Predicting Multiclass Java Code Readability: A Comparative Study of Machine Learning Algorithms

Budi Susanto, Ridi Ferdiana, Teguh Bharata Adji
Department of Electrical and Information Engineering-Faculty of Enginering,
Universitas Gadjah Mada, Indonesia

Abstract—**The classification of program code readability has traditionally focused on two target classes: readable and unreadable. Recently, it has evolved into a multiclass classification task in three categories: readable, neutral, and unreadable. Most of the existing approaches rely on deep learning. This study investigated the multiclass classification of Java code readability using four feature metric datasets and 14 supervised machine learning algorithms. The dataset comprises 200 labeled Java function declarations. Readability features were extracted using Scalabrino's tool, generating three datasets: Scalabrino, Buse-Weimer, a combined set (Dall), and a fourth (Dcorr) via feature selection based on interfeature correlation. Each model underwent hyperparameter tuning via a Randomized Search and was evaluated through 30 iterations of a five-fold cross-validation. Scaling techniques (MinMax, Standard, Robust, and None) were also compared. The best performance, with an average accuracy of 61.1% and minimal overfitting, was achieved by Random Forest with MinMax scaling on Dcorr. Feature importance analysis using permutation methods identified 22 key metrics related to comments: code complexity, syntax, naming, token usage, and density. Despite its moderate accuracy, the findings offer valuable insights and highlight essential features for advancing code readability research.**

Keywords*—*Code readability; machine learning; multiclass classification; hyperparameter tuning; future selection

I. INTRODUCTION

Reading source code is common for software developers, and readability significantly affects the software quality [1]. While readability pertains to the ease of reading code syntax, comprehensibility involves understanding code semantics [2]. These two concepts are related, but distinct, and readability may fluctuate throughout the software lifecycle, thereby affecting comprehensibility [3].

Automated classification methods have been widely used to predict code readability, typically categorizing it as "readable" or "unreadable." Various machine learning techniques, such as Logistic Regression, Bayesian Networks, Perceptron, Random Forest, and Support Vector Machines (SVM), have been applied to classify code readability into two labels, as demonstrated by researchers such as Buse and Weimer [4] and Scalabrino et al. [5].

Posnett et al. [6] developed a regression model for classifying source code readability into two classes. Dorn [7] also utilized logistic regression to determine the weights of various features to construct a readability metric model for source code. Mi et al. [8]–[10] further explored convolutional neural network (CNN)-based architectures and hybrid neural network models for two-class readability classifications.

Recent advancements have extended the readability classification into three categories: "readable," "neutral," and "unreadable" [11]. Mi et al. [11], [12] employed convolutional neural networks (CNNs) and graph neural networks (GCNs) for this task. The GCN model achieved state-of-the-art accuracy, reaching 72.5% for a three-class classification [11]. These models were trained on a Java corpus from Scalabrino et al. [13] containing 200 Java function declaration snippets. Overall, the research findings indicate that deep learning-based methods outperform traditional machine learning approaches in terms of readability classification accuracy.

However, existing comparative studies have predominantly focused on binary readability classification tasks. In contrast, limited attention has been given to the more complex three-class readability classification, which involves categorizing code as "readable," "neutral," or "unreadable." The state-of-the-art model addressing this task proposed by Mi et al. [11] employs a Graph Neural Network (GNN) framework. The research gap lies in the insufficient exploration of alternative machine learning classification approaches for this three-class problem, which may offer competitive or complementary performance to GNN-based methods. In light of this, further empirical investigation is warranted to evaluate the effectiveness of diverse machine learning algorithms in handling multiclass code readability classification.

This study comprehensively evaluated 14 classification methods applied to several code readability metrics: Buse and Weimer (BW), Scalabrino (Scal), and a combination of BW, Scal, and Posnett metrics. To enhance the classification performance, we employed two feature selection methods: one based on feature correlation and its relationship with target classes, and another based on the importance scores derived from the best-performing classification method. The dataset used in this experiment consisted of Java code snippets from Scalabrino et al. [13], which were categorized into three readability classes by Mi et al. [11] with a 1:2:1 ratio for readable, neutral, and unreadable classes. Each classification method was optimized for hyperparameter tuning using a Randomized Search method. Huyen [14] stated that the random search method is a form of soft AutoML.

This study aimed to address two key research questions: RQ1: Which classification method performs best? We investigated 12 classification methods and two calibration techniques for the Gaussian Naïve Bayes classifier to classify the Java corpus from Scalabrino et al. into three readability classes established by Mi et al. The evaluation metrics used included the accuracy and weighted F1 scores. Our analysis aims to provide a comparative overview of the performance of three-class readability classifiers in state-of-the-art models [11]. RQ2: Which features contribute the most to the best-performing classification method? The most important features were based on the best-performing model in the three-class readability classification for each classification method. Identifying these key features will be the foundation for developing improved source code readability metrics.

The remainder of this paper is organized as follows. The background and related works in Section II reviews the related literature, focusing on previous research in program code readability classification, particularly multiclass classification approaches using machine learning techniques. The methodology in Section III outlines the methodology, details the dataset used, classification models applied, and evaluation framework. The results and discussion in Section IV presents the experimental results and discussion of the findings, including a comparative analysis of the classification performance. The conclusion in Section V concludes the study by summarizing key insights, highlighting contributions, and suggesting directions for future research.

II. Background and Related Works

A. Code Readability Metrics

Evaluating code readability is a complex challenge because of its inherently subjective nature. Various metrics that consider factors such as code size, entropy, and structural properties have been proposed [6]. However, these metrics often fail to align with developers' perceptions of readability improvements [15]. Fakhoury et al. [15] utilized Scalabrino's, Dorn's, and combined Buse-Weimer and Posnett metrics to classify code readability based on changes made by programmers. Their findings suggested incorporating additional features, such as the number of incoming invocations and code styling elements, into code readability metrics.

Furthermore, textual features such as identifiers and comments have been shown to provide valuable supplementary information beyond structural characteristics [13]. Other factors, including coding style, application domain [16], structural constraints, and programming paradigms (e.g., reactive programming) [17] also influence readability assessments. Moreover, further evidence is required to clarify the impact of specific attributes, such as code size, complete-word identifiers, and comments, on readability and understandability [18]. These challenges underscore the need for more versatile and adaptive readability metrics from the perspectives of various programming styles and developers.

Predicting the readability of source code involves extracting several features from code snippets, collectively referred to as code readability metrics. Extensive research has been conducted to develop readability metrics. Buse and Weimer [4] focused on structural characteristics, whereas Dorn [7] introduced visual and spatial (metric based) characteristics. Scalabrino et al. [5] extended these metrics by incorporating textual characteristics. Alawad et al. [19] enhanced Buse and Weimer metrics using text readability metrics such as the Automated Readability Index (ARI). Mi et al. [9] defined readability features in terms of visual, structural, and semantic characteristics that are represented as embedding vectors. Additionally, Mi et al. [11] introduced a representation combining Abstract Syntax Tree (AST) graphs with data and control flow extracted from the source code.

Buse and Weimer defined 25 attributes to develop a readability classification model, categorizing readability into two classes: more and less readable. Posnett et al. [6] simplified Buse and Weimer's parameters into three attributes to construct a readability weight-based model. Dorn [7] identified approximately 59 code attributes, focusing on visual, spatial, and linguistic aspects. Scalabrino et al. [5] defined 20 attributes related to the textual properties and structural characteristics. Choi et al. [20] proposed seven attributes for linear regression modeling to derive readability weights for the source code. Mi et al. [21] extracted character-level, token-level, and node-level representations of the source code and applied them within a Convolutional Neural Network (CNN). Mi et al. [9] further advanced this approach by extracting visual codes, tokens, segment embeddings, and character metric representations for structural modeling. Readability classification models have been applied to two primary categories: readable and unreadable.

B. Methods for Measuring Code Readability

Several manual and heuristic methods have been developed to assess the readability of program codes. Although these methods are not explicitly designed for readability measurement, they provide approaches for evaluating code complexity, maintainability, and human-scale readability indicators. Some of these methods include syntactic-based metrics, such as variable length, lines of code (LOC), and cyclomatic complexity, which serve as potential indicators for measuring code readability, and structure and logic-based metrics, which evaluate code readability by analyzing the structural aspects of the code, including loops, branching, and function separation.

Well-structured modular code is generally easier to read and understand. Several types of metrics are commonly used to evaluate readability. Rule-based metrics focus on adherence to coding conventions, such as the use of descriptive variable names, consistent formatting, and sufficient comments [22]. Tools such as Checkstyle[1] and Pylint[2] automatically check and enforce these rules. Complexity metrics assess the readability by measuring the logical complexity of a code. For example, Halstead metrics estimate how difficult it is for a program to understand and maintain, whereas cognitive complexity metrics [23] account for factors such as deep nesting and long conditional statements that may make it harder to follow. Among the complexity metrics and code readability metrics, Tashtoush and Darwish [24] asserted that there is an influence between these two metrics. This empirical study explored the bidirectional relationship between code readability and software complexity by using 12,180 Java files from the Eclipse project. By applying machine learning models, particularly decision trees, to 25 readability features (based on Buse and Weimer) and seven complexity metrics, the study achieved over 90% accuracy in predicting how readability influences complexity and vice versa. Key contributing factors include formatting-related features (e.g., indentation and character usage) and complexity measures such as Halstead volume.

[1]https://checkstyle.sourceforge.io/
[2]https://www.pylint.org/

This study also challenges prior assumptions by showing that readability features are influenced by code size, thereby offering new insights into improving software quality and maintainability.

Finally, human-scale indicators rely on developers direct assessments of code readability. These evaluations are often used as training data for machine learning models that aim to automatically predict code readability [4], [13]. These approaches seek to align readability metrics with how developers perceive the code quality. To capture what programmers do to make source code more readable, Roy et al. [25] analyzed the history of programmer code changes. This study introduces a machine learning model that detects incremental readability improvements in source code aligned with developer perceptions. Trained on 2,781 manually validated Java file changes, the model leveraged static code metrics before and after commits and achieved 79.2% precision and 67% recall. Key insights reveal that readability improvements typically involve modifications to existing lines, whereas non-readability changes add new lines. This study lays the groundwork for integrating readability scoring into code review tools to support more efficient evaluation of code changes. Future research could explore how large language models influence code readability [26] as well as the development of more advanced readability models that adapt to the continual evolution of software development practices.

III. Methodology

Fourteen machine learning algorithms for multiclass classification of code readability were evaluated using a dataset of 200 Java code snippets from Scalabrino et al. [13], which were categorized into three readability classes by Mi, et al. [11]. Mi et al. used a systematic approach to establish three categories of code readability in their study. This study used a dataset originally compiled by Scalabrino et al. [13] consisting of 200 code fragments extracted from four open-source Java projects: jUnit, jHibernate, jFreeChart, and ArgoUML. These fragments ranged from 10 to 50 lines of code and were free of syntax errors, ensuring that the readability evaluation was unaffected by syntax or compilation issues.

The dataset was then divided into three readability groups based on Scalabrino's readability score: easy to read (top 25%), difficult to read (bottom 25%), and neutral (middle 50%). This classification follows a 1:2:1 ratio, allowing for meaningful comparison with Scalabrino's two-class readability distribution. Additionally, this distribution more accurately reflects real-world scenarios, in which neutral readability is more common than extreme readability or legibility.

From these 200 Java code snippets, a set of readability metric values were derived based on the readability metrics proposed by Scalabrino, Buse-Weimer, and Posnett. These metric values were computed using the Readability Assessment Tool[3] provided by Scalabrino et al. [5]. Scalabrino's tool generates 110 attributes representing values from the Scalabrino, Buse-Weimer, Posnett, and Dorn models. Upon analysis, it was found that 12 of Dorn's 59 attributes had over 41.5% missing values across 200 data points. Consequently, Dorn's attributes and readability score attributes from the Scalabrino, Buse-Weimer, and Posnett models were excluded, leaving 48 attributes for the classification experiment. The combined dataset with the 48 selected attributes was labeled as D_{all}.

[3]https://dibt.unimol.it/report/readability/files/readability.zip

TABLE I. Metrics with > 10% Outliers among 200 Samples

No.	Attribute	% outlier data
1	Scalabrino.expression_complexity_maximum	41.0%
2	Scalabrino.number_of_senses_maximum	27.5%
3	BW.loops_average	23.0%
4	BW.comments_average	14.5%
5	Scalabrino.abstractness_words_maximum	14.5%
6	BW.operators_average	13.0%
7	Scalabrino.commented_words_average	11.0%

Based on the definition of D_{all} metric features, this study created another dataset containing the definition of a series of feature metrics based on the scalabrino model (D_{scal} uses 20 feature metrics) and Buse-Weimer model (D_{bw} uses 25 feature metrics). Both feature metric definitions are subsets of the entire metric feature set. The purpose of this is to measure how well the Scalabrino and Buse–Weimer models perform in multiclass code readability classification.

Regarding the characteristics of the D_{all} dataset with 200 data points and 48 attributes when analyzed using the Interquartile Range (IQR), seven metric attributes (14.58%) can be said to have outlier data of more than 10%. Table I displays the seven metric attributes whose data had more than 10% detection as outliers. One approach that can be applied to handle outlier data is data scaling [27]. Therefore, this study applies a configuration with three scaling techniques: Standard, Minmax, and Robust.

Fig. 1 illustrates the workflow of the classification experiment. The process begins with the formation of a dataset using Scalabrino et al.'s Readability Assessment Tool, applied to a corpus of Java code snippets categorized into three readability classes by Mi et al. [11]. This recalculation is necessary to ensure that the attribute values are derived directly from the code snippet dataset, which consists of function declarations written in Java. The tool generates attribute values based on the readability metrics proposed by Scalabrino, Buse, Weimer, and Posnett. The generated values were then converted into a CSV file. The final dataset consists of attribute values and three corresponding readability class labels.

Based on D_{all}, this study calculates the correlation between each attribute, including the target classification label. This feature correlation analysis step was performed to identify highly correlated metric features. Attribute analysis of the combined dataset D_{all} was performed by examining the correlation value between its attributes. Correlation-based attribute selection uses the minimum correlation value as the constraint ($\geq$ 0.4 or $\leq$ -0.4) as a strong correlation between any two attributes. This constraint is based on a study by Diesing [28], which recommends a minimum correlation of 0.4 0.5. The selected attributes with error values ($\geq$ 0.4 or $\leq$ -0.4) were not unique, but duplicate attributes existed. Of the duplicated attributes selected with the highest correlation value, so in the end, we obtained as many as 35 attributes. This resulted in an additional dataset D_{corr} (35 attributes), which contains only strongly correlated features.

Each dataset, that is, D_{all}, D_{scal}, D_{bw}, and D_{corr}, was processed using different data pre-processing scaling techniques, including MinMax, Standard, Robust, and without scaling (none). For each dataset with or without scaling techniques, Randomized Search [29] was employed to optimize the hyperparameters of each of the 14 classification algorithms. Cross-validation was performed using five-fold cross-validation for each validation and testing phase.

For each dataset, before entering the process of building a classification model using the 14 classification algorithms, training and testing sets were formed. From the 200 sets of metrics, we divided the dataset into 80% (160 data points) for training and 20% (40 data points) for testing. We split the datasets for training and testing using the `train_test_split()` function with the stratification parameter to maintain the proportion of classes in the training and testing sets.

The 14 classification algorithms can be broadly categorized based on their underlying approach. Tree-based models, including Random Forest (RF) [30] and Decision Tree (D3) [31] models, are known for their ability to handle nonlinearity in data without requiring feature scaling. Distance-based models, such as k-Nearest Neighbors (KNN) [32] and Support Vector Classifier (SVC) [33], rely heavily on the measurement of feature distances, making them particularly sensitive to scaling transformations. Probabilistic classifiers, including Gaussian Naïve Bayes with Isotonic Calibration (GNB-Isotonic), Gaussian Naïve Bayes with Sigmoid Calibration (GNB-Sigmoid), and Gaussian Naïve Bayes (NB) [34], [35] function under the assumption of feature independence and may benefit from certain types of pre-processing. Linear models, such as Logistic Regression (LogReg) [36], Linear Discriminant Analysis (LDA) [37], [38], Perceptron [39], Perceptron optimization based on Stochastic Gradient Descent (SGD) [39, p. 43-46], and Passive-Aggressive Classifier (PA) [40], assume linear relationships between features and classes and typically require scaling to enhance numerical stability. Bayesian and quadratic models, such as Quadratic Discriminant Analysis (QDA) [41]. Finally, neural network-based methods, such as Multi-Layer Perceptron (MLP) [42], rely on gradient-based optimization, which is highly sensitive to feature magnitudes.

The execution of the Randomized Search for each algorithm was repeated 30 times. Using 30 iterations in a Randomized Search provides a pragmatic balance between computational efficiency, a thorough exploration of the hyperparameter space, and model robustness. This implementation ensured that the best-performing model was selected without incurring excessive computational costs, thus making it a well-justified approach for this classification experiment.

The flowchart (in Fig. 2) illustrates a structured approach for training and selecting an optimal classification model using Randomized Search. The process began by splitting the dataset into 80% training and 20% testing to ensure that unbiased evaluation. Both subsets underwent a scaling transformation to standardize the feature magnitudes, thereby enhancing the model performance. A diverse set of classification algorithms—including SVM, Logistic Regression, k-NN, LDA, QDA, Gaussian Naïve Bayes, Decision Trees, MLP, Random Forest, Perceptron, SGD, and Passive Aggressive classifiers— was prepared for evaluation, employing 5-fold cross-validation to ensure robust assessment. An iterative loop executes Randomized Search for each classifier to optimize the hyperparameters efficiently.

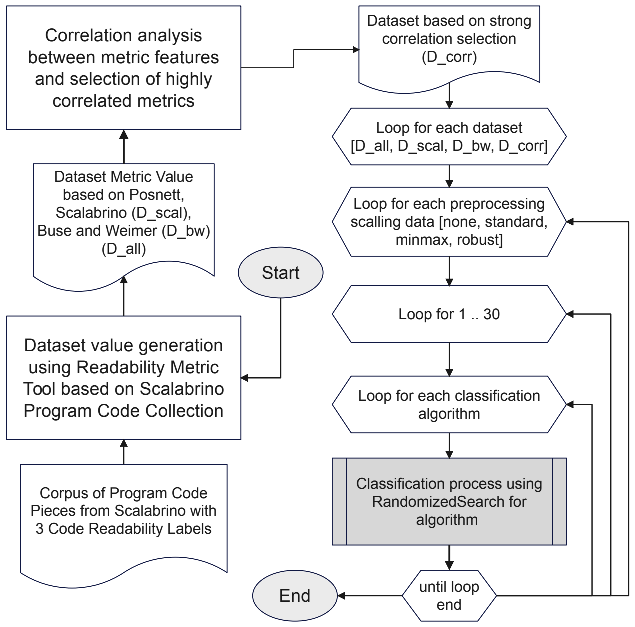
Correlation analysis between metric features and selection of highly correlated metrics
Dataset based on strong correlation selection (D_corr)
Dataset Metric Value based on Posnett, Scalabrino (D_scal), Buse and Weimer (D_bw) (D_all)
Loop for each dataset [D_all, D_scal, D_bw, D_corr]
Loop for each preprocessing scalling data [none, standard, minmax, robust]
Start
Loop for 1 .. 30
Dataset value generation using Readability Metric Tool based on Scalabrino Program Code Collection
Loop for each classification algorithm
Classification process using RandomizedSearch for algorithm
Corpus of Program Code Pieces from Scalabrino with 3 Code Readability Labels
End
until loop end

Fig. 1. The process flow for testing the 3 readability class classification.

Following training, cross-validation scores were measured to assess the generalization ability of the model and detect potential overfitting. Overfitting detection was performed to determine whether the best model generated by the algorithms was overfitted. This study implemented a quantitative method to assess whether a trained machine learning model overfits by comparing its performance on cross-validation and test data. Eq. (1) shows the conditions for detecting overfitting by using the quantitative method applied in this study.

$$\frac{|\text{mean\_cv\_score} - \text{mean\_test\_score}|}{\text{mean\_cv\_score}} > 0.10 \qquad (1)$$

The if condition in the given overfitting condition establishes a threshold-based criterion for detecting overfitting in the trained classification model. It evaluates the relative difference between the mean cross-validation score (mean_cv_score) and test score (test_score). Specifically, the condition checks whether the absolute difference between these scores, normalized by the mean cross-validation score, exceeds 10% (0.10). If this condition holds, the model is considered overfitting, indicating that it performs significantly better on the training data during cross-validation than on the independent test set, suggesting poor generalization. Conversely, if the relative difference remains within the 10% threshold, the model is deemed not to overfit, implying a balanced performance between the training and testing phases. This approach provides a quantitative measure for assessing the generalization ability of a model beyond the training dataset.

The best-performing model, determined based on the cross-validation performance, was then tested on the scaled testing dataset for the final validation. The selected model represents the most effective classification approach for a given dataset,

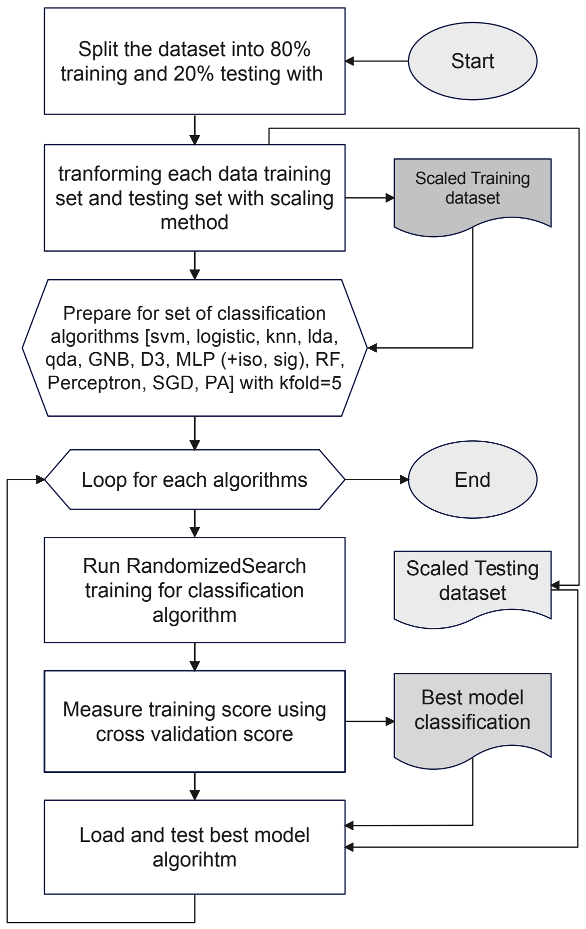
Start
Split the dataset into 80% training and 20% testing with
tranforming each data training set and testing set with scaling method
Scaled Training dataset
Prepare for set of classification algorithms [svm, logistic, knn, lda, qda, GNB, D3, MLP (+iso, sig), RF, Perceptron, SGD, PA] with kfold=5
Loop for each algorithms
End
Run RandomizedSearch training for classification algorithm
Scaled Testing dataset
Measure training score using cross validation score
Best model classification
Load and test best model algorihtm

Fig. 2. The process flow for classification process with parameter optimization.

ensuring optimized accuracy and reliability in classification tasks. This workflow establishes a reproducible methodology for systematic machine learning model selection and evaluation.

IV. Result and Discussion

A. Classification Performance Results

After performing 30 iterations of the classification process for each of the 14 classification algorithms using Randomized Search across four datasets (D_{all}, D_{scal}, D_{bw}, and D_{corr}) with four different scaling configurations (without scaling, standard, min-max, and robust), the average accuracy and weighted F1-score were computed. An analysis was conducted to determine whether the classification model exhibited overfitting by evaluating the difference between the average cross-validation score and the test accuracy for each iteration.

In this study, the best average accuracy was obtained when the overfitting ratio did not exceed 6.67% (i.e., no more than two overfitting occurrences out of 30 iterations). For instance, in the case of the GNB-Isolate method, the best average accuracy was chosen as 0.53 with an overfitting ratio of 0% when using MinMax Scaling, rather than an average accuracy of 0.6 with an overfitting ratio of 100% when using Standard Scaling. Table II presents the results of the selection of the highest average accuracy while ensuring compliance with the overfitting threshold. Fig. 3 presents the optimal average accuracy of 14 classification algorithms under different data-scaling techniques: MinMax, Standard, Robust, and no scaling (none). The performance of each method is depicted using bar plots, in which the highest accuracy values for each classifier are highlighted numerically above the respective bars.

The exploration of 14 classification algorithms optimized using a Randomized Search revealed that the dataset D_{corr} serves as the most optimal alternative for multiclass code readability classification. The D_{corr} dataset comprises a selected set of combined metric features derived from the correlation analysis among the Scalabrino, BW, and Posnett metric groups. Based on the classification results, it can be inferred that the 48 combined metrics could be effectively represented by 35 attributes (approximately 73%).

By analyzing the best average accuracy across all classification algorithms, the dataset distribution based on scaling configurations indicated that D_{corr} contributed to 42.86% (six algorithms) of the best accuracy results among the 14 classification algorithms, followed by D_{bw} at 35.71%, D_{all} at 14.29%, and D_{scal} at 7.14%. However, when factoring in the overfitting constraint, where only one to two occurrences of overfitting are acceptable within 30 iterations, certain algorithms, including MLP, Perceptron, D3, SGD, and PA, cannot be considered optimal for multiclass classification tasks. These algorithms exhibit an overfitting rate exceeding 10% across all scaling configurations, making them less reliable for generalization.

After eliminating these five suboptimal algorithms, the dominance of the D_{corr} dataset in representing the code readability metrics for multiclass classification became even more pronounced, accounting for 35.71% (five algorithms) of the best accuracy results among the remaining classification algorithms. The classifiers that achieve the most optimal performance using the D_{corr} dataset are Random Forest (RF), Support Vector Classifier (SVC), K-Nearest Neighbors (KNN), Gaussian Naïve Bayes with Isotonic Calibration (GNB-Isotonic), and Quadratic Discriminant Analysis (QDA).

Dataset D_{corr} demonstrates that the selection of measurement metrics based on correlated yet relevant attributes can be leveraged optimally using tree- and distance-based classification algorithms. The performance of the Random Forest algorithm (a tree-based model) is particularly effective when applied to datasets in which features exhibit strong correlations with one or more other features. These findings highlight the necessity of considering relevant correlations, whether positive or negative, between readability metrics when constructing a comprehensive set of code-readability metrics. Similarly, the characteristics of distance-based classification algorithms include Support Vector Classification (SVC) (although not entirely distance-based) and k-nearest neighbors (KNN) (which are fully distance-based), allow them to effectively utilize the D_{corr} dataset.

The average classification accuracy of RF, SVC, and kNN significantly benefits from the application of dataset scaling on D_{corr}, particularly with MinMax and Robust scaling. This finding also highlights that the application of scaling techniques can help mitigate the risk of overfitting during

TABLE II. BEST AVERAGE ACCURACY IN MULTICLASS CLASSIFICATION OF CODE READABILITY

Methods	Minmax			Standard			Robust			None			Best	
	Acc	% Over	Data	Acc	% Over	Data	Acc	% Over	Data	Acc	% Over	Data	Acc	Data
RF	**0.611**	3.33%	D_{corr}	0.607	6.67%	D_{corr}	0.601	3.33%	D_{corr}	0.609	3.33%	D_{corr}	**0.611**	D_{corr}
SVC	**0.590**	0%	D_{corr}	0.528	10%	D_{all}	0.565	0%	D_{corr}	0.550	0%	D_{all}	**0.590**	D_{corr}
KNN	0.553	43.33%	D_{all}	0.540	53%	D_{all}	**0.572**	0%	D_{corr}	0.498	56.67%	D_{scal}	0.572	D_{corr}
GNB-Isotonic	**0.570**	0%	D_{corr}	0.530	100%	D_{scal}	0.550	0%	D_{corr}	0.550	0%	D_{corr}	0.570	D_{corr}
QDA	0.500	0%	D_{scal}	0.570	0%	D_{scal}	0.550	0%	D_{corr}	**0.570**	0%	D_{corr}	0.570	D_{corr}
LogReg	0.552	46.67%	D_{bw}	0.536	20%	D_{bw}	0.550	100%	D_{bw}	**0.554**	6.67%	D_{bw}	0.554	D_{bw}
MLP	0.536	33.33%	D_{bw}	0.532	40%	D_{all}	**0.548**	10%	D_{all}	0.528	73.33%	D_{bw}	0.548	D_{all}
Perceptron	0.531	56.67%	D_{bw}	0.492	26.67%	D_{bw}	**0.541**	60%	D_{bw}	0.506	20%	D_{scal}	0.541	D_{bw}
D3	**0.531**	16.67%	D_{bw}	0.524	20%	D_{bw}	0.525	33.33%	D_{bw}	0.517	6.67%	D_{bw}	0.531	D_{bw}
GNB-Sigmoid	**0.530**	0%	D_{scal}	0.500	100%	D_{all}	0.470	0%	D_{corr}	0.500	0%	D_{corr}	0.530	D_{scal}
LDA	**0.530**	0%	D_{bw}	0.530	0%	D_{bw}	0.530	0%	D_{bw}	0.530	0%	D_{bw}	0.530	D_{bw}
SGD	0.521	20%	D_{bw}	**0.525**	33.33%	D_{bw}	0.520	46.67%	D_{bw}	0.474	33.33%	D_{bw}	0.525	D_{bw}
PA	0.493	30%	D_{bw}	**0.506**	43.33%	D_{corr}	0.492	23.33%	D_{all}	0.447	30%	D_{scal}	0.506	D_{corr}
NB	0.420	0%	D_{corr}	0.400	0%	D_{corr}	**0.450**	0%	D_{all}	0.420	0%	D_{corr}	0.450	D_{all}

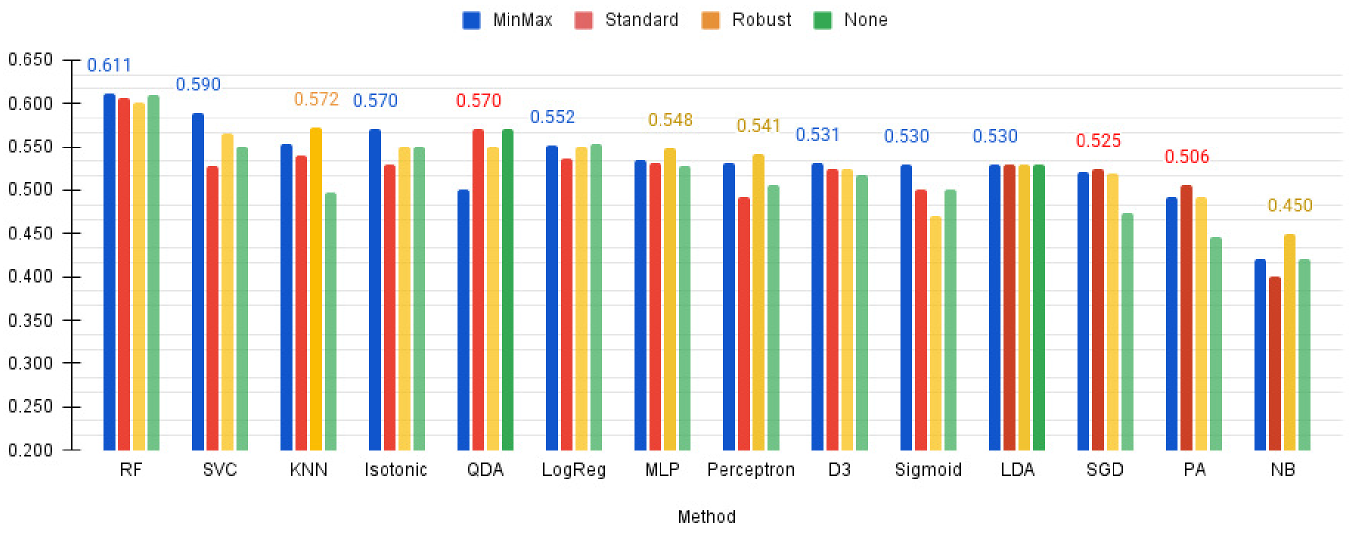

Fig. 3. Best average accuracy results for multiclass code readability classification (k-fold = 5).

multiclass classification model development. However, not all multiclass classification algorithms perform optimally even when utilizing D_{corr} with scaling. In this study, the Passive-Aggressive (PA) algorithm exemplifies this limitation. Because the PA is inherently designed for binary classification, its performance in multiclass classification is suboptimal. This is evident from the fact that 43.3% of the classification models generated over 30 iterations using the PA exhibited overfitting.

In addition to dataset D_{corr}, the use of dataset D_{bw}, which consists of metric features defined by Buse and Weimer [4], does not yield an optimal performance in multiclass code readability classification. The highest average accuracy achieved was 55.4% using the Logistic Regression algorithm without data scaling and 53% using the Linear Discriminant Analysis (LDA) algorithm. By aligning the average accuracy results with the overfitting percentage from 30 iterations of model training, both the Logistic Regression and LDA demonstrated the ability to produce multiclass classification models with an acceptable level of overfitting. These findings suggest that the D_{bw} dataset is more suitable for classification models in which the decision boundary formulation is based on a linear function. Conversely, the Perceptron, Decision Tree (D3), and Stochastic Gradient Descent (SGD) algorithms failed to achieve optimal performance, as this exploration indicates that all three algorithms exhibit an overfitting rate exceeding 6.67%.

Dataset D_{all} contains the largest number of metric features, as it integrates the metrics proposed by Scalabrino, Buse, and Weimer (BW) and Posnett. Among the evaluated multiclass classification algorithms, multilayer perceptron (MLP) and Gaussian Naïve Bayes (NB) demonstrated the most optimal utilization of this dataset. The optimization of the average accuracy of these algorithms is primarily influenced by the application of Robust scaling to D_{all}. However, the average accuracy of MLP did not fully satisfy the overfitting threshold of less than 6.67% because three iterations still exhibited overfitting. Conversely, the NB algorithm achieved an accuracy of 0.45 on D_{all}, making it the least effective among the evaluated algorithms. The fourth dataset, D_{scal}, comprised 20 readability metrics derived from the model proposed by Scalabrino et al. [5]. The only classification algorithm that effectively utilizes this dataset is the Gaussian Naïve Bayes with a sigmoid activation function (GNB-sigmoid), particularly when MinMax scaling is applied. The results from 30 classification trials using

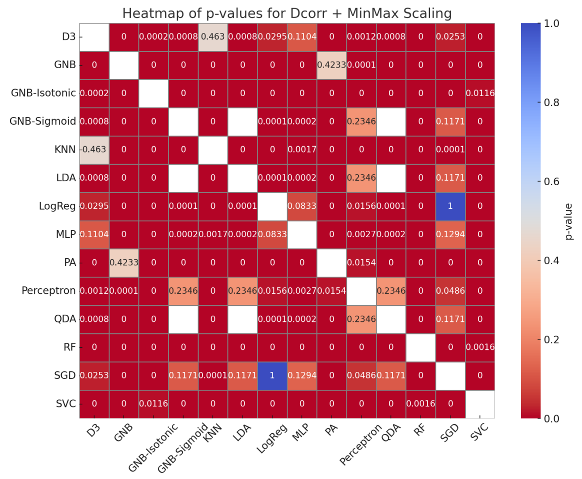

Fig. 4. Heatmap of p-Values for D_{corr} with minmax scaling.

GNB-Sigmoid showed no signs of overfitting, regardless of whether MinMax, Robust, or scaling was applied.

The average accuracy results obtained by utilizing the dataset with various scaling configurations, as presented in Table II, highlight the significance of data scaling in influencing a dataset. Scaling serves as a crucial pre-processing step before building a multiclass classification model for code readability. However, the findings of this study indicate that the impact of scaling is not consistently definitive in yielding superior average accuracy. The effectiveness of scaling techniques depends on the classification model employed. Specifically, MinMax scaling was optimally utilized by the RF, SVC, GNB-isotonic, GNB-sigmoid, and LDA algorithms. Meanwhile, Standard scaling enhances the performance of QDA without causing overfitting. Although Robust scaling can be beneficial, it can lead to overfitting in some cases. The KNN and NB algorithms can leverage Robust scaling to optimize their performance without overfitting. These findings underscore the importance of selecting an appropriate scaling strategy that aligns with the fundamental assumptions of classifiers and their sensitivity to the distribution of metric-based dataset features, particularly for code readability in multiclass classification.

As part of the validation stage, in addition to applying rule-based overfitting checking and 5-fold cross validation, this study also conducted Statistical Significance Testing (Shapiro-Wilk test, paired t-test, or Wilcoxon singed-rank test) to validate the superiority of the selected models. Fig. 4 shows the p-value heatmap of the significance test results between the algorithms in the best configuration, namely D_{corr} with MinMax scaling. Based on the p-value heatmap, it can be stated that Random Forest is consistently superior to the other algorithms because it shows a truly significant difference, not by chance.

Based on the accuracy of the results, several key findings answered the first research question (RQ1). Random Forest (RF) is the best-performing classification method for multiclass code readability classification, as it achieves the highest accuracy across most datasets and remains stable across different scaling techniques. SVC also performs well but is more sensitive to feature scaling, with MinMax scaling being the most beneficial. These findings suggest that ensemble methods such as Random Forest are more effective for code readability multiclass classification, particularly when feature selection is applied (as in D_{corr}).

Overall, dataset analysis underscores the importance of choosing the correct data pre-processing strategy based on the nature of the classifier. Feature correlation (D_{corr}) appears to enhance the accuracy of tree- and distance-based models, whereas balanced weighting (D_{bw}) benefits linear models. Complete feature sets (D_{all}) are favorable for neural networks and probabilistic models, whereas scaling transformations (D_{scal}) offer a limited but occasionally beneficial effect on certain classifiers.

B. *Attribute Role Analysis Based on Importance Score*

This section outlines the process of analyzing the contribution of attributes (features) from the D_{corr} dataset, which are critical for classifying source code readability into three classes. The analysis was conducted using the correlated dataset D_{corr} with MinMax scaling and the Random Forest classification model because this configuration demonstrated the highest accuracy in the conducted trials. The D_{corr} dataset comprises 35 attributes derived from the combined features proposed by Scalabrino, Buse-Weimer (BW), and Posnett.

Attribute contribution analysis was performed by computing the average `importance_mean` for each attribute exclusively in the D_{corr} dataset. The classification model that achieved the highest performance, which employed D_{corr} with MinMax scaling, was evaluated using the test data generated during the classification assessment process. The attribute importance weights were computed using the `permutation_importance` function from the `sklearn.inspection`. Attribute selection is based on the average importance score (mean_threshold) and standard deviation (std_threshold) of the highest `importance_mean` values, with selection criteria determined by predefined thresholds: a minimum average error threshold of [0.005, 0.01, and 0.02] and a maximum standard deviation error threshold of [0.015, 0.02]. The average (mean) was used to determine the overall contribution of the features (attributes) to the model performance based on 30 iterations. An average value greater than zero (> 0) indicates that the attribute consistently positively contributes to model performance. Conversely, if the average is approximately zero (≈ 0), then the attribute is likely to be irrelevant.

Similarly, if the average is negative (< 0), the attribute is likely to negatively impact the model performance. In addition to the mean threshold, the standard deviation of the `permutation_importance` values was considered. The standard deviation measures the variation (spread) in an attribute's importance score across the iterations. This variation reflects the consistency of the impact of the features on the model performance. A low standard deviation indicates that the contribution of the feature remained stable across all 30 iterations, whereas a high standard deviation suggests that the importance of the attribute is inconsistent, implying that its influence may vary, at times being beneficial, unimportant, or even detrimental.

TABLE III. AVERAGE BEST ACCURACY OF MULTICLASS CLASSIFICATION USING Ds_1, Ds_2, Ds_3, Ds_4, Ds_5, AND Ds_6

Method	Avg Accuracy	Overfitting %	Dataset	Scaling
LDA	0.550	0.0%	Ds_4	MinMax
	0.550	0.0%	Ds_4	Standard
	0.550	0.0%	Ds_4	Robust
	0.550	0.0%	Ds_4	None
RF	**0.583**	**0.0%**	Ds_6	**Robust**
KNN	**0.590**	**0.0%**	Ds_2	**Robust**
SVC	0.517	0.0%	Ds_1	MinMax
	0.545	0.0%	Ds_3	Standard
	0.579	0.0%	Ds_5	Standard

The next step involved constructing a dataset in which D_{corr} served as the baseline, with attributes selected based on predefined thresholds. Attribute selection was performed according to threshold conditions for the mean and standard deviation, resulting in six distinct attribute groups. For each of these groups, a corresponding dataset was derived from the source dataset D_{corr}. Consequently, six new datasets were generated, each based on different attribute selections: Ds_1 (11 attributes), Ds_2 (22 attributes), Ds_3 (8 attributes), Ds_4 (17 attributes), Ds_5 (16 attributes), and Ds_6 (27 attributes).

Each dataset formed through this selection process was subsequently subjected to classification trials along with similar trials conducted for the D_{all}, D_{scal}, D_{bw}, and D_{corr} datasets. The primary objective of these tests was to identify the attributes from D_{corr} that contributed the most significantly to the best classification accuracy, thereby providing a foundation for developing alternative attributes or metrics for program code readability models. In addition, this analysis aimed to evaluate whether the six datasets resulting from attribute selection produced better classification performance than the D_{all}, D_{scal}, D_{bw}, and D_{corr} models.

The best-performing models were selected based on an overfitting threshold $\leq 6.67\%$ across 30 iterations for each dataset. The selection results, as shown in Table III, indicate that the K-Nearest Neighbors (KNN) algorithm achieves the best performance when applied to the Ds_2 dataset compared to other datasets. KNN achieved the highest average accuracy (59%), followed by Random Forest (RF) at 58.3%, Support Vector Classifier (SVC) at 57.9%, and Linear Discriminant Analysis (LDA) at 55%. The Ds_2 dataset (containing 22 metric attributes) is better suited for instance-based learning algorithms, such as KNN, whereas Ds_6 (27 attributes) is more effective for decision tree-based algorithms, particularly Random Forest. However, the highest average performance results obtained from these six newly formed datasets remained suboptimal compared with those achieved with the original D_{corr} dataset.

The primary objective of the classification experiment using the six datasets, Ds_1, Ds_2, Ds_3, Ds_4, Ds_5, and Ds_6, was to identify which set of feature attributes (metrics) played a significant role in classification. Based on the results of this study, 22 feature attributes from dataset Ds_2 were identified as key metrics for measuring code readability. Table IV provides a detailed overview of the 22 feature attributes in the program code, including their corresponding metric categories.

By selecting 22 readability metric features from the feature set in dataset D_{corr}, this finding also addressed the second research question (RQ2). Among the Scalabrino metrics, eight key metrics play a significant role in multiclass code readability classification. For the Buse and Weimer metrics, 12 features were identified as being important for multiclass classification. Based on Posnett's metrics, two features were found to contribute significantly to the code readability classification experiment. Thus, based on the results of the multiclass code readability classification, it can be concluded that comment text readability, code complexity and structure, syntax and formatting, identifier naming and token usage, and code size and density are crucial factors in classifying Java source code into three readability categories: unreadable, neutral, and readable.

C. Discussion

The performance results of multiclass classification based on 14 machine learning algorithms, although not optimal, show that the utilization of the definition of the code readability metric feature set, particularly the result of selecting features based on their correlation, can still be an alternative in multiclass classification of code readability. This study offers a practical and interpretable alternative based on the code readability metric features used in classification compared with the application of deep learning. Deep learning often requires significant computational resources and lacks transparency. The establishment of a feature metric based on the correlation result D_{corr} that can be utilized by Random Forest to produce consistent performance can still be used to explain the model through the importance of the code readability feature. This makes machine learning based on metric features more suitable for applications that require interpretation and positions the machine learning framework as a complementary method to more complex deep learning approaches.

The different average accuracy performance results among the dataset definitions D_{all}, D_{scal}, D_{bw}, and D_{corr} can be explained by variations in feature composition and selection strategies. D_{all} integrates features from Scalabrino, Buse-Weimer, and Posnett, offering broader coverage but potential redundancy. In contrast, D_{scal} and D_{bw} focused on defining a specific set of metric features according to their respective models. By contrast, D_{corr}, which is formed from a correlation-based selection of D_{all}, can be said to be a metric feature selection that minimizes redundancy and retains only highly relevant features.

These differences affect the performance of the 14 machine learning classification algorithms. For example, tree-based models, such as Random Forest with D_{corr}, can effectively handle correlated features. Distance-based models, such as SVC and KNN, also perform well on D_{corr} when proper scaling (e.g., MinMax, Robust) is applied because distance-based model algorithms are sensitive to feature magnitude and distribution. Linear models such as Logistic Regression and LDA showed better performance against D_{bw} utilization, which seems to be more in line with the linear separability assumption. Probabilistic models, such as the calibrated Gaussian Naïve Bayes, perform reasonably well with D_{scal} and D_{corr} when scaling is used to reduce the risk of overfitting.

To optimize the readability interpretation based on the Scalabrino, Buse-Weimer, and Posnett readability metric fea-

TABLE IV. 22 Selected "Code Readability" Metrics

Metric	Category	Source
Scalabrino.commented_words_average	Comment and Text-Based Readability	Scalabrino
Scalabrino.synonym_commented_words_average		
Scalabrino.synonym_commented_words_maximum		
Scalabrino.comments_readability		
Scalabrino.semantic_text_coherence_standard		
Scalabrino.expression_complexity_average	Code Complexity and Structure	
Scalabrino.method_chains_average		
Scalabrino.method_chains_maximum		
BW.commas_average	Syntax and Formatting	Buse & Weimer
BW.indentation_average		
BW.periods_average		
BW.spaces_average		
BW.indentation_maximum		
BW.comments_average	Identifiers and Token-Based Complexity	
BW.identifiers_length_average		
BW.number_of_identifiers_average		
BW.number_of_identifiers_maximum		
BW.numbers_maximum		
BW.char_maximum		
BW.words_maximum		
Posnett.volume	Code Size and Information Theory	Posnett
Posnett.lines		

tures, in this study, feature selection based on the permutation importance method was performed on the best Random Forest model applied to D_{corr} dataset with MinMax scaling. The results of the selection form six combinations of metric features, and then choose which combination is the best based on multiclass classification tests on each of the six combinations of features. Of the 35 features available from D_{corr}, 22 metric features were selected because they provided consistent positive contributions and low variance in the average accuracy of the multiclass classification model. These features included comment readability, code complexity, syntax, identifier naming, and information density.

V. Conclusion

A multiclass classification study for machine learning-based code readability was conducted by utilizing metric feature definitions from the Scalabrino, Buse and Weimer, and Posnett models. Utilization of soft AutoML hyperparameter tuning, namely Randomized Search, produced an optimal multiclass classification model based on 200 Java codes from Scalabrino et al. The set with 35 metric features resulting from correlation-based future selection (forming the D_{corr} dataset) consistently exhibited the highest average accuracy and a weighted F1 score. The Random Forest algorithm provides the highest average accuracy among the algorithms with or without utilizing MinMax, Standard, and Robust scaling transformation techniques on the data of each readability label with minimal overfitting conditions. The validity test based on Statistical Significance Testing of the classification performance results also shows that the RF algorithm is consistently and significantly superior to the other classification algorithms.

In this study, the most important feature metric was extracted using a permutation importance function based on the results of the previous best classification model. From the resulting six combinations, 22 out of 35 D_{corr} metric features play an important role in the multiclass classification of code readability. Metric features include comment readability, code complexity, syntax, naming, and density. Overall, the average multiclass classification accuracy results generated in this study could not surpass the 72.5% accuracy of the GNN model proposed by Mi et al. Thus, in future research, it will be necessary to refine the definition of code readability metric features to better represent the code readability metric. In addition, the utilization of hybrid machine learning methods for multiclass classification of code readability can be explored to obtain better performance than the application of machine learning algorithms.

Authors' Contributions

Conceptualization, BS, RF, TBA; methodology, BBS, RF, TBA; validation, BS; investigation, BS; resources, BS; data curation, BS; writing—original draft preparation, BS; writing—reviewing and editing, BS, RF, TBA; visualization, BS; supervision, RF, TBA; project administration, BS, RF, TBA; funding acquisition, RF, TBA.

Acknowledgment

This research is supported by the 2024 Doctoral Dissertation Research Grant number 2773/UN1/DITLIT/PT.01.03/2024 from the Directorate General of Higher Education, Research, and Technology, Ministry of Education, Culture, Research and Technology of the Republic of Indonesia.

References

[1] S. E. Sorour, H. E. Abdelkader, K. M. Sallam, R. K. Chakrabortty, M. J. Ryan, and A. Abohany, "An analytical code quality methodology using latent dirichlet allocation and convolutional neural networks," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 8, Part B, pp. 5979–5997, Sep. 2022.

[2] C. E. C. Dantas and M. A. Maia, "Readability and understandability scores for snippet assessment: An exploratory study," in *Anais do IX Workshop de Visualização, Evolução e Manutenção de Software (VEM 2021)*. Sociedade Brasileira de Computação - SBC, Sep. 2021, pp. 46–50.

[3] V. Piantadosi, F. Fierro, S. Scalabrino, A. Serebrenik, and R. Oliveto, "How does code readability change during software evolution?" *Empirical Software Engineering*, vol. 25, no. 6, pp. 5374–5412, Nov. 2020.

[4] R. P. L. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Trans. Software Eng.*, vol. 36, no. 4, pp. 546–558, 2010.

[5] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk, "A comprehensive model for code readability," *J. Softw. (Malden)*, vol. 30, no. 6, pp. 1–23, Jun. 2018.

[6] D. Posnett, A. Hindle, and P. Devanbu, "A simpler model of software readability," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: Association for Computing Machinery, May 2011, pp. 73–82.

[7] J. Dorn, "A general software readability model," https://citeseerx.ist.psu.edu/pdf/e24f9095a15f30f45cd4b23e84c5abe2f0095a17, 2012, accessed: 2023-10-22.

[8] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and X. Mei, "An inception architecture-based model for improving code readability classification," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. New York, NY, USA: ACM, Jun. 2018, pp. 139–144.

[9] Q. Mi, Y. Hao, L. Ou, and W. Ma, "Towards using visual, semantic and structural features to improve code readability classification," *J. Syst. Softw.*, vol. 193, pp. 1–11, Nov. 2022.

[10] Q. Mi, Y. Xiao, Z. Cai, and X. Jia, "The effectiveness of data augmentation in code readability classification," *Information and Software Technology*, vol. 129, pp. 1–4, Jan. 2021.

[11] Q. Mi, Y. Zhan, H. Weng, Q. Bao, L. Cui, and W. Ma, "A graph-based code representation method to improve code readability classification," *Empirical Software Engineering*, vol. 28, no. 4, pp. 1–26, May 2023.

[12] Q. Mi, L. Wang, L. Hu, L. Ou, and Y. Yu, "Improving multi-class code readability classification with an enhanced data augmentation approach (130)," *Int. J. Software Engineer. Knowledge Engineer.*, vol. 32, no. 11n12, pp. 1709–1731, Nov. 2022.

[13] S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Improving code readability models with textual features," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, May 2016, pp. 1–10.

[14] C. Huyen, "Designing machine learning systems: An iterative process for production-ready applications," Sebastopol, CA, pp. 173–174, May 2022.

[15] S. Fakhoury, D. Roy, A. Hassan, and V. Arnaoudova, "Improving source code readability: Theory and practice," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, May 2019, pp. 2–12.

[16] M. Akour and B. Falah, "Application domain and programming language readability yardsticks," in *2016 7th International Conference on Computer Science and Information Technology (CSIT)*. IEEE, Jul. 2016, pp. 1–6.

[17] G. Holst and F. Dobslaw, "On the importance and shortcomings of code readability metrics: A case study on reactive programming," *arXiv [cs.SE]*, Oct. 2021.

[18] T. V. Ribeiro and G. H. Travassos, "Attributes influencing the reading and comprehension of source code – discussing contradictory evidence," *CLEI Electron Journal*, vol. 21, no. 1, pp. 5:1–5:33, Apr. 2018.

[19] D. Alawad, M. Panta, M. Zibran, and M. R. Islam, "An empirical study of the relationships between code readability and software complexity," in *27th International Conference on Software Engineering and Data Engineering (SEDE)*, F. C. Harris, Jr, S. Sharma, and S. Dascalu, Eds. International Society for Computers and Their Applications, Aug. 2019, pp. 122–127.

[20] S. Choi, S. Kim, J.-H. Lee, J. Kim, and J.-Y. Choi, "Measuring the extent of source code readability using regression analysis," in *Computational Science and Its Applications – ICCSA 2018*. Springer International Publishing, 2018, pp. 410–421.

[21] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and Y. Gao, "Improving code readability classification using convolutional neural networks," *Information and Software Technology*, vol. 104, pp. 60–71, Dec. 2018.

[22] M. Motwani and Y. Brun, "Better automatic program repair by using bug reports and tests together," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1225–1237.

[23] G. A. Campbell, "Cognitive complexity: an overview and evaluation," in *Proceedings of the 2018 International Conference on Technical Debt*, ser. TechDebt '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 57–58.

[24] Y. Tashtoush, N. Abu-El-Rub, O. Darwish, S. Al-Eidi, D. Darweesh, and O. Karajeh, "A notional understanding of the relationship between code readability and software complexity," *Information*, vol. 14, no. 2, pp. 81:1–81:26, Jan. 2023.

[25] D. Roy, S. Fakhoury, J. Lee, and V. Arnaoudova, "A model to detect readability improvements in incremental changes," in *Proceedings of the 28th International Conference on Program Comprehension*. New York, NY, USA: ACM, Jul. 2020, pp. 25–36.

[26] N. Al Madi, "How readable is model-generated code? examining readability and visual inspection of GitHub copilot," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, Oct. 2022, pp. 205:1–205:5.

[27] X. H. Cao, I. Stojkovic, and Z. Obradovic, "A robust data scaling algorithm to improve classification accuracies in biomedical data," *BMC Bioinformatics*, vol. 17, no. 1, pp. 359:1–359:10, Sep. 2016.

[28] E. G. Ávalos, "Interactive comment on "deep-sea sediments of the global ocean" by markus diesing," May 2020, accessed: 2023-10-22. [Online]. Available: https://essd.copernicus.org/preprints/essd-2020-22/essd-2020-22-RC1.pdf

[29] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[30] J. Schlenger, "Random forest," in *Computer Science in Sport*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2024, pp. 201–207.

[31] H. Li, "Decision tree," in *Machine Learning Methods*. Singapore: Springer Nature Singapore, 2024, pp. 77–102.

[32] H. L, "K-nearest neighbor," in *Machine Learning Methods*. Singapore: Springer Nature Singapore, 2024, pp. 55–66.

[33] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, Apr. 2011.

[34] R. D. S. Raizada and Y.-S. Lee, "Smoothness without smoothing: why gaussian naive bayes is not naive for multi-subject searchlight studies," *PLoS One*, vol. 8, no. 7, pp. e69 566:1–e69 566:10, Jul. 2013.

[35] M. Schonlau, "The naive bayes classifier," in *Applied Statistical Learning: With Case Studies in Stata*. Cham: Springer International Publishing, 2023, pp. 143–160.

[36] H. Li, "Logistic regression and maximum entropy model," in *Machine Learning Methods*. Singapore: Springer Nature Singapore, 2024, pp. 103–125.

[37] C. Gambella, B. Ghaddar, and J. Naoum-Sawaya, "Optimization problems for machine learning: A survey," *Eur. J. Oper. Res.*, vol. 290, no. 3, pp. 807–828, 2021.

[38] S. Zhao, B. Zhang, J. Yang, J. Zhou, and Y. Xu, "Linear discriminant analysis," *Nat. Rev. Methods Primers*, vol. 4, no. 1, pp. 1–16, Sep. 2024.

[39] H. Li, "Perceptron," in *Machine Learning Methods*. Singapore: Springer Nature Singapore, 2024, pp. 39–53.

[40] S. Shalev-Shwartz, "Online passive-aggressive algorithms," *Journal of Machine Learning Research*, vol. 7, pp. 551–585, 2006.

[41] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Twenty-first international conference on Machine learning - ICML '04*. New York, New York, USA: ACM Press, 2004, pp. 116–124.

[42] R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher, "Multi-layer perceptrons," in *Texts in Computer Science*, ser. Texts in computer science. Cham: Springer International Publishing, 2022, pp. 53–124.