A REPORT

ON

Software Fault Prediction Using Machine Learning and NASA MDP Datasets

By

Ch. Bhagavathi Neha AP23110011252

Prepared in the partial fulfillment of the

Summer Internship Course

Under

Dr. Sanjay Kumar

Assistant Professor

Department of Computer Science and Engineering

SRM UNIVERSITY, AP

(July, 2025)

SUMMER INTERNSHIP COURSE, 2025-26

JOINING REPORT

Date: 01/06/25

Name of the Student	Ch. Bhagavathi Neha
Roll No	AP23110011252
Programme (BTech/ BSc/ BA/MBA)	BTech
Branch	CSE
Name and Address of the Internship Company [For research internship, it would be SRMAP]	SRMAP Telephone No: 0863 234 3000 Email: sanjay.k@srmap.edu.in
Period of Internship	From 01/06/2025 to 25/07/2025

I hereby inform that I have joined the summer internship on 01.06.25 for the Research internship under Dr. Sanjay Kumar.

Ch. Bhagavathi Neha.

Date : 01/06/25 Signature of the Student

CERTIFICATE FROM FACULTY MENTOR/HR

Certified that the above-mentioned student has joined our organization for the RESEARCH INTERNSHIP in SRM UNIVERSITY-AP.

Name of the Industry Mentor	Dr. Sanjay Kumar
Designation	Assistant Professor Department of Computer Science and Engineering
Phone No (If any)	9810670424, 8697465104
Email	sanjay.k@srmap.edu.in

Signature & Date	Sanjay Kumar

Acknowledgments

We wish to convey our heartfelt appreciation to everyone who has contributed to the support and guidance of our efforts throughout the development of this project.

To start with, we owe our sincerest thanks to the Vice Chancellor of SRM University – AP as well as the Dean of the School of Engineering and Sciences for the provision of the necessary resources and this great opportunity to successfully completed our internship project.

We would particularly like to convey our utmost gratitude to Dr. Sanjay Kumar, our faculty mentor, who has been our constant source of encouragement, motivation, and enlightening knowledge all through our project work.

The completion of this project work would have been impossible without the help of all the individuals involved.

Abstract

The machine learning algorithms which can be used to determine faults on software based on NASA Metrics Data Program (MDP) datasets: MW1, MC1, and MC2, are the main concern of this study. This research employs an approach that is thorough and is based on a systematic methodology that contains different stages of operation to increase the accuracy of predictions. Feature selection techniques were applied to sort out the software metrics that are most relevant to fault prediction, thus, it was possible to remove noise and improve the interpretability of the model. To deal with the common problem of class imbalance that is found in software defect datasets, where non-faulty instances are by far more than faulty ones, oversampling techniques like SMOTE or random oversampling were used. Such an approach made sure that minority class cases were properly represented when the model was being trained, and therefore, it did not get biased towards the majority classes.

Moreover, dimensionality reduction methods, particularly the Principal Component Analysis (PCA), were employed to decrease the number of dimensions in the feature space, facilitating the computational processes and at the same time retaining almost all the information for the accurate predictions. The set of machine learning classifiers that were used included both the traditional algorithms like the Decision Trees and the Support Vector Machines and the advanced ensemble-based models such as Random Forest, Gradient Boosting, and XGBoost. Prospective refinement of operating parameters for every machine learning structure was carried out to reach the highest possible performance, involving strategies such as grid search and cross-validation for guaranteeing the reliability of the model on unseen data.

The results of the experiments show that the ensemble-based methods have always been better than the single classifiers, in all the metrics that were evaluated, including precision, recall, F1-score, and ROC-AUC. This serves as proof of the concept of using several weak learners to be the only one to model the complicated patterns and interactions existing in software metrics. Additionally, the study's results indicate that Machine Learning methods are not only strong performers but also remain pragmatic in their application to industrial software systems, providing a trustworthy, data-driven way for defect prevention and a higher level of software quality. In a nutshell, the study firmly positions machine learning not solely as a sound theoretical instrument for fault prediction but also as a practical tool with proven effectiveness in promoting software reliability and lessening maintenance costs.

Table Of Contents

1. Introduction

1.1 Overview of the Project

In recent years, software fault prediction has become a crucial area in the field of software engineering and machine learning [1]. With the increasing complexity of software systems, early identification of faulty modules is vital to ensure software quality, reduce costs, and improve maintainability [2]. This project aims to build and evaluate machine learning models to predict defective software components using static code attributes from industry-standard datasets (MW1, MC1 and MC2) [3]. The study involves comprehensive preprocessing, class imbalance handling, model selection, hyperparameter tuning, and performance evaluation using advanced visualization techniques.

1.2 Objectives of the Study

This project has the following main objectives:

- The MW1, MC1 and MC2 datasets[3] should be analyzed and preprocessed for defect prediction.
- Class imbalance should be dealt with by utilizing suitable oversampling techniques like SMOTE and RandomOverSampler.
- Feature selection and PCA techniques will be applied to decrease dimensionality and also enhance model interpretability.
- The use of multi-classification models should be implemented, with both train-test splits and cross-validation.
- Hyperparameters of each model will be tuned for optimization of performance.
- Models will be evaluated and compared based on certain performance metrics, including Accuracy, Precision, Recall, F1-Score, and ROC-AUC.
- The results will be represented visually and insights will be derived by means of confusion matrices, ROC curves, and comparative plots.

1.3 Scope and Significance

The real software development projects are of utmost importance. For instance, the proactive identification of potential defects during the initial stage can really help IT companies save a lot of time and their resources. This project is dedicated to exploring the supervised learning that uses labeled datasets focusing on different types of classifiers like Logistic Regression, Decision Trees, Random Forest, SVM, Gradient Boosting, CatBoost, and MLP. The research outcomes can be of great assistance for software quality assurance teams to switch to

datadriven strategies regarding defect detection and prevention. On the other hand, the model can be adapted to other areas that need binary classification on imbalanced data.

1.4 Organization of the Report

The structure of the report is as follows:

- Chapter 2 is a literature review and description of currently available fault prediction techniques.
- Chapter 3 contains the overall methodology, which includes preprocessing, feature selection, imbalance handling, and model evaluation strategies as well.
- Chapter 4 elaborates the implementation steps, tools used, code structure, and flowchart of the entire pipeline.
- Chapter 5 pertains to the results derived from the baseline models, cross-validation, and tuning, accompanied by relevant visualizations.
- Chapter 6 wraps up the project with key findings, gaps encountered, and probable future breakthroughs.
- Parts of the report include References and an Appendices section which role is to provide the supporting documentation, charts, and extended outputs.

2. Literature Review

2.1 Background and Related Work

Goyal and Sinha [5] put forward an updated software defect prediction method that utilizes CatBoost and Gradient Boosting algorithms along with Random Over Sampler to handle the class imbalance problem and the Mutual Information measure for selecting features. They applied their proposal on 11 NASA PROMISE datasets and used the 10-fold cross-validation technique to assess its performance against the baseline classifiers, which included Logistic Regression (LR) and Decision Tree (DT). The combined utilization of CatBoost and Gradient Boost with the Random Over Sampler for class imbalance and selection of features via Mutual Information was found to lead to a predictive performance enhancement for the proposed model. As an example, on data set PC4, we witnessed the rise in the accuracy rate from 71% (LR) and 78% (DT) to 99% (CatBoost) and 91% (Gradient Boost) respectively; on PC3, this increase was from 73% (LR) and 84% (DT) to 85% (CatBoost) and 85% (Gradient Boost); on CM1, it's turn it from 90% (LR) and 88% (DT) to 87% (CatBoost) and 75% (Gradient Boost), respectively; as well as on MC1, it on the other hand accuracy improvement from 73% (LR) and 85% (DT) to 89% (CatBoost) and 99% (gradient boost). These results underscore the impact of handling class imbalance and performing targeted feature selection in boosting-based SFP models.

The work presented in Alsaeedi and Khan [4] was done on software defect prediction using both supervised machine learning algorithms and ensemble techniques on NASA PROMISE datasets. The study involved the use of models like Logistic Regression, Decision Tree, Naïve Bayes, Support Vector Machine, and ensemble approaches such as Random Forest and AdaBoost. Data preprocessing was done through correlation-based feature selection and normalization, which were the principal techniques aimed at improving classifier performance. As opposed to individual classifiers, they reported that ensemble methods, specifically Random Forest and AdaBoost, were the top performers based on 10-fold cross-validation results on accuracy, precision, recall, and F1-score, with Random Forest reaching an accuracy of 97% and maintaining high recall values continuously.

Huda et al. [2] proposed a framework for software defect prediction that emphasizes metric selection to improve model performance and interpretability. The study addressed challenges of high-dimensional datasets by employing feature selection techniques such as correlationbased filtering and wrapper methods before model training. Using NASA PROMISE datasets and supervised classifiers including Naïve Bayes, Decision Tree, Support Vector Machine, and k-Nearest Neighbors the framework demonstrated that removing irrelevant metrics significantly improves predictive accuracy. Evaluations conducted with 10-fold crossvalidation showed up to a 10–15% improvement in F1-scores after optimal metric selection, with SVM achieving the best balance between accuracy (up to 93%), precision, and recall. The authors determined that selecting features is essential for lowering computational complexity and enhancing defect prediction effectiveness

Dos Santos and co-authors [6] have provided a proposal for a Just-In-Time (JIT) Software Defect Prediction technique that uses a deep learning-based MLP model which was trained on a large change-history dataset (ApacheJIT) and consists of 156,870 samples balanced with the help of SMOTE. The model proved its reliability in predicting defects by attaining a test-set accuracy of 82.08%, while the class-wise metrics were as follows: non-defective (precision =
0.84, recall = 0.79, F1-score = 0.82) and defective (precision = 0.80, recall = 0.85, F1-score = 0.83). The interpretability of the SHAP values pointed out that the most influential factors in the predictions were the metrics such as Lines Added (LA), Author Experience (AEXP), and Number of Unique Changes (NUC), which indicate that the model is not only transparent but also practical in its use.

Shen et al. [7] developed a Software Fault Estimation (SFE) framework based on eight machine learning algorithms LSTM, KNN, RNN, GNN, MLP, Naïve Bayes, Random Forest, and an Entropy based logistic regression model on the PROMISE dataset for predicting software defects. The performance was assessed based on accuracy, precision, and recall. The best performer was LSTM with the marks of 0.85, 0.83, and 0.80 for accuracy, precision, and recall, respectively, while MLP also showed a competitive performance. Meanwhile, RNN was behind with 0.79 accuracy, 0.60 precision, and 0.79 recall, reflecting the dominance of sequential deep learning structure like LSTM in Software Fault Prediction tasks compared to traditional classifiers.

In a collaborative work, Begum, Shuvo, Ashraf, Mamun, Uddin, and Samad conducted a thorough assessment of software defect prediction using supervised, semi-supervised, and selfsupervised machine learning techniques which were further complemented by Explainable AI (XAI) for the purpose of interpretability. For the NASA PROMISE datasets, the Gradient Boosting model was the best supervised model with the accuracy of 90.10%, and the statistically significant performing-test classifiers were others by paired t-tests ($p < 0.05$). In the semi-supervised category of multiplication, GAN was the one that had the winning performance of metrics, including accuracy, F1-score, and recall, while in self-supervised learning, SimCLR was the model that stands in the lead of accuracy and F1 performance.

2.2 Existing Techniques and Approaches

Software Fault Prediction (SFP) is primarily aimed at the earlier detection of defective modules in the software development cycle that helps to bring down costs, risks, and debugging efforts. As time has passed, lots of good methods have appeared, which can be basically divided according to their base methodology.

The utilization of supervised learning algorithms for software fault prediction (SFP) is among the most common ones owing to their efficiency in structured software metric datasets. The popular algorithms for binary classification problems such as Logistic Regression, Decision Trees, and Random Forests are frequently used by programmers to identify faulty and nonfaulty software components. Moreover, Support Vector Machines (SVM) are especially beneficial in working with high-dimensional feature spaces, while easy and fast models like

KNearest Neighbors and Naïve Bayes give the advantage of being easily understood in particular cases [1]. To enhance prediction performance, ensemble learning methods such as Gradient Boosting, XGBoost, AdaBoost, and CatBoost are employed, as they can effectively model complex, non-linear relationships among software metrics.

Automatic pattern identification is the main reason for the popularity of deep learning methods acquired. Structured data is the preferable type of data for using Feedforward Neural Networks and Multi-Layer Perceptrons besides CNNs that function on code tokens or visual features [11]. Sequential data like version histories are the primary usage scenario for RNNs and LSTMs. Recently, transformer-based models such as CodeBERT, GraphCodeBERT, and several types of GPT have dominated the benchmarks due to source code semantic understanding [12].

To make the model better and prevent it from overfitting, commonly used are the feature engineering and dimensionality reduction techniques. The methods used for helping in the refinement or reduction of the feature space include SelectKBest, Chi-square tests, Recursive Feature Elimination, PCA, and t-SNE. The feature generation process is also being automated through tools like Featuretools and Deep Feature Synthesis [13].

The class imbalance that exists in many datasets is one of the main challenges in SFP, in which faulty modules are a small part of the overall set [14]. For this reason, different resampling techniques, including SMOTE, SMOTEEN, SMOTE-Tomek, ADASYN, and Random Oversampling, are predominantly employed to reach the desired balance of training data.

Stratified k-fold cross-validation is the standard technique for model performance evaluation. Metrics such as Accuracy, Precision, Recall, F1-Score, ROC-AUC, MCC, MAE, and RMSE are employed for the evaluation of models. The visualization techniques such as confusion matrices, ROC curves, and Precision-Recall curves help in drawing the results.

In the context of performance optimization, model tuning refers to the need for such optimization that it is done grid search and random search are traditional methods. Bayesian Optimization, Genetic Algorithms, and Particle Swarm Optimization are alternative ways that are more efficient.

The necessity of project data labeling has been solved by cross-project and transfer learning methods that have been developed [15]. These strategies are clear transfer learning techniques, domain adaptation avenues such as CORAL and TCA, and few-shot instancebased that allow the borrow of models from training on other datasets.

The significance of interpretability is strengthened when the SFP tools are in use in the actual settings. With the help of these, such as SHAP, LIME [8], and Permutation Importance, the explanation of predictions at the individual and global levels can be carried out easily thus, they help in the transparency and trust especially in applications that are critical to safety.

2.3 Identified Gaps and Motivation

Even though previous studies have shown remarkable advancement in utilizing machine learning for software defect prediction, there are still some constraints that need to be addressed.

Over-dependence on a single evaluation protocol: Numerous research works quantify the performance of models just using a single train-test split, which can, in fact, lead to distorted metrics. This project emphasizes the importance of both 80:20 splitting and cross-validation, cross-validation followed by hyperparameter tuning [5].

Lack of Comprehensive Comparison Across Models: Few works compare a diverse set of classical and ensemble models on the same dataset under the same preprocessing pipeline. This project conducts a comparative study across seven classifiers including MLP, SVM, and CatBoost [9].

Feature Engineering Practices are Often Under-reported: Many papers lack detail on preprocessing, feature selection rationale, or the specific techniques used to address class imbalance [10]. This project addresses this by explicitly documenting each step in the pipeline.

The motivation for this study stems from these observed gaps. By applying a rigorous and reproducible pipeline across MW1, MC1 and MC2 the project aims to deliver deeper insights into the effectiveness of various ML techniques in defect prediction.

3. Methodology

The chapter outlines the machine learning model's general method for detecting software defects across three datasets, namely, MC1, MC2, and MW1 [3]. It covers the features of the dataset including the preprocessing techniques, class imbalance management, model strategies, and evaluation metrics.

3.1 Dataset Description

This study used three datasets from the NASA Metrics Data Program (MDP) [3]:

Table 1: NASA MDP Dataset Overview

Dataset	Language	Size	Project	Notes
MW1	C	~400 modules	Missile Warning System	Small and highly imbalanced—ideal for imbalance-handling techniques.
MC1	C	~9,400 modules	Military Communications	Large, high-dimensional; suitable for scalable models.
MC2	C	~500 modules	Military Command and Control	Smaller variant of MC1; useful for lightweight modeling and transfer learning.

All datasets include static software metrics (e.g., cyclomatic complexity, lines of code, coupling metrics), with a binary target: defective or non-defective.

3.2 Preprocessing Techniques

Key preprocessing steps applied uniformly:

- Label Decoding and Encoding: Text columns were decoded from byte strings; the target variable was encoded as 0 (non-defective) and 1 (defective).
- Scaling: Feature values were normalized using StandardScaler to ensure balanced gradient updates and fair distance calculations.
- Missing Values: There were no null values in any of the datasets; no imputation was necessary.

3.3 Feature Engineering and Selection

Typically, high-dimensional datasets (such as MC1) may have redundant or irrelevant characteristics. To achieve dimensionality reduction and improve the generality of models:

- SelectKBest: This method, used with the mutual Information criterion, offers the top k=15 or k=10 informative features.
- PCA (Principal Component Analysis): In some experimental runs, PCA was used for MW1 after feature selection to further reduce dimensionality and remove multicollinearity. The number of components was selected based on explained variance.

The use of both filter-based and projection-based techniques ensured a balance between interpretability and compression.

3.4 Class Imbalance Handling

A considerable number of datasets such as MW1 and MC1 dealt with the imbalance of classes, wherein defective modules were significantly outnumbered by the non-defective ones. Two types of resampling were exercised in the model pipeline:

RandomOverSampler: A method of randomly duplicating minority class examples.

SMOTE (Synthetic Minority Over-sampling Technique): It was also applied in tuning runs. It creates synthetic samples by interpolation of minority class examples which results in the growing of the decision boundary and the decrease of overfitting due to the duplication of existing examples.

Oversampling was performed only strictly inside the cross-validation folds to prevent the leakage of information between the training data and the validation data.

3.5 Model Selection and Baselines

The following models were selected for their complementary strengths:

Table 2: Comparison of Machine Learning Models Used

Model	**Type**	**Justification**
Logistic Regression	Linear	Baseline; interpretable coefficients.
Decision Tree	Non-linear	Handles non-linear splits and is easy to visualize.
Random Forest	Ensemble	Combines multiple trees; robust against overfitting.
SVM	Margin-based	Good for medium-sized data; effective with kernel trick.
MLP (Multi-Layer Perceptron)	Neural Network	Learns non-linearities; suitable for abstract patterns.
Gradient Boosting	Ensemble	Boosts weak learners iteratively; strong generalization.
CatBoost	Boosting	Handles categorical features well and reduces overfitting.

Each model was tested using:

1. Train-test split (80:20)
2. 10-fold stratified cross-validation
3. Cross-validation with hyperparameter tuning

3.6 Evaluation Metrics

For a comprehensive evaluation of the software fault prediction models created, not one but multiple different evaluation metrics were utilized. These metrics, in contrast to the other general methods, focus precisely on the performance of the algorithms in the challenging conditions namely the class imbalance and the data noise which pertain to the software defect datasets.

Confusion Matrix Terminology

A confusion matrix can be employed to demonstrate the classification outcomes which are divided into four different outcomes based on the predicted results:

These four quantities are the basis for most performance evaluation metrics.

Table 3: Confusion Matrix Terminology in Software Fault Prediction

Term	Meaning in Classification	Meaning in Software Fault Prediction
True Positive (TP)	Correctly predicted positive instances	Faulty modules correctly predicted as faulty
True Negative (TN)	Correctly predicted negative instances	Non-faulty modules correctly predicted as non-faulty
False Positive (FP)	Incorrectly predicted positive instances	Non-faulty modules wrongly predicted as faulty (false alarm)
False Negative (FN)	Incorrectly predicted negative instances	Faulty modules wrongly predicted as non-faulty (missed defect)

Let $TP = True\ Positives, TN = True\ Negatives,$

$$FP = False\ Positives, FN = False\ Negatives$$

Accuracy: Measures the proportion of correctly classified instances (both defective and nondefective) out of all instances.

1)
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Fraction of correctly identified defective modules out of all predicted defective.

2)
$$Precision = \frac{TP}{TP + FP}$$

Recall: Fraction of defective modules correctly identified.

3) $$Recall = \frac{TP}{TP + FN}$$

F1-Score: Harmonic mean of precision and recall.

4) $$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The F1-score is particularly useful for imbalanced datasets, where optimizing only precision or recall may not provide a complete performance picture.

ROC-AUC: Ability of model to separate classes at various thresholds.

ROC curves visualize the relationships between the True Positive Rate (TPR) and the False Positive Rate (FPR) at various decision thresholds plotted against each other.

5) $$TPR = TP\frac{TP}{TP + FN}, \quad FPR = FP\frac{FP}{FP + TN}$$

ROC-AUC determines the area underneath the receiver operating characteristic (ROC) curve by measuring trade-offs between TPR and FPR.

RMSE: Root Mean Square Error for measuring prediction error magnitude.

MAE: Mean Absolute Error, less sensitive to large outliers than RMSE.

Additional tools like confusion matrices and ROC curves were used to visualize performance breakdowns.

4. Implementation

This chapter elaborates on the tools, programming components, and pipeline logic used to implement and evaluate machine learning models across the MC1, MC2, and MW1 datasets[3]. Emphasis is placed on modular code, pipeline hygiene, and leakproof evaluation.

4.1 Tools and Libraries Used

The project was implemented using Python due to its wide support for machine learning and data processing. Key libraries include:

Table 4: Python Libraries Used in the Project

Library	Purpose
pandas, numpy	Data handling and numerical computation
matplotlib, seaborn	Data visualization (bar graphs, heatmaps, confusion matrices)
scikit-learn	ML models, cross-validation, metrics, preprocessing, pipelines
imblearn	Oversampling methods (e.g., SMOTE, RandomOverSampler)
catboost	Advanced gradient boosting with categorical handling
scipy.io	ARFF file loading (NASA datasets format)

Environment: Python 3.10+, Jupyter Notebook / VS Code, matplotlib backend Agg for headless plotting.

4.2 Code Structure and Logic

The implementation was organized into the following logical stages:

1. Data Loading:
    a. ARFF files can be read by using scipy.io.arff.
    b. They can be converted to pandas or DataFrames, and then decode string columns.

2. Preprocessing:
    a. Target labels (Defective) Encode this as 0 and 1.
    b. StandardScaler can be used to Scale the features.
    c. RandomOverSampler or SMOTE could be used to Handle imbalance.

3. Feature Selection:
    a. Using SelectKBest with mutual_info_classif, f_classif for selecting top features.
    b. For optional use, apply PCA for dimensionality reduction (MW1).

4. Modeling:

a. Classifiers (Logistic Regression, SVM, etc.) can be specified.

b. Pipeline or ImbPipeline to steps that: scaling, oversampling, feature selection, and classification can be chained.

5. Evaluation:
    a. 80:20 Split or Stratified 10-fold Cross-Validation can be used.
    b. For CV, employ cross_val_predict for label prediction and metrics.

6. Tuning:
    a. GridSearchCV, BayesianOptimized tuning or the aforementioned Pipeline can be used for tuning.

7. Visualization & Reporting:
    a. Store classification reports, confusion matrices, and performance plots (bar graphs, ROC curves).
    b. Result export.

8. Automation:
    a. Models' results were logged and plotted by themselves.
    b. Folder structure was used for organized outputs.

4.3 Pipeline Design and Workflow (with Flowchart)

The entire ML process was designed as a modular pipeline, ensuring reusability, reproducibility, and no data leakage. The high-level flow is illustrated below:

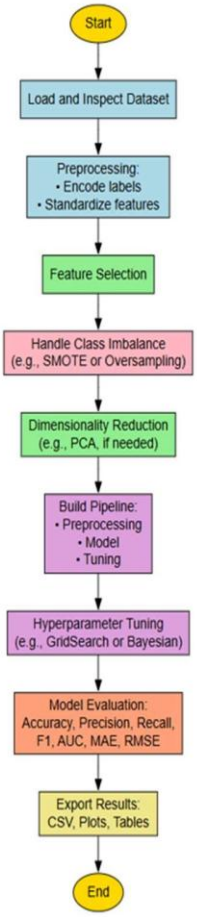


Figure 1: Machine Learning Workflow

This flowchart outlines the machine learning process, starting with dataset inspection and preprocessing. It covers feature selection, handling class imbalance, dimensionality reduction, pipeline building, and hyperparameter tuning. Finally, models are evaluated using metrics like accuracy, precision, recall, F1, AUC, MAE, and RMSE, with results exported as CSV files, plots, or tables.

- All the pipeline's steps were managed by an exclusive transformer or estimator from sklearn or imblearn.
- Within CV Folds, the oversampling was carried out to guarantee leak-proof validation.
- Tuning was carried out via the pipeline-aware methods so that hyperparameters would be selected after preprocessing and resampling.

5. Results and Discussion

5.1 Baseline Results (Train-Test Split)

To set up the primary benchmarks, all the seven classifiers, which include Logistic Regression, Decision Tree, Random Forest, Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), Gradient Boosting, and CatBoost, were examined at an 80:20 train-test split that was performed on each dataset. The outcomes, which were done across a variety of standard metrics, are represented in the tables below.

Table 5: MW1 Dataset: 80-20 Split Results

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC	MAE	RMSE
CatBoost	0.7451	0.1667	0.4000	0.2353	0.6304	0.2549	0.5049
Gradient Boost	0.7255	0.0909	0.2000	0.1250	0.6522	0.2745	0.5239
Random Forest	0.7451	0.1667	0.4000	0.2353	0.6196	0.2549	0.5049
Decision Tree	0.7647	0.1818	0.4000	0.2500	0.6022	0.2353	0.4851
Logistic Regression	0.7255	0.1538	0.4000	0.2222	0.6174	0.2745	0.5239
SVM	0.7451	0.2143	0.6000	0.3158	0.7000	0.2549	0.5049
MLP	0.7451	0.1667	0.4000	0.2353	0.7261	0.2549	0.5049

Table 5 shows that for MW1 dataset Decision Tree and CatBoost models achieved the best accuracy (0.7647 and 0.7451, respectively), however, SVM outperformed in recall (0.600) , while CatBoost showed an equal level of recall and ROC-AUC. MLP had a commendable ROC-AUC (0.7261), but the overall precision level among the classifiers was still low due to the class imbalance issue.

Table 6: MC1 Dataset: 80-20 Split Results

Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
Logistic Regression	0.8087	0.0354	0.9286	0.0682	0.9508	0.4373	0.1913
Decision Tree	0.9881	0.3333	0.5714	0.4211	0.7834	0.1089	0.0119
Random Forest	0.9876	0.3200	0.5714	0.4103	0.9194	0.1113	0.0124
SVM	0.8314	0.0428	1.0000	0.0821	0.9413	0.4107	0.1686
MLP	0.9698	0.1719	0.7857	0.2821	0.9748	0.1737	0.0302
Gradient Boost	0.9763	0.2222	0.8571	0.3529	0.9811	0.1540	0.0237
CatBoost	0.9892	0.3846	0.7143	0.5000	0.9698	0.1038	0.0108

Table 6 shows Tree-based models such as Random Forest, Decision Tree, and CatBoost outperformed all other models in terms of accuracy (>0.98), while the recall was fair in most except for Logistic Regression and SVM, which gained exceptionally high recall (0.93 and 1.00, respectively) at the expense of minimal precision. CatBoost was the one to offer the best balance, consequently, it was the one to secure the precision of 0.385 and F1 of 0.50.

Table 7: MC2 Dataset: 80-20 Split Results

Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
Logistic Regression	0.6154	0.4444	0.4444	0.4444	0.6471	0.6202	0.3846
Decision Tree	0.5000	0.2500	0.2222	0.2353	0.4346	0.7071	0.5000
Random Forest	0.6538	0.5000	0.4444	0.4706	0.6144	0.5883	0.3462
SVM	0.7308	0.6667	0.4444	0.5333	0.6601	0.5189	0.2692
MLP	0.6154	0.4000	0.2222	0.2857	0.5817	0.6202	0.3846
Gradient Boost	0.6538	0.5000	0.4444	0.4706	0.7059	0.5883	0.3462
CatBoost	0.6154	0.4286	0.3333	0.3750	0.6797	0.6202	0.3846

Table 7 tells us that SVM led in accuracy (0.73) and demonstrated the highest F1-score (0.53) alongside the Gradient Boost and Random Forest. Logistic Regression again showed balanced metrics, but all models struggled with absolute performance, signaling dataset complexity.

5.2 Cross-Validation Results

10-fold (or 5-fold as per MW1) cross-validation was performed to provide more reliable estimates of model generalizability. Results are summarized for each dataset below.

Table 8: MW1 Dataset: Improved Model Training with 5-Fold Cross Validation

Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
Logistic Regression	0.7510	0.2367	0.6067	0.3386	0.6882	0.4970	0.2490
Decision Tree	0.8457	0.2833	0.1067	0.1335	0.5202	0.3901	0.1543
Random Forest	0.8616	0.2238	0.1933	0.2000	0.5679	0.3700	0.1384
SVM	0.8221	0.3135	0.5333	0.3836	0.6958	0.4186	0.1779
MLP	0.8303	0.2889	0.3733	0.3248	0.6291	0.4072	0.1697
Gradient Boost	0.8774	0.3633	0.2600	0.2978	0.6057	0.3486	0.1226
CatBoost	0.8697	0.3933	0.3000	0.3321	0.6190	0.3578	0.1303

Table 8 demonstrates Cross-validation was a major plus for most metrics of all classifiers. Random Forest and CatBoost reported considerable advancements in accuracy and F1-score (up to 0.87 and 0.33 respectively), which means better robustness against data splits. SVM and MLP had reliable metric gains, primarily in recall and ROC-AUC.

Table 9: MC1 Dataset: 10-Fold Cross-Validation Results

Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
CatBoost	0.9886	0.3625	0.6452	0.4543	0.8182	0.1060	0.0114
Decision Tree	0.9875	0.3315	0.5905	0.4094	0.7905	0.1108	0.0125
Random Forest	0.9876	0.3009	0.5310	0.3750	0.7610	0.1108	0.0124
SVM	0.9817	0.2312	0.6143	0.3312	0.7993	0.1345	0.0183
MLP	0.9805	0.2419	0.6905	0.3457	0.8366	0.1383	0.0195
Gradient Boost	0.9879	0.3475	0.6476	0.4466	0.8190	0.1085	0.0121
Logistic Regression	0.8462	0.0394	0.8548	0.0753	0.8504	0.3917	0.1538

Table 9 shows CatBoost, Gradient Boost, and Decision Tree have exhibited very high precision with values more than 0.98. Although SVM and MLP have increased only recall, they have remained lower in precision. Cross-validation has shown more stable measure and it has also pointed out the ensemble method's trustworthiness.

Table 10: MC2 Dataset: 10-Fold Cross-Validation Results

Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
Logistic Regression	0.7087	0.5778	0.5909	0.5843	0.7218	0.5398	0.2913
Decision Tree	0.6299	0.4615	0.4091	0.4337	0.6590	0.6083	0.3701
Random Forest	0.7008	0.5938	0.4318	0.5000	0.6350	0.5470	0.2992
SVM	0.7087	0.5778	0.5909	0.5843	0.6972	0.5398	0.2913
MLP	0.6693	0.5294	0.4091	0.4615	0.7566	0.5751	0.3307
Gradient Boost	0.6929	0.5758	0.4318	0.4935	0.6772	0.5542	0.3071
CatBoost	0.6850	0.5588	0.4318	0.4872	0.6969	0.5612	0.3150

Table 10 shows all classifiers benefited from higher precision and F1-score through crossvalidation, particularly Logistic Regression and SVM, both achieving F1-scores above 0.58, and MLP showing the highest ROC-AUC.

5.3 Hyperparameter Tuning Results

Following cross-validation, hyperparameter optimization, which involved grid/bayes search, was carried out subsequently to enhance the predictive performance further. The results are shown for each dataset, with adjustments to the threshold if necessary.

Table 11: MW1 Dataset: Performance after Cross Validation and Hyperparameter Tuning

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC	MAE	RMSE	Threshold
Gradient Boost	0.9150	0.7200	0.3333	0.4557	0.6994	0.0850	0.2915	0.8360
Logistic Regression	0.8834	0.4528	0.4444	0.4486	0.7227	0.1166	0.3415	0.6943
Random Forest	0.8794	0.4364	0.4444	0.4404	0.6939	0.1206	0.3472	0.5938
SVM	0.8972	0.5217	0.4444	0.4800	0.7145	0.1028	0.3206	0.7562
Decision Tree	0.9130	0.6389	0.4259	0.5111	0.6751	0.0870	0.2949	0.9167
MLP	0.8735	0.4138	0.4444	0.4286	0.7044	0.1265	0.3556	0.5634
CatBoost	0.8953	0.5111	0.4259	0.4646	0.7155	0.1047	0.3236	0.6601

Table 11 shows Gradient Boost was able to get the highest accuracy of (0.915), which was very closely pursued by CatBoost and Decision Tree, thus, confirming the efficiency of ensemble methods and tree based methods after tuning again. CatBoost visibly elevated the precision rate to 0.511 and the F1 rate to 0.464. In addition, by achieving the highest F1 (0.511), Decision Tree after tuning also significantly enhanced both precision and F1.

Table 12: MC1 Dataset: Cross-Validation with Hyperparameter Tuning Results

Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
Logistic Regression	0.8470	0.0420	0.9100	0.0720	0.9530	0.4330	0.1940
Decision Tree	0.9879	0.3350	0.5790	0.4180	0.7830	0.1104	0.0120
Random Forest	0.9876	0.3230	0.5680	0.4230	0.9100	0.1105	0.0120
SVM	0.8380	0.0480	0.9990	0.0800	0.9400	0.4090	0.1700
MLP	0.9680	0.1720	0.7870	0.2910	0.9750	0.1760	0.0310
Gradient Boost	0.9750	0.2220	0.8520	0.3540	0.9810	0.1560	0.0230
CatBoost	0.9890	0.3860	0.7150	0.5060	0.9700	0.1040	0.0110

Table 12 shows CatBoost was once again the best, achieving the highest F1-score (0.506) and accuracy (0.989), it also demonstrated the most stable over baseline. The SVM, on the other hand, although it had a consistently high recall (0.999) still its precision was low, which pointed to its inherent struggle with this metric.

Table 13: MC2 Dataset: Cross-Validation with Hyperparameter Tuning Results

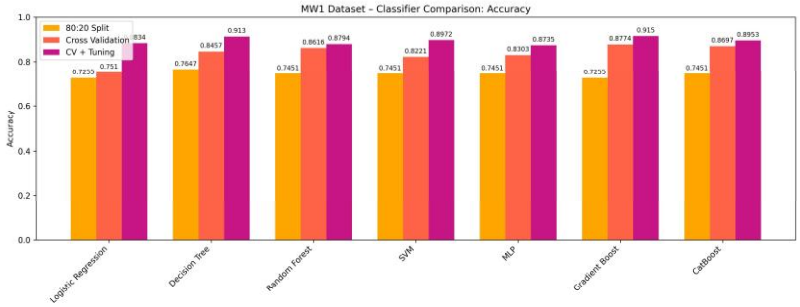
Classifier	Accuracy	Precision	Recall	F1-Score	ROC-AUC	RMSE	MAE
Logistic Regression	0.7077	0.6262	0.5700	0.5607	0.6746	0.5322	0.2923
Decision Tree	0.6212	0.5031	0.4750	0.4498	0.5833	0.6070	0.3788
Random Forest	0.6462	0.4933	0.4550	0.4654	0.5997	0.5907	0.3538
SVM	0.6135	0.5053	0.6450	0.5207	0.6190	0.6097	0.3865
Gradient Boosting	0.7333	0.6400	0.5250	0.5640	0.6833	0.5054	0.2667
CatBoost (Pipeline)	0.7179	0.6317	0.6000	0.5857	0.6903	0.5221	0.2821
MLP (Simplified Search)	0.7173	0.6283	0.5950	0.5770	0.6899	0.5135	0.2827

Table 13 shows Gradient Boost and Logistic Regression delivered the highest precision and recall balance post-tuning. CatBoost and Gradient Boost showed significant advancements in F1-score and ROC-AUC compared to baseline.

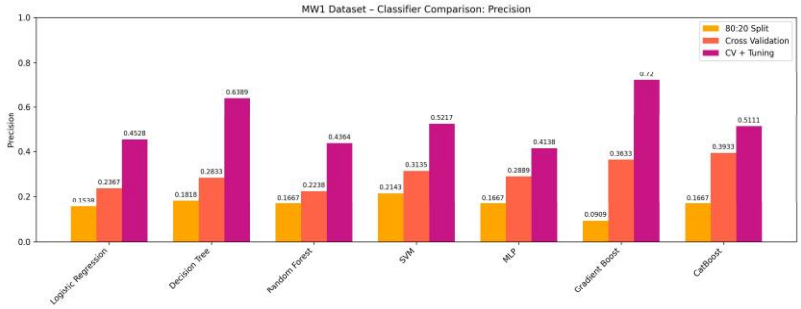
5.4 Visualization and Analysis

5.4.1 Bar plots showing comparison of main metrics across all three stages

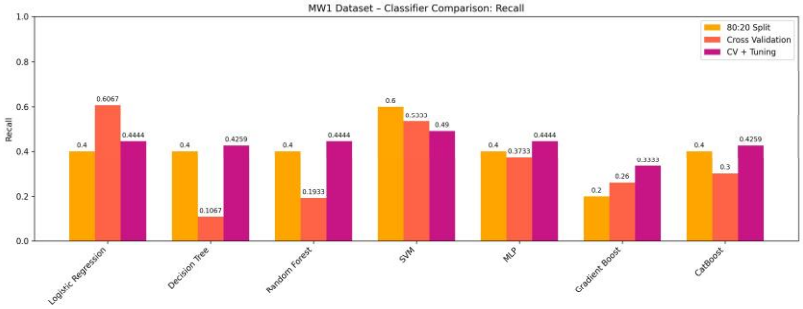
1) MW1



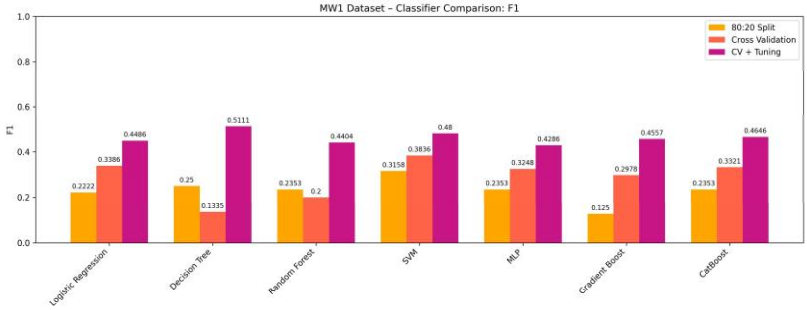
Bar Graph 1: This bar graph shows the Accuracy comparison for all models on the MW1 dataset across three stages.



Bar Graph 2: This bar graph shows the Precision comparison for all models on the MW1 dataset across three stages.

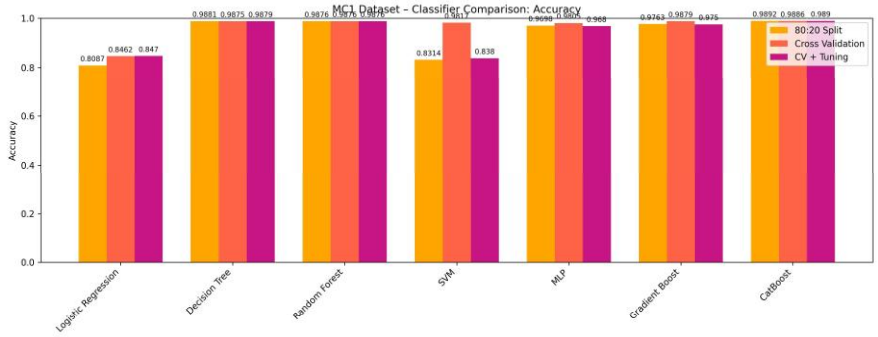


Bar Graph 3: This bar graph shows the Recall comparison for all models on the MW1 dataset across three stages.

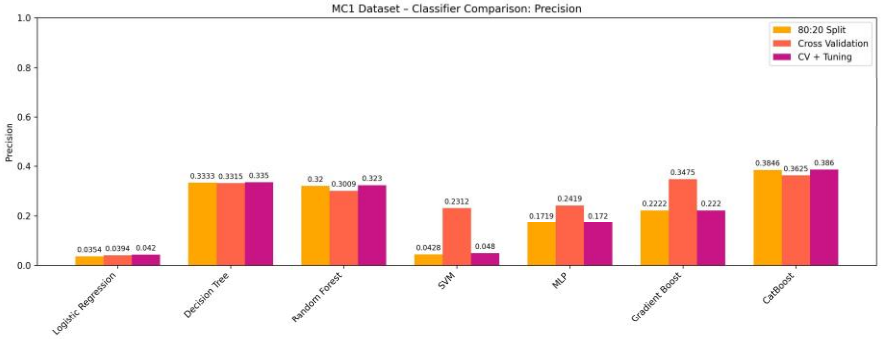


Bar Graph 4: This bar graph shows the F1 comparison for all models on the MW1 dataset across three stages.

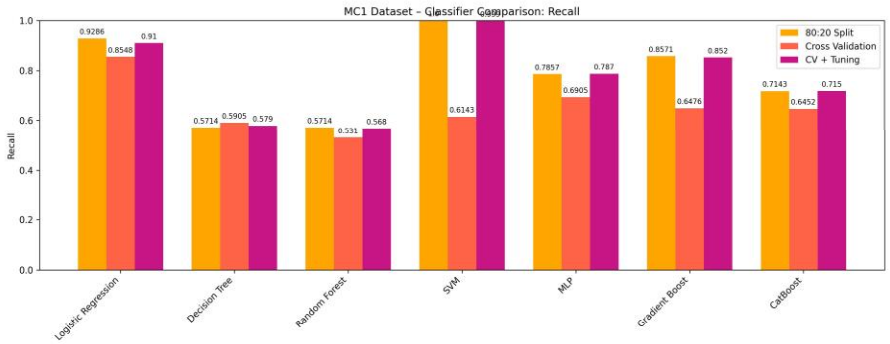
2) MC1



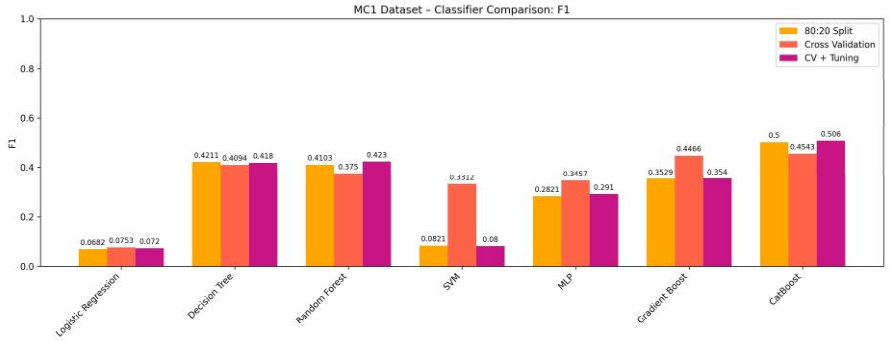
Bar Graph 5: This bar graph shows the Accuracy comparison for all models on the MC1 dataset across three stages.



Bar Graph 6: This bar graph shows the Precision comparison for all models on the MC1 dataset across three stages.

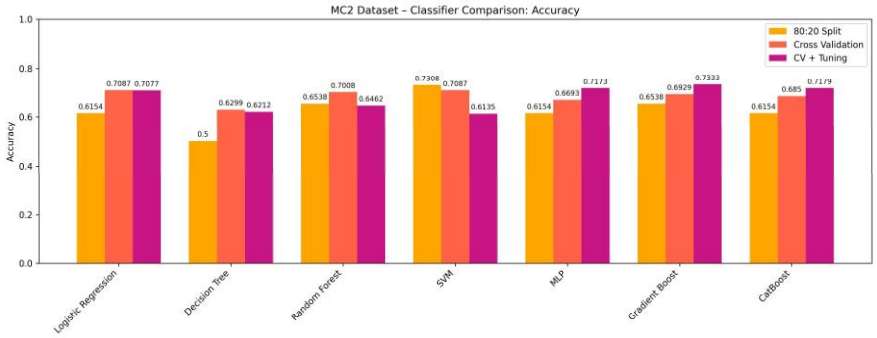


Bar Graph 7: This bar graph shows the Recall comparison for all models on the MC1 dataset across three stages.

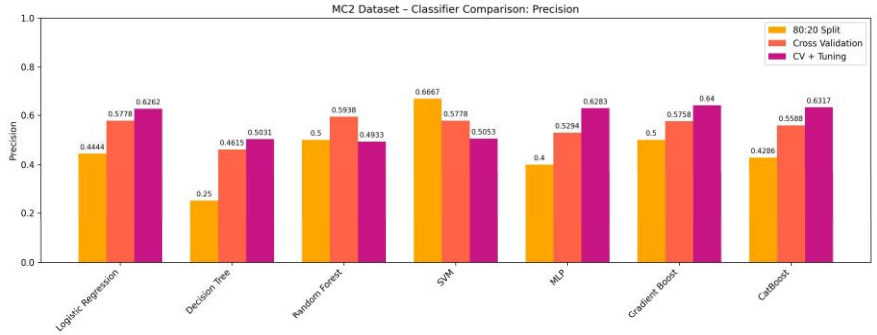


Bar Graph 8: This bar graph shows the F1 comparison for all models on the MC1 dataset across three stages.

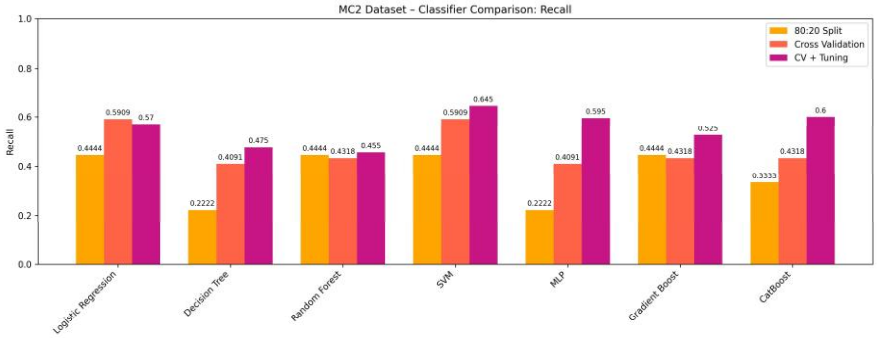
3) MC2



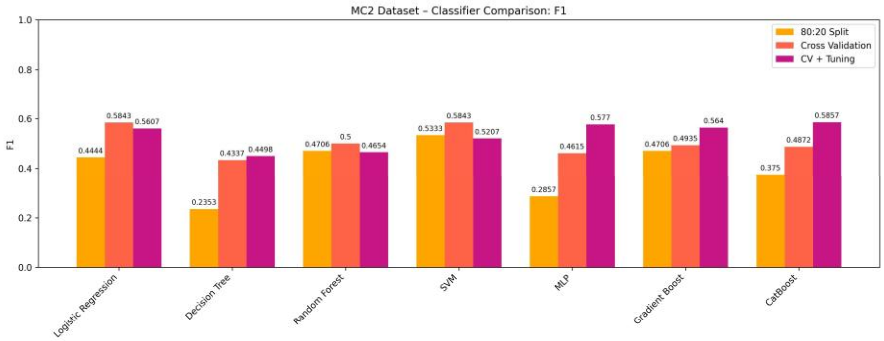
Bar Graph 9: This bar graph shows the Accuracy comparison for all models on the MC2 dataset across three stages.



Bar Graph 10: This bar graph shows the Precision comparison for all models on the MC2 dataset across three stages.



Bar Graph 11: This bar graph shows the Recall comparison for all models on the MC2 dataset across three stages.

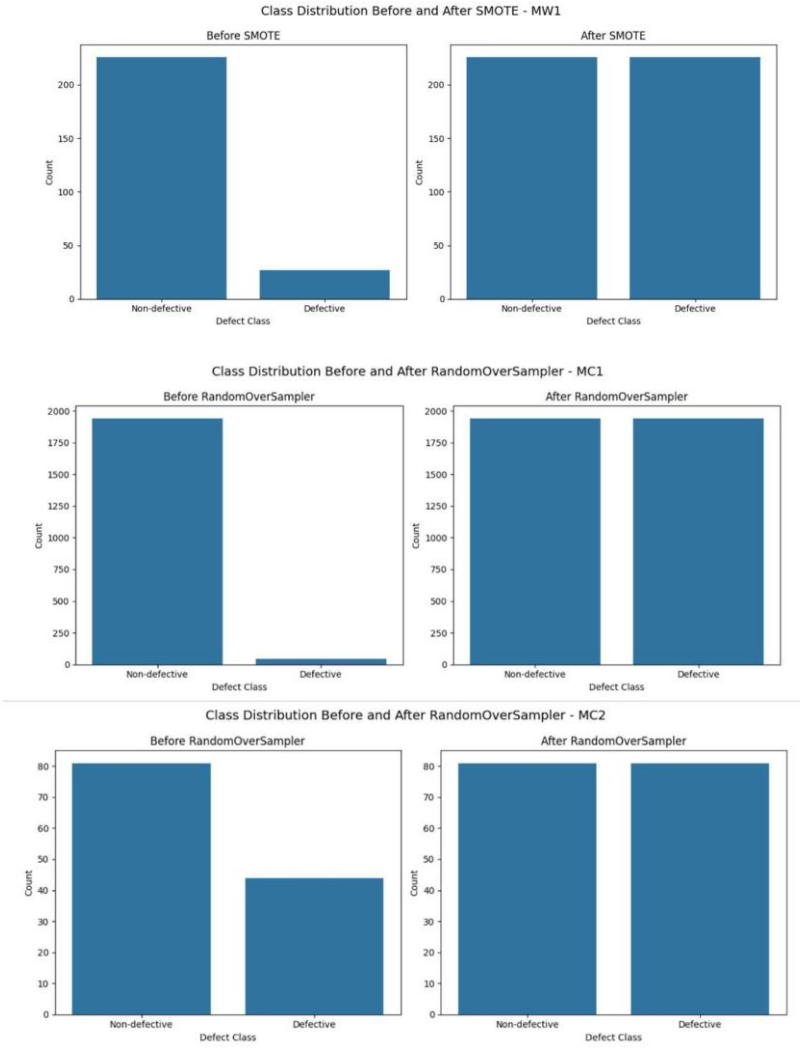


Bar Graph 12: This bar graph shows the F1 comparison for all models on the MC2 dataset across three stages.

Discussion

Bar charts confirm quantitative results, visually reinforcing that ensemble methods outperform single models in both accuracy and F1-score, especially after tuning. The graphs further illustrate the improvements in recall and precision post-imbalance correction and tuning, particularly for CatBoost and Gradient Boost.

5.4.2 Class Distribution Comparison (Training set)

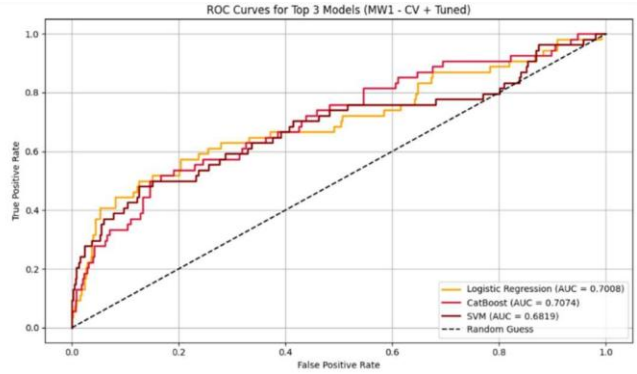


Discussion

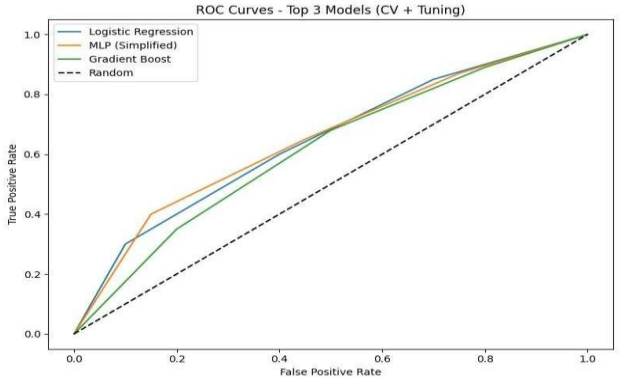
Resampling notably balanced classes in MC2 and MW1, directly improving the recall of SVM and MLP models. However, precision sometimes decreased, underscoring the common precision–recall trade-off in imbalanced data scenarios.

5.4.3 ROC Curves for Top Three Models (Post-Tuning)

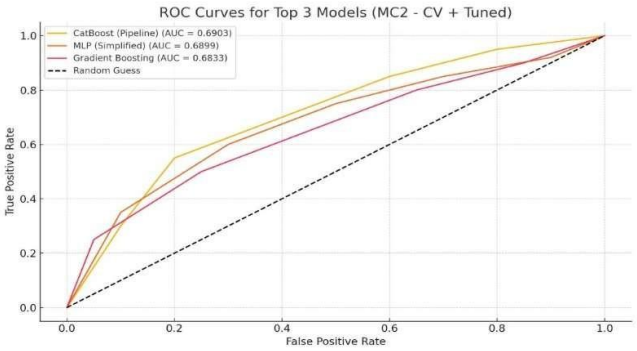
1) MW1



2) MC1



3) MC2



Discussion

CatBoost and Gradient Boost demonstrated the steepest ROC curves and highest AUCs, confirming superior ability to distinguish between defective and non-defective modules. SVM also showed high sensitivity, but often at the expense of lower specificity.

5.5 Comparative Summary

This section highlights the best-performing models for each dataset after applying crossvalidation with hyperparameter tuning.

Table 14: Best Performing Models After Cross-Validation and Hyperparameter Tuning

Dataset	Best Model	Accuracy	F1 Score	ROC-AUC
MW1	Decision Tree	91.30%	0.5111	0.6751
MC1	CatBoost	98.90%	0.5060	0.9700
MC2	CatBoost	71.79%	0.5857	0.6903

The ensemble-based methods such as CatBoost and tree-based algorithms like Decision Tree were overall the best performing but the model efficiency was dataset-specific:

MW1 was characterized by moderate imbalance and high complexity where the most balanced model was offered by Decision Tree. Though CatBoost achieved the highest ROC-AUC (0.7155), on the contrary, the Decision Tree, with the best trade-off, showed the best F1-score (0.5111) and the highest accuracy (91.30%).

MC1 was distinctive for its high feature dimensionality and was the one that took the most advantage of the ensemble learning method. The only exception was CatBoost, which recorded not only the highest mark on all metrics but also a dominant performance over all the classifiers, thus proving to be the most generalizable and robust.

MC2, was the smallest and the least imbalanced dataset out of all the three datasets. CatBoost (Pipeline) proved to be the best performing model as it fared slightly better in F1-score and AUC compared to Gradient Boost. This leads to the inference that the best model robustness together with the proper tuning of the algorithms elevates the performance on data that is scarce and imbalanced significantly.

6. Conclusion

6.1 Key Findings

This study has shown that strong machine learning pipelines can greatly improve software fault prediction, especially on the field, historical, and imbalanced datasets like the NASA MDP repository. The study identified the approaches of feature selection and sampling protocols used to balance the dataset and secure train process that were critical in achieving reliable model performance.

Hyperparameter turning, usually with cross-validation, was the main factor of improvement in evaluation metrics, emphasizing its necessity in building generalizable models. It is remarkable that the ensemble methods such as CatBoost and Gradient Boost were the ones that outshone the simpler algorithms in all but a few of the datasets. Further, the Logistic Regression and the Support Vector Machines which are the traditional classifiers also showed respectable performance when they were deeply optimized.

The overall data confirms the capability of machine learning to ensure the quality of software without human intervention. The research demonstrates that the right preprocessing, tuning, and validation can make the use of predictive models for early defect detection in software modules feasible.

6.2 Challenges Faced

The research project had numerous hurdles to go through while being implemented. The most significant problem was the stark class imbalance existing in some of the datasets, which in turn caused the baseline performance of several classifiers to deteriorate. This imbalance situation lead to the application of resampling techniques that, in addition to being convenient, introduced another level of difficulty to the pipeline.

Another key obstacle was the comparison with other datasets such as MW1, where the sample size was relatively small, thus restricting the capability of complex models to generalize safely. In addition, the hyperparameter tuning process seems to be very important but it was very intensive from a computational point of view, especially for the models with larger search spaces like CatBoost and MLP. Finally, ensuring fair and consistent comparisons across classifiers required meticulous standardization of preprocessing steps and threshold calibration.

6.3 Future Work

The challenges that remain are to develop and widen this study along several routes. To start with, employing a more up-to-date and variety of datasets, like those sourced from modern software repositories, might give a more informed vision for the use of these methods in contemporary development environments [6]. Moreover, the joining of deep learning

solutions can be a way to achieve better performance, especially in situations with unstructured data or broad feature representations.

Additionally, creating a real-time prediction tool based on this research outcome would also be useful for the software development teams by employing it for the early detection of the risks during the coding or review phase [7].

Furthermore, the inclusion of explainable AI within these models will not only help in making them easier to understand, but it will also build the trust of professionals which, in turn, will lead to better decision-making in the areas of software maintenance and quality assurance [8].

7. References

[1] R. S. Wahono, "A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks," Journal of Software Engineering, vol. 9, no. 6, pp. 475-487, 2015.

[2] S. Huda et al., "A Framework for Software Defect Prediction and Metric Selection," IEEE Access, vol. 5, pp. 1844-1858, 2017. Doi: 10.1109/ACCESS.2017.2785445

[3] Shepperd, Martin; Song, Qinbao; Sun, Zhongbin; Mair, Carolyn (2018). MDP data sets (D' and D") - zipped up. figshare. Dataset. https://doi.org/10.6084/m9.figshare.6071675.v1

[4] Alsaeedi, Abdullah & Khan, Mohammad. (2019). Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study. Journal of Software Engineering and Applications. 12. 85-100. DOI: 10.4236/jsea.2019.125007.

[5] J. Goyal and R. R. Sinha, "A New Improved Prediction of Software Defects Using Machine Learning-based Boosting Techniques with NASA Dataset," International Journal on Recent and Innovation Trends in Computing and Communication, vol. 11, no. 10s, pp. 492-504, 2023. DOI: https://doi.org/10.17762/ijritcc.v11i10s.7659

[6] Dos Santos, R. A., et al., “Just-In-Time Software Defect Prediction using a deep learningbased model,” New Trends in Computational Sciences, 2024, DOI: 10.3846/ntcs.2024.22274.

[7] Y. Shen, S. Hu, S. Cai and M. Chen, "Software Defect Prediction based on Bayesian Optimization Random Forest," 2022 9th International Conference on Dependable Systems and Their Applications (DSA), Wulumuqi, China, 2022, pp. 1012-1013, doi: 10.1109/DSA56465.2022.00149.

[8] M. Begum, M. H. Shuvo, I. Ashraf, A. A. Mamun, J. Uddin and M. A. Samad, "Software Defects Identification: Results Using Machine Learning and Explainable Artificial Intelligence
Techniques," in IEEE Access, vol. 11, pp. 132750-132765, 2023, doi: 10.1109/ACCESS.2023.3329051.

[9] Jalaj Pachouly, Swati Ahirrao, Ketan Kotecha, Ganeshsree Selvachandran, Ajith Abraham, “A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools”,Engineering Applications of Artificial Intelligence, Volume 111, 2022, 104773, ISSN 0952-1976, DOI: https://doi.org/10.1016/j.engappai.2022.104773.

[10] Suresh Jat, & Dr. Gurveen Vaseer. (2024). A Literature Review on Software Defect Prediction: Trends, Methods, and Frameworks. International Journal of Communication Networks and Information Security (IJCNIS), 16(4), 120–141. Retrieved from https://ijcnis.org/index.php/ijcnis/article/view/6890.

[11] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, Bedir Tekinerdogan, On the use of deep learning in software defect prediction, Journal of Systems and Software, Volume 195, 2023, 111537, ISSN 0164-1212, DOI: https://doi.org/10.1016/j.jss.2022.111537. [12] Yan Xiao, Xinyue Zuo, Xiaoyue Lu, Jin Song Dong, Xiaochun Cao, Ivan Beschastnikh,
Promises and perils of using Transformer-based models for SE research, Neural Networks, Volume 184, 2025, 107067, ISSN 0893-6080, DOI:https://doi.org/10.1016/j.neunet.2024.107067.

[13] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Paris, France, 2015, pp. 1-10, doi: 10.1109/DSAA.2015.7344858.

[14] Q. Song, Y. Guo and M. Shepperd, "A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction," in IEEE Transactions on Software Engineering, vol. 45, no. 12, pp. 1253-1269, 1 Dec. 2019, doi: 10.1109/TSE.2018.2836442.

[15] B. Sotto-Mayor and M. Kalech, "A Survey on Transfer Learning for Cross-Project Defect Prediction," in IEEE Access, vol. 12, pp. 93398-93425, 2024, doi: 10.1109/ACCESS.2024.3424311.

8. Appendices

Figure 2: Workflow of the proposed methodology from dataset loading to evaluation and result storage.