

An Intelligent Machine Learning Framework for Predicting Software Defects Using Repository Mining

Deep Patel (2504308)

RESEARCH METHODS AND PROFESSIONAL DEVELOPMENT (ACFA7004)

Table of Contents

1. Introduction.....	2
1.1 Context and Background	2
1.2 Research Problem.....	2
1.3 Research Aim and Objectives	3
1.4 Significance of the Research.....	4
2. Literature Review.....	5
2.1 Introduction.....	5
2.2 Machine Learning for Software Defect Prediction	5
2.3 Hybrid and Ensemble Learning Approaches	5
2.4 Deep Learning in Software Defect Prediction	6
2.5 Role of Source Code Metrics	6
2.6 Explainable AI in Software Defect Prediction	7
2.7 Research Gaps in Existing Literature	7
2.8 Positioning of the Present Study	7
3. Research Design and Methodology	8
3.1 Research Design.....	8
3.2 Research Methods and Materials	8
4. Expected Outcomes.....	12
4.1 Implications.....	12
4.2 Limitations	12
5. Timeline (12 Weeks).....	13
6. Overall Presentation of the Proposal	14
References	15

List of Figures

Figure 1: Project Timeline (Gantt Chart)	13
--	----

1. Introduction

1.1 Context and Background

Software is essential to today's economic, government, health, education and industrial processes. With the growing reliance on software-based operations, the importance of software maintainability and reliability issues has ascended the corporate agenda. Software defects can result in economic losses, productivity loss, damage to reputation and, in certain critical areas, to major societal impacts.

Well-established defect detection methods such as manual inspection, static analysis, integration testing and post-shipment defect repair remain critical contributing factors for quality control. But these could be costly, time-consuming and even reactive. For many development teams, defects are detected only after a significant number of resources are spent on code production or testing.

To address these challenges, there is growing interest in predictive analytics in the field of software engineering. Software defect prediction aims to predict which modules, components or files will contain defects before the software undergoes extensive testing [13]. By identifying these components, better test resource allocation can be made, developers can focus on high-risk areas, and maintenance costs could possibly be minimised.

And advances in machine learning have helped to fortify this research area. Machine learning techniques can learn from past software metrics and labels of defects to predict software that has a history of being more prone to faults. Techniques like Logistic Regression, Decision Trees, Random Forest and gradient boosting algorithms have shown promising results in classification problems with structured engineering data.

1.2 Research Problem

Despite advances in software analysis tools, development processes and testing techniques, software defects continue to plague developers. Numerous organisations still find defects late in the development process, where they are more expensive to fix than other defects addressed early in the software development. Detecting software defects late in the development cycle can also risk software launches being postponed, technical debt and customer dissatisfaction.

While there has been considerable research on software defect prediction, there remain a few unexplored questions. First, previous research shows mixed results about the effectiveness of different machine learning models. For example, models obtained with one data set may not perform well on other data sets due to different class distributions, performance metrics or software projects. This can be confusing for decision makers looking to implement predictive systems.

Second, certain research focuses on improving predictive performance above the understandability or practicality of implementation. In the industry, complex models might be

difficult to adopt if systems developers cannot comprehend and trust the predictions. So, there needs to be a trade-off between performance and interpretability [7].

Third, many teams, especially small and medium-sized development teams, do not have the infrastructure necessary to develop complex data engineering pipelines from source data. They need frameworks that can make good use of structured historical data without incurring large data engineering costs. Here, benchmark data sets and generic modelling code are very useful.

Fourth, class imbalance is common with defect datasets due to the much larger number of non-defective modules than defective modules. This can result in models being "accurate" without identifying defective modules. Failure to detect defects can have consequences, so accuracy alone is insufficient.

This is an issue because the software industry needs informed advice on the models to prioritise, the performance criteria to assess and how predictive technologies can assist in quality assurance activities.

1.3 Research Aim and Objectives

Research Aim

To implement and assess a novel machine learning approach for software defect prediction techniques by applying machine learning approaches using repository-mined benchmark data.

Research Objectives

1. To provide a critical review of the current state-of-the-art research in software defect prediction, software repository mining and machine learning in software engineering.
2. To clean and profile the KC1 benchmark data set to better understand issues with data quality, feature distributions and imbalance in predictive modelling.
3. To apply some of the chosen supervised machine learning models (i.e. Logistic Regression, Decision Tree and Random Forest) for defect prediction.
4. To assess and compare the efficacy of these models with suitable classification measures such as Accuracy, Precision, Recall, F1-score, Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) and confusion matrices.
5. To determine a practically preferable model based on performance metrics and other aspects of model suitability, such as interpretability, robustness and practical implementation.
6. To develop a prototype intelligent system that can help software development teams to select vulnerable modules as part of quality assurance.

These goals aim to be tangible, ordered and meet our research challenge criteria. They range from theoretical to empirical and applied.

1.4 Significance of the Research

Theoretically, the study contributes to the ongoing discussion about the effectiveness of various machine learning methods in software-defect prediction [1]. Given existing literature has obtained context-specific results, more evidence is needed. The study will also contribute to the debate of whether simpler models outperform more complex ensemble methods in a benchmarking experiment.

Pragmatically, the software development world is often short of time, people and budget for testing. Thorough tests on all the modules are inefficient. A sound prediction model can guide practitioners to prioritise software components for quality assurance, leading to improved efficiency of the software testing process and possibly lower cost of maintenance [11].

In terms of the research methodology, the study avoids single-metric generalisations and rigorous assessments. Class imbalance can render accuracy unreliable. The addition of Recall, Precision and F1-score metrics adds to a better interpretation of model performance. This is because false negatives may have a greater impact on how a software product is categorised than false positives [2].

The research is pedagogically and industrially relevant. As smart software engineering gains prominence, students and professionals need to be familiar with potential applications of machine learning to support software lifecycle management. So, the conceived framework can be used as a benchmark for future research and development.

Potential applications include:

- Risk-based software testing
- Maintenance prioritisation
- Code review targeting
- Quality assurance planning
- Supporting decision-making in agile dev

Finally, the research is important because it has tackled a key industry problem by using an evidence-based, technically sound and relevant approach.

2. Literature Review

2.1 Introduction

Software defect prediction has emerged as a significant research topic in software engineering due to the impact of software defects on software reliability, software maintenance and user satisfaction. With the growing complexity of systems, it becomes less effective to identify defects on the basis of testing or post-release. As a result, researchers have investigated prediction models that can detect software defects at early phases in the software life cycle.

Machine learning is one of the most popular techniques for software defect prediction due to its ability to learn from data relating to prior software metrics and software defects. These approaches leverage data from the version control system (VCS), including metrics such as complexity, size, coupling and the history of software defects to predict whether software components are defective or not. But while significant gains have been made, there is still no agreement about which methods are best and most practically effective across different data sets.

This review discusses the most recent research (2022-2024) on software defect prediction. The review is based on five topics: traditional machine learning techniques, hybrid and ensemble learning, deep learning, source code metrics and explainable artificial intelligence. This is done to understand the latest trends, challenges and gaps in the research for the proposed study.

2.2 Machine Learning for Software Defect Prediction

Machine learning is still a major focus of software defect prediction. Albattah and Alzahrani [1] empirically compared the use of machine learning and deep learning for software defect prediction. They confirmed the effectiveness of traditional machine learning models, despite the shift towards deep learning. This finding is significant, as it suggests that solving software prediction problems does not necessarily require complex models.

Machine learning methods continue to be important due to the nature of most software defect datasets. Software defect prediction benchmark datasets (such as KC1) are typically based on tabular numerical data. Algorithms such as Logistic Regression, Decision Tree, Support Vector Machine and Random Forest work well in such contexts [1].

But research by Albattah and Alzahrani [1] finds that classifier accuracy differs between datasets. This implies that no one classifier can be generally recommended as best. Instead, the choice of the model should be based on the characteristics of available data, interpretability and the constraints of the project [1].

2.3 Hybrid and Ensemble Learning Approaches

There has been a growing trend towards ensemble learning for better predictions. Typically, hybrid methods may involve the use of feature selection from one method and the classification capability of another. Hybrid approaches generally involve combining the

features of different algorithms, for instance, feature selection with one algorithm and classification with another.

Likewise, Kumar, Nayak and Behera [5] developed an ensemble learning technique involving stacking based on software quality code features. They found that ensemble methods can enhance classification performance, as these methods combine predictions from different base learners into a final prediction. This finding is common in the machine learning literature, with ensemble methods tending to decrease variance and increase generalisation.

While the results are encouraging, there are also drawbacks to hybrid and ensemble methods. These can include increased computational demands, lack of interpretability and additional tuning. In the context of software engineering, these factors can hinder their adoption. Even if an accurate model exists, it may not be used if engineers do not understand or cannot manage the model [5].

So, while ensemble methods may have better predictive accuracy, simpler models might be preferred in situations where interpretability and practical usage are important.

2.4 Deep Learning in Software Defect Prediction

Deep learning is now in the spotlight due to its capacity for automatic modelling of the complex, non-linear nature of relationships. A recent review by Giray et al. [6] of the application of deep learning for software defect prediction observed a growing interest in neural network-based approaches. The ability to automatically model interactions between variables is an advantage of deep learning methods.

But the study also documented a number of issues. First, deep learning typically needs more data than is typically available for software defect prediction. Existing benchmark datasets typically have relatively small numbers of observations, thereby limiting the potential of deep neural networks. Second, deep learning models can be less understandable and slower to execute. Third, studies have shown mixed results with improvements over traditional machine learning [6].

These results indicate that deep learning may not necessarily be superior. In smaller structured data like KC1, traditional approaches may still be preferred and equally effective.

2.5 Role of Source Code Metrics

For software defect prediction, the quality of predictor variables is of utmost importance. Rebro, Chren and Rossi [3] have studied the role of source code metrics for software defect prediction and found that measurable code features play an important role in prediction tasks. Variables relating to complexity, size, coupling and counts of operations continue to be useful for predicting defects.

This is important since the KC1 dataset contains many such metrics. The value of these variables helps to validate the use of the chosen dataset for this research.

But source code metrics are not perfect. Complexity may be indicative of errors, but complex modules may not necessarily be defective. Nor are small modules necessarily free of critical logic errors. As a result, metrics should be considered probabilistic [3].

In other words, the key to predictive success might lie not just in the variables of the dataset, but also in the modelling approach to combine them.

2.6 Explainable AI in Software Defect Prediction

With predictive systems increasingly making professional recommendations, explainability is an important issue. Halдар and Capretz [4] explored explainable software defect prediction through project metrics across companies with machine learning models. They're part of a trend that recognises it is not enough for predictions to be accurate; users also need to understand why predictions were made.

This is particularly so in software engineering workflows. Programmers and managers may not be willing to adopt black-box systems that operate in an opaque manner when resource allocation for testing, priorities for code review or decisions for release are driven by model output [4].

As such, explainable AI facilitates organisational acceptance and accountability. Simple predictive models like Logistic Regression and Decision Trees will be naturally more explainable than complex ensemble or neural networks.

2.7 Research Gaps in Existing Literature

There are several remaining gaps in the reviewed research.

First, several studies focus on model accuracy but ignore model interpretability and deployability. Second, a few papers assess different models on varied data, making comparisons challenging. Third, deep learning is receiving much attention, despite limited evidence of better performance on benchmark tabular data. Fourth, there is no agreed-upon balance between interpretability and performance.

These motivate the planned dissertation, which will address several machine learning models over a benchmark data set using a comprehensive multi-metric assessment to assess performance, as well as model interpretability and implementability.

2.8 Positioning of the Present Study

The current study bridges the gap in the literature by analysing the performance of chosen machine learning models on the KC1 benchmark dataset for software defect prediction. In contrast to research which aims solely at improving prediction accuracy, the proposed study will consider the measures of Accuracy, Precision, Recall, F1 and ROC-AUC, as well as interpretability.

This places the study in an academically important as well as useful position.

3. Research Design and Methodology

3.1 Research Design

This study will use a quantitative experimental research design. This is suitable because the primary goal is to compare the predictive accuracy (performance) of machine learning models using data in a structured numeric format. Quantitative experimentation allows variables to be carefully controlled, procedures to be repeated, and performance to be measured.

Other types of experimentation, such as interviews or case studies, would be valuable in investigating user perceptions, but they would not directly address the central research question relating to the effectiveness of models. Likewise, purely theoretical studies would not be satisfactory for a study that needs to be empirically tested.

The research also has design science research elements, as it is expected to deliver an artefact: an intelligent predictive model using Python. But experimental comparison is the prevailing methodological approach.

The research process will encompass data preparation, data exploration, training, testing, comparison and prototyping. This is a progressive process from contributing to understanding, to practical results.

3.2 Research Methods and Materials

The research approach is through supervised classification modelling. The metrics describing past software assets will be predictors, and the occurrence of defects will be a target variable. Several algorithms will be trained on the prepared (normalised) dataset under similar configurations for a fair comparison [12].

The focus is not just on achieving the highest score, but on evaluating trade-offs between accuracy, recall, interpretability and actionability of the model. This is crucial as a model that is closest to the "true" model may not be the easiest to implement [9].

Research Methods

A supervised learning approach will be used for the study because the available data includes class labels, indicating whether a code module is faulty or not. This type of learning is suitable when there are historical examples of which class they belong to, and we want to predict the class of new examples. Here, the problem will be binary classification.

We will implement three different machine learning algorithms: Logistic Regression, Decision Tree and Random Forest. The models chosen were not done so arbitrarily.

Logistic Regression is used to provide a strong statistical baseline with good interpretability. Interpretable models are still desirable in practice because teams of engineers often seek to understand why software modules are "risky".

A decision tree is chosen as it can capture complex or nonlinear patterns while being easily visualised. It can provide easily explainable rules based on software metrics, which may be useful to gain managerial confidence [6].

Random Forest is chosen as an ensemble of trees, which tends to enhance the reliability and predictive performance [10]. It is a very popular benchmark for tabular data.

These three models also vary in terms of complexity and therefore provide a way to trade off model simplicity, interpretability and predictive accuracy.

Models with greater complexity (e.g. deep neural networks) will be de-emphasised in this study. While potentially useful in some applications, the usefulness of these models on mid-sized tabular benchmark datasets is unclear, and interpretability and implementation costs can be inferior. The "no more complex than needed" mindset is a strength.

Data Preparation Strategy

Data preparation is vital as the results yield excellent predictions only if the data are of good quality. The following steps will be undertaken:

1. Data Inspection - to check data types, missing values, duplicate records and outlier values.
2. Target Encoding - numerical encoding of the target variable (defective or not).
3. Feature Scaling - normalisation, if necessary (e.g., for Logistic Regression). Less sensitive to tree-based algorithms.
4. Review of Outliers - removing influential cases that can skew some methodologies.
5. Train-Test Split – holding out test data to avoid data leakage.
6. Cross-Validation - estimating models on the training data using k-fold validation.

Class Imbalance Handling

We will check the class distribution in the KC1 data before determining whether class imbalance correction is needed. This is much better than blindly applying resampling.

If the dataset is only slightly imbalanced and models exhibit satisfactory Recall values for "bad" modules, resampling might be avoided. If imbalanced data seriously impairs the ability to detect the minority class, then controlled resampling (class weighting and SMOTE) may be investigated [15]. Their use will be motivated through empirical evaluation vis-à-vis performance.

This avoids problems from previous studies that haphazardly apply imbalance techniques.

Evaluation Methods and Metrics

A thorough evaluation approach will be adopted. We will evaluate using:

- Accuracy - overall correct predictions.
- Precision - fraction of "defective" modules that are defective.

- Recall - proportion of true defectives predicted.
- F1-score - mean of Precision and Recall.
- ROC-AUC - ability to discriminate different thresholds.
- Confusion Matrix - in-depth classification.

Accuracy will not be considered the only critical metric, as in defect prediction analysis, it may be more important not to miss defective modules than to make some false predictions. So, Recall and F1-score will receive a heavy weight as interpretation [10].

When necessary, we will report the means and standard deviations of cross-validation runs instead of the results of a single run.

Artefact Development

Practical artefacts will be developed as a simple Python-based prototype. The prototype will take input software metrics and produce a predicted defect risk category. This study will add value to the operational value of this thesis by exemplifying how to implement predictive studies rather than just leaving them as academic theory.

Materials

Research Data

This study will utilise the KC1 dataset from the NASA PROMISE repository, a widely used benchmark dataset in software defect prediction research. The dataset contains software module-level metrics and defect labels, making it suitable for supervised machine learning classification tasks. Its widespread usage ensures comparability with prior studies and enhances the validity of experimental results.

There are a number of reasons the KC1 dataset is suitable.

First, it is widely used in the software defect prediction literature; it's possible to compare results with previous experiments. Second, it is a non-null-terminated data set, conducive to supervised machine learning. Third, it's small enough for experimentation by postgraduate students to work on while remaining complex enough to require modelling. Fourth, it has a well-defined software Metric, matching the theme of quality analytics derived from software repositories.

A benchmark public dataset also facilitates study reproducibility to the extent that other researchers can repeat the study [14].

Ethical Considerations Related to Data

The dataset does not contain any personal, confidential or sensitive human data. It does not include people's names or behavioural data. As such, the dataset poses minimal risks.

But best practice requires:

- correct source of data

- transparent preprocessing decisions
- reporting results correctly
- no made-up claims
- recognising the limitations of the data set

Software Requirements

The implementation environment will include:

- Python
- Jupyter Notebook or Google Colab
- pandas
- NumPy
- scikit-learn
- matplotlib
- seaborn

These are suitable as they are accepted in most research in machine learning, open-source, and the scope of the work is limited.

Hardware Requirements

The research does not need state-of-the-art GPU hardware support because the chosen models are light, and tabular data is low-dimensional, easily processed. Any recent desktop or cloud-based computing environment with average memory capacity will be sufficient [8].

This is a benefit as it shows that predictive systems can be trained without a substantial hardware investment.

4. Expected Outcomes

At the time of the proposal, the results were still not available. But several evidence-informed outcomes can be identified.

First, it is expected that machine learning will have some predictive power in identifying which software modules may be defective. According to previous research, Random Forest may provide the best performance in terms of predictive power, owing to its ensemble learning approach and ability to learn non-linear relationships.

Second, it's anticipated that Logistic Regression will offer moderate or weak predictive performance compared with Random Forest, but will offer greater interpretability. This could be important in some cases [2].

Third, a decision tree may provide comprehensible rules, but could be less robust than other models.

Fourth, it is likely that evaluating only with Accuracy may be misleading. Multipurpose assessment will most likely reveal that some models may perform better in terms of Recall, Precision or a balance of both.

And fifth, the artefact (prototype) is likely to show the feasibility of translating research findings into a lightweight decision support system.

4.1 Implications

The study could lead to:

- better planning for testing efforts
- better detection of risky modules, sooner
- recommendations for model choice
- usable application of smart quality assurance

4.2 Limitations

1. A single benchmark dataset was used, which may affect generalisation.
2. Past data might not capture current DevOps practices.
3. Results could be sensitive to data preprocessing and tuning [15].
4. Defects are predictive (not explanatory).

5. Timeline (12 Weeks)



Figure 1: Project Timeline (Gantt Chart)

The project time plan, over twelve weeks, is planned to enable the systematic and gradual implementation of the research. It assigns adequate time to each of the key phases, such as literature review, data preparation, model development, evaluation, prototype implementation, and final report writing. This sequential system allows efficient time management, consistency of the development and the timely delivery of not only scholarly but also technical elements of the study.

6. Overall Presentation of the Proposal

The proposal will be presented in a well-organised academic format consistent with the high expectations of postgraduate academic standards. The sections will be laid out in a manner that allows for a smooth flow of argument from the establishment of the research context and problem, to the setting of research aims and objectives, justification of the research methodology, and discussion of the expected outcomes. This order is designed to show the congruence between the research problem, the proposed methodology and the anticipated outcomes.

The use of tables, figures, and diagrams will be limited to instances where they add value through their ability to clarify, condense information, enhance the display of chronologies or flows and processes in the methodology [16]. This means that tables, figures and diagrams will be used for their analytical rather than aesthetic value. In summary, the delivery of the proposal will be expected to be indicative of integrity, professionalism and effective academic communication skills.

References

- [1] W. Albattah and M. Alzahrani, “Software defect prediction based on machine learning and deep learning techniques: An empirical approach,” *AI*, vol. 5, no. 4, pp. 1743–1758, 2024. <https://www.mdpi.com/2673-2688/5/4/86>
- [2] H. Kumar and V. Saxena, “Software defect prediction using hybrid machine learning techniques: A comparative study,” *J. Softw. Eng. Appl.*, vol. 17, no. 4, pp. 155–171, 2024. <https://doi.org/10.4236/jsea.2024.174009>
- [3] D. A. Reboredo, S. Chren, and B. Rossi, “Source code metrics for software defects prediction,” in *Proc. 38th ACM/SIGAPP Symp. Appl. Comput.*, pp. 1469–1472, Mar. 2023. <https://dl.acm.org/doi/pdf/10.1145/3555776.3577809>
- [4] S. Haldar and L. F. Capretz, “Explainable software defect prediction from cross company project metrics using machine learning,” in *2023 7th Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, pp. 150–157, May 2023. <https://arxiv.org/pdf/2306.08655>
- [5] P. S. Kumar, J. Nayak, and H. S. Behera, “Model-based software defect prediction from software quality characterized code features by using stacking ensemble learning,” *J. Eng. Sci. Technol. Rev.*, vol. 15, no. 2, pp. 137–155, 2022. <https://www.jestr.org/downloads/Volume15Issue2/fulltext171522022.pdf>
- [6] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, “On the use of deep learning in software defect prediction,” *J. Syst. Softw.*, vol. 195, pp. 1–26, 2023. <https://www.sciencedirect.com/science/article/pii/S0164121222002138>
- [7] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, “Software defect prediction analysis using machine learning techniques,” *Sustainability*, vol. 15, no. 6, p. 5517, 2023. <https://www.mdpi.com/2071-1050/15/6/5517>
- [8] P. Mohapatra, “Predictive defect analysis in software engineering: Approaches using artificial intelligence and machine learning,” *Intell. Assurance: Artificial Intelligence-Powered Software Testing in the Modern Development Lifecycle*, vol. 4, p. 55, 2025. <https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=5372806#page=61>

- [9] A. S. Nyamawe, "Mining commit messages to enhance software refactorings recommendation: A machine learning approach," *Mach. Learn. Appl.*, vol. 9, p. 100316, 2022. <https://www.sciencedirect.com/science/article/pii/S2666827022000354>
- [10] I. Mehmood, S. Shahid, H. Hussain, I. Khan, S. Ahmad, S. Rahman, N. Ullah, and S. Huda, "A novel approach to improve software defect prediction accuracy using machine learning," *IEEE Access*, vol. 11, pp. 63579–63597, 2023. <https://ieeexplore.ieee.org/iel7/6287639/6514899/10155117.pdf>
- [11] S. D. Palma, D. Di Nucci, and D. Tamburri, "RepoMiner: A language-agnostic python framework to mine software repositories for defect prediction," *arXiv preprint arXiv:2111.11807*, 2021. <https://arxiv.org/pdf/2111.11807>
- [12] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Y. Ghadi, and M. A. Khan, "Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning," *PeerJ Comput. Sci.*, vol. 10, p. e1860, 2024. <https://peerj.com/articles/cs-1860.pdf>
- [13] F. Matloob, T. M. Ghazal, N. Taleb, S. Aftab, M. Ahmad, M. A. Khan, S. Abbas, and T. R. Soomro, "Software defect prediction using ensemble learning: A systematic literature review," *IEEE Access*, vol. 9, pp. 98754–98771, 2021. <https://ieeexplore.ieee.org/iel7/6287639/6514899/09477596.pdf>
- [14] T. O. Olaleye, D. A. Aborishade, O. Arogundade, A. Abayomi-Alli, and O. J. Adeniran, "Multilayer perceptron of software complexity metrics for explainable multicollinearity mitigation and defect localization," *Cureus J. Comput. Sci.*, vol. 17, pp. 1–17, 2025. <https://www.cureusjournals.com/articles/2871-multilayer-perceptron-of-software-complexity-metrics-for-explainable-multicollinearity-mitigation-and-defect-localization.pdf>
- [15] M. Owusu-Adjei, J. Ben Hayfron-Acquah, T. Frimpong, and G. Abdul-Salaam, "Imbalanced class distribution and performance evaluation metrics: A systematic review of prediction accuracy for determining model performance in healthcare systems," *PLOS Digit. Health*, vol. 2, no. 11, p. e0000290, 2023. <https://journals.plos.org/digitalhealth/article/file?id=10.1371/journal.pdig.0000290&type=printable>

[16] L. Dube and T. Verster, “Enhancing classification performance in imbalanced datasets: A comparative analysis of machine learning models,” *Data Sci. Finance Econ.*, vol. 3, no. 4, p. 354, 2023. https://www.researchgate.net/profile/Lindani-Dube/publication/374734735_Enhancing_classification_performance_in_imbalanced_datasets_A_comparative_analysis_of_machine_learning_models/links/652d4e7f6725c324010cc0f1/Enhancing-classification-performance-in-imbalanced-datasets-A-comparative-analysis-of-machine-learning-models.pdf