Novel Approach for Handling Class and Attribute Noise in Lines-of-Code

By

Usama Bin Israr

(Registration No.: MS-SE-20-327324)

<u>Supervisor:</u>

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD

August, 2024

Annex A

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by NS **Usama Bin Israr** Registration No. **00000327324**, of College of E&ME has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the thesis.

Signature : [illegible]

Name of Supervisor: Dr Wasi Haider Butt

Date: 12-08-2024

Signature of HOD: [illegible]
(Dr Usman Qamar)
Date: 12-08-2024

Signature of Dean: Rashid
(Brig Dr Nasir Rashid)
Date: 12 AUG 2024

Novel Approach for Handling Class and Attribute Noise in Lines-of-Code

By

Usama Bin Israr

(Registration No.: 00000327324)

A thesis submitted to the National University of Sciences and Technology, Islamabad,
in partial fulfillment of the requirements for the degree of

Master of Science in Software Engineering

Supervisor:

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD

August, 2024

DEDICATION

I dedicate this work wholeheartedly to the Almighty Allah; without Whose guidance it would not have been possible. I thank Him for His guidance, strength, protection, health and so much more that I may or may not know. I am thankful to all my family members, teachers and friends, especially my grandmother, mother, and sister, who gave me the strength and hope to complete this achievement even when I was on the verge of giving up.

ACKNOWLEDGEMENTS

In the name of Allah Almighty, the most Gracious and the most Merciful. All praises to Him for the blessings and strengths in completing this thesis. Special appreciation for my supervisor, Dr. Wasi Haider Butt, for his cooperation, supervision, and constant support.

I would also like to thank all my teachers, my family members, especially my grandmother, mother, and sister. They provided a continuous source of inspiration, moral, spiritual, emotional, and financial support.

Last but not the least, my deepest gratitude goes to my friends, Ammar Ahmed, Ammar Khan, Waqar Ahmed & Muhammad Zubair, who helped me and provided wisdom and strength during this period of my life. To those who indirectly contributed to this thesis and degree, your help and guidance means a lot to me. Thank you very much for everything.

ABSTRACT

Attribute and class noise is a pervasive issue in software quality interpretation that has caught ample consideration due to its substantial impression on classification algorithms. This study delves into composite interplays amidst attribute and class noise as regards to software quality datasets and demonstrates advancements in model performance that originated from enquiring effective means for reducing specific forms of noise. It uses a broad-spectrum of field research, applying random forest as key classification approach and uniting various data sampling methods, to review the significance of attribute and class noise. This study delves into the intricacies of attribute noise along with the domain of class noise, examining its effects on model performance. Comparable changes in accuracy, precision, recall and F-score are observed as attribute noise levels increase. The experimental data points out quantifiable merits of skillful noise reduction when assessing software quality. In particular, the study demonstrates that significant gains in recall, accuracy, precision, and F-score are closely correlated with noise reduction. Eminently, important advances are observed by converting from unclean data to class-noise cleaned data. The results demonstrate the importance of noise handling approaches and the effect of noise on the accuracy and dependability of machine learning models. The proposed algorithm achieves significant gains 94.59%, 97.74%, 94.79% and 96.24% in accuracy, precision, recall and F-score respectively, that exhibit how necessary noise reduction strategies are and how extensive of an effect they have on the performance of an ML model.

Keywords: Class Noise, Attribute Noise, Noise Reduction Strategies, Random Forest, Continuous Integration.

TABLE OF CONTENTS

LIST OF TABLES

Page No.

LIST OF FIGURES

Page No.

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

BP	Build Prediction
CI	Continuous Integration
ML	Machine Learning
RF	Random Forest
LOC	Lines-of-Code
NLP	Natural Language Processing
ATNODE	Attribute Noise Detection
PANDA	Pairwise Attribute Noise Detection Algorithm

CHAPTER 1: INTRODUCTION

System code quality and reliability are particularly important for continuously developing and maintaining the software systems eventually. Noise, in any of its forms, is a serious problem as it not only negatively affects the quality but also the validity of the entire process of development as well as the maintenance phase. There are several reasons for the existence of noise in the code such as incorrect code changes, lack of documentation or the difference in coding standards of team members. It makes it difficult for the developers to understand the functionality required and therefore its behaviour. It also increases the likelihood that issues will be introduced in subsequent code versions and complicates the process of finding and fixing faults. Hence, software development proposals might display reduced output, prolonged development times, and additional costs, thus, dealing with the subject of attribute and class noise is critical for the development of secure and tenable software systems.

1.1. Background and Context

Software engineering tasks are rapidly being automated using machine learning (ML) models [1], [2], [3], [4]. To determine which test cases ought to be added to test suites following each build in continuous integration (CI), to optimize software regression testing, is one of the instances machine learning is used for. Regression testing is routinely carried out (after each commit), producing huge amounts of data that contain test execution outcomes. When such vast amounts of data are accessible for study, this presents an opportunity to use ML.

Several different strategies, in the literature, have addressed the subject matter of predicting defects and selection of test cases during CI. Static code analysis [5], [6], NLP [7], [8] and static code metrics [9], [10] are some of the examples. Code is either identified as defective (in need of testing) or functional or predicted if test cases will be unsuccessful using ML models trained on datasets with historical flaws.

For predicting the results of test case execution, while developing an algorithm, one complication that arises is the volume of noise present in the data. This problem is usually significant in the field of testing as many automated test case executions might unintentionally produce noise. Research on a comprehensive vocabulary of noise subtypes is still ongoing [11]. The research [12], [13], [14], [15], however, mainly discusses two types of noise namely attribute and class noise. Contrary to attribute noise, which results from choosing attributes that are irrelevant for describing the training instances and their connections to the target class or from using redundant or empty attribute values [12], [16], class noise is caused by either contradictory entries or incorrect labelling of training entries [12].

The class noise may be seen in the CI build prediction domain, for instance, when the identical code line occurs many times in the data with distinct class labels (build outcomes) for the same build. For predictors, these repeated occurrences of the same line cause class noise, which lowers the accuracy of their classification. In similar terms, Van Hulse et al [17] asserts that the reason attribute noise arises is when at least one of its characteristics differ from the average dissemination of other attributes. In other words, attribute noise emerges when code lines are created using distinct coding patterns. Code lines drafted in the less common pattern will have characteristics that differ from lines of a similar nature written in the majority styles in terms of frequency. These variances might cause code lines created using less common coding styles to stand out in the data at hand, which can have a detrimental effect on learning performance.

Numerous research studies suggested a variety of methods for dealing with attribute and class noise [15], [17], [18], [19]. These fall into three groups: toleration, eradication or filtration, and alteration or polishing. In the case of tolerance, errors residing within the data are addressed by preserving the noise and constructing machine learning algorithms that can withstand specific levels of noise. Eradication approaches aim to find noisy data and eradicate it from the dataset. Entries that are believed to be fabricated (such as those with incorrect labels or duplicates) are rejected and eradicated from the training set. In the last group, the noisy entries are fixed by having their values changed to more suitable ones rather than being removed. With each of these strategies, there are a variety of benefits and drawbacks. In the tolerance category, cleaning the data is not necessary but building a learner using unclean data will result in a learner that may perform poorly. To save cleaner instances of the data, we sacrifice information loss by filtering noisy instances. By correcting noisy instances, we run the danger of showing unwanted qualities while maintaining the most information possible in the data.

1.2. Problem Statement

The existence of attribute and class noise in code lines substantially impedes software system development and maintenance. Noise, in code fragment, unfavorably affects the balance of final software product and the entire characteristic of the generated code. The challenges of existing noise control methods depend on human code checks, scalability problems, and reduced fidelity when handling complex noise patterns. These limitations, besides limiting the proficiency of software development, raise the prospect of proposing software defects into the system. A novel and useful outcome is surely required to identify and deal with attribute and class noise in arrays of code, diminishing its adverse impacts and improving software development methods.

Noise, attribute, or class, in lines of code poses significant challenges for software engineers. The negative impacts noise may have on code quality, software reliability, and development efficiency make it essential to design an automated and efficient technique for noise identification and removal. The proposed method should be able to forecast CI build results with high reliability, even in the presence of attribute and class noise. Therefore, the main objective of this research is to present a novel and clever method that can successfully handle noise at several levels of code granularity and adapt to different noise patterns. This will be accomplished by utilizing advanced machine learning, natural language processing, and data mining techniques to provide a solid and complete solution.

This project will address the following research question for managing attribute and class noise:

RQ: How can we create a machine learning algorithm with nominal computational and execution time costs that can handle attribute and class noise in actual datasets?

The software engineering society would benefit substantially from dealing with the problem of noise in lines of code. By managing attribute and class noise, the suggested technique will lead to persistent software systems. Furthermore, it will upgrade development methods by diminishing the chance of noise-induced flaws and reducing debugging efforts. Long-term maintenance requirements will be decreased, and productivity of programmers will rise because of automating noise detection and removal. In conclusion, this study aims to provide useful information to the area of software engineering and push the sector into better and authentic software development applications.

1.3. Proposed Approach

In machine learning literature, the obstacle of accomplishing good learning outcomes in noisy settings has been extensively discussed. For better learning potential of ML classifiers, various proposals have been drafted [12], [13], [21]. Although it has been determined that attribute and class noise still have a negative consequence on learning, hence it should be tackled ahead of practice. We present our proposed procedures regarding attribute and class noise in this part.

We suggest handling class noise by relabeling code lines that are repeated and have distinct class values. This proposed approach was presented by Al-Sabbagh et al [8]. The repeated lines in the code may be the result of a variety of circumstances, including copying code [22] and merging transactions [14]. The first case occurs when code that has already passed testing and integration is reused by simply "copy-pasting" it. The second situation occurs when developers from one or more teams utilize code that is comparable to that which has been committed and merged from separate branches while working on dedicated branches for feature development [14]. A minor fraction of the total chunk of code modifications is frequently taken up by defective lines. As a result, it is more likely that a line, selected at random, from the chunk of code that was overall deemed failed was not the reason for the failure. As a result, our choice is to reclassify lines as "passed" if they had previously been identified as components of segments that did not fail before.

Attribute noise occurs because of selecting attributes that are unnecessary for identifying training instances. The code snippet being analyzed, for instance, contains fragments that are drafted using different coding syntax or if several conditions, function declaration or statements differ in syntax from other similar lines of code. For addressing the problem of attribute noise, we propose a novel approach called *Attribute Noise Detection algorithm (ATNODE)*. The proposed approach is derived from Pairwise Attribute Noise Detection Algorithm (PANDA), which was introduced by Van Hulse et al [17], [23]. The PANDA technique was modified considering the computational expense suffered by the dimensions of the dataset. A method is proposed for detecting and reducing attribute noise in datasets. This method iterates through each column of the dataset and applies different operations to each column. When the dataset is partitioned, the values of each column are used for sorting. Each section is statistically analyzed to determine the presence of noise, and the algorithm repeats this process for each column in the dataset. In the process, the noise values for the various sections are quickly calculated. Given the circumstances, this method enhances the quality of the data and makes additional analysis easier by offering a methodical framework for evaluating and tracking noise in recorded data. More information on sound processing techniques is provided in Chapter 3.

1.4. Significance of Research

Software is rendered less efficient, and mistakes are more likely to occur due to shortcomings in current noise management techniques, such as vulnerabilities, scalability challenges with human programming, and limited accuracy in controlling complicated noise forms. The reliability of the final product and the overall standard of the product are significantly harmed by noise in code fragments. Sometimes it is challenging to build the technology and locate the real fault in the code during debugging because of the noise that exists between the lines of code.

This study suggests a novel strategy for efficiently detecting, controlling, and removing noise to get rid of incorrect code and enhance program performance. It does this by utilizing machine learning methods, natural language processing, and recent data mining techniques. By efficiently locating and eliminating noise between lines of code, this technique should provide clean, maintainable code. It can assist programmers in locating live issues in code, saving a substantial amount of development time, increasing overall programming productivity, and requiring less diagnostic work. Reducing maintenance needs, raising customer satisfaction and product reliability, and directly enhancing the stability and dependability of software systems are just some advantages of improving code quality.

The objective of the study is to evaluate the effectiveness of the platform approach as well as its applicability and scalability for multiple software systems using a real code base. The dataset of Al-Sabbagh et al [24], [25], which allows increasing the importance of this study, provides practical validation, and the most probable and best proposed approach will have a positive impact on real software systems. Insights from this study can help practitioners and researchers in the Industrial Electronic Noise Program at LOC.

1.5. Aims and Objectives

The objectives for this scientific survey are listed below:

- Utilizing a dataset of Build Prediction in CI Using Textual Analysis of Source Code for experimentation, develop a method for addressing attribute and class noise in lines of code.
- Enhance code feature by virtually finding and eliminating noise in lines of code.
- Minimize the effect of noise (on the source code integrity to improve software maintenance activities), reduce debugging efforts and increase development efficiency through noise handling.
- Validate the effectiveness of the proposed approaches in handling attribute and class noise in real-world codebases and compare their performances with existing methods to identify its strengths and contributions.

1.6. Structure of the Research

Four key chapters are discussed in this study. A detailed review of the literature is discussed in chapter 2, enlightening on the advanced attribute and class noise in software quality datasets. The data processing and machine learning techniques used, as well as the experimental research approach, are covered in detail in chapter 3. The results of the experimental studies as well as the impact of noise and how to reduce it on model performance is presented in chapter 4. When it comes to managing attribute and class noise in software quality datasets, chapter 5 concludes with a comprehensive review and recommendations for further study.

CHAPTER 2: LITERATURE REVIEW

To enhance expected accuracy of machine learning models, various research dissertations have been dedicated to problem resolution led by attribute and class noise. The usage of machine learning models themselves to recognize and thoroughly monitor depictions of attribute and class noise has acquired substantial focus in this domain. To identify noisy data points and whether overcome them or handle them accurately, these attempts require the start of algorithms and methods that harness the benefit of machine learning. In later sub-headings, we will provide information regarding dealing with class noise, attribute noise and both attribute and class noise correspondingly.

2.1. Class Noise Handling Research

This study [26] exhibited an innovative technique for determining and rectifying instances with invalid labels in large or scattered datasets and utilized a unique Partitioning Filter method to resolve this matter and was aimed to handle obstacles led by the data range. Finding instances that could have had inaccurate labels due to labelling errors was the primary goal. Step one was to segregate the dataset into more convenient sets for induction algorithm processing. Practical standards were produced per subset and implemented to assess the entire dataset. The error count variable for each instance is used to track the number of times all classes are classified as noise. Samples with incorrect labels are likely to show a higher number of errors. Two different strategies were used to classify cases as noise, majority, and non-objection. While the no-objection approach focused on situations where noise requirements were not met regularly, the policy of majority approach limited the frequency of what was classified as noise across all categories. In each iteration of the process, some correctly labeled samples and detected noise cases were removed. This process is repeated until the specified stopping condition is met, improving classification accuracy, and reducing the negative impact of mislabeled examples on training. The effectiveness and ability of the proposed Partitioning Filter method to identify mislabeled cases and thereby improve the confidence of clusters is demonstrated using real datasets and empirical analysis. The research provided a substantial contribution to the domains of data preparation and machine learning and underlined the need of efficiently managing big or distributed datasets around noise detection.

Table 2.1: Class noise handling approaches

Ref.	**Title**	**Year**	**Technique**
[26]	Eliminating Class Noise in Large Datasets	2003	Partitioning Filter (PF)
[27]	An Empirical Comparison of Three Boosting Algorithms on Real Data Sets with Artificial Class Noise	2003	Adaboost, Logitboost, Brownboost
[11]	Rule-Based Noise Detection for Software Measurement Data	2004	Rule-Based Classification Model (RBCM)
[13]	Identifying And Handling Mislabeled Instances	2004	Removal Technique, Relabeling Technique
[28], [29]	Cost-guided class noise handling for effective cost-sensitive learning Class Noise Handling for Effective Cost-Sensitive Learning by Cost-Guided Iterative Classification Filtering	2004, 2006	Cost-Guided Iterative Classification Filter (CICF)
[19]	Identifying Noise in An Attribute of Interest	2005	PANDA
[30]	Bridging Local and Global Data Cleansing: Identifying Class Noise in Large, Distributed Data Datasets	2006	Classification Filter (CF) Partition Filter (PF)
[31]	Class noise detection using frequent itemsets	2006	Frequent Itemsets

[32]	Classification in the presence of class noise using a probabilistic Kernel Fisher method	2007	Probabilistic Fisher Probabilistic Kernel Fisher
[33]	Class Noise Mitigation Through Instance Weighting	2007	Pair-Wise Expectation Maximization (PWEM)
[34]	Robustness of learning techniques in handling class noise in imbalanced datasets	2007	C4.5, Naïve Bayes, k-NN
[35]	Software quality modeling: The impact of class noise on the random forest classifier	2008	Random Forest (RF), C4.5, Naïve Bayes
[36]	Genre-based decomposition of email class noise	2009	Dynamic Markov Compression Algorithm (DMC), Relaxed Online SVM (ROSVM), BogoFilter
[37]	Advances in Class Noise Detection	2010	Naïve Bayes, Random Forest, SVM
[15]	Identifying Mislabeled Training Data with The Aid of Unlabeled Data	2011	CFAUD, MFAUD
[38]	Class noise detection based on software metrics and ROC curves	2011	ROC, Naïve Bayes
[39]	A Study on Class Noise Detection and Elimination	2012	Ensemble Classifier (SVM, k-NN, CART, C4.5, Random Forest, Naïve Bayes, MLP), Edited Nearest Neighbor (ENN), Repeated ENN (RENN)
[40]	Bagging schemes on the presence of class noise in classification	2012	Bagging Credal Decision Tree
[41]	Noise in Bug Report Data and the Impact on Defect Prediction Results	2013	FS-ID3, Fuzzy-Logic Decision Tree

[42]	An empirical study of the classification performance of learners on imbalanced and noisy software quality data	2014	C4.5, Random Forest (RF), k-NN, RIPPER, Logistic Regression (LR), Naïve Bayes (NB), Multilayer Perceptron (MLP), Support Vector Machine (SVM)
[43]	Relating ensemble diversity and performance: A study in class noise detection	2015	Support Vector Machine (SVM), CN2, Random Tree (RT), J48, Naïve Bayes, JRip, Multilayer Perceptron (MLP), Random Forest (RF), SMO, k-NN
[44]	Modelling Class Noise with Symmetric and Asymmetric Distributions	2015	Boundary Conditional Class Noise (BCN), Logistic Regression (LR), Probit Regression (PR), Linear SVM
[45]	Class noise removal and correction for image classification using ensemble margin	2015	Classification and Regression Trees (CART), k-NN, AdaboostM1
[21]	Evaluating The Classifier Behavior with Noisy Data Considering Performance and Robustness: The Equalized Loss of Accuracy Measure	2016	C4.5, Support Vector Machine (SVM)
[46]	On the Influence of Class Noise in Medical Data Classification: Treatment Using Noise Filtering Methods	2016	Support Vector Machine (SVM), C4.5, Nearest Neighbor (NN)
[47]	Complete Random Forest Based Class Noise Filtering Learning for Improving the Generalizability of Classifiers	2019	Complete Random Forest (CRF), CRF based Noise Filtering Learning framework (CRF-NFL), Support Vector Machine (SVM), k-NN, Decision Tree (DT), Logistic Regression (LR), XGBoost

[48]	The Effect of Class Noise on Continuous Test Case Selection: A Controlled Experiment on Industrial Data	2020	Bi-Gram Bag of Words Model, Random Forest (RF), Mathew Correlation Coefficient (MCC)
[8]	Improving Data Quality for Regression Test Selection by Reducing Annotation Noise	2020	Method Using Bag of Words for Test Case Selection (MeBoTS)
[49]	Improving class noise detection and classification performance: A new two-filter CNDC model	2020	Class Noise Detection and Classification (CNDC) model
[50]	A Novel Class Noise Detection Method for High-Dimensional Data in Industrial Informatics	2021	Sequential Ensemble Noise Filter (SENF)

The aim of the research [27] was to correlate how three popular boosting algorithms, Adaboost, Logitboost, and Brownboost, performed when implemented with monitored algorithmic learning. Boosting methods integrate various flawed ideas to develop potent ensemble classifiers. The performance of the algorithms was assessed analytically applying five actual datasets, especially to test error rates. The boosting algorithms were trained on the rest of two-thirds of the data in every trial applying binary stumps as core learners, with one-third of the data being arbitrarily selected as a test set. Outcomes were highlighted with a 95% confidence rate, and the last practice and trial error values were recorded. The researchers also randomly altered 20% of the class labels in the datasets to provide fake class noise to assess the noise tolerance of the algorithms.

The main outcomes of the research were listed as follows:

- In every dataset, Logitboost outperformed Adaboost in terms of test error rates when fake class noise was removed.
- It was suggested that because Logitboost places a maximum limit on weight changes, it may be less susceptible to overfitting, which accounts for its higher performance.

- As a result of Adaboost's strong sensitivity to class noise, generalization performance suffered when noise was present.
- In line with prior findings, Brownboost demonstrated resilience in the presence of class noise.
- The study done with prior knowledge of noise levels revealed the difficulty in assessing noise levels in actual situations.

Given the circumstances, the study provided valuable insights into the relative performance of various boosting methods as well as their susceptibility to class noise, which have implications for machine learning applications in the real world.

In this study [11], data quality was mentioned as an important factor in the classification of tasks, especially in the category of defective and working parts of the software. A novel noise detection method, based upon Boolean logic rules, was presented for detecting and removing noise from the training data. Different attributes containing noise were introduced at various noise levels to evaluate this proposed method. The C4.5-based classification filter and the proposed approach were compared, and the results showed that as the noise increases the latter approach operates more efficiently than the former approach. The findings show that the proposed model is a great alternative to common noise filtering techniques due to its simplicity and Boolean logic.

The main goal of this work [13] was to increase the classification accuracy in the presence of noisy data and untrained training samples. A new method is proposed for noise identification using a graph of geometric parameters as a pre-filter. The classification results can be improved by removing or reclassify such fuzzy patterns. The deletion procedure performs better than the reclassifying test using the restricted 1-NN method by the UCI Machine Learning Repository, especially when the classes are clearly separated. By detecting and resolving outliers, this method increases class separation and reduces noise in performance estimation. The results showed the effectiveness of this strategy in developing accurate class distributions.

Recent advances in machine learning and data mining have led to many methods using cost-sensitive classification. These methods are usually used under the expectation that no significant noise is present in the dataset, which is often not true in real-world datasets. To tackle this issue, the authors of [28], [29] proposed a Cost-guided Iterative Classification Filter (CICF) for investigating the effect class noise has on cost-sensitive learning. CICF is adaptable enough to include the rejection of misclassification costs by focusing on detailed classes and is more vigilant as compared to other approaches. The noise detection method of CICF consists of a classification filter and cost estimation method. It has been proven to be more effective on datasets with large noise-to-cost ratio and it can help researchers reduce the misclassification cost by using cost-sensitive categories in environments containing noise.

A novel noise detection method, using the user defined Attribute of Interest (AOI) method, in [19] which was developed considering the poor quality of data in data mining. It provided a flexible way to improve data quality by allowing specific noise conditions to be classified into functions. The use of this technique is illustrated by an example of class noise detection, where AOI classes are used to classify noise structure at high resolution. The proposed method demonstrated its effectiveness by outperforming hierarchical and clustering filters in the dataset under question. It has proven to be a useful tool for improving data quality in real-world datasets with various integrity issues, especially noise.

Separating the dataset into a testing and training subgroups is an approach mostly used by set-oriented approaches when cleaning data for efficient extrapolation, and then using a classifier trained on the training subgroup to detect noise in the testing one. However, as the amount of data increases, the drawbacks of this approach make memory pretraining intensive, time-consuming, or impossible, limiting its usefulness. To address these issues, this study [30] presents a denoising technique for large or distributed datasets that combines local and global analysis. Instead of using one large data set, several smaller data sets are used, each treated as a local subset of the inductive process. High-quality rules are then generated from these local subsets and used to evaluate the entire dataset. An error count variable is created that tracks instances of noise in each dataset. An instance is more likely to be faulty when the error count is high, and accuracy is low. The majority and safe thresholds are analyzed to detect and remove noise. To reduce the noise in a partitioned dataset, partition filtering techniques are proposed to divide the dataset into smaller partitions and organize each instance according to local rules. The main features of this method are small datasets, focusing on the local condition and using good samples in each round of cleaning and each local source sample as its own noise detection matrix. Experimental evaluations and reliability tests show that this noise detection method is dependable and effectively improves classification accuracy.

Each event in the dataset is assigned a noise variable that uses a frequency function to assign the probability of the event to the noise class. The importance of voice recognition in intelligent analysis tasks is demonstrated by this paper [31], which also proposes a new method for detecting noise types in datasets to be used for classification tasks. Itemsets are frequently described as a collection of items that satisfy a user-specified minimal support condition and share attribute values. The feature structure and its interactions were explained by these itemsets. Based on the quantity of each sort of item it included, a class was assigned to each continuous group of objects. There were occurrences marked as noise in itemsets that were dominated by the opposite class. Several case studies using practical software measurement datasets with both induced and inherent noise were used to evaluate the effectiveness of the proposed technique. The results demonstrated how successfully the new algorithm identified situations with class noise. The study placed a strong emphasis on the need of noise identification since conclusions drawn from noisy data may not be dependable. By putting the algorithm to the test on several datasets with various noise-generating methods, a thorough evaluation was produced. The study focused on how applicable the approach was since it offered accurate class noise detection and was not affected by changes in parameters. The proposed approach was also compared with two popular class noise detection methods, Ensemble Filter and Classification Filter, and the unique method outperformed both filters in class noise identification.

The focus of this study [32] was to address the issue of class noise in machine learning, which may significantly reduce classifier performance if it exists in a dataset. The paper introduced two new classifiers that were built on a probabilistic model that Lawrence and Schölkopf first put out in 2001 [51]. These suggested methods build on the earlier work of Lawrence and Schölkopf in two essential aspects, with the goal of tolerating and managing class noise effectively. First, they altered the distribution assumptions used in the prior work, and second, they included a special integration of the probabilistic noise model into the Kernel Fisher discriminant. Using simulated noisy datasets and actual comparative genomic hybridization (CGH) data, these approaches' efficacy was assessed. PF and PKF, two noise-tolerant classifiers that were based on the probabilistic noise model of Lawrence and Schölkopf, were introduced in this paper's conclusion after a comprehensive investigation of the class noise problem. These classifiers produced linear and non-linear classifiers in the original feature space by optimizing the projection direction in noisy data. They made an important decision by not relying on explicit distribution assumptions in the input space. The study also included a component-based probabilistic algorithm (CPA) to the probabilistic model to handle non-Gaussian datasets. The potential of the suggested noise models was shown by the experimental findings. In general, these models outperform conventional classifiers when used properly. When dealing with non-Gaussian datasets and datasets with a disproportionately large number of features in comparison to the sample size, PKF showed notable benefits. Notably, PKF accurately identified mislabeled samples when used on the BRCA1 dataset. It was important to note that kernel-based approaches, like the ones suggested in this study, have computational difficulties, particularly when dealing with huge datasets. For bigger datasets with several repetitions, kernelization techniques can get quite complicated and may not be practical.

In this study [33], a brand-new method for dealing with class noise in machine learning datasets was presented. This method assigns a class membership probability vector to each training instance and weighs the learning process based on the current label confidence. The goal is to provide an example of a correctly classified sample with low confidence in the existing label and high confidence in the true label, and an example of a clean sample with high confidence in the existing label. This method considers two special cases of instance weighting: instance deletion and instance correction. The method described in this article uses clustering to determine the probability distribution of naming classes for each case. Experiments showed that this scheme improves classification accuracy compared to the original scheme. The article also explains that instance weighting is a better way to reduce class noise than noise elimination. This study proposed Probability Weighted Expectation Maximization (PWEM), a probabilistic strategy to reduce the impact of class noise. PWEM produces a probability distribution of class labels for each condition and uses clustering to learn the true class label of the training set. According to the practical data, PWEM and instance weighting considerably improved classifier accuracy, even coming close to the theoretical optimal performance under different noise levels.

The problem of unbalanced datasets, where one class considerably outnumbers the other(s), was the main topic of this research [34]. Although there are various methods for dealing with unbalanced datasets, most of them presuppose that the incoming data is noise-free. In real-world situations, data frequently contain noise that can have an impact on the accuracy of the data, the models based on it, and the judgements drawn from it. The study assessed how well-suited current methods are at managing unbalanced datasets with added class noise. Seven unbalanced datasets were used in the assessment, and the findings revealed that the MetaCost technique seemed to be more dependable as the amount of class noise increased. The paper's conclusion emphasized the need for more study in dealing with multi-class unbalanced datasets and suggested the creation of more durable approaches to deal with class noise in such datasets.

In this study [35], the influence of rising amounts of artificial class noise on the categorization of software quality was investigated. The performance of three distinct classifiers was inspected, namely Naïve Bayes, C4.5 and Random Forest (RF100), which were tested on seven software engineering measurement datasets subjected to class noise in the study. The RF100 classifier was chosen, given its superior performance, especially in contrast to leasing classifiers like Naïve Bayes and C4.5, which were frequently employed in the field of software quality classification. The two main experimental variables of this study were the volume of class noise and the proportion of few occurrences that were infused with noise. No matter the amount of noise, the proportion of minority cases with noise, the performance metric (AUC or KS), or the degree of noise in the tests, the empirical data consistently showed that the random forest classifier (RF100) beat the other classifiers. The study found that RF100 was an exceptionally reliable classifier for identifying software quality, particularly when there was class noise. Although the building and combining of 100 decision trees increased the execution time, software quality classification tasks often included relatively modest datasets in terms of both the number of instances and characteristics, thus this was generally not a significant problem. The study also highlighted how class noise significantly affects how well software quality classification algorithms work. Particularly in cases where instances should have been classified as not-fault-prone when they were falsely classified as fault-prone, it brought attention to the need of data quality and the necessity to examine and revise class labels.

The problem of class noise (label noise) in email spam filtration and its effects on classification issues were the main subjects of this study [36]. It challenged the uniform distribution assumption assumed by many data cleaning techniques, which is often incorrect in practical settings. According to the study, class noise can display significant content-specific bias based on email spam filtering data; in other words, certain email genres or types were more likely to be incorrectly classified than others. The study also examined classifier-confidence based noise detection techniques and found that these approaches often identify examples that human evaluators would also likely mislabel. The study suggested using genre modelling to quantify the content bias in the class noise distribution depending on email genres. It was suggested that other text categorization problems might display similar patterns of content-based bias. The results demonstrated that classifier confidence-based data cleaning methods could effectively identify cases that were mislabeled, providing a less expensive solution than human review. To increase spam filtering accuracy, the research also recommended using genre membership signals in the classifier learning process. For the benchmark collections, the performance findings of the study outperformed previously released figures.

Finding noisy examples in data preparation to be reviewed by domain experts at the data comprehension stage was the aim of this research [37]. Unlike traditional noise filtering methods that aim to increase classifier accuracy, the purpose of this study was to obtain high precision in class noise detection. The F-Score was used to represent the trade-off between accuracy and recall. The F0.5 measure provided a trade-off between recall and noise detection accuracy, was used to evaluate several applications of noise filtering algorithms based on classification. According to the research results, the new highly consistent random forest filter was found to be the most effective strategy with a 70%

agreement rate between decision tree classifiers using random forest methods. In the case of F0.5 score and accuracy, these well-connected random forests outperformed other simple classification filters.

This work [15] proposes a unique approach to efficiently detect and correct mislabeled training sessions, which can improve the accuracy of supervised learning systems. This unique approach improves the performance by utilizing unlabeled instances to help discover cases that are incorrectly classified, unlike previous systems that rely only on labeled data. With some modifications, this strategy can easily be used with common noise detection techniques such as majority filtering (MF) and consensus filtering (CF). New versions of MF and CF are released as MFAUD and CFAUD, respectively, using unlabeled data to improve accuracy. Experimental studies confirm the effectiveness of this strategy and show that MF and CF are better than before, especially in noisy environments.

The purpose of this paper [38] was to present a technique especially for detecting class noise. The noise features and classes included in this scenario can reduce the performance of the machine learning classifier, so it was important to find the noise entries to improve the overall performance. Examining software metrics, called Receiver Operating Characteristic (ROC), allowed them to determine the threshold values used in the proposed voice detection method. Five public NASA datasets were used in the case studies, and a Naïve Bayes model was created to predict software failures both after and before the use of noise detection techniques. Empirical results showed how effective this noise detection method was in detecting class noise. As seen by lower false positive, false negative, and error rates, the Naïve Bayes based software fault prediction models performed substantially better when the class labels of modules having noise were amended. The technique for detecting noise conditional to threshold values of software metrics, which successfully enhanced the functionality of software fault prediction models, was the main contribution of this study.

This work [39] focused on noise in classification datasets, specifically in the class labels of the target attribute. Various factors, such as errors in data collection, transmission, and storage, might introduce noise into the data. Machine learning models may perform worse when there is noisy data present, with longer training durations and worse predicted accuracy. To solve classification challenges, the study presented and empirically tested several noise detection and removal methodologies. In the trials, five UCI datasets that were initially clean and consistent were subjected to regulated noise levels. The experiment's findings showed that the methods for managing noise suggested were potentially useful. While conservative, the consensus voting method performed worse at locating noisy data instances. However, for some pre-processed datasets, the original data categories show high classification performance and accurate detection rate even with noisy data. The study also shows how clustering techniques affect data quality and how reclassification algorithms can be used to remove randomness and noise.

The authors of this study [40] used Bagging Cradle decision trees to classify noisy datasets. These uncertainty and probability measures are used to analyze these decision trees. To generate reliable decision trees, this study performed continuous attribute analysis and improved the flexibility and ease of use of the original method [52] by incorporating missing data. For noisy datasets, this experimental study shows that a Bagging credal decision tree, using measures of error probability and uncertainty, performs well in handling noisy data. Extensive research and conflicting results show that decision tree recognition significantly reduces the number of different sources of error when processing datasets with high classification noise. This performance advantage is achieved through a simpler design with fewer features than previous approaches.

The question of evaluating noise defect prediction in bug report data and how it can affect the accuracy of the defect classification process was answered, in this study [41], and the impact of different sources of noise on the accuracy of error prediction was also investigated. Some of the common causes of noise in bug report data are wrong bug reports, ignored or overlooked problems, incorrect measurements, missing links between modules and other software errors. These components increased the overall "dirtiness" of the real-world information that was used to predict defects. The effect of class noise on defect prediction was tested by using an industrial software system. Surprisingly, the defect prediction findings remained accurate, with over 95% in most cases, even at noise levels of up to 20%. The models could occasionally even handle noise levels of 30% or more.

The study [42] looked closely at the impact of class differences and class noise on the classification models that are used to identify program modules that are prone to errors in software quality datasets. 12 datasets generated from actual software quality data were used to assess different levels of class noise and imbalance. Eleven classification algorithms and seven data sampling strategies were assessed using noisy and imbalanced datasets. Initially, as the degree of class imbalance increased, most classification algorithms and sampling techniques outperformed one another. This improvement was attributed to the decrease in data noise with increasing degrees of imbalance. Severe imbalance might still have a detrimental effect on classifier performance, it was discovered. Second, various classification algorithms' responses to the use of sampling strategies differed considerably. When sampling was used, several algorithms showed notable gains, especially when there was more imbalance. As an illustration, the Radial Basis Function (RBF) classifier showed notable improvement with sampling, particularly in circumstances with high imbalance. The Naïve Bayes (NB) classifier, in contrast, seems to be mostly unaffected by sampling. Thirdly, the study pinpointed sampling methods that consistently outperformed others in a range of imbalance and noise levels. Notably, across all datasets, the Random Under-Sampling (RUS) method consistently produced excellent results. Additionally, the Weighted Ensemble (WE) method performed well, especially as the degree of imbalance rose. However, other methods, such as One-Sided Sampling (OSS) and Cluster Based Over Sampling (CBOS), produce inconsistent results. This study also focuses on applying classification algorithms using sampling techniques. For each dataset, Naïve Bayes (NB) and support vector machine (SVM) were the most efficient classifiers, consistently outperforming other techniques. Unlike many other methods, increasing imbalance has a significant impact on the Radial Basis Function (RBF) classification. The results of this study demonstrated a complex relationship between noise and the types of differences between categories of software quality attributes. This provides useful information about the robustness of sampling strategies and classification algorithms in these complex environments.

The purpose of this work [43] was to determine whether diversity metrics helps in selecting the efficient ensembles for noise detection and to study the relationship between ensemble variation and class noise detection in an ensemble-based environment. In an experimental study involving the detection of multiple sets of heterogeneous noise detection ensembles, two voting systems were considered: a majority voting system and a consensus voting system. The results were surprising, especially in the group with more noise, which showed better memory but lower accuracy in identifying noisy conditions. While increasing ensemble diversity was found to have no beneficial effect on noise detection ability—in fact, in some cases, it had the opposite effect—the majority voting system, which placed a higher priority on high recall than precision, was found to be successful for expert-guided noise detection. Despite efforts to link diversity values with noise detection performance, there was not enough information available in this scenario to choose effective noise detection ensembles. However, consensus-based noise detection ensembles, which demonstrated high noise detection precision but potentially worse recall, benefited more from ensemble variation. These ensembles were suitable for unsupervised noise detection where precision was crucial, and the probability of false noise identification was to be minimized. Less ensemble diversity improved recall and F-scores, but more ensemble diversity improved class noise detection precision, according to the experiments. Diversity metrics can be used as a guide to select effective ensembles for noise detection, as demonstrated by the significant relationships that the ambiguity diversity measure and Kohavi-Wolpert measures shown with F-Score and noise recall, while the diversity measure for noise detection accuracy was regarded as “bad" during this process.

The investigation of class noise in classification problems was the goal of this work [44]. The researchers created the concept of boundary-conditional class noise (BCN), which depends upon the hypothesis that samples near the class border are more likely than those farther away to have erroneous annotations. To mimic how class noise is created, they proposed symmetric and asymmetric noise models using unnormalized Gaussian and Laplace distributions. Furthermore, the newly proposed models taking class noise into account was reinterpreted and compared to Logistic regression and Probit regression. Empirical studies used to test these models showed that asymmetric noise models consistently outperformed benchmark linear models. In terms of overall performance, the asymmetric Laplacian noise model outperformed other proposed models.

In the research study [45], the problem of incorrect classification of training data was solved, which presented a challenge for classifiers including ensembles. A method for identifying, removing, or correcting mislabeled educational articles using four different ensemble margins was proposed. This approach was based on thresholding misclassified data instances and the concept of hierarchical noise classification. To demonstrate the effectiveness of this strategy, it was applied to image classification. A comparative study between the majority filter, that is, the class noise filter, and the proposed method was conducted. The results showed that the proposed margin-based method outperforms most filters in detecting and correcting mislabeled entries. The study concludes with a demonstration of a margin-based clustering method for detecting and processing erroneous training data, including removal and optimization of noise labels. Two commonly used ensemble margin definitions were evaluated, and an unsupervised alternative was proposed to address the mislabeling problem. It seemed that this method could improve the accuracy of image classification.

In real datasets, noise often degrades classification performance, necessitating the use of noise-tolerant classifiers. However, discussing classifier efficiency and its resilience separately can lead to different perspectives. To overcome this limitation, this study [21] developed a unique metric that takes power and performance into account: the Equalized Loss of Accuracy (ELA). The limitations of existing metrics made it difficult to evaluate different methods on the same data or interpret concepts correctly. Because of these challenges, ELA was developed to provide a comprehensive understanding of the behavior of a classifier in a noisy environment. This study demonstrated the advantages of ELA over other reliability measures and how it could accurately predict classifier behavior in the presence of noise. Real-world examples and empirical analysis were used to accomplish this. About comparing multiple classifiers on the same dataset, the proposed measure enhanced the process and yielded useful information for selecting classifiers in scenarios when noise is present.

This article [46] focused on the impact of class noise on medical data classification algorithms. The study showed that categorization algorithms are essential for automated data analysis in medical decision support. However, the accuracy of the training dataset used to create the classification models determines how well these systems perform. When instances are wrongly labelled, class label noise arises, which can negatively impact training and ultimately classification performance. The study investigated the use of noise filtering techniques to deal with class label noise in the context of classifying medical data. On real-world medical datasets that were known to be impacted by class noise, many tests were carried out. The study emphasized that class noise in medical data might be caused by mistakes made by human experts or inaccurate data gathering techniques. Even if steps like double-checking data or obtaining expert consensus might decrease labelling mistakes, they are frequently time and money consuming. The effectiveness of three classification algorithms; C4.5, SVM, and NN; both with and without the use of noise filtering was examined in the study. The results showed that even low amounts of class noise might dramatically impair classification performance across twelve medical datasets with various degrees of noise. SVM performed best when noise filtering was not used. However, it was discovered that noise filtering was essential, particularly when dealing with loud noise. The most successful noise filters in the research's tests were EF, IPF, and NCNE. It was emphasized that using noise filters would not always result in better performance than doing no preprocessing at all. It was urged to carefully evaluate various classifiers and noise filters as a result, as their efficacy can be affected by the unique features of the medical data being analyzed.

This research [47] developed a noise filtering learning framework (CRF-NFL) that efficiently identifies class noise in complicated data situations and offered a complete random forest-based technique for class noise identification. Compared to other approaches that depend on distance measurements, overall distribution, or classifiers, the voting mechanism made the proposed system robust for datasets with feature noise and suitable for various machine learning techniques. The study developed the framework to incorporate numerous widely used classifiers, including k-means tree, GBDT, and XGBoost, and produced a distributed version for large-scale data. The performance of classical classifiers and a relative density-based (RD) method was contrasted with that of CRF-NFL-based classifiers on UCI datasets and high-dimensional ImageNet datasets. The outcomes proved the effectiveness of CRF-NFL-based classifiers, which showed appreciable improvements in test accuracy across different datasets. On UCI datasets, these classifiers increased test accuracy by an average of 0.7% to 11.54% and, at their highest test accuracy, by up to 30.60%. Notably, their advantage was much more obvious on high-dimensional datasets with class noise, with improvements in average test accuracy ranging from 1.08% to 30.60% and in maximum test accuracy reaching 30.60%.

The study [48] dug into the continuous integration and testing space, concentrating on utilizing rich data on code flaws to train predictive learners for efficient test suite selection. The noise included in training data, which has an impact on classification performance, was a major problem. The impact that class noise has on test case selection was examined, during this study, using a controlled experiment on an industrial dataset. A substantial correlation between classroom noise levels and learner performance was discovered using stringent criteria like Precision, Recall, F-score, and MCC. Increased class noise ratios led to test suite absence and a rise in false alarms when the noise ratio was above 30%, highlighting the trade-offs and complexity in noise treatment. By providing a formula to estimate class noise levels, the study enhanced practical insights and helped testers make decisions about noise management tactics. Even though class noise management procedures had some success, difficulties persisted, necessitating more research into effective handling techniques and comparison with attribute noise effects. The study's consequences included assessing the effectiveness of machine learning in foretelling test case failures, looking at the impact of code formatting on noise collection, and researching how well various learning models can categorize and assign noise.

Research [8] on the integration of machine learning and big data models in software engineering focuses on the selection of test cases for continuous integration. New noise reduction techniques solve the problem of inconsistent learning methods. A case study of the proposed method using sampled data for regression testing showed a significant improvement of 70% recall, 59% F-score, and 37% precision. By renaming rather than systematically deleting, this new approach improves test case suggestion, class noise, and pattern recognition.

This work [49] attempts to address the problem of noise in the classification process, which can have serious consequences such as reduced accuracy and increased model complexity. A novel Class Noise Detection and Classification (CNDC) model was proposed that used two filters, noise detection and noise classification, to effectively handle class noise. To solve the embedded noise identification problem using the distance filter and the removal and reclassification (REM-REL) method to improve the overall performance was used for noise detection and noise classification, respectively. Six original datasets were used to evaluate the efficiency of CNDC where F0.5 index, precision and recall rate were calculated for each dataset. The results showed that simulated CNDC model identified noisy data effectively and provided flexible filtering methods. The findings also showed that the REM-REL method significantly improved content quality compared to traditional deletion and renaming methods. It also showed that the CNDC model outperformed the existing approaches, and its stability was tested using ROC curves.

The big data misclassification problem and how noise and unwanted artifacts affect the noise detection performance of an algorithm was discussed in this research study [50]. Earlier approaches used a two-dimensional approach to separate important signals from noise, yet they do not provide sufficient performance for noise detection as both, feature selection and noise detection, are performed separately. A novel approach called Sequential Ensemble Noise Filter (SENF) was proposed to resolve this issue. It combines noise detection and reduction into one package. Tested on a variety of high and low noise datasets, the suggested SENF algorithm performs noticeably better than the advanced noise detection techniques. This advantage is most obvious when it comes to complexity and high class noise ratios. Statistical analyses were performed to confirm the results of this study and compare the performance of SENF with conventional methods. However, it showed that in cases where the class noise ratios and feature counts are the same, the majority filtering (MF) method is the best choice due to its simplicity and efficiency.

2.2. Attribute Noise Handling Research

Reliability and optimization of algorithms for training accuracy requires overcoming issues such as noise, unpredictability, and novelty, as these issues significantly affect algorithm training, storage requirements, and overall accuracy. These issues are addressed by three improved nearest neighbor algorithms, described in this study [53], which served as dependable incremental learners. The presented approach was aimed at combating noise, measuring the significance of lines, and analyzing observed differences. Algorithms can be contaminated with noise, redundant features can hinder learning, new features can destroy the learning process. The proposed algorithm tries to solve these problems by combining methods to increase the flexibility of the algorithm. A common feature of these algorithms is their ability to perform well in support of learning. The effects of noise, novelty, and uncertainty are eliminated, resulting in reliable and accurate classification. Using the induction method, filtering algorithms have been improved and new functions have been added.

Table 2.2: Attribute noise handling approaches

Ref.	Title	Year	Technique
[53]	Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms	1992	Instance Based Leaning (IBL) Algorithms
[23]	Identifying noisy features with the Pairwise Attribute Noise Detection Algorithm	2005	Pairwise Attribute Noise Detection Algorithm (PANDA)
[17]	The Pairwise Attribute Noise Detection Algorithm	2006	Pairwise Attribute Noise Detection Algorithm (PANDA)
[54]	Attribute Noise Detection Using Multi-Resolution Analysis	2006	Discrete Wavelet Transforms, PANDA
[55]	Empirical Case Studies in Attribute Noise Detection	2009	Pairwise Attribute Noise Detection Algorithm (PANDA)
[56]	Classification algorithm sensitivity to training data with non-representative attribute noise	2009	J4.8, AdaBoostM1, SMO, IBk, Logistic (Ridge logistic regression)
[57]	Pairwise attribute noise detection algorithm for detecting	2016	Pairwise Attribute Noise Detection Algorithm (PANDA), SVM

	noise in surface electromyography recordings		
[58]	Attribute Noise, Classification Technique, and Classification Accuracy	2017	SVM, Principal Component Analysis (PCA), Robust Principal Component Analysis (RPCA), Random Forest (RF)
[59]	Clustering data with the presence of attribute noise: a study of noise completely at random and ensemble of multiple k-means clustering	2019	LCE, HBGF, EAC-SL, EAC-AL, CSPA, HGPA, MCLA, AggF, AggLF, AggLR, QMI, MM, IVC, TOME, LWEA
[60]	Attribute Noise Robust Binary Classification	2019	Symmetric Dependent Attribute Noise Model (Sy-De), Asymmetric independent Attribute Noise Model (Asy-In)
[61]	Can machine learning paradigm improve attribute noise problem in credit risk classification?	2020	PCA, Linear Discriminant Analysis (LDA), Multidimensional Scaling (MDS), Kernel PCA (KPCA), Restricted Boltzmann machine (RBM), Classification and Regression Tree (CART)
[62]	ANCES: A novel method to repair attribute noise in classification problems	2022	Attribute Noise Corrector based on Error Scores (ANCES)

A critical issue in knowledge mining and discovery is data quality. This work [23] provided useful information about quality characteristics in electronic data and a unique technique for detecting noise characteristics. To detect instances of noisy objects, this method uses the excellent Pairwise Attribute Noise Detection Algorithm (PANDA) to detect instances that contain attribute noise. Case studies were conducted to evaluate how the method detects noise using real-time software measurement data and simulated noise injection. The primary goal of this study was to find noise patterns associated with classes or attributes that could be modified or removed prior to analysis. However, to assess alternative analyses, this study emphasized the need to understand the noise field in certain properties. The proposed technique continuously determines the noise classification characteristic using PANDA. Attribute noise is detected by carefully removing noise from each instance and evaluating its effect on the classification of the sample. By establishing a relationship between noisy data and attributes, this method provides domain experts with valuable information about the quality of those attributes. Experiments with data measurement software have demonstrated the effectiveness of this technique in detecting sound characteristics. The case of noise classification has a significant effect, and this method is effective when identifying noise sources.

The assessment of data quality is the main goal in the development of data mining algorithms. Since the accuracy of the input data has a significant impact on the performance of these models, an efficient noise detection technique is required. Although much attention has been paid to noise detection, significant results have not been achieved because high noise levels are difficult to detect. This paper [17], introduces a new method for finding noise attributes, namely the Pairwise Attribute Noise Detection Algorithm (PANDA). Using a real-time measurement software case study, the proposed method was validated and compared to the distance-based detection method (abbreviation DM). This study focuses on general noise problems in class and focuses on the details of sound perception. The PANDA method was developed to find occurrences of the attribute noise, which has the advantage that it does not require class identifier information. The effectiveness of this approach is evaluated through a case study, using software measurement datasets of NASA, and its applicability to several domains. Software engineering experts have continuously tested the PANDA and DM algorithms and found noise in the samples of both methods. The results show that PANDA is more effective than DM in finding noisy entries. Furthermore, this study highlights the importance of using real noise rather than artificially produced noise when evaluating real data.

Research shows how important data quality is to extract knowledge from database information. Finding noise in a dataset is important as it can distort and devalue the information found. In this study [54], a new approach to identify noisy features in software-metrics using multi-resolution discrete wavelet transforms. To validate the proposed approach, data collected by Military Command, Control and Communications System (CCCS) and Metrics Data Program (MDP) of NASA were used. Comparing the experimental results of the method with those of the PANDA and MDP datasets, the latter appears to be more appropriate, but comparing 300 datasets shows different results. Through several case studies, including the deliberate introduction of known generated noise into certain properties without adding class noise, all results were thoroughly validated. The importance of data quality in decision-making processes involving data across a variety of areas was highlighted in the article. The study's empirical methodology was built on meticulously calibrated datasets, guaranteeing the existence of noise. It is noteworthy that the suggested solution did not rely on a particular attribute, like a class attribute. The JM1 dataset’s results showed good comparisons to PANDA, with only a few slight differences in attribute noise ranks. The results of the CCCS, however, did not agree with PANDA as well, most likely because the dataset’s multivariate normality varied. The case study 5 resulted in a 100% similarity ratio when a "De-noising" technique was used to support the premise that CCCS demonstrates greater multivariate normality than JM1 in the research. Case Study 6 looked at non-normal data, notably the JM1 data subsets, to further confirm the theory of dataset normality. This study's contradictory findings prevented attribute ranking. In conclusion, this study made significant contributions to our understanding of the use of discrete wavelet transforms and other linear signal processing techniques to attribute noise detection in data mining.

To emphasize the significance of data quality in domain-specific data mining, this research [55] introduced a novel method for sorting characteristics in a dataset according to the amount of noise present in the data. Data analysts received insights for efficient data treatment, such as removing or cleaning noisy features, specific to the data mining application, by recognizing and rating noisy attributes. The efficiency of the method was demonstrated in several case studies utilizing synthetic and real-world datasets, exhibiting correct attribute noise rankings and possible applicability in classification scenarios. Data-driven algorithms and knowledge discovery projects stand to gain from the empirical investigation's potential to improve data quality and enhance data cleaning techniques.

In this work [56], an empirical comparison of classification algorithms was done where attribute noise levels in the training data were not accurate representations of field data. The goal of the study was to determine how sensitive various classification algorithms were to different noise levels and if it was beneficial to make the necessary expenditures to get realistic noise levels. The experimental design was creative and covered elements like the algorithm, training set size, noise intensity, and noise condition. The relative performance change was the performance metric employed. The study's findings confounded common knowledge by arguing that it might not always be required to make investments to reach representational noise levels. Overrepresenting training noise was more of a risk compared to underrepresenting, the study revealed, which should be avoided. Field data cleaning frequently led to performance gains. Due to the interplay between the training set size, the algorithm and level of noise, it was highlighted that there could be exceptions to these general conclusions. Internal validity was the focus of this study, but it was concluded that it could not be applied to all fields. In conclusion, this study shed light on how variations in attribute noise levels within training and real world settings are reacted upon by classification algorithms and it also brought to attention the understanding of considering the environment and the relationships between variables while evaluating algorithm sensitivity to noise.

The primary objective of [57] was to tweak an existing algorithm, first created for data mining software metrics, for the appraisal of surface electromyography (sEMG) signals. The PANDA algorithm was utilized to differentiate between noisy and clear sEMG signals with an emphasis on distinct forms of noise. The three phases of this study were configuration and testing using recorded and baseline data. Boundary settings, the amount of baseline signals, the feature set and the number of bins were the variables that were investigated to create an algorithm that worked efficiently. After initial testing using artificial sEMG data, a shift to recorded data was made for further consideration which marked constraints resulting from the differences with real data. Another element (normalcy) was added to the design of the algorithm for recorded data after which receiver operation characteristic testing was used to choose the correct boundary settings and bin numbers. PANDA was then evaluated using five distinct types of noise, i.e., motion artifact, saturation, powerline interference and their combinations, at diverse contamination levels. PANDA could correctly detect clean signals with a false alarm rate of either 9.1% or 4.2%, according to the results. Additionally, depending on the kind of noise, it showed a high sensitivity of 100% in recognizing different noise types and combinations until transition points. The Support Vector Machine (SVM), another tool for evaluating quality, and PANDA's performance were evaluated in the study. All noise types could be distinguished more accurately by PANDA than by the SVM, although blended noise types (such as power line interference and motion artefact or power line interference, motion artefact, and saturation) were particularly well-identified. PANDA's success in identifying signals with numerous noise sources was a noteworthy accomplishment given that Clean EMG usually deals with single sources of noise.

In cyber-security, where many response variables have a binary character, binary data categorization is critical. The selection of the classification method and the caliber of the data are important drivers of classification accuracy, among other variables. The necessity of choosing the right classification approach as well as recognizing and reducing noise is highlighted by the fact that noise in the data may considerably reduce classification accuracy. These issues were addressed, and classification accuracy was improved in this study [58]. Creating a noise reduction algorithm and analyzing the influence of noise on classification accuracy were the primary objectives of this study. The dataset used comprised of online credit card transactions to check for fraudulent transactions. The negative effect of noise on the categorization was seen in the experimental results. Random forest algorithm typically surpassed the other classification algorithms indicating its flexibility in the face of noise, however SVM algorithm performed adequately with low levels of noise but robust principal component analysis (RCPA) algorithm surpassed SVM when noise levels were high. The effect of data skewness on classification accuracy was also investigated and it was concluded that it had a varying effect on accuracy relying on the method used. SVM was prone to while random forest algorithm was resilient to skewed data and it became clear that principal component analysis algorithm can be used to effectively classify noisy and skewed data. To improve the classification accuracy in the presence of noise, a novel noise removal technique was proposed which produced promising results when compared to Cook's distance method. Optimal sample size ratios for the training and testing dataset were also investigated which demonstrated a direct correlation between ratios and classification accuracy.

The primary clustering algorithm k-means was utilized to perform a detailed experimental investigation of the effectiveness of consensus clustering approaches in the analysis of noisy data during the study [59]. The primary inspiration behind this study was based upon the conclusion that ensemble methods may deliver precise findings when working with noisy data even though the noise is added instantly. Two main research topics were the focus of their study. The authors' first goal was to comprehend how consensus clustering technique's function when used on data with various amounts of noise. Second, when used in the context of cluster ensembles, they investigated the effects of adding negligible amounts of noise to data on centroid-based clustering methods like k-means. Eight UCI datasets, thirteen well-known cluster ensemble techniques, and two distinct quality metrics were used in a wide range of studies. They conducted several trials for each noise ratio and experimental condition to generalize their findings. The findings showed that while most approaches performed well at low noise levels (between 1 and 5 percent), they became less effective at higher noise levels. To increase accuracy, it was advised to apply data pretreatment techniques to cut down on noise before using consensus clustering approaches. It was interesting to see that several algorithms, such as EAC-AL, LCE, and TOME, performed quite well even in the presence of minimal noise. In several cases, they even provided better clustering results than the original, noise-free data. Experiments employing both uniform random and Gaussian-based noise creation provided evidence in favor of this conclusion.

The intricacies of learning linear classifiers while working with labels and binary features also the difficulty of noisy features, was the purpose of this study [60], where features may be reversed with an unknown frequency. The two attribute noise models that were tested were the ASY-In model (which allows more independence in distribution over a 2-dimensional feature space) and the Sy-De model (in which every feature has an equal chance of being noisy). It was revealed that the loss function was not flexible enough to noise in the Sy-De model while the widely used squared loss function was which disclosed that squared loss could be better for learning in the presence of attribute noise when the features are binary. In the case of ASY-In model the loss function is flexible enough towards noise with a distribution across a 2-dimensional feature space while squared loss is not. The experimental results supported the flexibility of squared loss in the Sy-De model for low to moderate noise rates.

By introducing a dual voting based learning model, the issue of attribute noise in credit risk classification was focused upon in this study [61]. To deal with attribute noise in a profitable manner a three step model was proposed. During the initial step, four indexes were created to assess the noise levels in the datasets which was the starting point for determining the severity of attribute noise. The proposed learning model focused on the noise level results of dual voting model and divided the features with different noise levels into assorted feature groups based on the noise level, in the second stage. The training dataset was generated using several denoising methods to determine the performance of the Classification and Regression Tree (CART) model. Finally, the dataset is compared to the set of unique features using different denoising techniques and learning algorithms. Experimental data demonstrated that the dual voting model outperformed the baseline method in terms of stability, speed, attribute noise resistance in credit risk classification, and accuracy. The study looked at how scattered data affected attribute noise, and it was found that it might boost the accuracy and stability of a particular noise reduction approach, and it also evaluated how well the algorithm performed on two publicly accessible credit datasets which demonstrated its success in both instances in reducing attribute noise. The proposed dual voting based learning model consistently outperformed benchmark techniques even though the datasets were diverse.

A novel attribute noise correction method (ANCES) was developed in this research [62], that alters attribute noise instead of removing noisy models. Most of the errors that affect class label are treated by typical noise filtering methods, whereas ANCES suggested another approach to settle attribute level noise. Every value in the dataset was assigned an error score in an iterative method to help in identifying potential noise. Afterwards, an optimization meta-heuristic was used to settle these values to get better quality of data. To improve noise detection, the study also considered various iterations of the initial dataset. Thorough experimental research comparing ANCES to alternative noise preprocessing methods, including without preprocessing the data, was done to confirm the efficiency of ANCES. For assessment, real-world datasets with varying degrees of attribute noise were employed. The outcomes of the studies showed that using ANCES has advantages over preparing the data. It was made clear that reducing attribute-level noise might improve classification performance. Furthermore, ANCES often performed better than noisy sample removal preprocessing methods. It was discovered that, with some datasets and noise levels, deleting noisy samples might still produce promising results, particularly when the noise was associated with class label data.

2.3. Attribute and Class Noise Handling Research

Concentrating on the improvement of selecting test cases for continuous integration, using predictive models, the study [63] considered the integration of ML models and big data in software engineering. The issue of data noise was addressed, especially attribute and class noise that affects the test selection model's capacity for prediction. The study used domain expertise to reduce class noise by relabeling inconsistent data and eradicating duplicates, and it also conducted an experiment to test how reducing attribute noise affects learning. The investigation highlighted how class-noise cleansed data promotes optimum learning and results in considerable improvements in accuracy, recall, and f-score measures. This study's intriguing conclusion, which controlling attribute noise may not be as important as previous research suggests, prompted additional investigation of attribute noise's subtle effects in various scenarios. This research laid the groundwork for a greater comprehension of noise reduction techniques and their implications for improving test case selection models.

The research [64] explored how noise management methods and data defects affect data analysis, and it offered three different approaches: robust algorithms, filtering, and polishing. Experimental research was done to gauge their effect. Moreover, the results showed that filtering and polishing may both reduce the negative effect of noise to avoid overfitting. It was notable that polishing regularly beat the other techniques, proving its ability despite more complications. However, due to the large occurrence of noisy instances (for example, at 10% noise level, over 50% of instances are noisy), the authors noted that utilizing fully filtered data as a baseline was unfeasible. This fraction also expanded rapidly with the rising of the noise levels. The research proposed combining many noise handling methods to reduce noise. Generally, the study highlighted that a better understanding of the behaviour is needed before increasing the efficiency of these methods when combined.

Table 2.3: Class and attribute noise handling approaches

Ref.	Title	Year	Technique
[63]	Improving Test Case Selection by Handling Class and Attribute Noise	2021	**Class Noise:** Relabeling (for handling contradictory entries) **Attribute Noise:** Pairwise Attribute Noise Detection Algorithm (PANDA)
[64]	A Comparison of Noise Handling Techniques	2001	C4.5 Algorithm
[16]	Combining Noise Correction with Feature Selection	2003	C4.5 Algorithm
[12]	Class Noise Vs. Attribute Noise: A Quantitative Study	2004	Classification Filter (CF) Partitioning Filter (PF)
[65]	Noise identification with the k-means algorithm	2004	C4.5 Algorithm
[66]	Enhancing software quality estimation using ensemble-classifier based noise filtering	2005	Ensemble created from 25 Diverse Base Classifiers
[67]	Improving Software Quality Prediction by Noise Filtering Techniques	2007	Partitioning Filter (PF) => Multiple-Partitioning Filter => Iterative-Partitioning Filter
[68]	Data Cleaning Techniques for Software Engineering Data Sets	2010	C4.5, Classification and Regression Trees (CART)
[69]	Reasoning with Noisy Software Effort Data	2014	Artificial Neural Network (ANN), Decision Tress (DT), k-NN, Logistic

			Discrimination Analysis (LgD), Naïve Bayes (NB), RIPPER, STOCHS, Support Vector Machine (SVM)
[70]	FECS: A Cluster Based Feature Selection Method for Software Fault Prediction with Noises	2015	Feature Clustering with Selection (FECS) strategies
[71]	A Promising Method for Correcting Class Noise in the Presence of Attribute Noise	2023	Principal Component Analysis (PCA), Decision Tree (DT), Support Vector Machine (SVM), Naïve Bayes (NB)

The research [16] explored the usage of the polishing noise reduction method on a dataset referring to amino acid orders and pointed change of the COLIA1 gene to classify the signs of the genetic disorder Osteogenesis Imperfecta (OI). Polishing particularly found and corrected noisy elements by using the connection between attribute and class values. Polishing could boost classification validity according to initial results of the study. The research also examined the impact of polishing on classifier performance when used as an attaining procedure for quality choice. Research on the OI dataset showed that feature selection and polishing both improved prediction accuracy separately. Notably, combining the two methods produced even better outcomes, demonstrating their ability to jointly enhance data quality by removing unnecessary features and correcting noisy values.

This study [12] examined ways to deal with noise in machine learning, specifically attribute noise, and assessed its effects. Using 17 datasets, the study categorized noise into two categories, attribute, and class noise, and determined the impact of each on system performance individually. Both attribute and class noise were discovered to have a considerable effect on learning algorithms. The classification accuracy was improved when class noise was reduced by eradicating instances that included it. Although often less destructive than class noise, attribute noise might still cause issues for learning systems. When tackling attribute noise, noise correction techniques had been demonstrated to increase the accuracy of learnt classifiers. Even though the model had been trained using a noise-corrupted training set, addressing attribute noise in the test set typically yielded more substantial advantages regarding classification accuracy. When addressing the test set's noise was not an option, clearing the training set's attribute noise nevertheless greatly increased classification accuracy. Depending on how closely an attribute correlated with a class, the effect of attribute noise changed. Attribute noise was more pronounced for attributes with stronger correlations. It was recommended to apply learning techniques to train a noise filter to detect and fix attribute noise. However, it was essential to identify which traits were foreseeable by others and the class beforehand by looking at relationships among attributes.

The noise problem in measurement datasets and its detrimental effects on classification models are the main topics of this research [65]. Erroneous or corrupted examples within the dataset are referred to as noise in this context, and they can cause mistakes in the learnt hypothesis and decreased classification accuracy. The study's main objective was to present a clustering-based noise detection method that used the k-means algorithm to locate and remove possibly noisy occurrences. For each instance in the dataset, the proposed method entails computing a novel measure known as the noise factor. This noise factor measured the likelihood that an instance will be noisy. The dataset was then divided using the k-means clustering method to find instances with high noise factors. These examples were eliminated from the dataset because they were believed to be most probable to be noisy. During the study, two case studies were presented, using software measurement data acquired from NASA software projects, to assess the efficacy of this method. In addition to thirteen software metrics, these statistics provided a class label for each program module i.e., 'fault-prone' or 'non-fault-prone'. The trials showed a clear increase in the accuracy of the C4.5 learner when more potentially noisy cases were eliminated from the dataset. This improvement implied that the initial reduction in classification accuracy noticed was really caused by the eliminated occurrences. In conclusion, the study presented a noise detection method based on clustering that efficiently located and removed possibly noisy occurrences from measurement datasets. This greatly improved the classification precision of machine learning models, especially when it came to estimating software quality. The method made a significant contribution to solving the problems brought on by noise in real-world datasets.

In this work [66], a method to improve the quality of training data by removing noise was presented to increase the accuracy of classification models. To eliminate noisy examples, the suggested method used an ensemble classifier made up of 25 distinct classification methods. To achieve the necessary level of conservatism in noise filtering, the ensemble filter's employment of a relatively large number of base-level classifiers allows for different degrees of noise reduction. Data from a highly guaranteed software project was used as the empirical case study in this study to highlight the competence of noise elimination method in revamping the classification accuracy. Given the difference between two types of misclassifications frequently seen in software quality classification and related areas, the study used the Normalized Expected Cost of Misclassification as a practical performance metric. The study showed that the predicted accuracy of software quality classification models increased when more inherent noise was eliminated. The possibility of unintentional learning bias from a small number of algorithms impacting the outcomes was decreased by using more classifiers in the noise removal phase. Furthermore, the research hypothesized that the most restrictive level of filtering could be able to manage exceptions to some extent as at least three of the 25 base-level classifiers could properly categorize instances that were regarded as "hard-to-classify" or "exceptions". In the discussion of classifier clustering, which was discussed in the paper's conclusion, two coherent clusters were found among the 25 classifiers. While retaining the efficiency and confidence in the ensemble classifier with 25 base classifiers, this clustering technique might be utilized to decrease the number of base-level classifiers.

To increase the precision of machine learning models, the quality of training datasets was the main emphasis of this work [67]. The method involves removing instances that the Partitioning Filter had classified as noisy. This filter divided the dataset into subgroups and induced several base learners on each subset. If a given number of base learners continually misclassified an instance, the combination of predictions was used to classify the instance as noisy. The Multiple-Partitioning Filter and the Iterative-Partitioning Filter were both used as partitioning filters. Comparing the prediction performance of final models created using filtered and unfiltered training datasets was the main objective of the study. Software measurement data from a high assurance software project were used in a case study to perform the research. The results showed that models developed on noisy, unfiltered training datasets consistently underperformed models generated on filtered training datasets and assessed on noisy test datasets. The assessment dataset's noise, however, had an impact on certain harsh filtering techniques. The research showed how final learners who had been educated on supposedly noise-free datasets and assessed on noisy test datasets performed, according to the paper's conclusion. By deleting instances from the training dataset that the Partitioning Filter had deemed noisy, the training dataset's quality was improved. The amount of noise removal may be altered by adjusting the filtering level or iteration count. The results showed that the final learners performed better on the test dataset for the majority of the investigated cost ratios when employing the Multiple-Partitioning Filter (MPF) without cross-validation restrictions or the Iterative-Partitioning Filter with a consensus method (IPFConS). The test dataset's quality did not significantly increase when the fit dataset was filtered using the Multiple-Partitioning Filter with Cross-Validation Constraints (MPFCV) or the Iterative-Partitioning Filter with Majority Voting (IPFMaj). Even with a noise-free training dataset, learners could still perform badly on noisy test datasets, according to the study, even while performance increase on the test dataset was less evident than on the fit dataset. Resources should be devoted to ensuring error-free data gathering to reduce noise in datasets. The study's analysis of the relative efficacy of various filtering levels led to the conclusion that MPF-23 and IPFConS-5 were reasonably effective filters for the software measurement dataset under consideration.

The article [68] discusses the importance of data quality in empirical software engineering and how it affects the reliability and completeness of the results. Despite its significance, data quality had often been overlooked in this area of study, casting a doubt on the accuracy of the findings. The research focused on the approaches for handling noise and how to use them to boost the quality of data. Three different recommendation tree-based noise control methods were proposed and examined on large real-world software engineering dataset in detail. These findings assessed how well the approaches improved predicted accuracy and reduced doubtful value samples. More information was gathered by an imitation exercise even though different approaches showed hopeful results by using known noise levels thoroughly to examine the capability of the system to settle noise. This study defined the impact of noise and outliers on data survey and stressed the importance of making this different. It provided a realistic approach for including noise settling methods into a larger data cleaning procedure while focusing on the need of documentation and repeatability for verifying the accuracy of the findings. Finally, the research pointed out the need for valid noise management algorithms and an exhaustive approach to data analysis while highlighting the problems with data quality in experimental software engineering.

The research [69], noticed how noisy data affected eight machine learning statistical design recognition methods that were used to predict software effort. The purpose of this study was to test the effectiveness of these methods in different noise orders and types. It has been shown that the performance of machine learning and pattern recognition algorithms can be significantly affected by the amount of noise in the data used to evaluate software performance. The performance of a classifier under increasing noise mainly depends on the noise relationship between the class labels and their features. The accuracy of the classifier is affected by noise in the test set attribute values or class labels. The study found that RIPPER had the lowest reliability of the classifiers evaluated, although STOCHS generally performed well on noisy data. In terms of throughput, the next best model, decision trees, performs better than Support Vector Machines (SVM). These results show that noise must be considered when designing machine learning systems that are being built to handle noise. Since the capabilities of noise classification techniques have a large impact on prediction accuracy, they must be thoroughly understood and mastered to make effective predictions in real-world software development environments.

The research [70] analyzes the problem of noise in software error estimation. An innovative denoising strategy is proposed: the Feature Clustering with Selection (FECS) method. The two main stages of FECS (feature selection and feature collection) use different detection methods. Real-world databases, such as the NASA and Eclipse datasets, simulate noisy datasets using feature and class level noise. The effectiveness of the FECS method is evaluated in comparison with traditional feature selection methods. The results show how FECS improves software failure prediction by emphasizing its fault tolerance and flexibility. In addition, this work provides useful suggestions for the implementation of FECS systems by investigating the impact different levels of noise or characteristic coefficients have on the efficiency of the system.

The study [71] presented a new technique for denoising in the presence of feature noise. To remove features, the proposed models consider how the feature is interpreted as part of the class label. To this end, local feature reduction and stepwise PCA were used in subsequent iterations. In addition, this heterogeneous method has been used to solve noise classification problems using a variety of filtration models, Decision Trees (DT), Support Vector Machines (SVM), and Naïve Bayes (NB). Later, most of the filtration processes were replaced by the traditional method. The researchers compared the proposed method with three datasets for binary classification problems using RF Majority Vote Filter (RF-MV-F). Different classification models, Random Forest (RF), AdaBoost, SVM and NB were used to evaluate these methods. Experimental results showed that the proposed technique can improve the prediction accuracy of the classifier even in a noisy environment in most RF-MV-F characteristics. This improvement has been demonstrated on numerous examples of clinical and non-clinical classifications and datasets. The recommended technique worked well in terms of optimizing the RF and AdaBoost models, both of which are known to be affected by noise to varying degrees. Moreover, the proposed strategy produced more consistent results for the SVM model, which was sensitive to noisy data due to its cost function. In datasets with high noise levels, the suggested technique outperformed RF-MV-F in terms of Naïve Bayes.

CHAPTER 3: METHODOLOGY

ML literature has extensively addressed the challenge of obtaining a satisfactory learning performance in the face of noise conditions. Numerous methods [12], [13], [21] have been developed to improve ML classifiers' learning capabilities. However, it has been shown that attribute and class noise still have an adverse effect on learning, therefore it must be addressed prior to training. To focus on the issue that the class noise poses in this first section, we will discuss the method [72], that Al-Sabbagh et al [8] introduced and was used in the research [63]. After that, a method based on [17] will be discussed which deals with attribute noise.

3.1. Proposed Class Noise Handling Approach

As previously pointed out, relabeling code lines with different class values that are repeated is the suggested method for addressing class noise. Al-Sabbagh et al [8] offered this suggested method. There are several possible reasons for the repeated LOC, such as code duplication [22] or code merging [14]. The first scenario involves "copy-pasting" code that has previously passed integration and testing to be reused. The second scenario arises when developers working on feature development-specific branches from one or more teams use code that looks like the code that has been committed and merged from other branches. Of the entire block of code revisions, a small portion sometimes contains lines of code that have some kind of defects. Consequently, it is more likely that the failure was not caused by a single line from a fragment that was considered failed overall. Therefore, in cases where lines were previously recognized as parts of segments that did not fail, we have decided to reclassify them as "passed".

Here is a detailed explanation of the methodology (Figure 3.1 & Algorithm 1):

- **Data Preprocessing:** To guarantee data quality, the method first loads the dataset and then performs preprocessing operations. The dataset is cleaned up and made

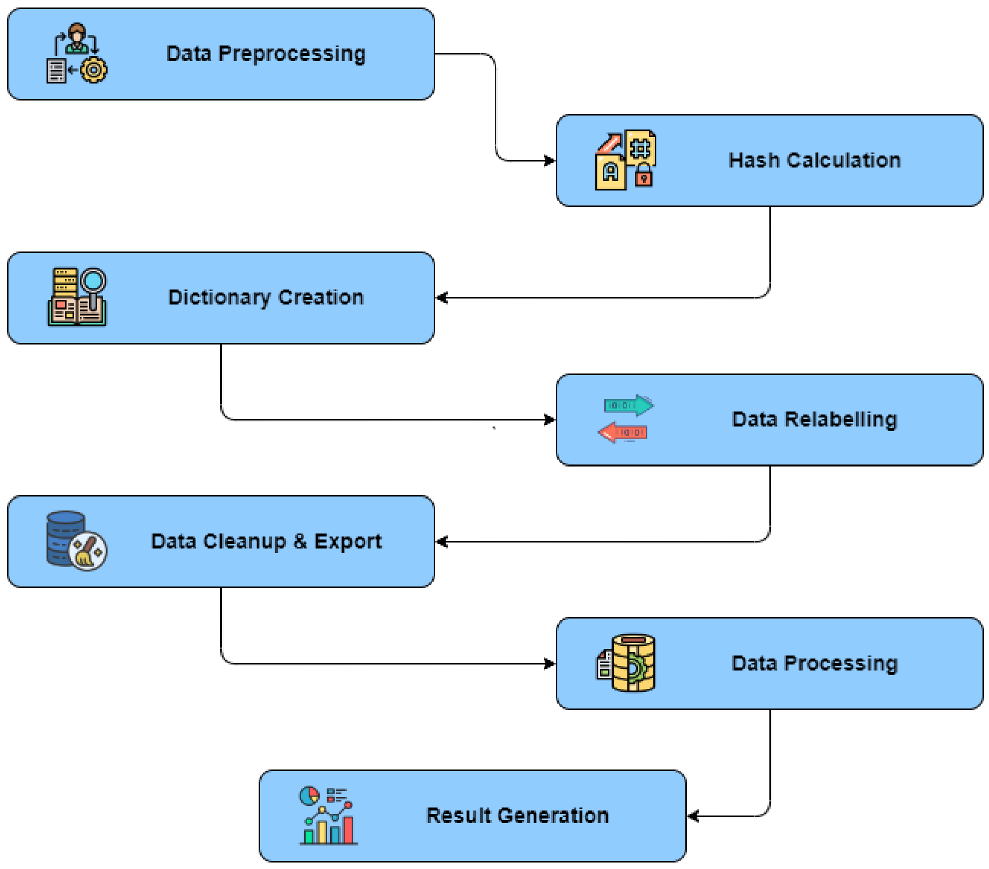

Figure 3.1: Flow Chart for Class Noise Handling Algorithm

easier to work with by removing trailing spaces from column names and removing any duplicate columns.

- **Hash Calculation:** Hashing is a method that allows for effective data comparison by transforming data into a fixed-length string of characters. The proposed technique creates a distinct 8-digit hash value for every row using the lines of code (given in the dataset).
- **Dictionary Creation:** The algorithm creates a dictionary to associate hash values with class verdicts. Every dictionary entry indicates a unique hash value together with the corresponding class verdict. The dictionary enables the algorithm to efficiently track the relationship between hash values and class verdicts.

Algorithm 1: Pseudocode for handling Label Noise

```
define class CRowInfo:
    iHash & iVerdict ← 0
define function generateHash():
    for each row in class_noise_dataset as CND:
        iHash ← 8-digit Hexadecimal Hash of LOC content
        CND ← iHash
    end for each
rowDict ← {}
define function generateDictionary():
    iLineIndex ← 0
    for row in class_noise_dataset as CND:
        if iLineIndex ≠ 0:
            iObj ← CRowInfo()
            iObj.iHash ← int(CND['hashed_line'])
            iObj.iVerdict ← int(CND['class_noise'])
            if iObj.iHash in rowDict:
                old_verdict ← rowDict[iObj.iHash].iVerdict
                if (old_verdict ≠ iObj.iVerdict) & (iObj.iVerdict = 1):
                    rowDict[iObj.iHash] ← iObj
                end if
                else:
                    Continue
                end else
            end if
            else:
                rowDict[iObj.iHash] ← iObj
            end else
        end if
        iLineIndex += 1
    end for
```

- **Relabeling Rows:** The algorithm goes over the dataset iteratively, checking to see if the hash value of each row matches an item in the dictionary for that row. The verdicts for that row in the dataset and dictionary are checked if a match is discovered. If the outcome in the dataset is 1 (passed) and the same instance has 0 (failed) in the dictionary, the value in the dictionary is updated to 1 (passed). If both values are 1 (passed), then that occurrence is skipped. In essence, this phase relabels rows based on the majority class selection for rows that have the same hash value.
- **Data Cleanup and Export:** Following the analysis of each row, the algorithm eliminates unnecessary columns from the data. After the dataset has been cleaned, it is saved to a new CSV file, which contains the final version of the data that includes the most current class verdicts.
- **Data Processing & Result Generation:** The initial and final version of the cleaned dataset are passed onto the learning algorithm (discussed in section 3.3). This helps to compare the improvements that were achieved from our proposed approach.

In summary, the unclean dataset is imported and cleaned of null values, duplicate columns, or trailing spaces in column names. An 8-digit hash is calculated for each code line, to uniquely identify a line-of-code even if it is repeated multiple times. A dictionary is created to store the hashes and class verdicts of each code line. Before adding a new entry in the dictionary, it is first compared to the elements already present in the dictionary. If the entry is already present and the verdict in the dictionary is ‘passed’ and the dataset has a ‘failed’ verdict, then the verdict in the dictionary is left as it is. On the other hand, if the scenario is reversed (i.e., dictionary verdict is ‘failed,’ and dataset has ‘passed’) then the verdict in the dictionary is relabeled to ‘passed.’ If the entries in the dataset and dictionary have the same verdict, then the entry is skipped. This process occurs in a loop to create an updated / relabeled dictionary of all the code lines from the dataset. The new verdicts are then mapped to their specific row in the dataset which results in a new class noise cleaned / relabeled dataset. This class noise cleaned dataset will be later used to handle attribute noise. To evaluate the progress of this proposed approach, the initial dataset and class noise cleaned dataset are evaluated using a ML model. The results of the class noise handling approach will be discussed in the next chapter.

3.2. Proposed Attribute Noise Handling Approach

As was already indicated, choosing attributes that are unnecessary for characterizing the training cases might lead to attribute noise. For instance, the code fragments that make up the analyzed code are written in diverse coding styles, or there are a few condition statements, code lines, function declaration, etc., whose structure differs from most lines that are comparable in the code. We provide a novel solution known as the *Attribute Noise Detection algorithm (ATNODE)* to deal with the attribute noise issue. Van Hulse et al [17], [23]. presented the Pairwise Attribute Noise Detection Algorithm (PANDA), from which the suggested method is derived. The PANDA technique was modified due to the computational expense suffered by the dataset's size. The suggested method has been developed to analyze the dataset to identify and address attribute noise. The method iterates through each column in the dataset, performing different actions on each one. The dataset is divided into sections and sorted based on the values of each column. Before allocating a noise score to each data point based on how far off it is from the mean in proportion to the standard deviation, it computes the mean and standard deviation of the data points in each division. This process is repeated by the algorithm for every column in the dataset. It computes noise ratings for different partitions rapidly and manages indices with care throughout the process. Given the circumstances, this methodology provides a systematic way to assess and control attribute noise in a dataset improving the quality of the data.

The proposed approach (Figure 3.2 & Algorithm 2) is detailed in the steps below:

- **Data Initialization:** The algorithm begins by doing preliminary data preparations by importing the class noise cleaned dataset from the CSV file. To make sure the dataset is tidy and well-organized, this involves deleting any extraneous or duplicate columns as well as trailing spaces from column names.

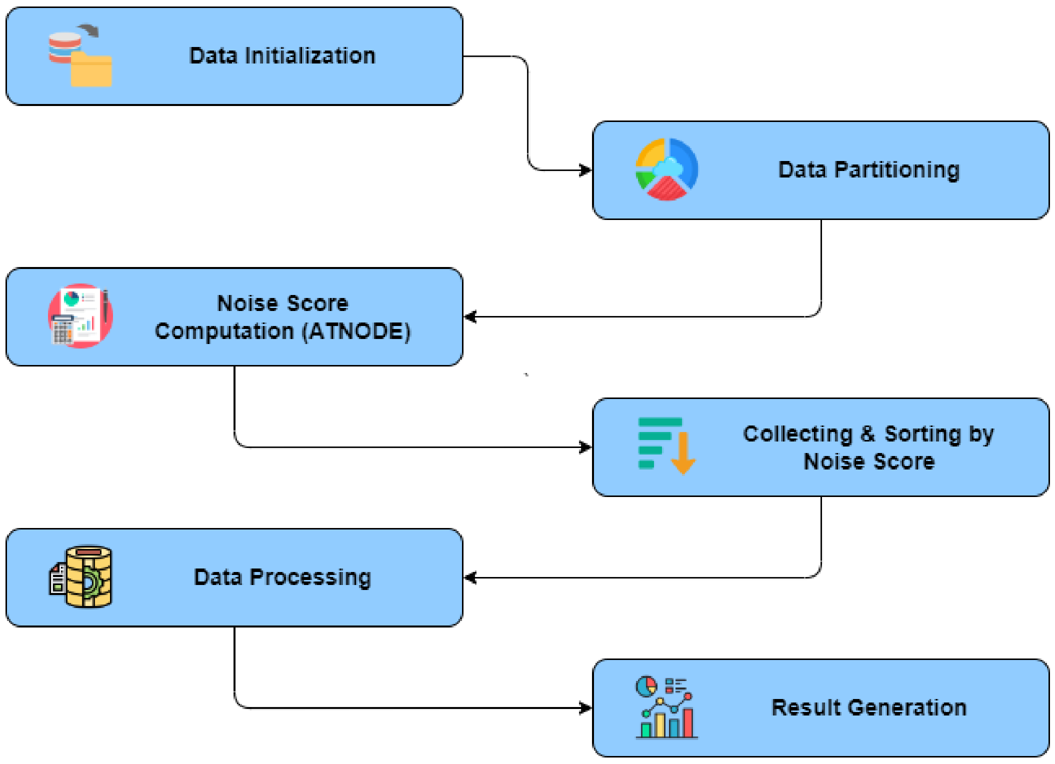

Figure 3.2: Flow Chart for Attribute Noise Handling Algorithm

- **Data Partitioning:** To make the noise score computation more efficient, the dataset is partitioned. In this instance, a predetermined number of segments, each with a comparable number of rows.
- **Noise Score Computation (ATNODE):** The iterative mechanism at the heart of the method determines noise ratings for every column (or attribute) in the dataset. After the mean and standard deviation of data within a partition are calculated, the ratio of mean-over-standard deviation for each column is also calculated. Based on the deviance of each data point from the estimated mean-over-standard deviation value, it uses this ratio to generate a noise score for each data point in the column.
- **Collecting & Sorting by Noise Scores:** The noise scores corresponding to every data point in the column are combined and kept in a new column which contains noise score for every row ("*max noise*"). Considering all columns, this aggregated

Algorithm 2: Pseudocode for handling Attribute Noise

```
define function run_ATNODE():

    bins_counter ← 0

    partition_values ← DataFrame()

    j_col ← DataFrame()


    for j in range(len(dataset.columns)):

        start_j ← time.now()

        clstrd_att ← dataset.sort(by=dataset.columns[j])

        bins_counter ← 0


        parition_mean ← mean(clstrd_att.iloc[0,j].values)

        partition_sd ← std(clstrd_att.iloc[0,j].values)

        if partition > 0:

            mean_over_sd ← parition_mean / partition_sd

        end if

        else:

            mean_over_sd ← 0

        end else

        noise_score ← DataFrame(for x in clstrd_att.iloc[0, j]:
                                 return abs(x - mean_over_sd))


        partition_values ← concat([partition_values, noise_score],
        axis=0)


        bins_counter ← bins_counter + bin_size

        j_col ← concat([j_col, partition_values], axis=1)


        end_j ← time.now()

        time_df ← end_j - start_j

    end for

    return j_col
```

noise score represents the total noise level connected to each row. Subsequently, the *max noise* values are used to sort the dataset in descending order. It is now simpler to recognize and manage noisy data instances due to this design, which places rows with higher noise ratings at the front of the collection.

- **Data Processing & Result Generation:** The attribute noise-sorted dataset was subsequently assessed at ten distinct levels of treatment, specifically from 5% to 50% with each level being a 5% increment of the previous one. In other words, the quantity of top noisy cases that corresponded to the treatment level were removed from the dataset and the remaining data was assessed again but now with less noise.

To recap, the class noise cleaned dataset is initialized, and data cleaning steps are taken to ensure that the data is clean of impurities. The dataset is divided into partitions, to improve the efficiency of the noise score computation, into a fixed number of segments, with a similar number of rows in each. Following the computation of the mean and standard deviation of the data inside a partition, the mean-over-standard deviation ratio is determined for each column. It utilizes this ratio to produce a noise score for each data point in the column based on how different each data point is from the predicted mean-over-standard deviation value. A new column with a noise score for each row is created by combining the noise scores for each data point in the column. The dataset is then sorted in decreasing order using the attribute noise score values. After sorting, the dataset was analyzed at 10 different levels of attribute noise. The dataset is then analyzed again after removing a certain number of high-noise events corresponding to the level of attribute noise. The next section describes the results of the proposed method.

3.3. Learning Model for Evaluation

The aim of this study is to extend the work of Al-Sabbagh et al [63] who used Random Forest (RF) as the learning model for evaluation. It Combines the results of multiple decision trees to get a single result. Its popularity is due to its adaptability, ease of use, and ability to solve classification and regression problems. The main reasons for choosing this were its white box architecture and lower processing costs compared to other deep learning models. The way it handles complex datasets and reduces overfitting makes it a valuable tool for solving a variety of machine learning prediction problems. It works efficiently even if the data contains zero or missing values and is not affected by the curse of dimensionality, because not all trees represent every feature.

The hyper parameters of the evaluation algorithm were maintained at the default configuration using the scikit-learn package. The n_estimator value in the RF model was not adjusted as this study was aimed to evaluate the impact attribute and class noise had on build predictions rather than to maximize the predictive performance of the model. We experimented using a version of the n_estimator, having the value 300, in the RF model to find out if this would have an impact on the predictive ability of the model. Appendix contains the generated results of the model's performance using n_estimator at 300.

CHAPTER 4: EXPERIMENTAL STUDIES AND RESULTS

We get into the fundamentals of our study in this chapter, where we outline the research topic, the extensive dataset analysis, and the fascinating findings from our studies. This chapter's main goal is to give a thorough description of the real-world experiments and evaluations carried out to gauge the viability of the suggested approaches. The results and discoveries covered in this chapter are crucial pieces of evidence that help to shape our perception of the efficiency, performance, and practicality of the techniques covered in previous chapters.

4.1. Dataset Analysis

Al-Sabbagh et al [24], [25] previously used the dataset that is being used in this work. The dataset includes results from build predictions made during source code integration. The dataset consists of lines of code (LOC) from a Java-written system. Initially, the dataset had 2816 dimensions, some of which were duplicates. At the time of execution, duplicate columns and null values were removed from the dataset. The final dataset had 4994 LOC with 1941 different dimensions and construct forecasts for each item.

4.2. Research Question

One of the biggest challenges for software developers is dealing with attribute and class noise in code segments. Designing an automated and effective method for noise identification and removal is crucial due to the detrimental effects noise may have on software dependability, code quality, and development productivity. This study aimed to give new and creative approaches—described earlier—that can manage noise at various granularities of code and accommodate a range of noise patterns. To give a strong and comprehensive solution, this was achieved by combining data mining, natural language processing, and sophisticated machine learning algorithms. In the present effort to manage attribute and class noise, the following research topic will be addressed:

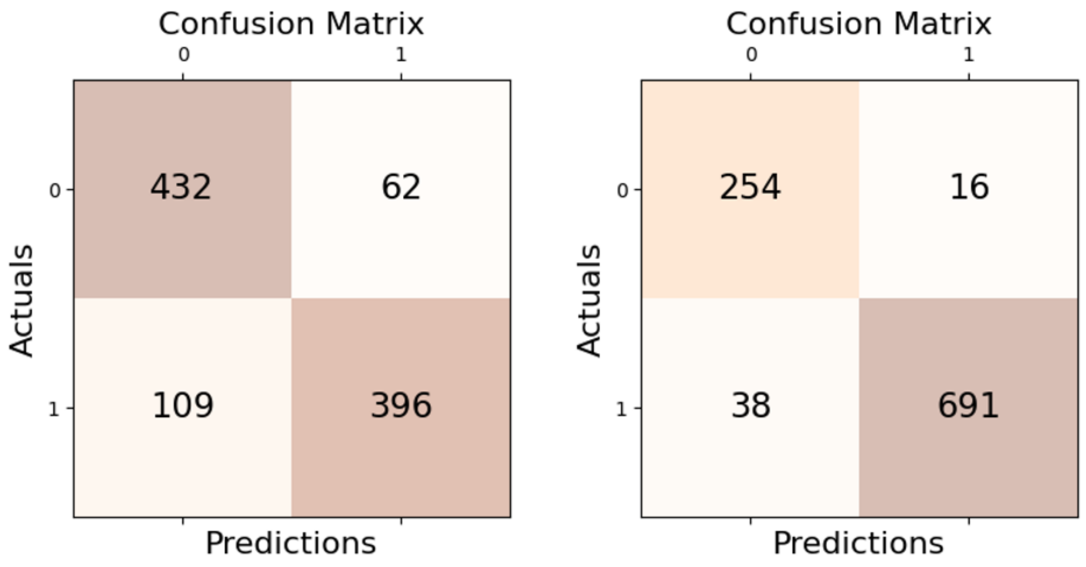

Figure 4.1: RF Results on Original (left) & Class Noise Cleaned (right) Dataset

RQ: How can we create a machine learning algorithm with nominal computational and execution time costs that can handle attribute and class noise in actual datasets?

4.3. Results

In the following section, we summarize and discuss the findings of our research. We give a comprehensive analysis of the collected data and draw significant implications from the results. The focus is on analyzing the data, identifying trends, and drawing conclusions from the data. Through the presentation and analysis of the data, we seek to foster a deeper comprehension of the experimental results and their implications.

4.3.1. Class Noise Approach

Figure 4.1 shows the comparison between the expected and actual values of the verdicts of build prediction for each instance in the dataset as unnormalized matrices. It uses confusion matrices to illustrate performance indicators for the Random Forest (RF) classifier, which was built using both uncleaned and class-noise cleansed data. The diagonal of the matrix (right) shows how many lines (691, as opposed to 396 in the original

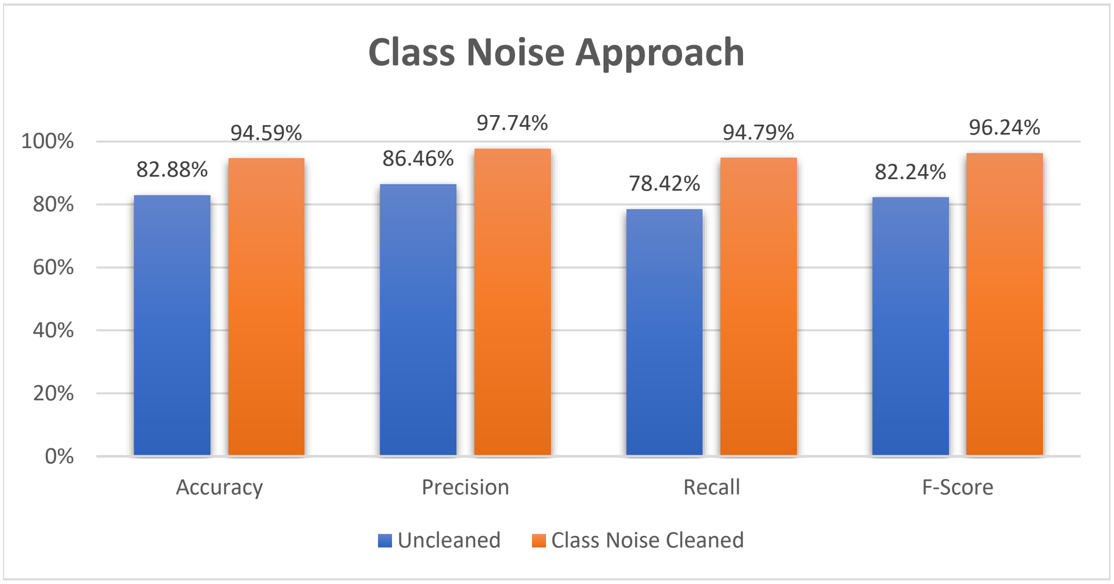

Figure 4.2: Learning Performance before and after Removing Class Noise

dataset) were correctly identified as non-defective as well as how many lines (254, as opposed to 432 in the original dataset) were correctly identified as defective without needing additional testing. The number of lines that are incorrectly categorized is reflected in the matrix's off-diagonal entries.

The performance of the classifier is shown in the bar graph (Figure 4.2) using class noise cleaned and the initial datasets. Learning from the class noise cleaned data shows a significant improvement in learning performance compared to learning from the original data. When compared to the original data, the class-noise cleaned data shows 16.37% improvement in recall, 11.71% improvement in accuracy, 14% improvement in F-score and 11.28% improvement in precision.

4.3.2. Attribute Noise Approach

This section focuses on assessing the RF classifier's performance using the dataset we were able to get through a methodical class noise reduction technique. The dataset is processed at different attribute noise levels (sometimes called treatment levels), from 0% to 50%. Here 0% treatment level will serve as the baseline (control group) for class noise

Table 4.1: Performance Metrics at Different Treatment Levels of Attribute Noise

Attribute Noise Removed	Accuracy	Precision	Recall	F-Score
0%	0.972	0.981	0.981	0.981
5%	0.980	0.985	0.987	0.986
10%	0.972	0.982	0.980	0.981
15%	0.972	0.983	0.979	0.981
20%	0.962	0.979	0.970	0.974
25%	0.965	0.982	0.972	0.977
30%	0.969	0.978	0.978	0.978
35%	0.969	0.984	0.971	0.978
40%	0.972	0.985	0.973	0.979
45%	0.956	0.978	0.956	0.967
50%	0.970	0.988	0.967	0.978

cleaned data. Each level corresponds to a distinct treatment scenario. To obtain an understanding of how the level of noise reduction affects the classifier's prediction skills, we evaluate its accuracy, precision, recall, and F-Score at these different treatment levels. The findings offer insightful details on the trade-offs between classifier performance and attribute noise removal, illuminating the ideal ratio between data purification and model correctness.

Table 4.1 lists the F-score, accuracy, precision, and recall values for the various noise reduction thresholds, which range from 0% to 50%. The performance indicators show a discernible pattern of improvement as the noise reduction percentage rises. Accuracy rises steadily up to 10% noise reduction and then stays mostly constant. Noise reduction usually leads to an improvement in precision, which peaks at 50%. Recall varies during the noise reduction process, but it consistently maintains high levels. The F-Score improves with more noise reduction, following a similar trend to accuracy.

4.4. ATNODE vs PANDA

A notable disparity in growth rates is seen between the suggested innovative algorithm, ATNODE, and the baseline approach, PANDA, when comparing their time and spatial complexities. PANDA shows a sharp rise in runtime as the input size, n, increases, with a cubic time complexity of ***$O(n^3)$***. Conversely, ATNODE has a quadratic temporal complexity of ***$O(n^2)$***, indicating a more advantageous development rate in contrast to PANDA. This disparity implies that when input size increases, performance loss of PANDA is substantially more than that of ATNODE. Consequently, it has been demonstrated that ATNODE outperforms PANDA in terms of runtime efficiency for large input sizes.

Also, the space complexity of PANDA is ***$O(n^2)$*** whereas ATNODE has ***$O(n)$*** which means that the latter requires less memory, based on the square of the input size, which indicates that it can process large amounts of data in limited memory space. Therefore, ATNODE has great advantages in increasing time and space efficiency.

In summary, the comparison between PANDA and ATNODE shows that both algorithms have polynomial complexities, but ATNODE is significantly better than PANDA in terms of time and space complexity. Given these differences, ATNODE provides a competitive alternative to PANDA when memory capacity and processing power are limited, or large amounts of data are involved. It also appears to be a more resource-efficient and scalable performance choice.

4.5. –Discussions

In this section, findings presented in 4.3.1 and 4.3.2 are discussed to address the research question, *how can we create a machine learning prototype with nominal computational and execution time costs that can handle attribute and class noise in actual datasets?* Also, a comparison is made of the efficiency of managing attribute noise against class noise. By looking at the accuracy, recall, and f-score in Figure 4.1, Figure 4.2 and Table 4.1, comparative findings are obtained. Remember from Section 4.3.2 that the baseline measurements are the performance metrics obtained for the control group at 0%

treatment level. The impact handling attribute noise has on the performance of the Random Forest ML model at various levels, is investigated using the remaining treatment levels.

Observations regarding class noise removal showcase that in the original data, the confusion matrix presents that the number of false negatives (FN) and false positives (FP) exceed those of true negatives (TN) and true positives (TP). Once the class noise is removed from the data, the matrix shows a significant decrease in both FN and FP and a significant increase in TN and TP, suggesting enhanced model performance. Significant gains are exhibited in recall, accuracy, F-score, and precision of class noise cleaned data indicating the efficacy of class noise management.

While observations regarding attribute noise removal showcase that for different attribute noise reduction levels, which range from 0% to 50% shown in the Table 4.1, accuracy, precision, recall, and F-score all show a discernible improvement with increasing percentages of attribute noise reduction. The performance of the model improved by a larger proportion of attribute noise reduction, based on enhanced accuracy and other performance indicators, as revealed by the data. The highest performance metrics are obtained after 5% of attribute noise is eliminated, with peak values being reached for F-score, recall, accuracy, and precision.

Overall, the findings demonstrate how crucial noise management is for enhancing the performance of classification models, both for attribute and class noise. Removing class noise improves performance measures such as model accuracy dramatically. Additionally, attribute noise removal improves the performance of the model; the biggest gains are shown at 5% noise removal.

4.6. Threats to Validity

In the context of software engineering experiments [73], addressing threats to validity is crucial to ensure the reliability and credibility of the research results. Here are some suggested ideas with reference to conclusion, internal, construct and external validities.

Internal Validity: It refers to the extent of how much may be implied about the causal link between dependent and independent variables.

- **Causal Relationship:** Making sure that the noise handling strategies are the true cause of the observed causal links between model performance and class noise treatment, and that no other confounding factors have an impact.
- **Experimental Design:** This type of threat can be minimized by carefully organizing our study and considering possible bias, random variation, and other factors that may affect the results.
- **Measurement Validity:** This is the process of confirming that the tools used to measure performance metrics such as F-score, accuracy, precision, and recall are dependable.

External Validity: It refers to the degree to which research findings can be generalized to other situations, people, contexts, and scales.

- **Sample Representativeness:** We can avoid this risk by ensuring that experimental datasets accurately reflect actual software quality data and by considering the possibility that results in other environments or regions may not be realistic.
- **Generalizability:** We can avoid this risk by agreeing to the fact that the results may not generalize across models or methods and may relate only to the specific classification algorithms and denoising techniques used.
- **Task Representativeness:** It is important to consider the possibility that the evaluation tasks used in the study may not adequately reflect the scope of software quality evaluation and that the results may not be generalizable to other systems.

Construct Validity: This is used to determine how well the test measures what it is supposed to measure.

- **Operationalization:** This risk can be avoided by making sure that the concepts being measured (such as model performance or separation noise handling) are clearly defined and implemented in a way that is consistent with the existing literature.

- **Experiment Setup:** It can be avoided by ensuring that all treatments and controls are exposed to the same experimental conditions as any number of configuration irregularities could jeopardize the reliability of the test design.

Conclusion Validity: It is used to judge whether the inferences we make about the relationships in our data are valid and dependable.

- **Statistical Analysis:** This risk can be avoided by evaluating the possibility that the statistical methods used to validate the data may affect the accuracy of the results and making sure that we select and use the correct statistical test.
- **Extraneous Variables:** Determining and taking into consideration any unrelated factors that might affect the validity of the inferences made from the studies.

CHAPTER 5: CONCLUSION

The investigations and empirical results of this study provide valuable insights into the key area of noise management in software quality dataset. The quality of training data can have a significant influence on the efficiency of a classification model and our results highlight the significance of managing noise in machine learning and data mining scenarios.

The accuracy of classification models can be improved using research-based denoising techniques. Relabeling significantly reduces class noise and improves model performance, as evidenced by high recall, precision, and precision values, as well as high F-scores. To make more accurate predictions, you need to use pre-processed data so that the machine learning model can easily distinguish between broken and working modules. Our study advances our understanding of the complex relationship between attribute and class noise through careful analysis of datasets with different noise levels in the context of software quality evaluation.

The need for preprocessing methods is highlighted to ensure the accuracy of the predictive model, since inaccurate data can significantly impact the results to begin with. By reducing noise and applying it to real world scenarios, we can see significant improvement in the performance of any model. The methods discussed in this study ultimately make the quality assessment more accurate and dependable.

5.1. Future Work

We expect that much work will be needed on further research on noise reduction in software quality datasets. This exciting topic for developing automatic noise detection systems requires further research. Advances in machine learning and artificial intelligence have made it possible to automatically detect and remove noise from systems using more efficient methods. Our goal is to improve software quality assessment by focusing on developing intelligent algorithms that improve noise control and automatically recognize and process noisy events and activities, reducing the need for significant human intervention.

In addition, the active role of noise in software quality dataset may be searched in succeeding study. Noise is an important fact that can alter with the passage of time and influence the accuracy and credibility of classification methods. It might be beneficial to explore dynamic noise control algorithms that revise the shifting noise design. These flexible techniques would constantly pay attention to the dataset recognizing changes in noise levels and to deal with the changing of the classification models. This procedure makes models more suitable for software engineering framework in the real world by verifying their flexibility to changing noise levels. By pursuing these research paths, we may improve software quality dataset's noise reduction capabilities and open the door to the development of classification models that are more precise, dependable, and resilient.

REFERENCES

[1] S. Kim, E. J. Whitehead, and Y. Zhang, “Classifying software changes: Clean or buggy?,” *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, Mar. 2008, doi: 10.1109/TSE.2007.70773.

[2] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, “Supporting Requirements Engineers in Recognising Security Issues,” in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2011, pp. 4–18. Accessed: Jul. 27, 2023. [Online]. Available: https://doi.org/10.1007/978-3-642-19858-8_2

[3] M. Ochodek, R. Hebig, W. Meding, G. Frost, and M. Staron, “Recognizing lines of code violating company-specific coding guidelines using machine learning: A Method and Its Evaluation,” *Empir Softw Eng*, vol. 25, pp. 220–265, Jan. 2020, doi: 10.1007/s10664-019-09769-8.

[4] H. Sajnani, “Automatic software architecture recovery: A machine learning approach,” in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, IEEE, 2012, pp. 265–268. Accessed: Jul. 27, 2023. [Online]. Available: https://doi.org/10.1109/ICPC.2012.6240501

[5] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” in *Proceedings of the 38th International Conference on Software Engineering*, IEEE Computer Society, May 2016, pp. 297–308. doi: 10.1145/2884781.2884804.

[6] Z. Cai, L. Lu, and S. Qiu, "An Abstract Syntax Tree Encoding Method for Cross-Project Defect Prediction," *IEEE Access*, vol. 7, pp. 170844–170853, 2019, doi: 10.1109/ACCESS.2019.2953696.

[7] K. W. Al-Sabbagh, M. Staron, and R. Hebig, "Predicting Test Case Verdicts Using TextualAnalysis of Commited Code Churns," in *International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, Haarlem, Netherlands, 2019, pp. 138–153. Accessed: Jun. 28, 2023. [Online]. Available: https://research.chalmers.se/publication/525483

[8] K. W. Al-Sabbagh, M. Staron, R. Hebig, and W. Meding, "Improving Data Quality for Regression Test Selection by Reducing Annotation Noise," in *Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, pp. 191–194. doi: 10.1109/SEAA51224.2020.00042.

[9] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*, Association for Computing Machinery, 2005, pp. 284–292. Accessed: Jul. 11, 2023. [Online]. Available: https://doi.org/10.1145/1062455.1062514

[10] T. Bin Noor and H. Hemmati, "Studying Test Case Failure Prediction for Test Case Prioritization," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, Association for Computing Machinery, Nov. 2017, pp. 2–11. doi: 10.1145/3127005.3127006.

[11] T. M. Khoshgoftaar, N. Seliya, and K. Gao, "Rule-based noise detection for software measurement data," in *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, IEEE Systems, Man, and Cybernetics Society, 2004, pp. 302–307.

[12] X. Zhu and X. Wu, "Class Noise vs. Attribute Noise: A Quantitative Study," *Artif Intell Rev*, vol. 22, pp. 177–210, 2004, Accessed: Jul. 11, 2023. [Online]. Available: https://doi.org/10.1007/s10462-004-0751-8

[13] F. Muhlenbach, S. Lallich, and D. A. Zighed, "Identifying and Handling Mislabelled Instances," *J Intell Inf Syst*, vol. 22, no. 1, pp. 89–109, 2004, Accessed: Jul. 11, 2023. [Online]. Available: https://doi.org/10.1023/A:1025832930864

[14] T. Zimmermann and P. Weißgerber, "Preprocessing CVS Data for Fine-Grained Analysis," *Proceedings of the First International Workshop on Mining Software Repositories*, 2004, Accessed: Jul. 27, 2023. [Online]. Available: https://www.researchgate.net/publication/228854395_Preprocessing_CVS_data_for_fine-grained_analysis

[15] D. Guan, W. Yuan, Y.-K. Lee, and S. Lee, "Identifying mislabeled training data with the aid of unlabeled data," *Applied Intelligence*, vol. 35, no. 3, pp. 345–358, Dec. 2011, doi: 10.1007/s10489-010-0225-4.

[16] C. M. Teng, "Combining Noise Correction with Feature Selection," in *Data Warehousing and Knowledge Discovery*, 2003, pp. 340–349. Accessed: Jul. 11, 2023. [Online]. Available: https://doi.org/10.1007/978-3-540-45228-7_34

[17] J. D. Van Hulse, T. M. Khoshgoftaar, and H. Huang, “The pairwise attribute noise detection algorithm,” *Knowl Inf Syst*, vol. 11, pp. 171–190, 2007, doi: 10.1007/s10115-006-0022-x.

[18] C. E. Brodley and M. A. Friedl, “Identifying and Eliminating Mislabeled Training Instances,” in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996, pp. 799–802. Accessed: Jul. 10, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/1892875.1892994

[19] T. M. Khoshgoftaar and J. Van Hulse, “Identifying Noise in an Attribute of Interest,” in *Fourth International Conference on Machine Learning and Applications (ICMLA’05)*, 2005. Accessed: Jul. 10, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1607431

[20] K.-A. Yoon and D.-H. Bae, “A pattern-based outlier detection method identifying abnormal attributes in software project data,” *Inf Softw Technol*, vol. 52, no. 2, pp. 137–151, Feb. 2010, doi: 10.1016/j.infsof.2009.08.005.

[21] J. A. Sáez, J. Luengo, and F. Herrera, “Evaluating the classifier behavior with noisy data considering performance and robustness: The Equalized Loss of Accuracy measure,” *Neurocomputing*, vol. 176, pp. 26–35, Feb. 2016, doi: 10.1016/j.neucom.2014.11.086.

[22] M. Balint, R. Marinescu, and T. Girba, “How Developers Copy,” in *14th IEEE International Conference on Program Comprehension (ICPC’06)*, IEEE, 2006, pp. 56–68. doi: 10.1109/ICPC.2006.25.

[23] T. M. Khoshgoftaar and J. Van Hulse, “Identifying noisy features with the Pairwise Attribute Noise Detection Algorithm,” *Intelligent Data Analysis*, vol. 9, no. 6, pp. 589–602, Dec. 2005, doi: 10.3233/IDA-2005-9606.

[24] Anonymous, “Build Prediction in Continuous Integration Using Textual Analysis of Source Code.” Accessed: Jun. 28, 2023. [Online]. Available: https://doi.org/10.5281/zenodo.6784987

[25] K. Al-Sabbagh, M. Staron, and R. Hebig, “Predicting build outcomes in continuous integration using textual analysis of source code commits,” in *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*, New York, NY, USA: ACM, Nov. 2022, pp. 42–51. doi: 10.1145/3558489.3559070.

[26] X. Zhu, X. Wu, and Q. Chen, “Eliminating class noise in large datasets,” in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, 2003, pp. 920–927. Accessed: Jul. 11, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/3041838.3041954

[27] R. A. McDonald, D. J. Hand, and I. A. Eckley, “An Empirical Comparison of Three Boosting Algorithms on Real Data Sets with Artificial Class Noise,” in *International Workshop on Multiple Classifier Systems*, 2003, pp. 35–44. doi: 10.1007/3-540-44938-8_4.

[28] Xingquan Zhu and Xindong Wu, “Cost-Guided Class Noise Handling for Effective Cost-Sensitive Learning,” in *Fourth IEEE International Conference on Data Mining (ICDM’04)*, IEEE, pp. 297–304. doi: 10.1109/ICDM.2004.10108.

[29] Xingquan Zhu and Xindong Wu, “Class Noise Handling for Effective Cost-Sensitive Learning by Cost-Guided Iterative Classification Filtering,” *IEEE Trans Knowl Data Eng*, vol. 18, no. 10, pp. 1435–1440, Oct. 2006, doi: 10.1109/TKDE.2006.155.

[30] X. Zhu, X. Wu, and Q. Chen, “Bridging Local and Global Data Cleansing: Identifying Class Noise in Large, Distributed Data Datasets,” *Data Min Knowl Discov*, vol. 12, no. 2–3, pp. 275–308, May 2006, doi: 10.1007/s10618-005-0012-8.

[31] J. Van Hulse and T. M. Khoshgoftaar, “Class noise detection using frequent itemsets,” *Intelligent Data Analysis*, vol. 10, no. 6, pp. 487–507, Nov. 2006, doi: 10.3233/IDA-2006-10602.

[32] Y. Li, L. F. A. Wessels, D. de Ridder, and M. J. T. Reinders, “Classification in the presence of class noise using a probabilistic Kernel Fisher method,” *Pattern Recognit*, vol. 40, no. 12, pp. 3349–3357, Dec. 2007, doi: 10.1016/j.patcog.2007.05.006.

[33] U. Rebbapragada and C. E. Brodley, “Class Noise Mitigation Through Instance Weighting,” in *European Conference on Machine Learning*, Berlin, Heidelberg:

Springer Berlin Heidelberg, 2007, pp. 708–715. doi: 10.1007/978-3-540-74958-5_71.

[34] D. Anyfantis, M. Karagiannopoulos, S. Kotsiantis, and P. Pintelas, “Robustness of learning techniques in handling class noise in imbalanced datasets,” in *IFIP International Conference on Artificial Intelligence Applications and Innovations*, Boston, MA: Springer US, 2007, pp. 21–28. doi: 10.1007/978-0-387-74161-1_3.

[35] A. Folleco, T. M. Khoshgoftaar, J. Van Hulse, and L. Bullard, “Software quality modeling: The impact of class noise on the random forest classifier,” in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, IEEE, Jun. 2008, pp. 3853–3859. doi: 10.1109/CEC.2008.4631321.

[36] A. Kolcz and G. V. Cormack, “Genre-based decomposition of email class noise,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA: ACM, Jun. 2009, pp. 427–436. doi: 10.1145/1557019.1557070.

[37] B. Sluban, D. Gamberger, and N. Lavra, “Advances in class noise detection,” in *Frontiers in Artificial Intelligence and Applications*, vol. 215, IOS Press, 2010, pp. 1105–1106. doi: 10.3233/978-1-60750-606-5-1105.

[38] C. Catal, O. Alan, and K. Balkan, “Class noise detection based on software metrics and ROC curves,” *Inf Sci (N Y)*, vol. 181, no. 21, pp. 4867–4877, Nov. 2011, doi: 10.1016/j.ins.2011.06.017.

[39] L. P. F. Garcia, A. C. Lorena, and A. C. P. L. F. Carvalho, “A Study on Class Noise Detection and Elimination,” in *2012 Brazilian Symposium on Neural Networks*, IEEE, Oct. 2012, pp. 13–18. doi: 10.1109/SBRN.2012.49.

[40] J. Abellán and A. R. Masegosa, “Bagging schemes on the presence of class noise in classification,” *Expert Syst Appl*, vol. 39, no. 8, pp. 6827–6837, Jun. 2012, doi: 10.1016/j.eswa.2012.01.013.

[41] R. Ramler and J. Himmelbauer, “Noise in Bug Report Data and the Impact on Defect Prediction Results,” in *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, IEEE, Oct. 2013, pp. 173–180. doi: 10.1109/IWSM-Mensura.2013.33.

[42] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Folleco, “An empirical study of the classification performance of learners on imbalanced and noisy software quality data,” *Inf Sci (N Y)*, vol. 259, pp. 571–595, Feb. 2014, doi: 10.1016/j.ins.2010.12.016.

[43] B. Sluban and N. Lavrač, “Relating ensemble diversity and performance: A study in class noise detection,” *Neurocomputing*, vol. 160, pp. 120–131, Jul. 2015, doi: 10.1016/j.neucom.2014.10.086.

[44] J. Du and Z. Cai, “Modelling Class Noise with Symmetric and Asymmetric Distributions,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, Feb. 2015, doi: 10.1609/aaai.v29i1.9612.

[45] W. Feng and S. Boukir, “Class noise removal and correction for image classification using ensemble margin,” in *2015 IEEE International Conference on Image Processing (ICIP)*, IEEE, Sep. 2015, pp. 4698–4702. doi: 10.1109/ICIP.2015.7351698.

[46] J. A. Sáez, B. Krawczyk, and M. Woźniak, “On the Influence of Class Noise in Medical Data Classification: Treatment Using Noise Filtering Methods,” *Applied Artificial Intelligence*, vol. 30, no. 6, pp. 590–609, Jul. 2016, doi: 10.1080/08839514.2016.1193719.

[47] S. Xia, G. Wang, Z. Chen, Y. Duan, and Q. liu, “Complete Random Forest Based Class Noise Filtering Learning for Improving the Generalizability of Classifiers,” *IEEE Trans Knowl Data Eng*, vol. 31, no. 11, pp. 2063–2078, Nov. 2019, doi: 10.1109/TKDE.2018.2873791.

[48] K. W. Al-Sabbagh, R. Hebig, and M. Staron, “The Effect of Class Noise on Continuous Test Case Selection: A Controlled Experiment on Industrial Data,” in *International Conference on Product-Focused Software Process Improvement*, 2020, pp. 287–303. Accessed: Jun. 28, 2023. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-64148-1_18

[49] Z. Nematzadeh, R. Ibrahim, and A. Selamat, “Improving class noise detection and classification performance: A new two-filter CNDC model,” *Appl Soft Comput*, vol. 94, p. 106428, Sep. 2020, doi: 10.1016/j.asoc.2020.106428.

[50] D. Guan *et al.*, “A Novel Class Noise Detection Method for High-Dimensional Data in Industrial Informatics,” *IEEE Trans Industr Inform*, vol. 17, no. 3, pp. 2181–2190, Mar. 2021, doi: 10.1109/TII.2020.3012658.

[51] N. D. Lawrence and B. Schölkopf, “Estimating a Kernel Fisher Discriminant in the Presence of Label Noise,” in *ICML ’01: Proceedings of the Eighteenth International Conference on Machine Learning*, 2001, pp. 306–313.

[52] J. Abellán and S. Moral, “Building classification trees using the total uncertainty criterion,” *International Journal of Intelligent Systems*, vol. 18, no. 12, pp. 1215–1225, Dec. 2003, doi: 10.1002/int.10143.

[53] D. W. Aha, “Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms,” *Int J Man Mach Stud*, vol. 36, no. 2, pp. 267–287, Feb. 1992, doi: 10.1016/0020-7373(92)90018-G.

[54] A. Folleco and T. Khoshgoftaar, “Attribute Noise Detection Using Multi-Resolution Analysis,” *International Journal of Reliability, Quality and Safety Engineering*, vol. 13, no. 03, pp. 267–288, Jun. 2006, doi: 10.1142/S0218539306002252.

[55] T. M. Khoshgoftaar and J. Van Hulse, “Empirical case studies in attribute noise detection,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 39, no. 4, pp. 379–388, 2009, doi: 10.1109/TSMCC.2009.2013815.

[56] M. Mannino, Y. Yang, and Y. Ryu, "Classification algorithm sensitivity to training data with non-representative attribute noise," *Decis Support Syst*, vol. 46, no. 3, pp. 743–751, Feb. 2009, doi: 10.1016/j.dss.2008.11.021.

[57] G. Phillips, "Pairwise attribute noise detection algorithm for detecting noise in surface electromyography recordings," University of New Brunswick, 2016. Accessed: Aug. 08, 2023. [Online]. Available: https://unbscholar.lib.unb.ca/handle/1882/13220

[58] R. I. P. Wickramasinghe, "Attribute Noise, Classification Technique, and Classification Accuracy," in *Data Analytics and Decision Support for Cybersecurity*, 2017, pp. 201–220. doi: 10.1007/978-3-319-59439-2_7.

[59] N. Iam-On, "Clustering data with the presence of attribute noise: a study of noise completely at random and ensemble of multiple k-means Clusterings," *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 3, pp. 491–509, Mar. 2020, doi: 10.1007/s13042-019-00989-4.

[60] A. Petety, S. Tripathi, and N. Hemachandra, "Attribute noise robust binary classification," *CoRR*, Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.07875

[61] L. Yu, X. Huang, and H. Yin, "Can machine learning paradigm improve attribute noise problem in credit risk classification?," *International Review of Economics & Finance*, vol. 70, pp. 440–455, Nov. 2020, doi: 10.1016/j.iref.2020.08.016.

[62] J. A. Sáez and E. Corchado, “ANCES: A novel method to repair attribute noise in classification problems,” *Pattern Recognit*, vol. 121, p. 108198, Jan. 2022, doi: 10.1016/j.patcog.2021.108198.

[63] K. W. Al-Sabbagh, M. Staron, and R. Hebig, “Improving test case selection by handling class and attribute noise,” *Journal of Systems and Software*, vol. 183, Jan. 2022, doi: 10.1016/j.jss.2021.111093.

[64] C. M. Teng, “A Comparison of Noise Handling Techniques,” in *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, 2001. Accessed: Jul. 11, 2023. [Online]. Available: https://aaai.org/papers/flairs-2001-052/

[65] W. Tang and T. M. Khoshgoftaar, “Noise identification with the k-means algorithm,” in *16th IEEE International Conference on Tools with Artificial Intelligence*, IEEE Comput. Soc, 2004, pp. 373–378. doi: 10.1109/ICTAI.2004.93.

[66] T. M. Khoshgoftaar, S. Zhong, and V. Joshi, “Enhancing software quality estimation using ensemble-classifier based noise filtering,” *Intelligent Data Analysis*, vol. 9, no. 1, pp. 3–27, 2005, Accessed: Aug. 07, 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/1239046.1239048

[67] T. M. Khoshgoftaar and P. Rebours, “Improving Software Quality Prediction by Noise Filtering Techniques,” *J Comput Sci Technol*, vol. 22, no. 3, pp. 387–396, 2007, doi: https://doi.org/10.1007/s11390-007-9054-2.

[68] G. A. Liebchen, “Data cleaning techniques for software engineering data sets,” Brunel University, London, 2010. Accessed: Jul. 10, 2023. [Online]. Available: http://bura.brunel.ac.uk/handle/2438/5951

[69] B. Twala, “Reasoning with noisy software effort data,” *Applied Artificial Intelligence*, vol. 28, no. 6, pp. 533–554, Jul. 2014, doi: 10.1080/08839514.2014.923165.

[70] W. Liu, S. Liu, Q. Gu, X. Chen, and D. Chen, “FECS: A Cluster Based Feature Selection Method for Software Fault Prediction with Noises,” in *2015 IEEE 39th Annual Computer Software and Applications Conference*, IEEE, Jul. 2015, pp. 276–281. doi: 10.1109/COMPSAC.2015.66.

[71] A. Nakhaei, M. M. Sepehri, and T. Khatibi, “A Promising Method for Correcting Class Noise in the Presence of Attribute Noise,” *International Journal of Hospital Research*, vol. 12, no. 1, 2023, [Online]. Available: http://ijhr.iums.ac.ir

[72] U. Bin Israr, “Label & Attribute Noise Handling Approaches.” Accessed: Jul. 29, 2024. [Online]. Available: https://github.com/uisrarCSE20/noisehandling

[73] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-29044-2.

APPENDIX

Attribute Noise Removed	Performance Metrics	Random Forest (RF)	
		n-estimator = 100	n-estimator = 300
0%	**Accuracy**	0.972	0.972
	Precision	0.981	0.981
	Recall	0.981	0.981
	F-Score	0.981	0.981
5%	**Accuracy**	0.980	0.980
	Precision	0.985	0.985
	Recall	0.987	0.987
	F-Score	0.986	0.986
10%	**Accuracy**	0.972	0.972
	Precision	0.982	0.982
	Recall	0.980	0.980
	F-Score	0.981	0.981
15%	**Accuracy**	0.972	0.969
	Precision	0.983	0.982
	Recall	0.979	0.976
	F-Score	0.981	0.979
20%	**Accuracy**	0.962	0.962
	Precision	0.979	0.979
	Recall	0.970	0.970
	F-Score	0.974	0.974
25%	**Accuracy**	0.965	0.965
	Precision	0.982	0.982
	Recall	0.972	0.972
	F-Score	0.977	0.977

30%	**Accuracy**	0.969	0.970
	Precision	0.978	0.978
	Recall	0.978	0.980
	F-Score	0.978	0.979
35%	**Accuracy**	0.969	0.969
	Precision	0.984	0.984
	Recall	0.971	0.971
	F-Score	0.978	0.978
40%	**Accuracy**	0.972	0.973
	Precision	0.985	0.985
	Recall	0.973	0.975
	F-Score	0.979	0.980
45%	**Accuracy**	0.956	0.956
	Precision	0.978	0.978
	Recall	0.956	0.956
	F-Score	0.967	0.967
50%	**Accuracy**	0.970	0.970
	Precision	0.988	0.988
	Recall	0.967	0.967
	F-Score	0.978	0.978