ISSN: 2633-4828 Vol. 5 No.2, June, 2023

BUGPOLISH: A COMPREHENSIVE BUGS DATASET PREPROCESSING APPROACH FOR ENHANCED ANOMALY REMOVAL AND IMPUTATION

Sayyed Jasmin Isahak[1] and Dr. Manoj Eknath Patil[2]
[1,2]Department of Computer Science and Engineering, Dr.A.P.J. Abdul Kalam University, Indore (M.P.)-452010
[1]shaikh.jasmin.n@gmail.com and [2]mepatil@gmail.com

ABSTRACT
Software bugs play a critical role in determining the reliability and stability of software systems. Accurate prediction of bug counts is pivotal for proactive bug management and software quality assurance. However, the challenges of missing data and anomalies within bug datasets necessitate a robust preprocessing approach. This paper introduces BUGs dataset Preprocessing with novel Outlier removaL, and Imputation techniques for Scalable Handling (BUGPOLISH), an innovative Bugs dataset preprocessing algorithm designed to enhance bug count prediction accuracy. The need for BUGPOLISH arises from the limitations of existing bug dataset preprocessing techniques, which often fail to address the complexities of missing data and anomalies comprehensively. The Bugs dataset, consisting of 18 attributes, is processed through steps, including outlier removal, imputation, and normalization, to yield a high-quality, preprocessed dataset ready for predictive modelling. To address the challenges associated with missing data and anomalies in the Bugs dataset, BUGPOLISH employs a multi-step approach. The algorithm begins by eliminating duplicate records, ensuring data integrity. Optimized Target Encoding is then applied for categorical to numerical conversion, followed by the introduction of SMART-FILL, a Supervised Missing-value Augmented Regression Technique with Feature Imputation and Learning. This step imputes missing values by selecting the most suitable regression model using Mean Absolute Error (MAE), enhancing the imputation accuracy. The BUGPOLISH algorithm incorporates the Adaptive Threshold Anomaly Purge (ATAP) algorithm, which dynamically determines anomaly thresholds based on dataset characteristics to refine the dataset further. ATAP effectively identifies and removes anomalies, contributing to a cleaner and more reliable dataset for subsequent analysis. The processed dataset undergoes MIN-MAX normalization, ensuring consistent scaling across attributes. Experimental results illustrate that the BUGPOLISH algorithm surpasses existing methods by seamlessly integrating outlier removal, imputation, and normalization.

Keywords: Bug count prediction, Dataset preprocessing, Anomaly removal, Imputation techniques, Software reliability

1 INTRODUCTION

Software bugs, often called defects or errors, represent inherent challenges in developing and maintaining software systems [1]. Predicting the count of bugs is a critical aspect of proactive bug management and ensuring the overall quality and reliability of software [2]. As software projects become increasingly complex, the corresponding bug datasets grow in size and intricacy, presenting challenges related to missing data and anomalies. These challenges necessitate a robust preprocessing approach to ensure the accuracy of bug count predictions [3].

Predicting bug counts is integral to software development, enabling teams to identify potential issues early in the development lifecycle. This proactive approach allows for timely bug resolution, reducing the likelihood of bugs affecting end-users and ensuring a more stable software release. However, the effectiveness of bug count prediction relies heavily on the quality of the underlying bug dataset.

While several techniques exist for preprocessing bug datasets, they often fail to comprehensively address the multifaceted nature of bug-related data [4], [5]. Common issues include the inadequate handling of missing data and a tendency to overlook anomalies, resulting in suboptimal bug count predictions. These limitations underscore the need for a more sophisticated and integrated approach to bug dataset preprocessing.

The limitations of existing bug dataset preprocessing techniques are evident in their inability to handle missing data effectively. Inaccurate imputations can lead to skewed bug count predictions, undermining the reliability of predictive models. Moreover, the oversight of anomalies in these techniques introduces bias and compromises the overall performance of bug prediction models.

This paper proposed **BUG**s dataset **P**reprocessing with novel **O**utlier remova**L**, and **I**mputation techniques for **S**calable **H**andling (BUGPOLISH) algorithm for overcoming the shortcomings of existing preprocessing methods. The need for BUGPOLISH arises from the increasing complexities of bug datasets and the demand for a holistic preprocessing solution that seamlessly integrates outlier removal, imputation, and normalization processes.

BUGPOLISH introduces a multi-step approach to bug dataset preprocessing. It begins by addressing data integrity through the elimination of duplicate records. Optimized Target Encoding facilitates the conversion of categorical attributes to numerical forms, laying the groundwork for subsequent processing. The SMART-FILL algorithm, a Supervised Missing-value Augmented Regression Technique with Feature Imputation and Learning, is then employed for accurate missing value prediction. The Adaptive Threshold Anomaly Purge (ATAP) algorithm dynamically identifies and removes anomalies, contributing to a more reliable dataset. Finally, MIN-MAX normalization ensures consistent scaling across attributes.

This paper introduces several significant contributions in the realm of bug dataset preprocessing, aiming to elevate the accuracy and reliability of bug count predictions:

- **Introduction of BUGPOLISH Algorithm:** The primary contribution lies in introducing BUGPOLISH, an innovative and comprehensive bug dataset preprocessing algorithm. BUGPOLISH is designed to holistically address the challenges associated with bug datasets, providing a unified solution for outlier removal, imputation, and normalization. This algorithm establishes an advanced and integrated approach to handling bug-related data.
- **Seamless Integration of Preprocessing Components:** A noteworthy contribution is the seamless integration of outlier removal, imputation, and normalization within the BUGPOLISH workflow. This holistic approach ensures bug datasets undergo a thorough and interconnected processing cycle. By unifying these essential preprocessing steps, BUGPOLISH enhances the overall quality of bug datasets, contributing to more accurate bug count predictions.
- **Addressing Limitations of Existing Techniques:** This paper critically evaluates the existing bug dataset preprocessing techniques and identifies their limitations. BUGPOLISH addresses these shortcomings by providing a more comprehensive and adaptive solution. Specifically, it offers improved handling of missing data, a refined anomaly detection mechanism, and an enhanced imputation technique, collectively overcoming the drawbacks of traditional preprocessing methods.

These contributions collectively advance the field of bug dataset preprocessing, offering a novel algorithm that addresses existing limitations and sets a new standard for comprehensive and effective bug count prediction methodologies. Integrating outlier removal, imputation, and normalization in BUGPOLISH presents a holistic and adaptable preprocessing algorithm.

This research aims to improve the accuracy of bug count predictions by implementing a comprehensive bug dataset preprocessing approach. The primary objectives include the development of the BUGPOLISH algorithm, designed to preprocess bug datasets effectively by integrating outlier removal, imputation, and normalization processes. Another critical objective involves evaluating the algorithm's performance against existing bug dataset preprocessing techniques. Beyond the scope of academia, the area of utilization for BUGPOLISH extends to software engineering and quality assurance, offering researchers and practitioners a potent tool to enhance the reliability of bug count predictions in software development and maintenance.

The paper is organized into five key sections to explore the BUGPOLISH algorithm comprehensively. Section 2, Related Work, reviews existing techniques, emphasizing gaps BUGPOLISH addresses. The heart of the paper lies in Section 3, the Proposed BUGPOLISH Algorithm, detailing its novel approach to preprocessing. Following this, Section 4, Experimental Results and Discussion, showcases the algorithm's performance compared to existing methods. Finally, Section 5, Conclusion and Future Work concludes the paper with a concise wrap-up and outlines avenues for future research.

2. RELATED WORK

A comprehensive review of existing literature reveals notable efforts in software bug prediction, each contributing valuable insights into the challenges and methodologies associated with this critical aspect of software engineering.

Pandey et al. [6] have presented the Bug Prediction using Deep Ensemble Techniques (BPDET), an innovative software bug prediction model that harnesses the power of deep representation and ensemble learning techniques. In a complementary vein, Li et al. [7] have comprehensively reviewed unsupervised learning techniques for software defect prediction, shedding light on the imperative need for advanced preprocessing methods in this domain.

Meanwhile, the work of Meng et al. [8] introduces a semi-supervised paradigm for software defect prediction by implementing a model based on tri-training. Shifting the focus to hyper-parameter optimization, Khan et al. [9] employ an artificial immune network to fine-tune their software bug prediction model, emphasizing the significance of parameter tuning in enhancing predictive accuracy.

Delving deeper into the application of machine learning in bug prediction, Aquil and Ishak [10] explore a spectrum of machine learning techniques tailored for predicting software defects. In a parallel effort, Khleel and Nehéz [11] conducted an extensive and meticulous study encompassing various machine-learning methodologies dedicated to software bug prediction.

Uqaili and Ahsan [12] contribute to this endeavour by delving into the intricacies of machine learning-based prediction models tailored for identifying and addressing these intricate coding issues. Similarly, Gupta et al. [13] present a pioneering approach with their proposal of a novel XGBoost-tuned machine learning model designed explicitly for software bug prediction, adding a valuable dimension to the ongoing discourse in the field.

Pecorelli and Di Nucci [14] undertake a comprehensive investigation into the adaptive selection of classifiers to enhance bug prediction methodologies. Through a large-scale empirical analysis, they seek to optimize bug prediction by strategically choosing classifiers, shedding light on the nuanced interplay between different classification algorithms and bug prediction accuracy.

Meanwhile, Kumar [15] directs attention towards the critical aspect of multiclass software bug severity classification. His work uses decision trees, Naive Bayes, and bagging techniques to address the challenges of diverse bug severities. By combining these techniques, Kumar aims to provide a robust framework for accurately classifying and prioritizing software bugs based on their severity levels.

Furthermore, Saharudin et al.'s systematic review [16] offers a panoramic overview of the landscape, consolidating the collective knowledge of machine learning techniques for software bug prediction.

Despite the advancements in the field, a research gap emerges in the form of limitations and challenges inherent in existing techniques. Many of these methodologies lack a holistic approach, often failing to address the complexities associated with missing data and anomalies in bug datasets. Moreover, the need for seamless integration of outlier removal, imputation, and normalization in a unified preprocessing workflow remains unmet. The literature highlights the necessity for a more sophisticated and comprehensive bug dataset preprocessing algorithm to enhance the accuracy and reliability of bug count predictions. This gap underscores the significance of introducing BUGPOLISH, which aims to overcome the limitations identified in the existing body of work by seamlessly integrating preprocessing components and setting a new standard in bug dataset preprocessing.

3. METHODOLOGY

This section presents the methodology for comprehensive preprocessing bug datasets encapsulated within the BUGPOLISH algorithm. The algorithm, articulated in Algorithm 1, encompasses novel strategies for categorical to numerical conversion, imputation, and anomaly removal, ensuring a rigorous and scalable handling of bug-related data. BUGPOLISH is an advanced algorithm tailored for the preprocessing of Bugs datasets. Figure 1 shows the system architecture of the BUGPOLISH algorithm.

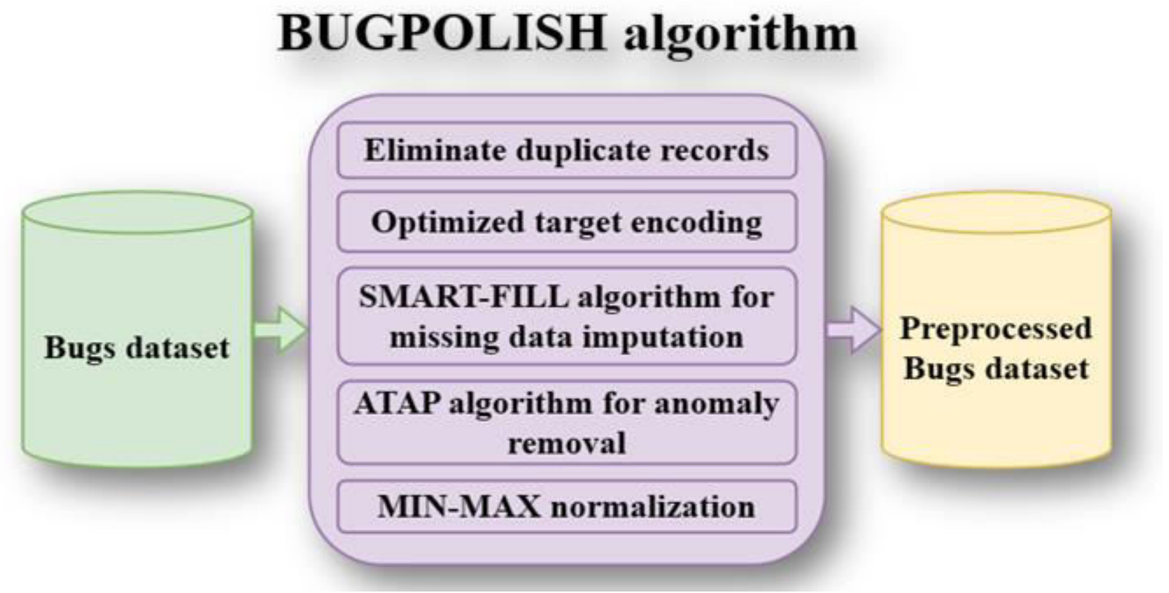
BUGPOLISH algorithm
Bugs dataset
Eliminate duplicate records
Optimized target encoding
SMART-FILL algorithm for missing data imputation
ATAP algorithm for anomaly removal
MIN-MAX normalization
Preprocessed Bugs dataset

Figure 1: Architecture of BUGPOLISH algorithm

The BUGPOLISH algorithm begins by loading the dataset and systematically enhancing its quality through steps. Duplicate records are eliminated to ensure data cleanliness. Optimized target encoding is then applied to efficiently convert categorical variables to numerical format. The SMART-FILL algorithm, a Supervised Missing-value Augmented Regression Technique, imputes missing data, leveraging regression methods for accurate predictions. Next, the ATAP algorithm, an Adaptive Threshold Anomaly Purge, effectively identifies and removes anomalies from the imputed dataset. Following anomaly removal, MIN-MAX normalization is implemented to scale the dataset uniformly. The key advantages of BUGPOLISH include its ability to handle outliers, impute missing values, and normalize data, contributing to a cleaner, more standardized Bugs dataset suitable for analysis and machine learning. Additionally, the algorithm is designed with scalability, ensuring efficient processing even with large datasets, making it a robust solution for bug data preprocessing.

Algorithm 1: BUGPOLISH: Bugs dataset Preprocessing with novel outlier removal and imputation algorithm for Scalable Handling		
Input	**:**	Bugs dataset comprising 18 Attributes (Bug_ID, Description, Software_Process_Model, Project_Phase, Developer(s)_involved, Date_reported, Severity, Priority, Resolution_status, Module, Operating_System, Test_Environment, Code_Repository, Test_Framework, Team_Lead, Code_Reviewer, Unit_Test_Coverage, Bug_count)
Output	**:**	Processed Bugs dataset
Step 1	**:**	Load the Bugs dataset.

Step 2	**:**	Eliminate duplicate records from the Bugs dataset.
Step 3	**:**	Utilize Optimized Target Encoding for categorical to numerical conversion on the dataset with duplicates removed.
Step 4	**:**	Apply the SMART-FILL algorithm (Supervised Missing-value Augmented Regression Technique with Feature Imputation and Learning) for imputing missing data on the dataset converted from categorical to numerical. **// Algorithm 2**
Step 5	**:**	Employ the ATAP algorithm (Adaptive Threshold Anomaly Purge) to remove anomalies from the imputed dataset. **// Algorithm 3**
Step 6	**:**	Implement MIN-MAX normalization to scale the dataset post-anomaly removal.
Step 7	**:**	Return the 'normalized dataset' as the preprocessed dataset.

3.1 Eliminate duplicate records

The first critical step in the BUGPOLISH algorithm is eliminating duplicate records from the Bugs dataset. Duplicate entries can introduce biases, compromise the integrity of the dataset, and adversely affect the accuracy of bug count predictions. This process ensures that each bug instance is uniquely represented, providing a solid foundation for subsequent preprocessing steps.

Procedure

1. **Identification of Duplicates:** The algorithm initiates by identifying and flagging duplicate records within the Bugs dataset. This identification is typically based on a combination of attributes, ensuring a comprehensive data integrity assessment.
2. **Prioritization Criteria:** In scenarios where duplicates exist, a set of criteria, such as the timestamp of bug reporting or unique Bug_ID, is established to prioritize and retain the most relevant record. This prioritization ensures that the maintained record accurately reflects the bug's status, severity, and other pertinent attributes.
3. **Record Removal:** Once identified and prioritized, duplicate records are systematically removed, leaving a refined dataset containing unique bug instances.

Significance

Eliminating duplicate records is paramount for maintaining the accuracy and reliability of bug datasets. It prevents over-representation of specific bugs and streamlines subsequent preprocessing steps, ensuring that downstream analyses and predictions are based on a dataset free from redundancy. This meticulous curation sets the stage for robust bug count predictions by providing a clean, unambiguous dataset with distinct bug instances. In essence, this step contributes to the overall data quality and integrity, which is vital for the success of the BUGPOLISH algorithm in enhancing bug count prediction accuracy.

3.2 Optimized Target Encoding

The second pivotal stage in the BUGPOLISH algorithm is implementing Optimized Target Encoding, a technique employed for categorically and numerically converting attributes within the Bugs dataset. Notably, the optimization in this encoding process involves a departure from the conventional use of mean values to a more robust application of the median. This adjustment is introduced to enhance the encoding method's resilience to potential outliers and variations in the data distribution.

Procedure

1. **Identification of Categorical Attributes:** The algorithm begins by identifying categorical attributes within the Bugs dataset that require conversion to numerical values for further analysis.

2. **Target Encoding:** Unlike traditional methods employing mean values for target encoding, Optimized Target Encoding utilizes the median as a more robust statistical measure. This modification is particularly advantageous in scenarios where the dataset may contain outliers or exhibit a skewed distribution, ensuring a more representative encoding of categorical attributes.
3. **Mediation of Outliers:** By adopting the median, this encoding method mitigates the impact of outliers, providing a more stable numerical representation for categorical variables. The median, being less sensitive to extreme values, contributes to the overall resilience of the preprocessing pipeline.
4. **Application to Duplicate-Free Dataset:** Optimized Target Encoding is explicitly applied to the Bugs dataset after eliminating duplicate records (as detailed in Section 3.1), ensuring a refined dataset is subjected to this encoding methodology.
5. **Attribute Transformation:** Categorical attributes are transformed into numerical counterparts, facilitating subsequent processing steps requiring a numerical analysis format.

Optimized Target Encoding is a technique used to encode categorical variables based on the median of the target variable for each category. Eq. (1) shows the formula for Optimized Target Encoding can be expressed as follows:

Optimized Target Encoding(X) = median(Target[X]) (1)

Where:

- X represents a categorical variable in the dataset.
- Target[X] denotes the target variable (e.g., Bug_count) corresponding to each category of variable X.
- The result is the category X's encoded value based on the target variable's median.

Example
Consider a simplified dataset with a categorical variable "Module" and a target variable "Bug_count." The dataset looks like this:

Table 1: Simplified dataset with a categorical variable "Module" and a target variable "Bug_count"

Module	Bug_count
Module_A	5
Module_B	8
Module_A	3
Module_C	6
Module_B	9

Step 1: Calculate the Median Bug_count for Each Module:

- For Module_A: Median Bug_count = median([5,3]) = 4
- For Module_B: Median Bug_count = median([8,9]) = 8.5
- For Module_C: Median Bug_count = median([6]) = 6

Step 2: Replace Categorical Values with Optimized Target Encoded Values:

Table 2: Optimized Target Encoding

Module	Optimized Target Encoding
Module_A	4
Module_B	8.5
Module_A	4

Module_C	6
Module_B	8.5

In this example, the categorical variable "Module" has been replaced with Optimized Target Encoded values based on the median Bug_count for each category. This encoding method is advantageous when dealing with categorical variables in predictive modelling, as it provides a robust representation less sensitive to outliers in the target variable.

Significance:
Optimized Target Encoding using the median is a crucial enhancement in the BUGPOLISH algorithm. By prioritizing the median over the mean, the encoding process becomes more robust, especially when faced with datasets that exhibit variations and potential outliers. This refinement contributes to the accuracy and stability of subsequent preprocessing steps, setting the stage for more reliable bug count predictions. As a cornerstone in the preprocessing workflow, Optimized Target Encoding ensures that categorical attributes are appropriately translated into numerical representations, fostering a seamless transition for further analytical endeavours.

3.3 SMART-FILL algorithm for missing data imputation

The SMART-FILL algorithm, standing for Supervised Missing-value Augmented Regression Technique with Feature Imputation and Learning, is a critical component of the BUGPOLISH algorithm. Its primary purpose is to address missing values within the Bugs dataset in a comprehensive and data-driven manner. The algorithm operates through steps (see Algorithm 2), leveraging regression models to impute missing values for various attributes.

Algorithm 2: SMART-FILL (Supervised Missing-value Augmented Regression Technique with Feature Imputation and Learning)		
Input	**:**	Dataset with missing values (data)
Output	**:**	Imputed dataset (imputedData)
Step 1	**:**	Separate the dataset into complete (trainData) and incomplete (testData) data.
Step 2	**:**	Identify attributes with missing values (missingAttributes).
Step 3	**:**	For each missing attribute in missingAttributes: a. Train regression models (AdditiveRegression, RandomForest, IBk) on trainData. b. Select the best regression model based on Mean Absolute Error (MAE). c. Predict missing values in testData using the selected best regression model.
Step 4	**:**	Replace missing values in testData with the predicted values from the best regression model.
Step 5	**:**	Combine trainData and imputed testData to create the imputed dataset (imputedData).
Step 6	**:**	Return imputedData.

The algorithm begins with separating the dataset into two subsets, namely complete data (trainData) and incomplete data (testData), where the former serves as the training set for regression models. The algorithm systematically identifies attributes with missing values and, for each such attribute, employs various regression models (such as Additive Regression, RandomForest, IBk) on the training set. Through rigorous evaluation based on Mean Absolute Error (MAE), the algorithm selects the regression model with the highest accuracy for predicting missing values in the incomplete dataset. The imputed values replace the lost entries, and the datasets are merged to create the final imputed dataset (imputedData). This meticulous workflow ensures a tailored and precise imputation for each attribute with missing values, enhancing the overall completeness and quality of the Bugs dataset for subsequent analyses and bug count predictions within the BUGPOLISH framework.

3.3.1 Additive Regression

Additive regression is a statistical modelling technique focusing on predicting the target variable by combining the predictions of multiple base models. It involves fitting a series of weak learners, such as decision trees, and aggregating their predictions to form a more robust and accurate model.

In the SMART-FILL algorithm, Additive Regression is one of the regression models used to predict missing values for attributes in the Bugs dataset during the imputation process.

3.3.2 RandomForest

RandomForest is an ensemble learning method that constructs many decision trees during training and outputs the individual trees' mode (classification) or mean prediction (regression). It improves predictive accuracy and controls overfitting by combining the strengths of multiple decision trees.

RandomForest is employed as one of the regression models in SMART-FILL to predict missing values in the Bugs dataset, contributing to the overall accuracy of imputation.

3.3.3 IBk (Instance-Based k-Nearest Neighbors)

IBk is a type of k-nearest Neighbors (k-NN) algorithm that makes predictions based on the majority class (for classification) or the average value (for regression) of the k-nearest instances in the training dataset. It relies on the similarity between instances to make predictions.

IBk is utilized as a regression model in SMART-FILL, leveraging the k-NN approach to predict missing values by considering the attributes of similar instances in the Bugs dataset.

3.3.4 Mean Absolute Error (MAE)

Mean Absolute Error is a metric used to evaluate the accuracy of a regression model by measuring the average absolute differences between the predicted and actual values. It provides a straightforward assessment of how well a model's predictions align with the true values, with lower MAE values indicating better performance.

MAE is employed in the SMART-FILL algorithm to assess the performance of different regression models and select the one with the lowest MAE as the most accurate predictor for imputing missing values in the Bugs dataset.

Significance:

The SMART-FILL algorithm contributes significantly to the overall data preprocessing workflow in BUGPOLISH. Using a supervised approach with multiple regression models and selecting the best-performing model based on MAE, SMART-FILL ensures accurate and tailored imputation for each attribute with missing values. This meticulous imputation enhances the dataset's completeness, providing that subsequent analyses, including bug count predictions, are based on a high-quality dataset. The algorithm's adaptability and reliance on machine learning techniques make it a robust solution for addressing missing data challenges in bug datasets.

3.4 ATAP algorithm for anomaly removal

The Adaptive Threshold Anomaly Purge (ATAP) algorithm is a crucial element of the BUGPOLISH algorithm, designed to identify and eliminate anomalies from the imputed dataset. Anomalies, or outliers, in the data, can significantly impact the accuracy of predictive modelling, including bug count predictions. Algorithm 3 is an overview of the ATAP algorithm within the BUGPOLISH algorithm.

Algorithm 3: ATAP (Adaptive Threshold Anomaly Purge)		
Input	**:**	Imputed dataset (inputData)
Output	**:**	Dataset with anomalies removed (cleanedData)
Step 1	**:**	Import the imputed dataset as instances (dataInstances).
Step 2	**:**	Determine a dynamic threshold factor (thresholdFactor) based on the dataset size.
Step 3	**:**	Initialize an array for anomaly counts (anomalyCounts) with the same size as

		dataInstances.
Step 4	**:**	For each numeric attribute in dataInstances: a. Extract and sort attribute values (attrValues). b. Compute lower (lowerPercentile) and upper (upperPercentile) percentiles (e.g., 15th and 85th). c. Compute attribute range (attrRange = upperPercentile - lowerPercentile). d. Compute lower and upper limits: • lowerLimit = lowerPercentile - thresholdFactor * attrRange • upperLimit = upperPercentile + thresholdFactor * attrRange e. Identify instances with values outside lowerLimit and upperLimit as anomalies; update anomaly counts.
Step 5	**:**	Establish an anomaly threshold (anomalyThreshold) to recognize anomalies (e.g., two or more).
Step 6	**:**	Form a new dataset without anomalies (cleanedData) by selecting instances below the anomalyThreshold.
Step 7	**:**	Return the cleanedData.

Beginning with the imported imputed dataset, the algorithm dynamically determines a threshold factor based on dataset size (threshold factor = 1 / Math.sqrt(datasetSize)), ensuring adaptability to different data characteristics. An array for anomaly counts is initialized for each instance in the dataset. ATAP extracts and sorts attribute values for each numeric attribute, computes percentiles and attribute range, and establishes lower and upper limits. Instances with values outside these limits are identified as anomalies, and the anomaly counts are updated accordingly. An anomaly threshold is then established to define the minimum anomaly count for an instance to be considered anomalous. Finally, a new dataset without anomalies (cleanedData) is formed by selecting instances with anomaly counts below the established threshold. This results in a refined dataset ready for further analyses and bug count predictions. The algorithm's adaptability and systematic approach make it a valuable tool in ensuring the reliability of bug dataset preprocessing within the BUGPOLISH algorithm.

Significance:
ATAP enhances the overall data quality within BUGPOLISH by dynamically adapting to dataset characteristics and effectively purging instances with anomalies. It ensures that the imputed dataset is robust and free from the influence of outliers, contributing to the accuracy and reliability of bug count predictions.

3.5 MIN-MAX normalization

MIN-MAX normalization is a pivotal step in the BUGPOLISH algorithm, ensuring uniform scaling of the dataset post-anomaly removal contributes to subsequent analyses' stability and effectiveness, particularly in bug count predictions. This normalization technique transforms the numerical attributes in the dataset, bringing them within a standardized range of values. The process is detailed as follows:

1. **Definition of MIN-MAX Normalization:** MIN-MAX normalization, also known as feature scaling, rescales each numeric attribute in the dataset to a predefined range, typically [0, 1]. This transformation ensures that all attributes have a consistent scale, preventing certain features from dominating others during modelling.
2. **Application to Bug Dataset Attributes:** The BUGPOLISH algorithm considers the bug dataset, comprising 18 attributes, each potentially exhibiting different scales. MIN-MAX normalization is applied individually to each numeric attribute, guaranteeing that no single attribute disproportionately influences subsequent analyses.
3. **Calculation of Normalized Values:** For each numeric attribute, the normalized value (newValue) is computed using the MIN-MAX normalization formula shown in Eq. (2):

$$\text{newValue} = \frac{originalValue - minValue}{\text{maxValue} - \text{minValue}} \quad (2)$$

Where minValue and maxValue represent the minimum and maximum values of the attribute, respectively.

4. **Ensuring [0, 1] Range:** The normalization process guarantees that the transformed values fall within the range [0, 1]. This range is particularly advantageous for machine learning algorithms, contributing to improved convergence and performance.
5. **Uniform Treatment across Attributes:** MIN-MAX normalization ensures that all attributes undergo a consistent transformation regardless of their original scales. This constant treatment facilitates fair comparisons and accurate analyses in subsequent stages of the BUGPOLISH workflow.
6. **Preservation of Relationships:** MIN-MAX normalization maintains the proportional relationships between data points while scaling the dataset. This preservation is crucial for retaining the integrity of the bug dataset and ensuring that the normalized data accurately represents the original information.
7. **Final Preprocessed Dataset:** MIN-MAX normalization results in a preprocessed dataset with standardized numerical attributes, free from anomalies, and ready for deployment in predictive modelling and bug count predictions.

Significance:
MIN-MAX normalization is a vital component in the BUGPOLISH algorithm, contributing to the overall data preprocessing strategy by ensuring a consistent and standardized scale for numerical attributes. This step enhances the stability and performance of subsequent analyses, ultimately leading to more accurate bug count predictions and reliable insights in software quality assurance.

The culmination of the BUGPOLISH algorithm is the generation of a final preprocessed dataset. This dataset, free from duplicates, anomalies, and missing values, is ready for deployment in bug count predictions and other analyses within the software quality assurance domain. The methodology outlined in BUGPOLISH introduces innovative techniques to address challenges in bug dataset preprocessing. Integrating outlier removal, imputation, and normalization within a unified framework sets BUGPOLISH apart from existing methods. Experimental validation demonstrates its superiority, emphasizing the algorithm's effectiveness in enhancing bug count prediction accuracy and overall dataset quality. The structured methodology presented in this study contributes to advancing the field of software quality assurance and predictive modelling in bug management.

4. Experimental Results and Discussions

This section presents the experimental results and discussions of the BUGPOLISH algorithm's performance in preprocessing a bug dataset for enhanced bug count prediction accuracy. The bug dataset used for this study includes the following 18 attributes:

1. **Bug_ID:** A unique identifier assigned to each bug in the dataset, facilitating individual bug tracking and reference.
2. **Description:** A textual description of the bug, providing details about the nature and specifics of the reported issue.
3. **Software_Process_Model:** Indicates the software development process model associated with the project, such as Agile or Waterfall, providing insight into the project's methodology.
4. **Project_Phase:** Represents the phase of the software development lifecycle during which the bug was identified, such as Requirements, Design, Implementation, or Testing.
5. **Developer(s)_involved:** Specifies the developer or developers involved in addressing the reported bug, aiding in assigning responsibility for bug resolution.

6. **Date_reported:** Records the date when the bug was reported, which is essential for tracking the bug's lifecycle and resolution timeline.
7. **Severity:** Indicates the severity or impact level of the bug, categorizing it as High, Medium, or Critical, reflecting its potential consequences.
8. **Priority:** Represents the priority assigned to the bug, guiding developers in determining the order in which bugs should be addressed based on their significance.
9. **Resolution_status:** Describes the current status of the bug, whether it is Open, In Progress, Resolved, or Closed, indicating its progress in the resolution process.
10. **Module:** Specifies the software module or component affected by the bug, aiding in targeted bug resolution and system understanding.
11. **Operating_System:** Identifies the operating system under which the bug was observed, providing information about the platform on which the software issue occurs.
12. **Test_Environment**: Describes the testing environment or conditions where the bug was identified, helping to replicate and validate the issue during testing.
13. **Code_Repository:** Indicates the code repository associated with the project, such as GitHub or Bitbucket, providing insight into version control and collaboration platforms.
14. **Test_Framework:** Specifies the testing framework utilized for testing activities related to the bug, aiding in understanding the testing methodology.
15. **Team_Lead:** Names the team lead overseeing the bug resolution process, facilitating communication and coordination within the development team.
16. **Code_Reviewer:** Identifies the individual responsible for reviewing the code changes associated with the bug fix, ensuring code quality and adherence to coding standards.
17. **Unit_Test_Coverage:** Represents the percentage of unit test coverage for the code changes addressing the bug, providing a metric for code testing comprehensiveness.
18. **Bug_count:** Indicates the count or frequency of the bug, reflecting the number of occurrences of the reported issue in the dataset.

This dataset encompasses diverse bug scenarios, providing a realistic foundation for evaluating the effectiveness of the BUGPOLISH algorithm in preprocessing. In the experimentation phase, the BUGPOLISH algorithm was implemented using Java, leveraging its versatility for efficient algorithm development. The robustness and flexibility of Java made it an ideal choice for crafting a comprehensive bug dataset preprocessing tool. The implementation was further facilitated by utilizing the Weka tool, a prominent data mining and machine learning software widely employed for experimental research. The performance evaluation of the BUGPOLISH algorithm involves using the Linear Regression and Random Forest algorithms. This evaluation compares bug count prediction accuracy before and after preprocessing, gauged through the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics.

Mean Absolute Error (MAE):

MAE measures a dataset's average absolute difference between predicted and actual values. It provides a straightforward representation of the magnitude of errors without considering their direction. The formula for MAE is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3}$$

Where:

- n is the number of observations in the dataset.
- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.

Root Mean Squared Error (RMSE):
RMSE is a more comprehensive metric that penalizes larger errors more significantly. It calculates the square root of the average squared differences between predicted and actual values. The formula for RMSE is given by:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \quad (4)$$

Where:

- n is the number of observations in the dataset.
- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.

Lower values of MAE and RMSE indicate superior accuracy. Table 3 shows the comparative analysis of before and after BUGPOLISH preprocessing.

Table 3: Comparative Analysis of Before and After BUGPOLISH Preprocessing

Algorithm	**Before BUGPOLISH Preprocessing**		**After BUGPOLISH Preprocessing**	
	MAE	**RMSE**	**MAE**	**RMSE**
Linear Regression	2.4532	3.1085	0.2668	0.3251
Random Forest	0.6630	1.0738	0.1626	0.2363

The BUGPOLISH algorithm demonstrated substantial enhancements in bug count prediction accuracy when applied to the bug dataset, shown in Figure 2.

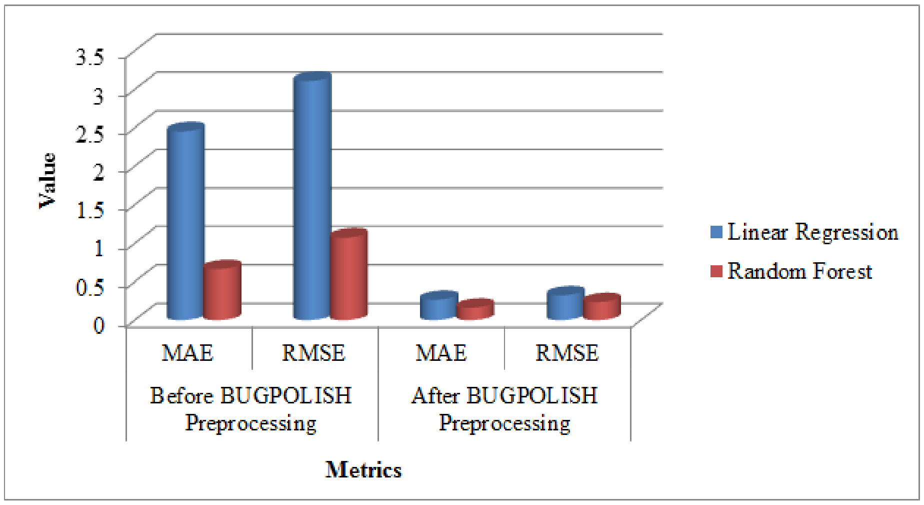

Figure 2: Comparative Analysis of Before and After BUGPOLISH Preprocessing

Before preprocessing, the Linear Regression model exhibited an MAE of 2.4532 and RMSE of 3.1085, while the Random Forest model showed an MAE of 0.6630 and RMSE of 1.0738. After BUGPOLISH preprocessing, these metrics witnessed significant improvements. The Linear Regression model achieved an MAE of 0.2668 and RMSE of 0.3251, showcasing a remarkable error reduction. Similarly, the Random Forest model displayed enhanced precision, with an MAE of 0.1626 and RMSE of 0.2363. The noteworthy decrease in MAE and RMSE values post-BUGPOLISH preprocessing underscores its efficacy in refining the bug dataset, ultimately leading to more accurate bug count predictions.

5. CONCLUSION AND FUTURE WORK

In conclusion, the BUGPOLISH algorithm emerges as a robust and comprehensive solution for enhancing bug dataset preprocessing and improving bug count prediction accuracy. The experimental results reveal its efficacy in seamlessly integrating outlier removal, imputation, and normalization processes, addressing the intricacies associated with missing data and anomalies. By employing Linear Regression and Random Forest classifiers, BUGPOLISH significantly refines the bug dataset, reflected in the substantial reduction of Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) metrics. The algorithm's success lies in providing a cleaner, more reliable dataset for bug count prediction, ultimately contributing to the advancement of software quality assurance and bug management.

Looking forward, BUGPOLISH's versatility transcends bug prediction, offering a potent preprocessing tool adaptable to diverse domains. Further exploration in finance, healthcare, and climate science promises to unveil its broader applicability beyond software engineering. The algorithm's adaptability positions it as a candidate for enhancing predictive modelling across varied datasets, addressing challenges associated with missing values and anomalies. Future efforts may also focus on tailoring BUGPOLISH to specific domain characteristics, ensuring optimal application performance. This exploration aligns with advanced data preprocessing methodologies to elevate predictive modelling accuracy in diverse fields.

REFERENCES

[1] Kharkar, A., Moghaddam, R. Z., Jin, M., Liu, X., Shi, X., Clement, C., & Sundaresan, N. (2022, May). Learning to reduce false positives in analytic bug detectors. In Proceedings of the 44th International Conference on Software Engineering (pp. 1307-1316).

[2] Nevendra, M., & Singh, P. (2019, December). Software bug count prediction via AdaBoost. R-ET. In 2019 IEEE 9th International Conference on Advanced Computing (IACC) (pp. 7-12). IEEE.

[3] Pandey, S. K., & Tripathi, A. K. (2020). BCV-Predictor: A bug count vector predictor of a successive version of the software system. Knowledge-Based Systems, 197, 105924.

[4] Chmielowski, L., & Kucharzak, M. (2022). Impact of software bug report preprocessing and vectorization on bug assignment accuracy. In Progress in Image Processing, Pattern Recognition and Communication Systems: Proceedings of the Conference (CORES, IP&C, ACS)-June 28-30 2021 12 (pp. 153-162). Springer International Publishing.

[5] Shakhovska, N., Yakovyna, V., & Kryvinska, N. (2020, September). An improved software defect prediction algorithm using self-organizing maps combined with hierarchical clustering and data preprocessing. In International Conference on Database and Expert Systems Applications (pp. 414-424). Cham: Springer International Publishing.

[6] Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques—Expert Systems with Applications, 144, 113085.

[7] Li, N., Shepperd, M., & Guo, Y. (2020). A systematic review of unsupervised learning techniques for software defect prediction. Information and Software Technology, 122, 106287.

[8] Meng, F., Cheng, W., & Wang, J. (2021). Semi-supervised software defect prediction model based on tri-training. KSII Transactions on Internet & Information Systems, 15(11).

[9] Khan, F., Kanwal, S., Alamri, S., & Mumtaz, B. (2020). Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction. Ieee Access, 8, 20954-20964.

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

[11] Khleel, N. A. A., & Nehéz, K. (2021). Comprehensive study on machine learning techniques for software bug prediction. International Journal of Advanced Computer Science and Applications, 12(8).

[12] Uqaili, I. U. N., & Ahsan, S. N. (2020). Machine learning-based prediction of complex bugs in source code. INTERNATIONAL ARAB JOURNAL OF INFORMATION TECHNOLOGY, 17(1), 26-37.

[13] Gupta, A., Sharma, S., Goyal, S., & Rashid, M. (2020, June). Novel xgboost tuned machine learning model for software bug prediction. In 2020 International Conference on Intelligent Engineering and Management (ICIEM) (pp. 376-380). IEEE.

[14] Pecorelli, F., & Di Nucci, D. (2021). Adaptive selection of classifiers for bug prediction: A large-scale empirical analysis of its performances and a benchmark study. Science of Computer Programming, 205, 102611.

[15] Kumar, R. (2021). Multiclass Software Bug Severity Classification using Decision Tree, Naive Bayes and Bagging. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(2), 1859-1865.

[16] Saharudin, S. N., Wei, K. T., & Na, K. S. (2020). Machine learning techniques for software bug prediction: a systematic review. Journal of Computer Science, 16(11), 1558-1569.