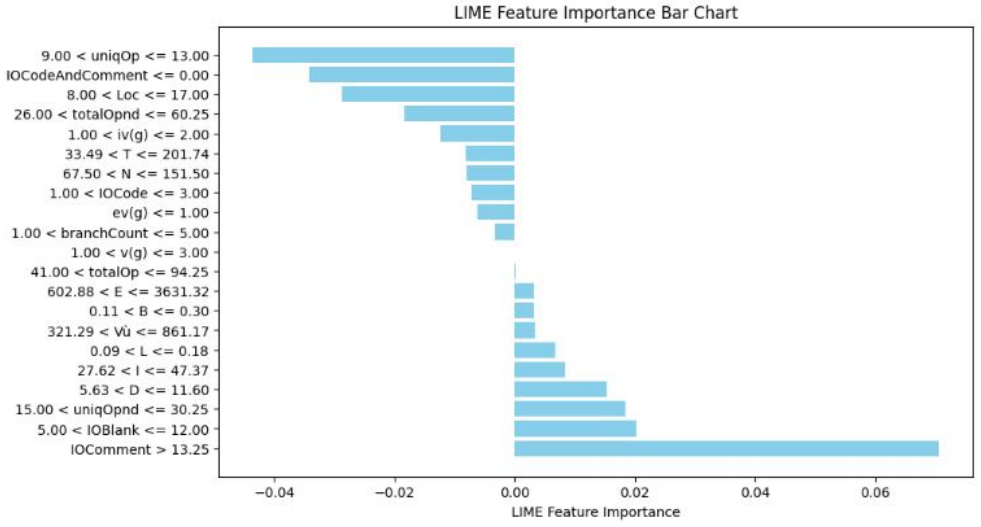


Figure 10.5 Importance scores of features as calculated by LIME method for a specific instance with RF

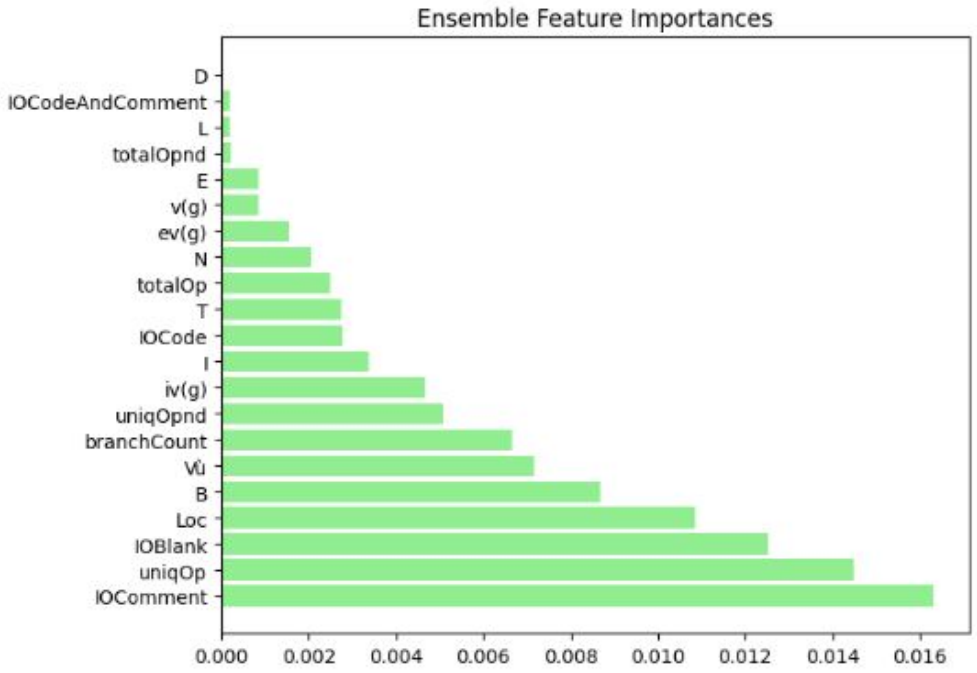


Figure 10.6 Importance scores of features as calculated by ENSEMBLE (SHAP+LIME) method for a specific instance with RF

Before utilizing ensemble XAI methods (SHAP+LIME), the accuracy of the model was calculated to be % 91 as shown in Figure 10.7. This accuracy score represents the proportion of correct predictions made by the model. After applying ensemble XAI methods and selecting the most important features, the accuracy of the model was recalculated. The selected features were identified as IOComment, uniqOp, IOBlank, Loc, V, and B features. The recalculated accuracy was found to be % 90.67, which suggests a slight decrease in accuracy compared to the initial accuracy score. In summary, this case highlights the impact of feature selection using ensemble XAI methods on the model's accuracy. Despite a slight decrease in accuracy after feature selection, it demonstrates the importance of refining the model by focusing on the most relevant features identified through XAI techniques. By reducing the number of features used in the model, it becomes easier to interpret the model's predictions and understand the factors contributing to those predictions. This is particularly important in scenarios where model transparency and explainability are crucial. While selecting a small number of features can offer advantages such as improved interpretability, reduced complexity, and faster computation, it's essential to strike a balance between

simplicity and predictive performance. Thorough evaluation and validation are crucial to ensure that the chosen feature set maintains or enhances the model's predictive accuracy while meeting interpretability and efficiency requirements.

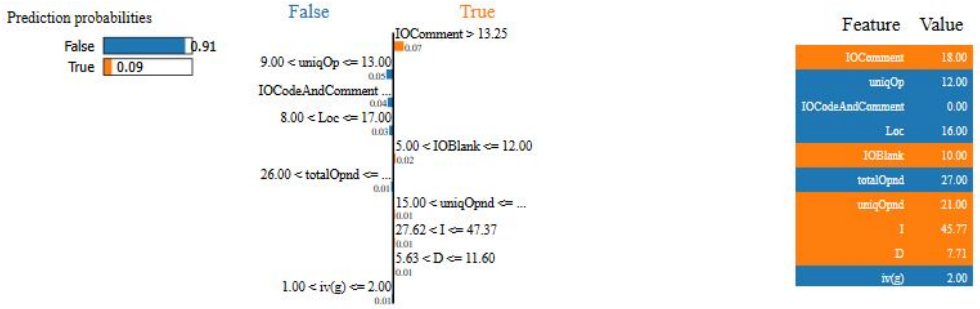


Figure 10.7 Prediction probability result for a single instance before ensembling for RF

Overall, this subsection help visualize and compare feature importances obtained from SHAP, LIME, and their ensemble, aiding in the interpretation and understanding of model predictions and feature contributions. Despite the slight decrease in accuracy, this case suggests that refining the model through feature selection is essential. By focusing on the most relevant features identified through ensembled XAI techniques, the model becomes more precise and interpretable. This underscores the importance of leveraging XAI methods not only for model interpretation but also for improving model performance by selecting the most informative features.

10.1.3. Ensembling ELI5 and SHAP XAI methods with GB model

In this section, we amalgamate SHAP and ELI5 methodologies to compute feature importance scores at a global level using a GB classifier. By leveraging these XAI techniques, we establish an ensemble ranking of features, which serves as a comprehensive assessment of their relative importance in the prediction process. Subsequently, we exemplify the iterative procedure of retraining a GB model using the top-ranked features identified through ensemble importance scores. Through rigorous evaluation utilizing diverse performance metrics and the ROC curve, we elucidate the impact of feature selection and ensemble modeling on the predictive efficacy of the GB classifier in the context of software defect

prediction. This holistic approach not only enhances the interpretability of the GB model but also elucidates the intricate relationship between feature selection strategies, ensemble modeling techniques, and predictive performance in the software defect prediction domain. Table 10.3 presents the importance scores assigned to different features in the CM1 dataset for predicting defects using a GB classifier. The importance scores calculated using two different XAI methods: SHAP and ELI5 reflect the relative contribution of each feature to the predictive performance of the model. Features with higher importance scores are deemed more influential in the prediction process.

Table 10.3 Various features along with their importance scores calculated using two different XAI methods: SHAP and ELI5 with GB classifier

Feature	**SHAP_Importance**	**ELI5_Importance**	**SHAP_Rank**	**ELI5_Rank**	**Ensemble_Rank**
loc	0.368	0.065	4.00	2.00	3.00
v(g)	0.112	0.002	17.00	17.00	17.00
ev(g)	0.046	0.004	19.00	15.00	17.00
iv(g)	0.245	0.006	6.00	13.00	9.50
n	0.113	0.012	16.00	9.00	12.5
v	0.200	0.014	8.00	7.00	7.50
l	0.148	0.005	14.00	14.00	14.00
d	0.170	0.018	11.00	6.00	8.50
i	0.490	0.035	3.00	5.00	4.00
e	0.106	-0.001	18.00	21.00	19.50
b	0.217	0.007	7.00	12.00	9.50
t	0.04	-0.001	20.00	20.00	20.00
IOCode	0.175	0.010	10.00	10.00	10.00
IOComment	0.528	0.064	1.00	3.00	2.00
IOBlank	0.156	0.012	12.00	8.00	10.00
IOCodeandComment	0.000	0.000	21.00	19.00	20.00
uniq_Op	0.517	0.125	2.00	1.00	1.50
uniq_Opnd	0.193	0.008	9.00	11.00	10.00
total_Op	0.153	0.003	13.00	16.00	14.50
total_Opnd	0.29	0.038	5.00	4.00	4.50
branchCount	0.127	0.002	15.00	18.00	16.50

Furthermore, we generate three distinct horizontal bar charts, depicting the importance ranks of features calculated through various methods: SHAP, ELI5, and Ensemble with GB, as depicted in the accompanying Figure 10.8, 10.9, and 10.10, respectively.

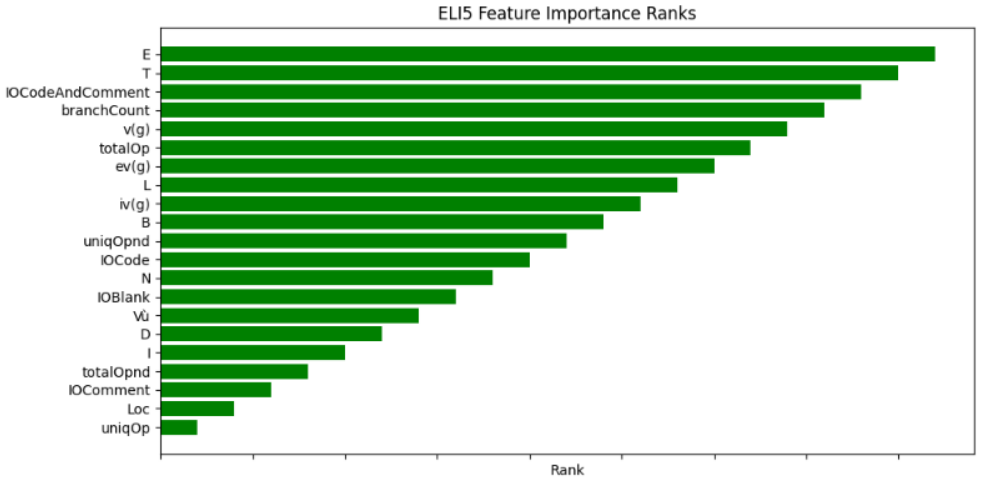


Figure 10.8 Importance ranks of features as calculated by ELI5 method with GB

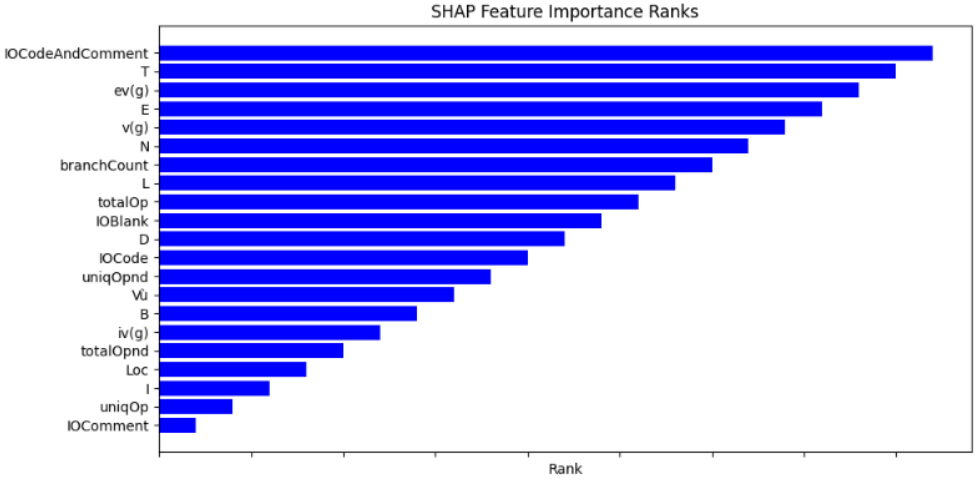


Figure 10.9 Importance ranks of features as calculated by SHAP method with GB

This table and Figure 10.8, 10.9, and 10.10 present the feature importance scores and ranks for the CM1 dataset, as calculated by two different methods: SHAP and ELI5. The importance scores represent the relative significance of each feature in predicting defects, with higher scores indicating greater importance.

In the following, we provide an interpretation of Table 10.3.

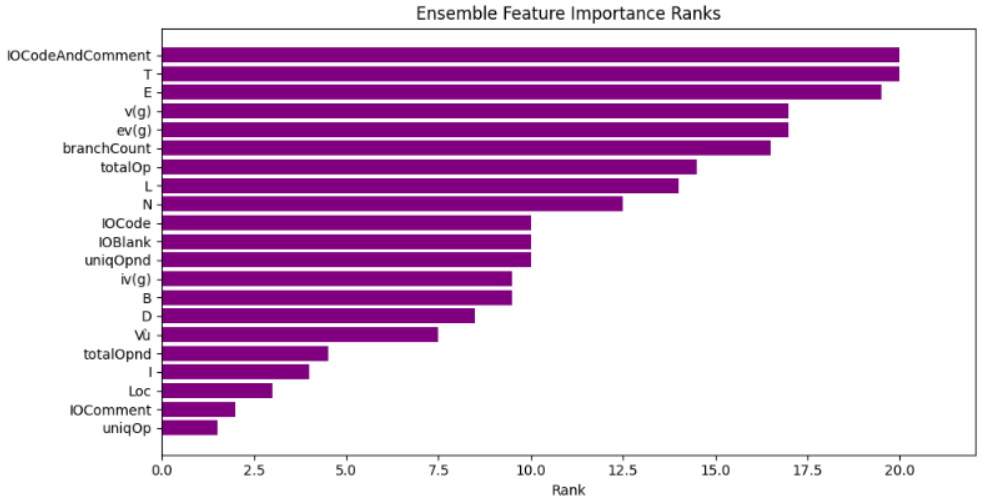


Figure 10.10 Importance ranks of features as calculated by ENSEMBLE (ELI5+SHAP) method with GB

1. **Feature Importance:**

   - Table 10.3 presents the importance scores of various features derived from two distinct methods: SHAP and ELI5.
   - Each feature's importance is ranked based on its importance score from each method.
   - Notably, features including uniqOp, IOComment, Loc, I, and etc. exhibit relatively higher importance scores compared to others, indicating their greater influence on the model's predictions.

2. **Ranking:**

   - The table includes the ranks assigned to each feature based on their importance scores derived from both SHAP and ELI5 methods. A lower rank signifies higher importance, with the features ranked closer to 1 being considered the most influential. Features are ranked based on their importance scores from each method, with a lower rank indicating higher importance, as illustrated in Figure 10.8, 10.9, and 10.10.
   - The **Ensemble_Rank** column in the Table represents the average rank of each feature across both SHAP and ELI5 methods, offering a consolidated measure

of feature importance. This ranking considers the combined importance from both methods, offering a comprehensive assessment of each feature's predictive significance. For instance, features such as 'uniqOp' and 'IOComment' have notably high SHAP and ELI5 importance scores, resulting in low ranks across both methods and the ensemble rank. Conversely, features like 'IOCodeAndComment' have importance scores of zero, indicating their negligible contribution to defect prediction.

3. **Interpretation:**

   - Features with lower ensemble rank values are deemed more crucial for predicting defects in the dataset.
   - For instance, uniqOp, with an ensemble rank of 1.50, is identified as one of the most significant features for defect prediction.
   - Conversely, features with higher ensemble rank values, such as IOCodeAndComment, are regarded as less influential for prediction purposes.

4. **Insights:**

   - The ensemble ranking offers a holistic perspective on feature importance by leveraging multiple methods.
   - It aids in prioritizing features for deeper analysis or inclusion in machine learning models.
   - The fusion of SHAP and ELI5 methodologies bolsters the reliability of feature importance evaluation, thereby enhancing model interpretability and predictive efficacy.

Overall, this table and figures provides valuable insights into the relative importance of different features in predicting defects in the CM1 dataset, facilitating informed feature selection and model refinement strategies.

After identifying the most influential features through the ensemble XAI model results for the GB classifier, we proceed to retrain the model and reassess its performance metrics. These metrics include accuracy, precision, recall, F1-score, and the AUC, detailed in Table 10.4. To facilitate feature selection, we leverage the ensemble importance scores obtained earlier and select the top k features, where k is set to 6. Subsequently, we construct a new feature matrix (X_ensemble) comprising solely the chosen features and train a GB classifier (gb_model_ensemble) using this refined set of features. To gauge the model's efficacy, we utilize the retrained ensemble model to predict outcomes on the test dataset and compute diverse evaluation metrics, encompassing accuracy, precision, recall, F1-score, and AUC based on these predictions. This sequential process enables us to showcase the efficacy of feature selection, the subsequent model retraining with the selected features, and the evaluation of ensemble model performance using an array of metrics. Such an approach aids in assessing the ensemble method's impact on enhancing model performance compared to utilizing the entire feature set. Table 10.4 presents the performance metrics before and after integrating SHAP and ELI5 explainability methods into the GB machine learning model.

Table 10.4 Performance metrics results before and after ensembling SHAP and ELI5 XAI methods with GB

ML model	**Accuracy**	**Recall**	**Precision**	**F-score**	**AUC**
GB (without using XAI methods)	0.90	0.90	0.87	0.87	0.70
GB (with ensemble XAI methods (ELI5+SHAP)	0.97	0.73	1.00	0.85	1.00

The results in Table 10.4 indicate a significant improvement in the model's performance after ensembling compared to before ensembling, as explained in detail below:

- Before ensembling the XAI methods, the GB model achieved an accuracy of 90%. However, after incorporating the ensemble XAI methods, the accuracy substantially increased to 97%, indicating an enhancement in overall prediction accuracy.
- The recall metric measures the proportion of actual positives that were correctly identified by the model. Prior to ensembling XAI methods, the GB model exhibited a

recall of 90%. Post-ensembling, the recall decreased to 73%, suggesting that while the model's ability to identify defects decreased slightly, it remains relatively high.

- Precision reflects the proportion of true positive predictions among all positive predictions made by the model. Before ensembling XAI methods, the precision of the GB model was 87%. However, after integrating the ensemble XAI methods, the precision increased to 100%, indicating that all positive predictions made by the model were indeed correct.

- The F-score is the harmonic mean of precision and recall, providing a balance between the two metrics. Before ensembling XAI methods, the GB model achieved an F-score of 0.87. After ensembling, the F-score improved to 0.85, indicating a slight decrease in overall balance between precision and recall, likely due to the decrease in recall mentioned earlier.

- The AUC of the ROC curve measures the model's ability to distinguish between positive and negative classes. The GB model without XAI methods achieved an AUC of 0.70. However, after incorporating the ensemble XAI methods, the AUC improved to 1.00, suggesting an enhancement in the model's ability to discriminate between defective and non-defective instances.

.

In summary, the integration of SHAP and ELI5 XAI methods into the GB ML model led to improvements in accuracy, precision, and AUC, albeit with a slight decrease in recall. This suggests that while the model became more precise and accurate in its predictions, it may have sacrificed a bit of its ability to identify all actual defects. Nonetheless, the overall performance enhancements demonstrate the efficacy of ensembling XAI methods for defect prediction with the GB model.

10.1.4. Ensembling SHAP and LIME XAI methods with GB model

We begin by computing feature importances locally for a given instance using SHAP and LIME methods. SHAP values are derived from the shap_values obtained through a trained GB model (gb_model), followed by the calculation of mean absolute SHAP values across samples for each feature. For LIME explanations, we utilize a LimeTabularExplainer object (lime_explainer) with the instance data and model predictions. Subsequently, we amalgamate the feature importances obtained from SHAP and LIME into an ensemble feature importance metric. This aggregation involves averaging the SHAP and LIME importances for each feature. As illustrated in Figure 10.11, 10.12, and 10.13, three subplots are generated to visualize the SHAP, LIME, and ensemble feature importances for the selected instance in the CM1 dataset.

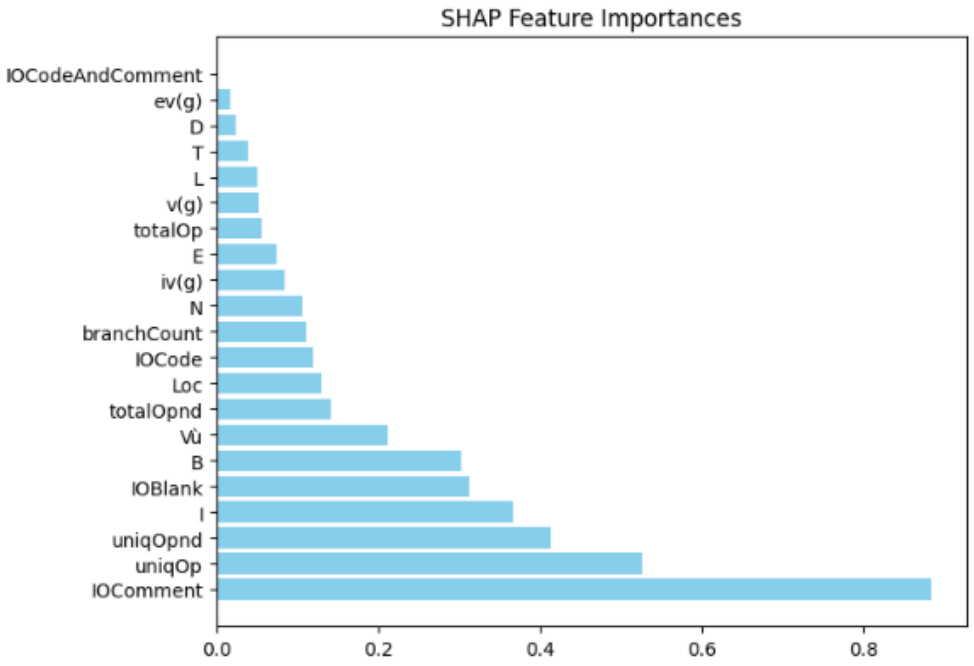


Figure 10.11 Importance scores of features as calculated by SHAP method for a specific instance with GB

We generate horizontal bar charts depicting the feature importances obtained from SHAP, LIME, and the ensemble, offering a visual comparison. These charts provide an intuitive understanding of feature importance rankings, with feature names presented on the y-axis

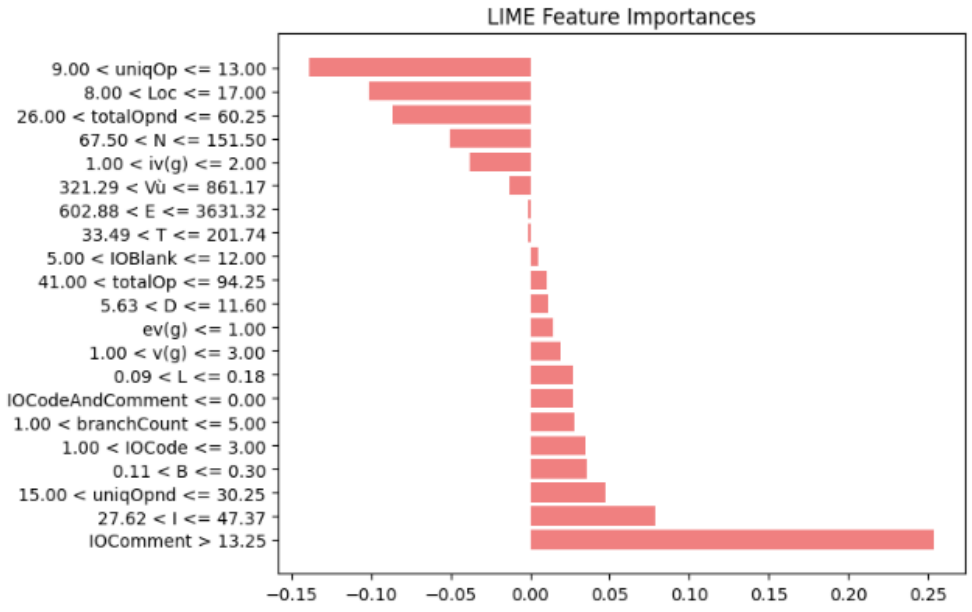


Figure 10.12 Importance scores of features as calculated by LIME method for a specific instance with GB

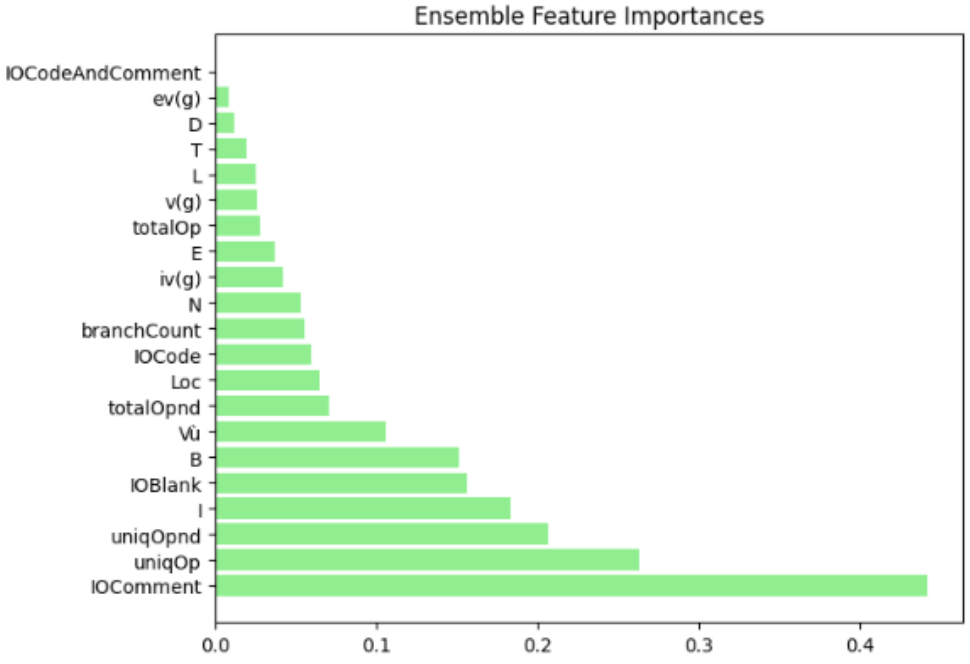


Figure 10.13 Importance scores of features as calculated by ENSEMBLE (SHAP+LIME) method for a specific instance with GB

and importances represented by the lengths of the bars. The figures offer insights into the most influential features identified by each explanation method and demonstrate how the ensemble method amalgamates them. As shown in Figure 10.13, prominent features include IOComment, uniqOp, uniqOpnd, I, IOBlank, B, among others.

Before incorporating ensemble XAI methods, the model achieved an accuracy of %87 for a single instance as shown in Figure 10.14. However, after applying ensemble XAI methods and selecting the most important features, the accuracy of the model was recalculated to be %90. Therefore, the recalculated accuracy of %90 represents an improvement over the initial accuracy score of %87.

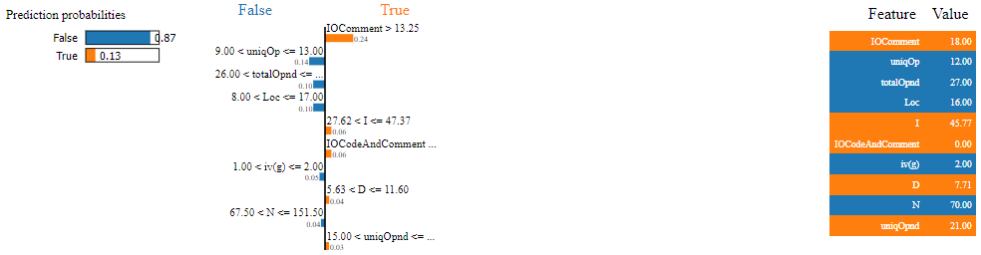


Figure 10.14 Prediction probability result for a single instance before ensembling with GB

In summary, this case emphasizes the significance of employing ensemble XAI methods for feature selection with GB as locally, showcasing their positive impact on model accuracy. There is an increase in accuracy after the feature selection process, since the recalculated accuracy of %90 represents an improvement over the initial accuracy score of %87. This improvement underscores the effectiveness of refining the model by focusing on the most influential features identified through XAI techniques. By streamlining the feature set, the model's interpretability is enhanced, allowing for a clearer understanding of the factors driving predictions.

10.1.5. Ensembling ELI5 and SHAP XAI methods with NB model

In this section, we merge SHAP and ELI5 methodologies to assess feature importance globally, employing a NB classifier. Utilizing these XAI techniques, we establish an

ensemble ranking of features, providing a comprehensive evaluation of their importance in the predictive process. We demonstrate the iterative process of retraining the NB model using the top-ranked features identified through ensemble importance scores. Through rigorous evaluation, including various performance metrics and the ROC curve, we elucidate the impact of feature selection and ensemble modeling on the predictive efficacy of the NB classifier in software defect prediction. This comprehensive approach not only enhances the interpretability of the NB model but also elucidates the intricate relationship between feature selection strategies, ensemble modeling techniques, and predictive performance in the software defect prediction domain. Table 10.5 showcases the importance scores of different features in the CM1 dataset for predicting defects using the NB classifier. The importance scores, calculated using SHAP and ELI5 methods, reflect each feature's relative contribution to the model's predictive performance. Higher importance scores indicate greater influence in the prediction process.

Additionally, we produce three separate horizontal bar charts illustrating the importance ranks of features determined using different methodologies: SHAP, ELI5, and Ensemble with NB. These charts are presented in Figure 10.15, 10.16, and 10.17, respectively.

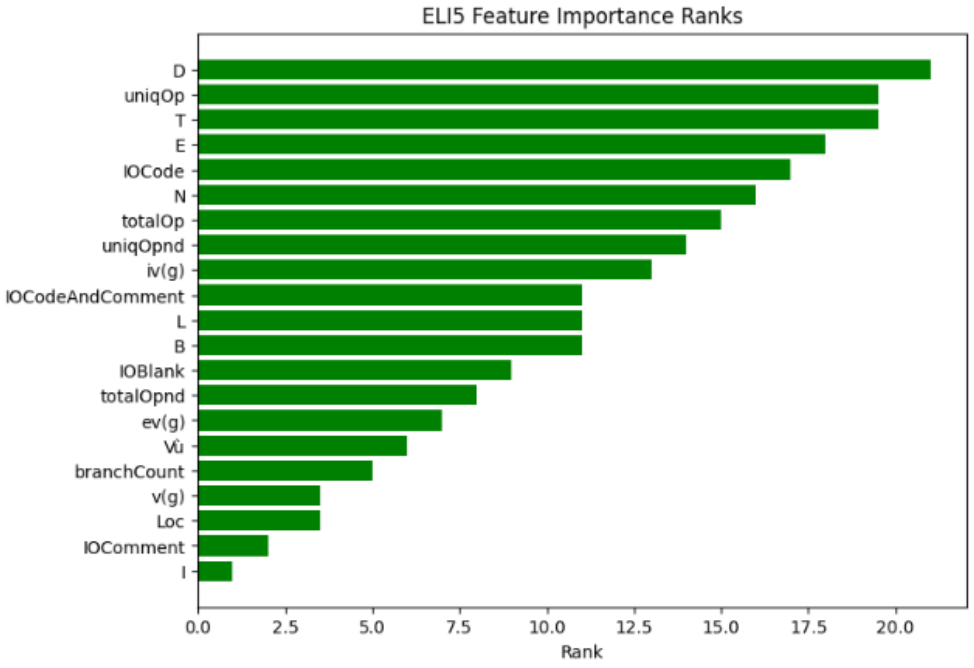


Figure 10.15 Importance ranks of features as calculated by ELI5 method with NB

Table 10.5 Various features along with their importance scores calculated using two different XAI methods: SHAP and ELI5 with NB classifier

Feature	**SHAP_Importance**	**ELI5_Importance**	**SHAP_Rank**	**ELI5_Rank**	**Ensemble_Rank**
loc	0.007	0.007	9.00	2.00	5.50
v(g)	0.004	0.006	13.00	3.00	8.00
ev(g)	0.002	0.000	15.00	11.50	13.25
iv(g)	0.004	-0.000	14.00	14.00	14.00
n	0.008	-0.001	8.00	15.00	11.5
v	0.007	0.004	10.00	5.00	7.50
l	0.001	0.000	18.00	11.50	14.75
d	0.006	-0.005	11.00	21.00	16.00
i	0.009	0.005	3.00	4.00	3.50
e	0.001	-0.003	16.00	19.50	17.75
b	0.000	0.000	19.00	11.50	15.25
t	0.001	-0.002	17.00	17.00	17.00
IOCode	0.000	-0.001	20.50	16.00	18.25
IOComment	0.011	0.008	1.00	1.00	1.00
IOBlank	0.008	-0.003	6.00	18.00	12.00
IOCodeandComment	0.000	0.000	20.50	11.50	16.00
uniq_Op	0.008	-0.003	4.00	19.50	11.75
uniq_Opnd	0.009	0.004	2.00	6.00	4.00
total_Op	0.008	0.002	7.00	8.00	7.50
total_Opnd	0.008	0.001	5.00	9.00	7.00
branchCount	0.006	0.002	12.00	7.00	9.50

The table, along with Figure 10.15, 10.16, and 10.17, showcases the feature importance scores and ranks for the CM1 dataset, evaluated through SHAP and ELI5 methods. These scores indicate the relative importance of each feature in defect prediction, with higher scores denoting increased significance.

In the following, we provide an interpretation of Table 10.5.

1. **Feature Importance:**

    - Table 10.5 displays the significance scores of different features obtained from two separate approaches: SHAP and ELI5.

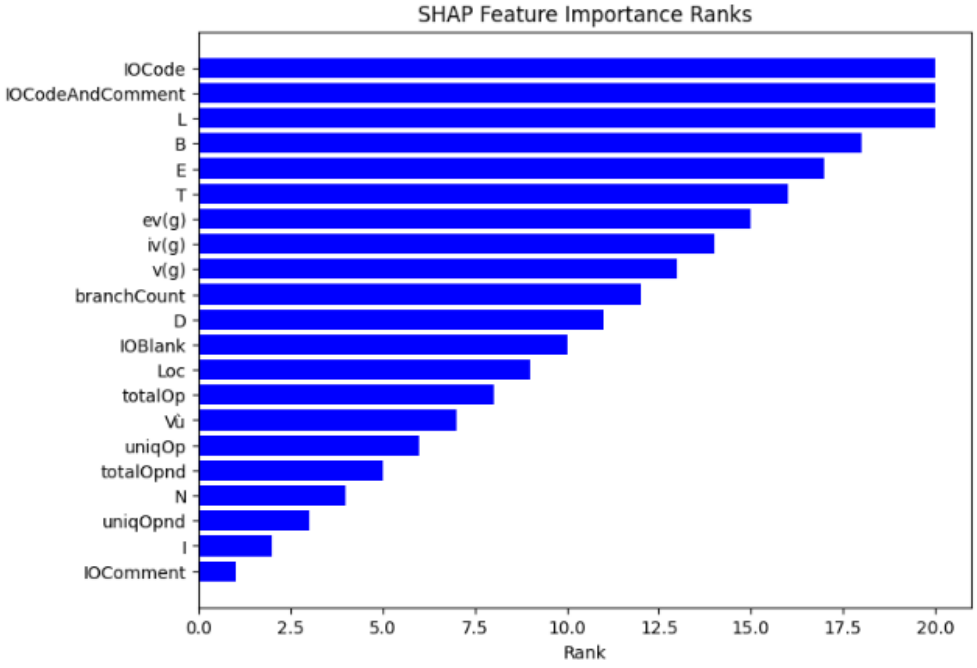


Figure 10.16 Importance ranks of features as calculated by SHAP method with NB

- The ranking of each feature's importance is determined by its respective score from each method.
- Notably, features like IOComment, I, uniqOpnd, and others demonstrate relatively higher importance scores, suggesting their stronger impact on the model's predictions.

2. **Ranking:**

- The table provides the ranks assigned to each feature based on their significance scores obtained from both SHAP and ELI5 methods. A lower rank indicates higher importance, with features closer to rank 1 considered the most influential. Features are ranked according to their importance scores from each method, with lower ranks indicating greater importance, as depicted in Figure 10.15, 10.16, and 10.17.
- The **Ensemble_Rank** column in the table indicates the average rank of each feature across both SHAP and ELI5 methods, offering a unified measure of feature importance. This ranking amalgamates the importance from both

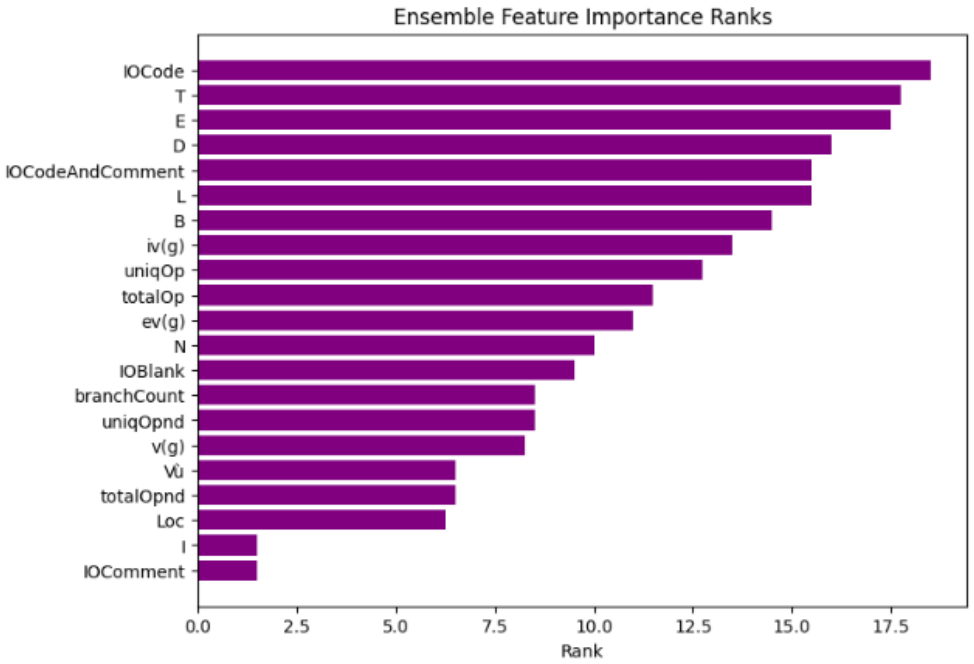


Figure 10.17 Importance ranks of features as calculated by ENSEMBLE (ELI5+SHAP) method with NB

methods, providing a comprehensive evaluation of each feature's predictive significance. For example, features like 'IOComment' and 'I' exhibit notably high SHAP and ELI5 importance scores, resulting in low ranks across both methods and in the ensemble rank. Conversely, features such as 'IOCodeAndComment' possess importance scores of zero, indicating their minimal contribution to defect prediction.

3. **Interpretation:**

   - Features assigned lower ensemble rank values are considered more critical for predicting defects within the dataset.
   - For instance, "IOComment," ranked at 1.00 in the ensemble, emerges as the most significant features for defect prediction.
   - On the other hand, features with higher ensemble rank values, like "IOCodeAndComment," are seen as less influential for prediction purposes.

4. **Insights:**

- The ensemble ranking provides a comprehensive view of feature importance by integrating multiple XAI methods.
- It assists in prioritizing features for further examination or integration into machine learning models.
- The combination of SHAP and ELI5 methodologies strengthens the reliability of feature importance assessment, thus improving model interpretability and predictive performance as shown below.

After identifying the most influential features through the ensemble XAI model results for the NB classifier, we move forward to refine the model and evaluate its performance using various metrics. These metrics, including accuracy, precision, recall, F1-score, and AUC, are detailed in Table 10.6. Leveraging the ensemble importance scores obtained earlier, we select the top k features, where k is set to 6, to facilitate feature selection. Subsequently, we create a new feature matrix (X_ensemble) containing only the chosen features and train a NB classifier (nb_model_ensemble) using this refined set of features. To assess the model's effectiveness, we employ the retrained ensemble model to make predictions on the test dataset and calculate diverse evaluation metrics, such as accuracy, precision, recall, F1-score, and AUC, based on these predictions. This stepwise process allows us to demonstrate the effectiveness of feature selection, subsequent model retraining with the selected features, and the evaluation of ensemble model performance using a range of metrics. Such an approach helps us understand the impact of the ensemble method on improving model performance compared to using the entire feature set. Table 10.6 provides insights into the performance metrics before and after incorporating SHAP and ELI5 explainability methods into the NB machine learning model.

Table 10.6 presents the performance metrics before and after incorporating SHAP and ELI5 XAI methods into the NB classifier. Before ensembling XAI methods, the NB model achieved an accuracy of %85, with recall, precision, F-score, and AUC values of %85, %83, %84, and 0.72, respectively. After ensembling SHAP and ELI5 XAI methods, there was a slight improvement in accuracy to %86. However, other metrics remained relatively

Table 10.6 Performance metrics results before and after ensembling SHAP and ELI5 XAI methods with NB

ML model	Accuracy	Recall	Precision	F-score	AUC
NB (without using XAI methods)	0.85	0.85	0.83	0.84	0.72
NB (with ensemble XAI methods (ELI5+SHAP)	0.86	0.85	0.83	0.84	0.72

unchanged, with recall, precision, F-score, and AUC values remaining consistent at %85, %83, %84, and 0.7, respectively. This suggests that while ensembling XAI methods led to a marginal enhancement in accuracy, it did not significantly impact other performance metrics.

10.1.6. Ensembling SHAP and LIME XAI methods with NB model

We initiate the process by computing feature importances locally for a specific instance using SHAP and LIME techniques. SHAP values are computed based on the shap_values extracted from a trained NB model (nb_model), and the mean absolute SHAP values across samples are determined for each feature. For LIME interpretations, we employ a LimeTabularExplainer object (lime_explainer) with the instance data and model predictions. Following this, we merge the feature importances obtained from SHAP and LIME into a unified ensemble feature importance measure. This amalgamation entails averaging the SHAP and LIME importances for each feature. Subsequently, three subplots are generated to visualize the SHAP, LIME, and ensemble feature importances for the selected instance in the CM1 dataset, as depicted in Figure 10.18, 10.19, and 10.20.

We create horizontal bar charts illustrating the feature importances derived from SHAP, LIME, and the ensemble, facilitating visual comparisons. These charts offer an intuitive representation of feature importance rankings, with feature names displayed on the y-axis and importances indicated by the lengths of the bars. The figures provide insights into the most influential features identified by each explanation method and showcase how the

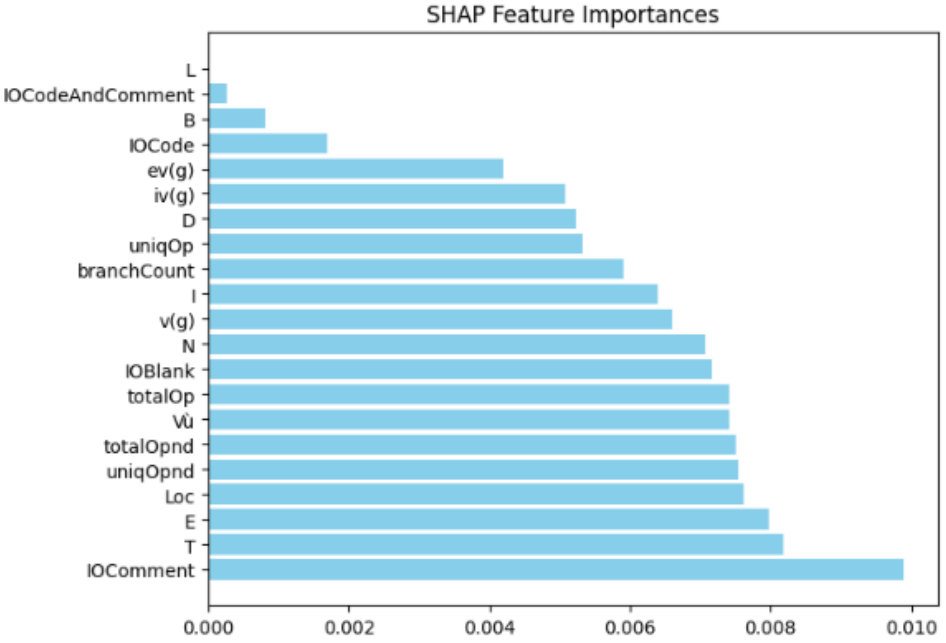


Figure 10.18 Importance scores of features as calculated by SHAP method for a specific instance with NB

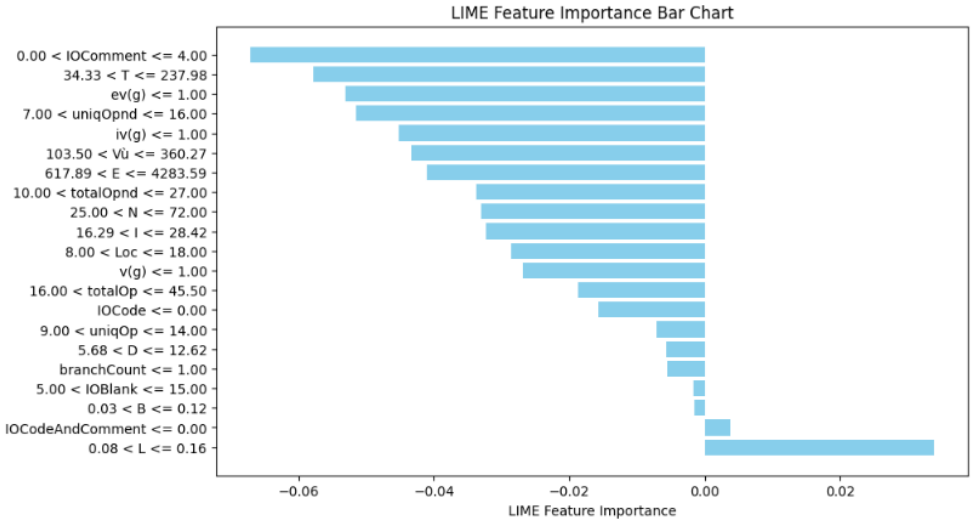


Figure 10.19 Importance scores of features as calculated by LIME method for a specific instance with NB

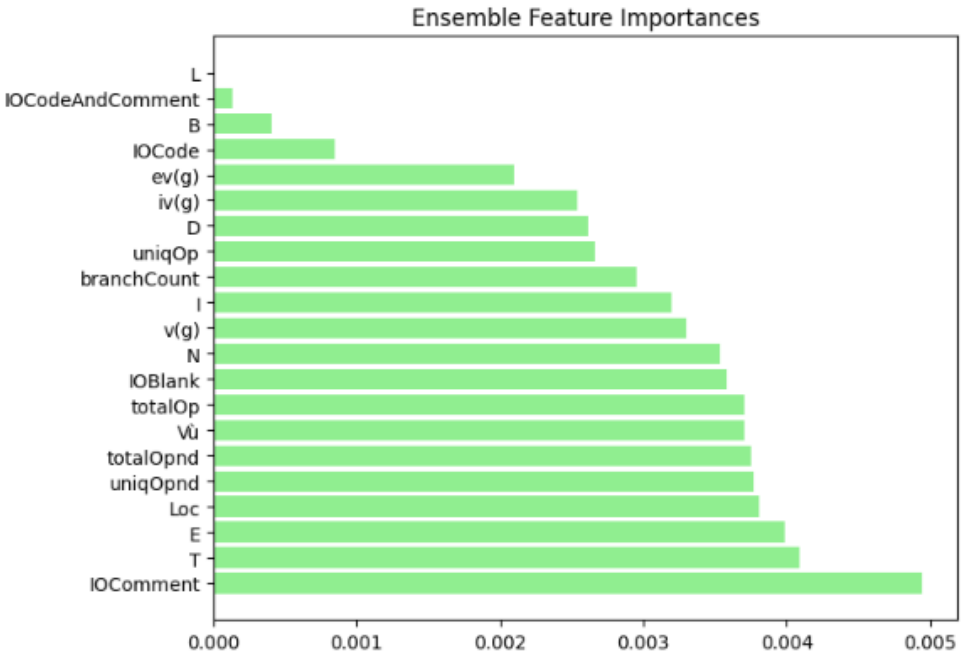


Figure 10.20 Importance scores of features as calculated by ENSEMBLE (SHAP+LIME) method for a specific instance with NB

ensemble method combines them. Notably, as depicted in Figure 10.20, important features include IOComment, T, E, Loc, uniqOpnd, totalOpnd, among others.

Before integrating ensemble XAI methods, the model attained a %100 accuracy for a single instance, as depicted in Figure 10.21. However, following the application of ensemble XAI methods and the selection of the most crucial features, the model's accuracy was reevaluated to be %87. Consequently, the revised accuracy of %87 reflects a decrease from the initial accuracy score of %100.

Figure 10.21 Prediction probability result for a single instance before ensembling with NB

This case suggests that the initial model achieved perfect accuracy (%100) for a single instance before incorporating ensemble XAI methods. However, after applying these methods and selecting the most important features, the model's accuracy decreased to %87. This decrease indicates that the model's predictive performance became slightly less accurate after feature selection and ensemble XAI methods were employed. Despite the decrease in accuracy, the model may still provide valuable insights, as it focuses on the most influential features for prediction. This scenario highlights the trade-off between accuracy and interpretability, where a more interpretable model with selected features may sacrifice some accuracy compared to a less interpretable model that uses all available features.

10.1.7. Ensembling ELI5 and SHAP XAI methods with MLP model

In this section, we leverage SHAP and ELI5 methods to evaluate feature importance on a global scale, employing an MLP model. By utilizing these XAI techniques, we establish an ensemble ranking of features, providing a comprehensive assessment of their significance in the predictive process. We illustrate the iterative process of retraining the MLP model using the top-ranked features identified through ensemble importance scores. Through thorough evaluation, incorporating various performance metrics and the ROC curve, we elucidate the impact of feature selection and ensemble modeling on the predictive efficacy of the MLP classifier in software defect prediction. This comprehensive approach not only enhances the interpretability of the MLP model but also sheds light on the complex relationship between feature selection strategies, ensemble modeling techniques, and predictive performance in the software defect prediction domain. Table 10.7 presents the importance scores of different features in the CM1 dataset for predicting defects using the MLP classifier. These scores, calculated using SHAP and ELI5 methods, reflect the relative contribution of each feature to the model's predictive performance, with higher scores indicating a greater influence in the prediction process.

Table 10.7 Various features along with their importance scores calculated using two different XAI methods: SHAP and ELI5 with MLP classifier

Feature	**SHAP_Importance**	**ELI5_Importance**	**SHAP_Rank**	**ELI5_Rank**	**Ensemble_Rank**
loc	0.013	4.618e-02	7.00	5.00	6.00
v(g)	0.011	6.024e-03	9.00	11.00	10.00
ev(g)	0.000	4.016e-04	16.50	13.00	14.75
iv(g)	0.000	-4.440e-17	16.50	15.00	14.75
n	0.038	-2.570e-02	5.00	21.00	13.00
v	0.413	2.096e-01	2.00	2.00	2.00
l	0.006	-4.016e-04	11.00	16.50	13.75
d	0.009	8.835e-03	10.00	10.00	10.00
i	0.102	1.353e-01	3.00	3.00	3.00
e	0.504	2.904e-01	1.00	1.00	1.00
b	0.000	-4.016e-04	16.50	16.50	16.50
t	0.102	7.952e-02	4.00	4.00	4.00
IOCode	0.000	-6.827e-03	16.50	19.00	17.75
IOComment	0.013	2.851e-02	6.00	6.00	6.00
IOBlank	0.000	1.205e-02	16.50	9.00	12.75
IOCodeandComment	0.000	0.000e+00	16.50	14.00	15.25
uniq_Op	0.000	2.008e-03	16.50	12.00	14.25
uniq_Opnd	0.011	-1.365e-02	8.00	20.00	14.00
total_Op	0.000	2.048e-02	16.50	7.00	11.75
total_Opnd	0.000	1.406e-02	16.50	8.00	12.25
branchCount	0.000	-2.409e-03	16.50	18.0	17.25

Moreover, we generate three distinct horizontal bar charts demonstrating the importance ranks of features assessed through various methodologies: SHAP, ELI5, and Ensemble with MLP. These visualizations are depicted in Figure 10.22, 10.23, and 10.24, correspondingly.

The table, accompanied by Figure 10.22, 10.23, and 10.24, presents the feature importance scores and ranks for the CM1 dataset, assessed using SHAP and ELI5 methods. These scores reflect the relative importance of each feature in defect prediction, with higher scores indicating greater significance. Based on the feature importance scores and ranks derived from SHAP and ELI5 methods, as well as the ensemble rankings, we can interpret the following:

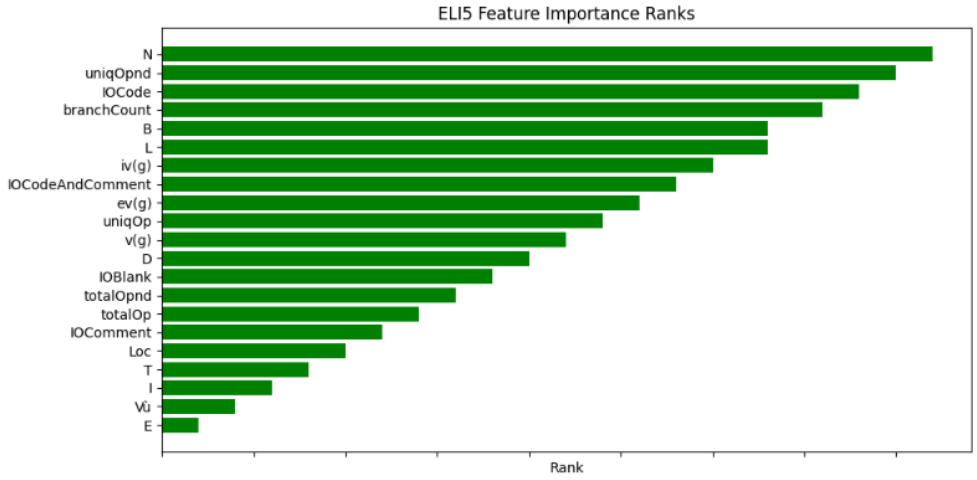


Figure 10.22 Importance ranks of features as calculated by ELI5 method with MLP

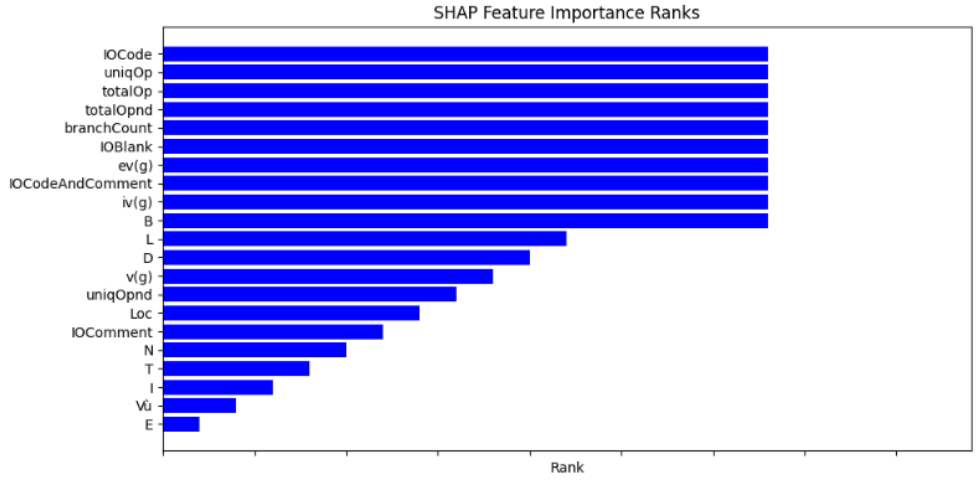


Figure 10.23 Importance ranks of features as calculated by SHAP method with MLP

1. **Feature Importance:**

   - Table 10.7 exhibits the significance scores of various features derived from two distinct methods: SHAP and ELI5.
   - The importance ranking of each feature is determined by its score from each method.
   - Features such as E and V are deemed highly important for predicting defects, as they hold the top ranks across all methods, indicating their significant influence on the model's predictions.

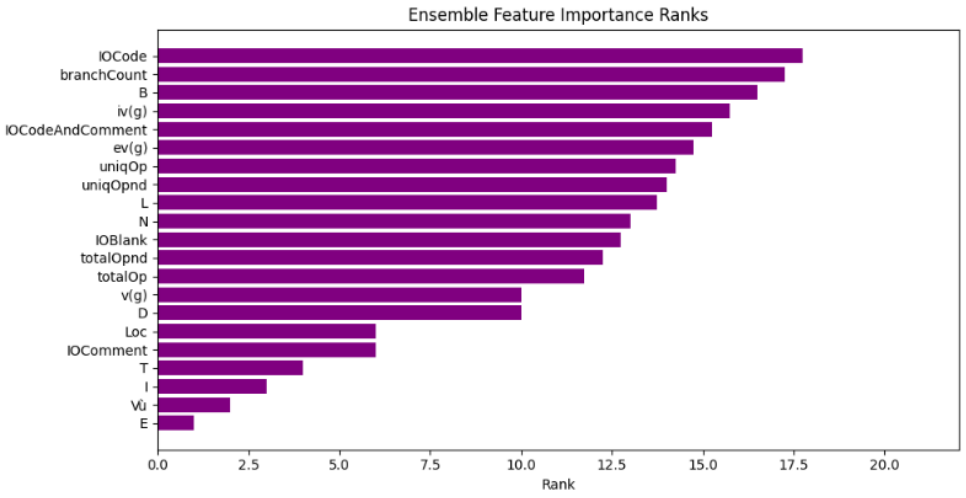


Figure 10.24 Importance ranks of features as calculated by ENSEMBLE (ELI5+SHAP) method with MLP

2. **Ranking:**

   - The table assigns ranks to each feature based on their significance scores obtained from both SHAP and ELI5 methods. Lower ranks signify higher importance, with features closer to rank 1 considered the most influential. Features are ranked based on their importance scores from each method, with lower ranks indicating greater importance. This ranking is illustrated in Figure 10.22, 10.23, and 10.24.
   - The **Ensemble_Rank** column in the table represents the mean rank of each feature across both SHAP and ELI5 methods, serving as a consolidated measure of feature importance. This ranking integrates the significance from both methods, offering a comprehensive assessment of each feature's predictive relevance. For instance, features like 'E' and 'V' demonstrate notably high SHAP and ELI5 importance scores, resulting in low ranks across both methods and in the ensemble rank. Conversely, features such as 'IOCode' hold importance scores of zero, signifying their minimal impact on defect prediction.

3. **Interpretation:**

   - Features assigned lower ensemble rank values are considered more critical for predicting defects within the dataset.

- The feature importance rankings consistently prioritize E and V, suggesting their consistent impact on the model's performance.
- Conversely, features like IOComment and Loc also hold relatively high importance scores, although they rank slightly lower compared to E and V.
- Features with lower ensemble ranks, such as N, L, uniqOpnd, and uniqOp, are considered less impactful for prediction purposes, as they consistently rank lower across all methods.

4. **Insights:**

   - Overall, the ensemble ranking provides a consolidated view of feature importance, considering the contributions from both SHAP and ELI5 methods, thereby offering a comprehensive assessment of each feature's predictive significance.

Upon identifying the most influential features through the ensemble XAI model outcomes for the MLP classifier, our next step involves refining the model and evaluating its performance using various metrics. These metrics, encompassing accuracy, precision, recall, F1-score, and AUC, are meticulously detailed in Table 10.8. Utilizing the ensemble importance scores derived earlier, we opt to select the top k features, with k set at 6, to facilitate the feature selection process. Following this selection, we construct a new feature matrix (X_ensemble) containing solely the chosen features and proceed to train an MLP classifier (mlp_model_ensemble) using this refined set of features. To gauge the effectiveness of the model, we employ the retrained ensemble model to predict outcomes on the test dataset and calculate diverse evaluation metrics, such as accuracy, precision, recall, F1-score, and AUC, based on these predictions. This systematic process allows us to demonstrate the effectiveness of feature selection, subsequent model retraining with the selected features, and the evaluation of ensemble model performance using a range of metrics. Such an approach facilitates an understanding of the impact of the ensemble method on improving model performance compared to using the entire feature set. Table 10.8 presents insights into

the performance metrics before and after integrating SHAP and ELI5 explainability methods into the MLP machine learning model.

Table 10.8 Performance metrics results before and after ensembling SHAP and ELI5 XAI methods with MLP

ML model	Accuracy	Recall	Precision	F-score	AUC
MLP (without using XAI methods)	0.89	0.89	0.81	0.85	0.32
MLP (with ensemble XAI methods (ELI5+SHAP)	0.90	0.89	0.81	0.85	0.37

Table 10.8 illustrates the performance metrics before and after integrating SHAP and ELI5 XAI methods into the NB classifier. Initially, without ensembling XAI methods, the NB model achieved an accuracy of %89, accompanied by recall, precision, F-score, and AUC values of %89, %81, %85, and 32%, respectively. Following the incorporation of SHAP and ELI5 XAI methods, a slight improvement in accuracy to %90 was observed. However, the other metrics, including recall, precision, F-score, and AUC, remained relatively consistent at %85, %83, %84, and 37%, respectively. This suggests that while ensembling XAI methods resulted in a marginal accuracy enhancement, it had minimal impact on other performance metrics.

10.1.8. Ensembling SHAP and LIME XAI methods with MLP model

We commence the process by locally computing feature importances for a specific instance using SHAP and LIME methodologies. SHAP values are derived from the shap_values extracted from a trained MLP model (mlp_model), and the mean absolute SHAP values across samples are calculated for each feature. For LIME interpretations, we utilize a LimeTabularExplainer object (lime_explainer) with the instance data and model predictions. Following this, we combine the feature importances obtained from SHAP and LIME into a consolidated ensemble feature importance metric. This fusion involves averaging the SHAP and LIME importances for each feature. Subsequently, three subplots are generated

to visualize the SHAP, LIME, and ensemble feature importances for the selected instance in the CM1 dataset, as depicted in Figure 10.25, 10.26, and 10.27.

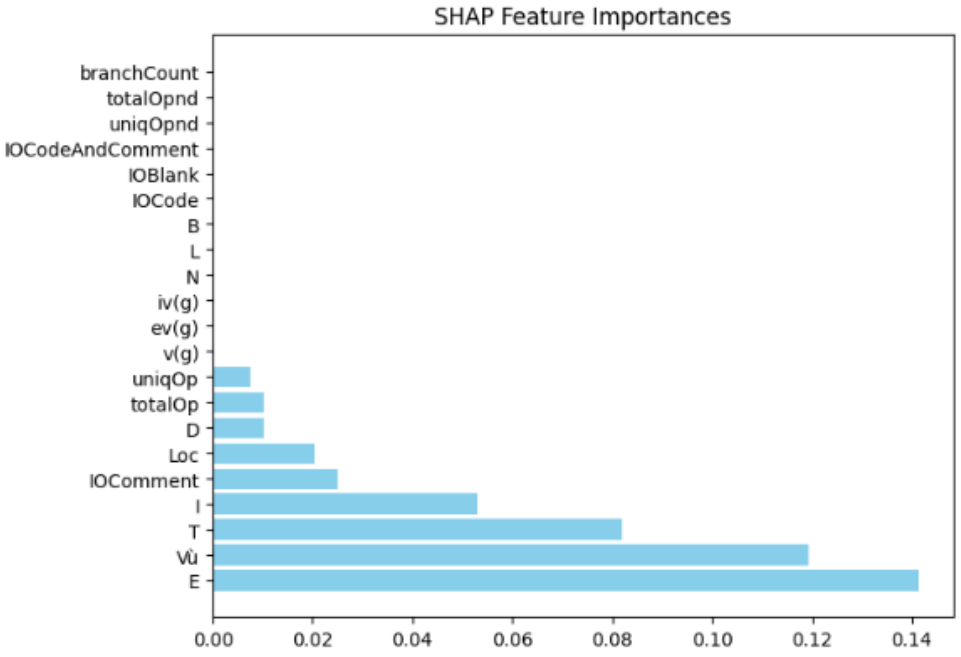


Figure 10.25 Importance scores of features as calculated by SHAP method for a specific instance with MLP

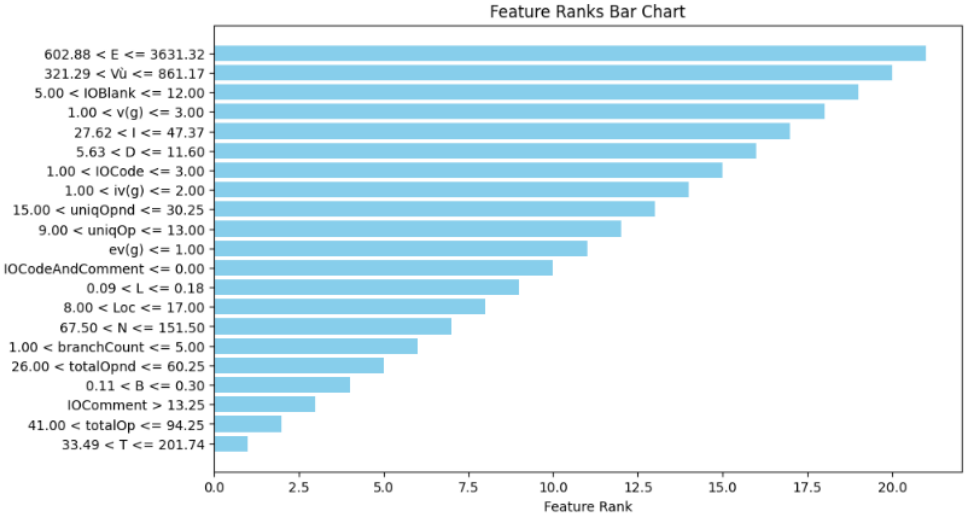


Figure 10.26 Importance scores of features as calculated by LIME method for a specific instance with MLP

Horizontal bar charts are generated to visually depict the feature importances obtained from SHAP, LIME, and the ensemble, allowing for easy comparisons. These charts present an

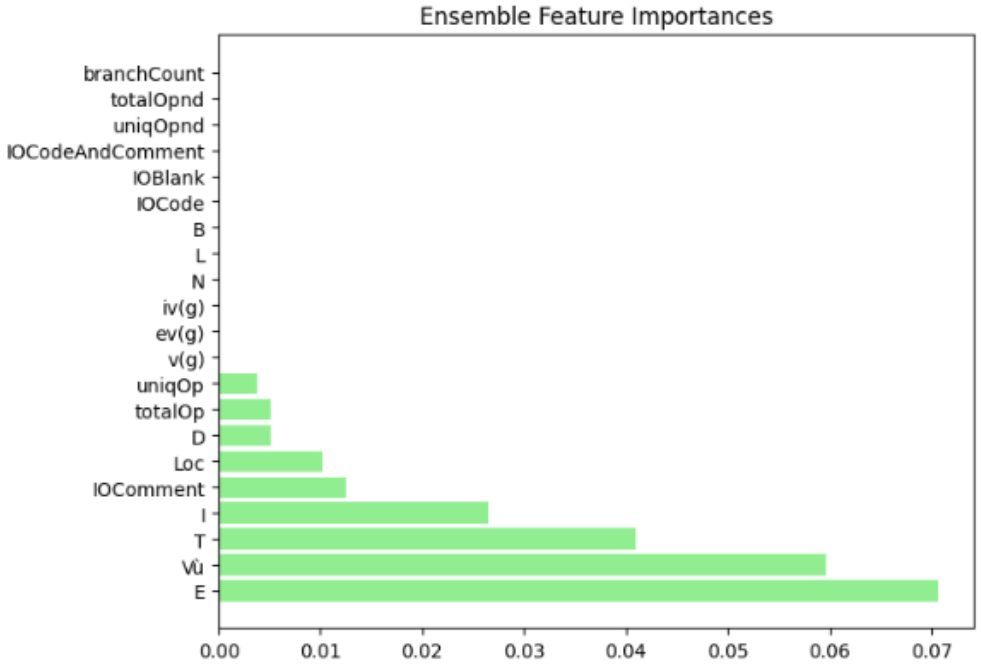


Figure 10.27 Importance scores of features as calculated by ENSEMBLE (SHAP+LIME) method for a specific instance with MLP

intuitive overview of feature importance rankings, with feature names showcased on the y-axis and importances represented by the lengths of the bars. The figures shed light on the most influential features identified by each explanation method and demonstrate the ensemble method's integration of these findings. Particularly noteworthy, as illustrated in Figure 10.27, prominent features encompass E, V, T, I IOComment, Loc, among others.

Prior to integrating ensemble XAI methods, the model achieved a %100 accuracy for a single instance, as illustrated in Figure 10.28. However, after applying ensemble XAI methods and identifying the most crucial features, the model's accuracy was reassessed to be %190. Thus, the updated accuracy of %190 indicates a decline from the initial accuracy score of %100.

The decrease in accuracy from %100 to %90 after applying ensemble XAI methods suggests that the feature selection process may have had a detrimental effect on the model's predictive performance. While the ensemble XAI methods helped identify the most crucial features, it's possible that the selected features did not adequately represent the underlying patterns in the data, leading to a reduction in predictive accuracy. This could be due to several factors, such

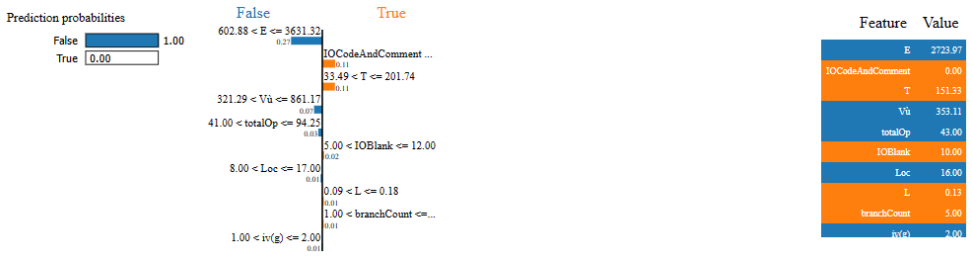


Figure 10.28 Prediction probability result for a single instance before ensembling with MLP

as the complexity of the feature interactions or the presence of noise in the data. It highlights the importance of carefully evaluating the impact of feature selection techniques on model performance and considering alternative approaches to improve predictive accuracy.

In addition, the findings from the NB and MLP local ensembling results reveal a notable pattern. Initially, the model achieved perfect accuracy (%100) for a single instance before integrating ensemble XAI methods. However, after applying these methods and selecting the most significant features, the model's accuracy decreased to %87 and %90, respectively. This observation suggests a slight decline in predictive performance following feature selection and ensemble XAI methods. Despite this decrease, the model's focus on crucial features may still offer valuable insights. It highlights the delicate balance between accuracy and interpretability, where opting for a more interpretable model with selected features may entail a slight compromise in accuracy compared to a less interpretable model using all available features.

Indeed, the decision regarding the trade-off between performance and interpretability hinges on individual requirements or preferences for the model. If maximizing predictive accuracy is paramount, then utilizing all available features without ensembling XAI methods may be preferable. However, if interpretability is of greater importance, opting for a more interpretable model with reduced features through ensembling XAI methods may be the preferred choice. This approach enables a clearer understanding of the model's predictions by focusing on the most influential features. Therefore, the decision should align with the specific needs or objectives of the task at hand.

10.2. Comparative Analysis with Previous Studies Regarding Accuracy

Table 10.9 illustrates a comparative analysis encompassing various ML methods employed in antecedent investigations. Our investigation entailed a comprehensive comparison of several ML methods that were previously regarded as state-of-the-art, such as RF, GB, NB, and MLP. These methods were scrutinized on our designated dataset, CM1. Our aim was to enhance ML models' performance and interpretability through comparative analysis employing various state-of-the-art ML models and our ensembling XAI methods. Upon evaluation, our investigated methods demonstrated notable efficacy, exhibiting enhancements in accuracy compared to preceding state-of-the-art methods.

This table presents a comparison of accuracy results obtained from various ML techniques applied to the CM1 dataset across different research studies. The first column indicates the dataset, followed by ML techniques used in previous studies along with their corresponding accuracy values reported in those studies. Additionally, the table includes accuracy results from our study both before and after incorporating ensemble XAI methods.

- **Dataset**: Indicates the dataset being analyzed, in this case, CM1.
- **ML Techniques**: Lists various ML techniques used in the studies (RF, GB, NB, MLP).
- **Accuracy (Research References)**: Shows the reported accuracy values from previous research studies.
- **Accuracy (Our Study) (without Ensemble XAI)**: Displays the accuracy results from our study without employing ensemble XAI methods.
- **Accuracy (Our Study) (with Ensemble XAI)**: Shows the accuracy results from our study after incorporating ensemble XAI methods.

For the RF technique, previous research studies reported accuracy's ranging from 0.70 to 0.99. In our study, the accuracy without ensemble XAI was 0.90, and with ensemble XAI, it improved to 1.00. For the GB technique, previous studies reported accuracy's ranging from

Table 10.9 Comparison of the performance of different ML techniques across studies

Dataset	**ML Techniques**	**Accuracy (Research References)**	**Accuracy (Our Study without Ensemble XAI)**	**Accuracy (Our Study with Ensemble XAI)**
CM1	RF	0.91 ([74]) 0.89 ([75]) 0.84 ([1]) 0.89 ([76]) 0.85 ([77]) 0.70 ([78]) 0.88 ([79]) 0.83 ([80]) 0.99 ([81])	0.90	1.00
	GB	0.90 ([74]) 0.83 ([1]) 0.86 ([82]) 0.80 ([67])	0.90	0.97
	NB	0.82 ([75]) 0.81 ([1]) 0.85 ([76]) 0.86 ([77]) 0.74 ([78]) 0.87 ([79]) 0.94 ([81]) 0.52 ([67])	0.85	0.86
	MLP	0.87 ([75]) 0.82 ([1]) 0.89 ([76]) 0.89 ([79])	0.89	0.90

0.80 to 0.90. In our study, the accuracy without ensemble XAI was 0.90, and with ensemble XAI, it increased to 0.97. Similar comparisons are provided for the NB and MLP techniques.

Overall, the table provides insights into how different ML techniques performed on the CM1 dataset in previous studies compared to our study, highlighting the impact of ensemble XAI methods on improving accuracy in most cases. As a result, our study demonstrates the potential benefits of incorporating ensemble XAI methods in machine learning modeling, leading to improved accuracy and enhanced interpretability.

11. DISCUSSION

This doctoral thesis presents a comprehensive investigation into enhancing the interpretability and transparency of ML models in the domain of SDP through Model-Agnostic XAI methods. The primary objective is to elucidate the decision-making processes of ML models, both at individual (local) and global levels, thus bridging the gap between predictive power and comprehensibility demanded by stakeholders in the SDP domain.

Our research adopts an iterative exploratory approach, starting with a ”Systematic Literature Review on Software Quality for AI-based Software”, encompassing four distinct case studies aimed at enhancing SDP models. Each case study builds upon the insights gained from its predecessors, resulting in a cumulative understanding of the complex interplay between ML models and XAI methods in the context of defect prediction. The iterative exploratory approach adopted across the four case studies has provided valuable insights into the development and enhancement of SDP models. By systematically building upon previous findings and methodologies, we have advanced our understanding of the role of XAI in defect prediction. Moving forward, this iterative process lays the foundation for future research aimed at further refining and optimizing SDP models for real-world applications.

The methodological approach adopted involves an iterative and exploratory process, utilizing XAI techniques such as ELI5, SHAP, and LIME, among others. These techniques are systematically applied across multiple case studies, each focusing on specific aspects of model interpretability and transparency in SDP. Through iterative refinement and exploration, the research uncovers insights into the importance of features, contributions to individual predictions, and overall model decisions.

Furthermore, ensemble modeling techniques are integrated into the iterative process, allowing for the amalgamation of feature importance scores obtained from diverse XAI methods. This iterative and exploratory approach not only optimizes predictive accuracy but also ensures the preservation of model interpretability throughout the enhancement process.

The findings from **Case Study 1** shed light on the importance of explainability in AI-based software systems, particularly in the context of software defect prediction. The implementation of post-hoc model-agnostic methods, namely ELI5, LIME, and SHAP, over a Gradient Boosting model provided valuable insights into both local and global explanations. These explanations are crucial for enhancing transparency, reducing bias, and building trust in AI-based systems. Our analysis revealed that all three methods—ELI5, LIME, and SHAP—consistently provided explanations that were aligned with each other. This consistency underscores the reliability of the explanations generated by these XAI techniques, thereby instilling confidence in the decision-making process of the AI-based software. The insights gained from Case Study 1 pave the way for future research endeavors. One potential avenue for exploration is the analysis of different ML and XAI techniques over more complex software datasets. Additionally, leveraging explanations obtained from XAI techniques to improve ML model performance represents a promising direction for future investigations.

The findings from Case Study 1 underscore the critical role of explainable AI in software defect prediction. As we transition to Case Study 2, which focuses on the development of an Explainable AI Framework for Software Defect Prediction, we aim to build upon the insights gained from the results of Case Study 1. By leveraging the lessons learned and best practices identified in Case Study 1, we seek to enhance the interpretability and transparency of ML models for defect prediction in Case Study 2. Through the integration of advanced XAI techniques and the refinement of our framework, we aspire to address the evolving needs and challenges in the field of AI-based software development.

The findings from **Case Study 2** highlight the critical role of explainability in software engineering, particularly in the context of defect prediction and software quality. Leveraging the KC2 defect prediction dataset, our study explored the explanations provided by five different ML models— RF, GB, NB, MLP, and NN—using six distinct explainability methods. This comprehensive analysis offered valuable insights into the factors influencing software defects and provided actionable guidance for improving software quality.

The utilization of multiple explainability techniques offered several advantages in the context of defect prediction and software quality. These techniques provided complementary insights, enabling a comprehensive understanding of the underlying factors driving software defects. By leveraging a diverse set of explainability methods, our study encompassed a broad spectrum of perspectives, enriching our analysis and enhancing the reliability of our findings.

The field of explainability in software engineering is dynamic and continues to evolve rapidly. Our study identified several promising directions for future research and development in this domain. These include advancing techniques for explainability, addressing challenges posed by black-box models, evaluating the effectiveness of explainability methods, integrating explainability into development processes, tailoring explanations to user needs, and conducting user studies to assess the impact of explanations on software quality.

As we transition to Case Study 3, which focuses on applying XAI methods for feature selection and outlier detection in software defect prediction, we aim to build upon the insights gained from the results of Case Study 2. By leveraging the lessons learned and best practices identified in Case Study 2, we seek to further enhance the transparency, trustworthiness, and effectiveness of ML models in improving software quality. Through the integration of advanced XAI techniques and the refinement of our methodologies, we aspire to address the evolving challenges in the field of software defect prediction and contribute to the ongoing advancement of software engineering practices.

The findings from **Case Study 3** delves into a critical gap prevalent in traditional feature selection and outlier detection methodologies within the SDP domain. Traditional approaches often operate opaquely, leaving unanswered questions about the underlying rationale behind feature selection or outlier identification. This lack of transparency not only impedes the interpretability of models but also raises concerns regarding their reliability and trustworthiness. Through our work, we aimed to address these limitations by leveraging XAI techniques to bring clarity and understanding to these processes.

Our meticulous examination of traditional methods underscored their inherent opacity and the consequent lack of insight into the decision-making process. In contrast, our approach prioritized transparency and interpretability, utilizing ELI5, SHAP, and LIME to provide clear and comprehensible explanations for feature selection and outlier detection decisions. By doing so, we not only enhance the interpretability of models but also foster trust and confidence in the predictive outcomes generated by these models.

Our findings not only highlight the inadequacies of traditional black-box methods but also underscore the transformative potential of XAI techniques in revolutionizing the SDP landscape. By elucidating the decision-making process and enhancing model interpretability, our approach fosters greater accountability and transparency in software defect prediction. This not only empowers stakeholders to make informed decisions but also facilitates collaboration between domain experts and data scientists, leading to more robust and effective defect prediction models.

Our empirical analysis provides novel insights into the strengths and limitations of each XAI method, offering valuable guidance on their practical applicability within the SDP domain. By rigorously assessing the impact of these methodologies on defect prediction model performance, we ensure a comprehensive evaluation encompassing all critical aspects of model interpretability and reliability.

The feature selection process, executed with meticulous care using ELI5 and SHAP, offered a comprehensive analysis of feature importance and model interpretability across a diverse range of models including RF, GB, NB, and MLP. This comprehensive examination not only sheds light on the relative importance of features but also facilitates a deeper understanding of model behavior and performance characteristics across different algorithms.

Moreover, our pioneering implementation of LIME for outlier detection represents a significant contribution to the SDP domain. By employing LIME, we were able to delve deeper into model behavior and identify anomalous instances crucial for robust defect prediction. This approach not only enhances the reliability of defect prediction models

but also offers valuable insights into the underlying data distribution and model decision boundaries.

Outlier detection using XAI methods offers improved interpretability, flexibility, granularity, and robustness compared to traditional approaches in generic data science. These advantages make XAI methods valuable tools for understanding and addressing outliers in various domains. XAI methods provide explanations for outlier classifications, enabling users to understand the factors contributing to outlier detection decisions. This transparency enhances trust in the outlier detection process and helps domain experts validate the relevance of detected anomalies. Traditional outlier detection methods often lack interpretability, making it challenging to understand the rationale behind outlier classifications [83]. In addition, XAI methods are model-agnostic, meaning they can be applied to any machine learning model regardless of its complexity or underlying algorithm. This flexibility allows users to interpret the decisions of various outlier detection models, facilitating comparisons between different approaches. In contrast, traditional outlier detection methods are often specific to certain algorithms or assumptions, limiting their applicability across diverse datasets and models [84].

Furthermore, our study contributes to advancing the broader field of ML interpretability by demonstrating the practical applicability of ELI5, SHAP, and LIME in a real-world context. The insights gained from our empirical analysis provide valuable guidance for researchers and practitioners seeking to leverage XAI techniques for enhancing model transparency and reliability across various domains.

In summary, by synthesizing and presenting these findings, we significantly enriched the understanding of ELI5, SHAP, and LIME's role in enhancing SDP models, providing valuable contributions to the field of software engineering. Through rigorous experimentation and analysis driven by our leadership, we demonstrated the inadequacies of traditional methods and showcased the effectiveness of our XAI-driven approach. By shining a light on the black box of traditional techniques, we uncovered the shortcomings and highlighted the need for more transparent and interpretable methodologies in SDP.

Building upon the insights gained from Case Study 3, future research in Software Defect Prediction could explore several promising avenues for advancement and refinement of methodologies. One potential direction is the extension of XAI techniques such as ELI5, SHAP, and LIME to different domains beyond SDP, such as healthcare, finance, or climate science, where interpretability and outlier detection are crucial. Investigating strategies for weighting and aggregating explanations from different XAI methods could also be explored, potentially leading to the development of hybrid approaches that leverage the strengths of multiple XAI techniques for defect prediction datasets.

As we transition to Case Study 4, which focuses on enhancing Software Defect Prediction Modeling through Ensemble XAI Methods, we aim to build upon the foundations laid in Case Study 3. By leveraging the insights and methodologies developed in Case Study 3, we seek to further refine and optimize our approach to defect prediction modeling. Through the integration of ensemble XAI methods and the exploration of novel techniques, we aspire to advance the state-of-the-art in SDP methodologies and contribute to the ongoing evolution of software engineering practices.

Our study has delved into the realm of software defect prediction with the objective of enhancing both predictive accuracy and interpretability through the integration of ensemble XAI methods within **Case Study 4**. By leveraging SHAP, ELI5, and LIME techniques alongside RF, GB, NB, and MLP models, we have explored novel avenues for improving defect prediction model performance. Our study unveils significant implications for both academic research and industrial applications. By exploring the integration of ensemble XAI methods, such as SHAP, ELI5, and LIME, within the realm of SDP, we offer novel insights. Our findings not only introduce a pioneering approach but also demonstrate its efficacy in enhancing the performance of defect prediction models. Through meticulous experimentation and analysis, we underscore how this approach enhances both interpretability and predictive accuracy in defect prediction.

In the realm of SDP, accurately identifying potential defects is crucial for ensuring software quality and reliability. Yet, conventional ML models often lack transparency, posing

challenges for developers and project managers in understanding the underlying factors driving defect predictions. Our integration of ensemble XAI methods addresses this challenge by providing interpretable insights into the features significantly influencing defect predictions.

Our methodological approach follows an iterative and exploratory path, employing XAI methods like ELI5, SHAP, and LIME across various ML methods. This iterative refinement and exploration unravel insights into feature importance, individual prediction contributions, and overall model decisions.

The flow diagram depicted in Figure 11.1 embodies the principles of design science research, delineating the XAI-enabled SDP process tailored for NASA's CM1 dataset. Serving as a structured implementation blueprint, it paves the way for future empirical studies. Our research aligns with the iterative problem-solving ethos of design science research by integrating XAI methods with ML models and domain-specific datasets, thereby enhancing decision-making in SDP using transparent and interpretable ML techniques.

Ensemble modeling techniques are seamlessly integrated into the iterative process, facilitating the amalgamation of feature importance scores derived from diverse XAI methods. This iterative and exploratory approach not only optimizes predictive accuracy but also ensures the preservation of model interpretability throughout the enhancement process

Through meticulous experimentation and analysis, we have demonstrated the effectiveness of ensembling XAI methods in uncovering the underlying patterns within software metrics. By doing so, we have facilitated more accurate and reliable defect predictions. Our findings underscore the importance of balancing predictive performance with interpretability, particularly in the software fect prediction domain, where transparency and insight into prediction factors are crucial for informed decision-making.

The integration of ensemble XAI methods offers a promising approach to addressing the inherent trade-offs between predictive accuracy and interpretability. By identifying the most influential features while maintaining model transparency, we provide developers and project

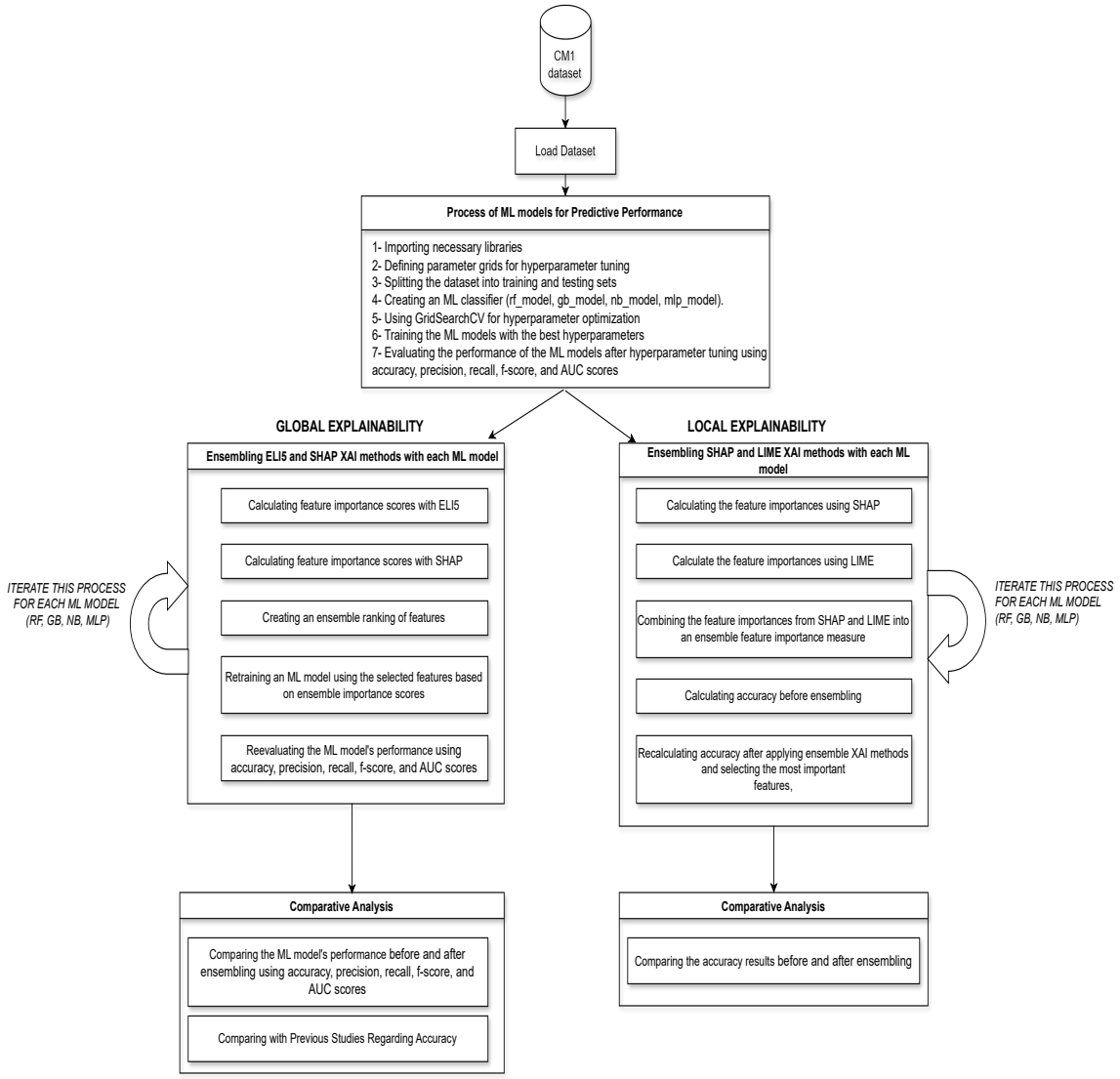


Figure 11.1 The flow diagram illustrates the XAI-enabled SDP process, tailored for NASA's CM1 dataset.

managers with actionable insights to prioritize testing and maintenance activities, ultimately enhancing software quality and reliability.

However, it's essential to acknowledge the potential limitations and challenges associated with ensembling XAI methods. These may include the need for careful evaluation of trade-offs and the necessity for domain-specific adaptation. Future research endeavors could focus on further refining ensemble XAI techniques, exploring additional machine learning models, and investigating their applicability across diverse software engineering contexts.

In summary, Case Study 4 contributes to advancing the field of software defect prediction by introducing an innovative approach that integrates ensemble XAI methods to enhance both predictive accuracy and interpretability. By bridging the gap between machine learning models and human understanding, we pave the way for more transparent, reliable, and effective defect prediction systems in software development practice.

To conclude, this research significantly contributes to the field of SDP by furnishing a thorough understanding of ML model decision-making processes. It enhances model interpretability and transparency, effectively addressing critical gaps in traditional feature selection and outlier detection methodologies. Moreover, it offers valuable insights into ensemble modeling approaches, elucidating their role in optimizing predictive accuracy while maintaining interpretability. Validation of the developed methodologies is conducted through rigorous empirical studies and comparative analyses, thus ensuring their effectiveness and usability in real-world SDP scenarios.

12. THREATS TO VALIDITY

Before delving into the findings of our study, it is imperative to acknowledge and address potential validity threats that may affect the robustness and generalizability of our results. By identifying and mitigating these threats, we aim to ensure the reliability and validity of our conclusions. In this section, we discuss various validity threats, including internal, external, construct, and reliability validity, and outline the actions taken to minimize their impact on our study. Through a comprehensive evaluation of validity threats, we strive to enhance the credibility and trustworthiness of our research findings in the domain of software defect prediction.

Internal Validity

- By using techniques such as cross-validation, we regularize ML models to prevent overfitting and improve generalizability.
- We perform hyperparameter tuning using techniques like grid search cross validation while ensuring robust validation methods to avoid bias in parameter selection.
- We conduct thorough data preprocessing, including handling missing values and document preprocessing steps to ensure transparency and reproducibility.
- To mitigate validity threats in the context of using ELI5 and SHAP for feature selection and LIME for outlier detection, we conducted comparative analysis and baseline Models.
- We compared the performance of SHAP and ELI5 against established baseline models and traditional techniques for feature selection.

External Validity

- We ensure the external validity of our study by validating model performance across multiple ML models and comparing our results with and without ensembling XAI methods. Additionally, we establish the external validity of our findings by benchmarking our performance metrics against studies from the literature that predominantly concentrate on the performance of traditional ML models. This comprehensive approach enables us to evaluate the generalizability of ensembling XAI techniques in enhancing defect prediction model performance, while also providing contextual insights by comparing our results with established methodologies in the field.

- The generalizability of the findings to other software defect prediction contexts may be a concern. The dataset used in the study may not fully represent the diversity of software projects, development practices, and defect types encountered in real-world scenarios, limiting the external validity of the findings

Construct Validity

- Despite their strengths, ELI5, SHAP, and LIME possess certain limitations, such as computational complexity and potential biases in interpretations, warranting cautious utilization and further research.

- The interpretability provided by ELI5, SHAP, and LIME, although comprehensive and detailed, might be too complex for non-technical stakeholders, contrasting with some traditional methods that offer simpler, albeit less detailed, explanations.

- We standardize the implementation of XAI methods and provide detailed documentation to ensure consistency and reproducibility. We validate the robustness of XAI techniques through cross-validation.

- We evaluate model complexity and considered simpler models when possible to enhance interpretability. We used feature selection technique to reduce complexity while maintaining predictive performance.

Reliability

- We document all steps of the modeling process, including data preprocessing, model training, hyperparameter tuning, and evaluation.
- We address missing values through robust data preprocessing techniques.

13. CONCLUSION

In conclusion, this doctoral thesis has adopted an iterative and exploratory approach, beginning with a comprehensive **"Systematic Literature Review on Software Quality for AI-based Software**" and progressing through four distinct case studies focused on enhancing Software Defect Prediction (SDP) models. Each case study serves as a building block, leveraging insights gained from its predecessors to deepen our understanding of the intricate relationship between Machine Learning (ML) models and eXplainable Artificial Intelligence (XAI) methods within the context of defect prediction. Through this iterative process, we have not only advanced the development and refinement of SDP models but also illuminated the pivotal role of XAI in augmenting predictive accuracy and interpretability. By systematically building upon prior findings and methodologies, we have laid a solid foundation for future research endeavors aimed at further optimizing SDP models for real-world applications. Our thesis underscores the importance of embracing iterative exploration in addressing the evolving challenges of AI-based software development, thereby contributing to the advancement of both theory and practice in the field.

Firstly, we performed a systematic literature review for software quality of AI-based software. The main goal of this study is searching for "How is quality defined or investigated for the AI-based software?." To answer this research goal, we searched within six different databases for determining the furthest appropriate papers on the topic and selected 29 primary studies. After all, we analyzed the data collected from these studies to answer the 23 research questions under five main headings. This SLR is among the first exhaustive reviews that examined the scope and discussion of quality characteristics and their assurance, challenges, solutions, and existing quality models for the software quality in AI-based software in the period from 1988 to 2020. Based on the findings from this SLR, we provided important insights about the challenges of AI-based software by categorizing them with different perspectives such as "Software Quality", "Software Development", "Design", "Social Aspect", and "Testing", traditional or AI-based software quality attributes, quality models, and the domain or type of the software that are subjected to the primary studies. The

results of this study are useful for observing the challenges about explainability for AI-based software. Because in contrast to traditional engineering-based approaches, ML systems often adopt a scientific-based development methodology. This shift introduces a level of uncertainty into system outputs, responses, and decision-making processes, emphasizing the importance of explainability.

By addressing the challenges related to explainability in AI-based software, the culmination of our research endeavors across all four case studies has yielded valuable insights and advancements in the domain of SDP through XAI methods. Each case study has contributed to our understanding of the decision-making processes underlying ML models, enhancing transparency and interpretability while optimizing predictive accuracy. Contributions and Importance of the Findings:

1. **Case Study 1: Explainable AI for Software Defect Prediction with Gradient Boosting Classifier:** In this study, we applied three post-hoc model-agnostic methods—ELI5, LIME, and SHAP—to a Gradient Boosting model for software defect prediction, examining both local and global explanations. Our findings indicate that all three methods consistently provide explanations that align with each other. However, there remains substantial potential for enhancing the reliability and robustness of eXplainable Artificial Intelligence (XAI) methods in the AI-based software, particularly within the ML industry. Addressing the limitations inherent in these methods is crucial, necessitating further development efforts. Generally, SHAP is favored for both local and global explanations; however, in cases where its computation cost becomes prohibitively high due to exponential runtime, ELI5 may serve as a suitable alternative for global explanations, and LIME for local ones.

   Looking ahead, our future research aims to extend our analysis to encompass various ML and XAI techniques applied to more intricate software datasets. Furthermore, we plan to leverage the insights gleaned from these explanations to enhance ML model performance. This may involve exploring different approaches, such as the

construction of new feature sets based on the explanations provided, thus facilitating a deeper understanding and optimization of AI-based software systems.

2. **Case Study 2: Explainable AI Framework For Software Defect Prediction:** Explainability plays a pivotal role in software engineering, particularly within the realm of defect prediction and software quality assessment, as demonstrated by the significance attributed to the KC2 defect prediction dataset. This dataset, a widely recognized benchmark in defect prediction research, encompasses 21 software metrics alongside binary labels indicating defect presence. The ability to elucidate the predictions of defect prediction models is paramount for comprehending the intricacies influencing software quality. Through the analysis of these explanations, software engineers gain valuable insights into the specific metrics or patterns contributing to defects. Armed with this knowledge, they can effectively prioritize efforts towards improving critical aspects of software, such as code complexity, coupling, and coding practices, thus bolstering software quality, reducing defects, and enhancing overall reliability and maintainability.

   In our study, we endeavor to comprehensively understand five distinct machine learning (ML) models—RF, GB, NB, MLP, and NN—both locally and globally. To achieve this, we leverage six diverse explainability methods—SHAP, LIME, ELI5, Partial Dependence Plots (PDP), Anchor, and Protodash. The utilization of multiple explainability techniques presents numerous advantages in the context of defect prediction and software quality assessment, facilitating a deeper comprehension of the factors underpinning software defects and offering actionable insights into software quality enhancement strategies.

   The landscape of explainability in software engineering is dynamic and continually evolving. As such, there exist numerous avenues for future research and development endeavors, spanning various ML models, explainability methods, and software quality considerations. The trajectory of explainability in software engineering involves advancing techniques, tackling challenges associated with black-box models, evaluating explainability methodologies, integrating explainability into development

workflows, tailoring explanations to user requirements, and conducting user-centric studies to gauge the impact of explanations on software quality. By embracing and exploring these directions, we can further augment the transparency, reliability, and efficacy of ML models in fortifying software quality standards. Through the implementation of post-hoc model agnostic methods such as ELI5, LIME, and SHAP, we elucidated the importance of local and global explanations in SDP. By answering research questions related to feature importance and its impact on individual and overall predictions, we laid the groundwork for subsequent studies.

3. **Case Study 3: Applying XAI Methods for Feature Selection and Outlier Detection in Software Defect Prediction:** As we transition to Case Study 3, which focuses on applying XAI methods for feature selection and outlier detection in software defect prediction, we aim to build upon the insights gained from the results of Case Study 2. By leveraging the lessons learned and best practices identified in Case Study 2, we seek to further enhance the transparency, trustworthiness, and effectiveness of ML models in improving software quality. Through the integration of advanced XAI techniques and the refinement of our methodologies, we aspire to address the evolving challenges in the field of software defect prediction and contribute to the ongoing advancement of software engineering practices.

   In this study, we addressed a critical gap in conventional feature selection and outlier detection methods within the realm of SDP. Traditional approaches often lack transparency and fail to provide clear insights into the rationale behind feature selection or outlier identification. In contrast, our work prioritized transparency and comprehensibility through the application of XAI techniques. By scrutinizing the limitations of conventional methods, we highlighted their opaque nature and the absence of insight into decision-making processes. This stands in stark contrast to our approach, which leveraged ELI5, SHAP, and LIME on the PC1 SDP dataset to provide transparent and interpretable results.

   The meticulous execution of feature selection using ELI5 and SHAP, across various models such as RF, GB, NB, and MLP, ensured a thorough examination of feature

importance and model interpretability. Furthermore, our pioneering use of LIME for outlier detection deepened our understanding of model behavior, which is crucial for robust defect prediction. Through empirical analysis, we not only evaluated the strengths and limitations of each XAI method but also comprehensively assessed their impact on defect prediction model performance. By synthesizing these findings, we made significant contributions to understanding the roles of ELI5, SHAP, and LIME in enhancing SDP models, while also shedding light on the shortcomings of traditional methods and advocating for more transparent and interpretable methodologies in SDP.

Future research stemming from this study, which utilized ELI5 and SHAP for feature selection and LIME for outlier detection, could explore several promising avenues to further advance and refine SDP methodologies. Additionally, investigating strategies for weighting and aggregating explanations from different XAI methods, and subsequently integrating these strategies to create a hybrid approach that harnesses the strengths of various XAI methods for defect prediction datasets, presents another potential avenue for future exploration.

4. **Case Study 4: Enhancing Software Defect Prediction Modeling through Ensemble XAI Methods:** In conclusion, our study delves into the domain of software defect prediction with the aim of improving both predictive accuracy and interpretability through the integration of ensemble XAI methods. By harnessing SHAP, ELI5, and LIME techniques alongside RF, GB, NB, and MLP models, we explore innovative pathways to enhance defect prediction model performance.

   Through meticulous experimentation and analysis, we showcase the efficacy of ensembling XAI methods in revealing underlying patterns within software metrics, thereby enabling more precise and dependable defect predictions. Our findings underscore the importance of striking a balance between predictive power and interpretability, particularly within the software development domain, where transparency and insight into prediction factors are essential for informed decision-making.

Nevertheless, it is crucial to acknowledge the potential limitations and challenges associated with ensembling XAI methods, including the necessity for careful evaluation of trade-offs and the need for domain-specific adaptation. Future research endeavors could concentrate on further refining ensemble XAI methods, exploring additional ML models, and examining their applicability across diverse software engineering contexts.

In summary, our study contributes to the advancement of SDP by introducing an innovative approach that integrates ensemble XAI methods to enhance both predictive accuracy and interpretability. By bridging the gap between ML models and human understanding, we pave the way for more transparent, reliable, and effective defect prediction systems in software development practices.

Looking ahead, future research aims to extend our analysis to encompass various ML and XAI techniques applied to more intricate software datasets. Furthermore, leveraging the insights gleaned from these explanations to enhance ML model performance is a promising avenue for exploration. This may involve constructing new feature sets based on the explanations provided, thus facilitating a deeper understanding and optimization of AI-based software systems. In addition, investigating the effectiveness of different XAI methods in explaining the decisions of LLM-Blackbox models across various tasks and datasets, evaluating the performance of CodeBERT in sequential modeling tasks such as code completion, code summarization, or code generation, assessing the feasibility and impact of integrating XAI methods into ML-Ops processes for model monitoring, debugging, and maintenance, exploring alternative application domains for vulnerability prediction models, such as network security, cybersecurity threat intelligence, or software supply chain security are the future research directions aim to advance our understanding and application of XAI methods, sequential modeling with CodeBERT, integration of XAI into ML-Ops processes, and exploration of vulnerability prediction in diverse domains.

The findings from each case study underscore the importance of transparency and interpretability in ML models for SDP. By synthesizing insights from diverse datasets, ML

algorithms, and XAI techniques, we have advanced our understanding of SDP methodologies and paved the way for future research directions.

Moving forward, our research opens avenues for exploring novel XAI methods, integrating ensemble techniques, and extending our analyses to diverse domains beyond SDP. By advocating for more transparent and interpretable ML models, we aim to foster trust, reliability, and effectiveness in software engineering practices, ultimately contributing to the advancement of the field.

REFERENCES

[1] Murat Cetiner and Ozgur Koray Sahingoz. A comparative analysis for machine learning based software defect prediction systems. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, **2020**.

[2] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE transactions on software engineering*, 34(4):485–496, **2008**.

[3] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, 33(1):2–13, **2006**.

[4] Yasutaka Kamei and Emad Shihab. Defect prediction: Accomplishments and future challenges. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)*, volume 5, pages 33–45. IEEE, **2016**.

[5] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, **2011**.

[6] Rana Özakıncı and Ayça Tarhan. Early software defect prediction: A systematic map and review. *Journal of Systems and Software*, 144:216–239, **2018**.

[7] Rana Özakıncı and Ayça Kolukısa Tarhan. A decision analysis approach for selecting software defect prediction method in the early phases. *Software Quality Journal*, 31(1):121–177, **2023**.

[8] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, **2019**.

[9] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25:2630–2660, **2020**.

[10] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. The promises and limitations of code metrics. *IEEE Transactions on Software Engineering*, 22(6):371–378, **1996**. doi:10.1109/32.50206.

[11] Beny N. Juang, David J. Robinson, and Atif M. Memon. A comparative study of static bug detection tools. In *Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, pages 82–87. IEEE, **2007**. doi:10.1109/ICIS.2007.40.

[12] Leon Moonen and Arie van Deursen. Predicting fault incidence using software change history. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 97–106. ACM, **2008**. doi:10.1145/1453101.1453116.

[13] Gabriele Bavota and Barbara Russo. A large-scale empirical study on self-admitted technical debt. In *Proceedings of the 13th international conference on mining software repositories*, pages 315–326. **2016**.

[14] C Aggarwal Charu. *Neural networks and deep learning: a textbook*. Springer, **2018**.

[15] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, **2009**.

[16] Simon S Haykin et al. Neural networks and learning machines/simon haykin., **2009**.

[17] Michael Collins. The naive bayes model, maximum-likelihood estimation, and the em algorithm. *lecture notes*, **2012**.

[18] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, **2012**.

[19] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, **2001**.

[20] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, **1998**.

[21] Fionn Murtagh and Mohsen M Farid. Pattern classification, by richard o. duda, peter e. hart, and david g. stork. *Journal of Classification*, 18(2):273–275, **2001**.

[22] Leo Breiman. Population theory for boosting ensembles. *The Annals of Statistics*, 32(1):1–11, **2004**.

[23] John D Kelleher, Brian Mac Namee, and Aoife D'arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, **2020**.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, **1998**.

[25] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, **2022**.

[26] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, **2009**.

[27] David Gunning. Broad agency announcement explainable artificial intelligence (xai). Technical report, Technical report, **2016**.

[28] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, **2018**.

[29] Morteza Lahijanian and Marta Kwiatkowska. Social trust: a major challenge for the future of autonomous systems. In *2016 AAAI Fall Symposium Series*. **2016**.

[30] L Richard Ye and Paul E Johnson. The impact of explanation facilities on user acceptance of expert systems advice. *Mis Quarterly*, pages 157–172, **1995**.

[31] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, **2019**.

[32] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, **2018**.

[33] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144. **2016**.

[34] Lloyd S Shapley. A value for n-person games. *Classics in game theory*, 69, **1997**.

[35] ELI5. Explain like i am five. *https://eli5.readthedocs.io/en/latest/overview.html*, **2023**.

[36] Partial Dependence Plot. Partial dependence plot — interpretml documentation. *https://interpret.ml/docs/pdp.html*, **2023**.

[37] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32. **2018**.

[38] Karthik S Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient data representation by selecting prototypes with importance weights. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 260–269. IEEE, **2019**.

[39] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, **2018**.

[40] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, **2020**.

[41] Wilson E Marcílio and Danilo M Eler. From explanations to feature selection: assessing shap values as feature selection mechanism. In *2020 33rd SIBGRAPI conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 340–347. Ieee, **2020**.

[42] Branka Hadji Misheva, Joerg Osterrieder, Ali Hirsa, Onkar Kulkarni, and Stephen Fung Lin. Explainable ai in credit risk management. *arXiv preprint arXiv:2103.00949*, **2021**.

[43] Samanta Knapič, Avleen Malhi, Rohit Saluja, and Kary Främling. Explainable artificial intelligence for human decision support system in the medical domain. *Machine Learning and Knowledge Extraction*, 3(3):740–770, **2021**.

[44] Sheikh Rabiul Islam, William Eberle, and Sheikh K Ghafoor. Towards quantification of explainability in explainable artificial intelligence methods. In *The thirty-third international flairs conference*. **2020**.

[45] Shraddha Mane and Dattaraj Rao. Explaining network intrusion detection system using explainable ai framework. *arXiv preprint arXiv:2103.07110*, **2021**.

[46] Michal Bugaj, Krzysztof Wrobel, and Joanna Iwaniec. Model explainability using shap values for lightgbm predictions. In *2021 IEEE XVIIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 102–106. IEEE, **2021**.

[47] Sebastian Palacio, Adriano Lucieri, Mohsin Munir, Sheraz Ahmed, Jörn Hees, and Andreas Dengel. Xai handbook: towards a unified framework for explainable ai. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3766–3775. **2021**.

[48] Upol Ehsan, Koustuv Saha, Munmun De Choudhury, and Mark O Riedl. Charting the sociotechnical gap in explainable ai: A framework to address the gap in xai. *arXiv preprint arXiv:2302.00799*, **2023**.

[49] Brian Hu, Paul Tunison, Bhavan Vasu, Nitesh Menon, Roddy Collins, and Anthony Hoogs. Xaitk: The explainable ai toolkit. *Applied AI Letters*, 2(4):e40, **2021**.

[50] Lindsay Sanneman and Julie A Shah. A situation awareness-based framework for design and evaluation of explainable ai. In *Explainable, Transparent Autonomous Agents and Multi-Agent Systems: Second International Workshop, EXTRAAMAS 2020, Auckland, New Zealand, May 9–13, 2020, Revised Selected Papers 2*, pages 94–110. Springer, **2020**.

[51] Syed Wali and Irfan Khan. Explainable ai and random forest based reliable intrusion detection system. *Authorea Preprints*, **2023**.

[52] Alexander Heimerl, Katharina Weitz, Tobias Baur, and Elisabeth André. Unraveling ml models of emotion with nova: Multi-level explainable ai for non-experts. *IEEE Transactions on Affective Computing*, 13(3):1155–1167, **2020**.

[53] Marcin Kapcia, Hassan Eshkiki, Jamie Duell, Xiuyi Fan, Shangming Zhou, and Benjamin Mora. Exmed: an ai tool for experimenting explainable ai techniques

on medical data analytics. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 841–845. IEEE, **2021**.

[54] Mostafa Amini, Ali Bagheri, and Dursun Delen. Discovering injury severity risk factors in automobile crashes: A hybrid explainable ai framework for decision support. *Reliability Engineering & System Safety*, 226:108720, **2022**.

[55] Katharina J Rohlfing, Philipp Cimiano, Ingrid Scharlau, Tobias Matzner, Heike M Buhl, Hendrik Buschmeier, Elena Esposito, Angela Grimminger, Barbara Hammer, Reinhold Häb-Umbach, et al. Explanation as a social practice: Toward a conceptual framework for the social design of ai systems. *IEEE Transactions on Cognitive and Developmental Systems*, 13(3):717–728, **2020**.

[56] Weina Jin, Jianyu Fan, Diane Gromala, Philippe Pasquier, and Ghassan Hamarneh. Euca: The end-user-centered explainable ai framework. *arXiv preprint arXiv:2102.02437*, **2021**.

[57] Zakaria Abou El Houda, Bouziane Brik, and Lyes Khoukhi. "why should i trust your ids?": An explainable deep learning framework for intrusion detection systems in internet of things networks. *IEEE Open Journal of the Communications Society*, 3:1164–1176, **2022**.

[58] Ghada El-khawaga, Mervat Abu-Elkheir, and Manfred Reichert. Xai in the context of predictive process monitoring: An empirical analysis framework. *Algorithms*, 15(6):199, **2022**.

[59] Zubaira Naz, Muhammad Usman Ghani Khan, Tanzila Saba, Amjad Rehman, Haitham Nobanee, and Saeed Ali Bahaj. An explainable ai-enabled framework for interpreting pulmonary diseases from chest radiographs. *Cancers*, 15(1):314, **2023**.

[60] Ahmad Haji Mohammadkhani, Nitin Sai Bommi, Mariem Daboussi, Onkar Sabnis, Chakkrit Tantithamthavorn, and Hadi Hemmati. A systematic

literature review of explainable ai for software engineering. *arXiv preprint arXiv:2302.06065*, **2023**.

[61] Geanderson Esteves, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and Nivio Ziviani. Understanding machine learning software defect predictions. *Automated Software Engineering*, 27(3):369–392, **2020**.

[62] Jiho Shin, Reem Aleithan, Jaechang Nam, Junjie Wang, and Song Wang. Explainable software defect prediction: Are we there yet? *arXiv preprint arXiv:2111.10901*, **2021**.

[63] Geanderson Esteves dos Santos and Eduardo Figueiredo. Failure of one, fall of many: An exploratory study of software features for defect prediction. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 98–109. IEEE, **2020**.

[64] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, Hoa Khanh Dam, and John Grundy. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering*, **2020**.

[65] Toshiki Mori and Naoshi Uchihira. Balancing the trade-off between accuracy and interpretability in software defect prediction. *Empirical Software Engineering*, 24(2):779–825, **2019**.

[66] Yazan Al-Smadi, Mohammed Eshtay, Ahmad Al-Qerem, Shadi Nashwan, Osama Ouda, and AA Abd El-Aziz. Reliable prediction of software defects using shapley interpretable machine learning models. *Egyptian Informatics Journal*, 24(3):100386, **2023**.

[67] Momotaz Begum, Mehedi Hasan Shuvo, Imran Ashraf, Abdullah Al Mamun, Jia Uddin, and Md Abdus Samad. Software defects identification: Results using machine learning and explainable artificial intelligence techniques. *IEEE Access*, **2023**.

[68] Bahar Gezici and Ayça Kolukısa Tarhan. Systematic literature review on software quality for ai-based software. *Empirical Software Engineering*, 27(3):66, **2022**.

[69] Hiroshi Kuwajima and Fuyuki Ishikawa. Adapting square for quality assessment of artificial intelligence systems. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 13–18. IEEE, **2019**.

[70] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, **2020**.

[71] Christian Murphy, Gail E Kaiser, and Marta Arias. A framework for quality assurance of machine learning applications. -, **2006**.

[72] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. -, **2007**.

[73] Ahmet Okutan and Olcay Taner Yıldız. Software defect prediction using bayesian networks. *Empirical Software Engineering*, 19(1):154–181, **2014**.

[74] Hamoud Aljamaan and Amal Alazba. Software defect prediction using tree-based ensembles. In *Proceedings of the 16th ACM international conference on predictive models and data analytics in software engineering*, pages 1–10. **2020**.

[75] Ahmed Iqbal, Shabib Aftab, Israr Ullah, Muhammad Salman Bashir, and Muhammad Anwaar Saeed. A feature selection based ensemble classification framework for software defect prediction. *International Journal of Modern Education and Computer Science*, 11(9):54, **2019**.

[76] Saiqa Aleem, Luiz Fernando Capretz, and Faheem Ahmed. Benchmarking machine learning technologies for software defect detection. *arXiv preprint arXiv:1506.07563*, **2015**.

[77] Bushra Mumtaz, Summrina Kanwal, Sultan Alamri, and Faiza Khan. Feature selection using artificial immune network: An approach for software defect prediction. *Intelligent Automation & Soft Computing*, 29(3), **2021**.

[78] Cagatay Catal and Banu Diri. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, 179(8):1040–1058, **2009**.

[79] Karim O Elish and Mahmoud O Elish. Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, 81(5):649–660, **2008**.

[80] Abdullah Alsaeedi and Mohammad Zubair Khan. Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. *Journal of Software Engineering and Applications*, 12(5):85–100, **2019**.

[81] Aimen Khalid, Gran Badshah, Nasir Ayub, Muhammad Shiraz, and Mohamed Ghouse. Software defect prediction analysis using machine learning techniques. *Sustainability*, 15(6):5517, **2023**.

[82] Md Nasir Uddin, Bixin Li, Md Naim Mondol, Md Mostafizur Rahman, Md Suman Mia, and Elizabeth Lisa Mondol. Sdp-ml: an automated approach of software defect prediction employing machine learning techniques. In *2021 International Conference on Electronics, Communications and Information Technology (ICECIT)*, pages 1–4. IEEE, **2021**.

[83] Jonas Herskind Sejr and Anna Schneider-Kamp. Explainable outlier detection: What, for whom and why? *Machine Learning with Applications*, 6:100172, **2021**.

[84] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. Outlier detection: Methods, models, and classification. *ACM Computing Surveys (CSUR)*, 53(3):1–37, **2020**.

Publications

1- Gezici, Bahar, and Ayça Kolukısa Tarhan. ”Systematic literature review on software quality for AI-based software.” Empirical Software Engineering 27.3 (2022): 66.

2- Gezici, Bahar, and Ayça Kolukisa Tarhan. ”Explainable AI for software defect prediction with gradient boosting classifier.” 2022 7th International Conference on Computer Science and Engineering (UBMK). IEEE, 2022.(Best Paper Award)

3- Gezici, Bahar, Demir, Merve Özdeş, and Ayça Kolukisa Tarhan. ”An Investigation on the Explainability of Tree-based Machine Learning Models” 2023 16. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS). IEEE, 2023.(Best Paper Award)

4- Demir, Merve Özdeş, Bahar Gezici, and Ayça Kolukısa Tarhan. ”Assessing The Explainability of Lgbm Model For Effort Estimation on ISBSG Dataset.” 2023 4th International Informatics and Software Engineering Conference (IISEC). IEEE, 2023.

5- Gezici, Bahar, and Ayça Kolukısa Tarhan. ”Explainable AI Framework For Software Defect Prediction.” The Journal of Systems & Software, 2024 (Under Review).

6- Gezici, Bahar, and Ayça Kolukısa Tarhan. ”Applying XAI Methods for Feature Selection and Outlier Detection in Software Defect Prediction.” Journal of Software: Evolution and Process, 2024 (Under Review).

7- Gezici, Bahar, and Ayça Kolukısa Tarhan. ”Enhancing Software Defect Prediction Modeling through Ensemble XAI Methods.” Information and Software Technology, 2024 (Under Review).