Understanding Regression Models on Stack Overflow Code: GBM Returns the Best Prediction Performance among Regression Techniques

Sherlock A. Licorish[1*], Brendon Woodford[1], Lakmal Kiyaduwa Vithanage[1], and Osayande Pascal Omondiagbe[2]

[1] School of Computing, University of Otago, Dunedin, Otago, New Zealand.
[2] Research Computing Services, University of Melbourne, 3010, Melbourne, Victoria, Australia.

*Corresponding author. Tel.: +643 479 8319; email: sherlock.licorish@otago.ac.nz
Manuscript submitted February 17, 2025; revised April 3, 2025, accepted May 8, 2025; published July 9, 2025
doi: 10.17706/jsw.20.2.51-65

Abstract: Practitioners are often dependent on Stack Overflow code during software development, where poor quality is occasionally reported. Research tends to focus on ranking content, identifying defects and predicting future content, but less attention is dedicated to identifying the most suitable techniques for modelling/prediction. Contextualizing the Stack Overflow code quality problem as regression-based, we examined the variables that predict Stack Overflow (Java) code quality, and the regression approach that provides the best predictive power. We observed answer count ($\beta = 0.138$), code length ($\beta = 0.382$), code spaces ($\beta = 0.099$) and lines of code ($\beta = 1.959$) as the strongest predictors of code quality on Stack Overflow. Six regression approaches were considered in our evaluation, where Gradient Boosting Machine (GBM) achieved superior performance (RMSE = 2.77, $R^2 = 0.99$, MAE = 0.79) compared to other methods including eXtreme Gradient Boosting (XGBoost) (RMSE = 3.12, $R^2 = 0.97$, MAE = 2.36), and Classification and Regression Trees (CART) (RMSE = 3.45, $R^2 = 0.96$, MAE = 1.77). In fact, even when evaluated against Deep Neural Networks (DeepNN), GBM's superior performance is maintained. Follow-up evaluations using two independent datasets on Electrical Grid Stability and USA Cancer Mortality confirm GBM's superior performance, supporting claims for generalizability of our findings. Outcomes here point to the value of the GBM ensemble learning mechanism and need for continued modelling techniques' experimentation.

Keywords: evaluation study, regression methods, stack overflow, code quality, electrical grid stability, USA cancer mortality.

1. Introduction

Practitioners use platforms such as Stack Overflow and public software code repositories to support their software development efforts [1, 2]. In recent years Large Language Models (LLMs) are also used, albeit the coding data used for training these models are typically taken from the same online code repositories [3]. While these online code platforms provide utility for the software development community, studies have shown that they can present content with questionable quality at times, which may lead to immediate software errors and long term defects [4], and these errors also propagate to code generated by LLMs [3].

Thus, a growing body of research is dedicated to studying these platforms and providing mechanisms for detecting errors and improving the quality of content that is available to practitioners online [5–8].

Researchers have also developed various datasets to support research efforts, where the academic and practitioner communities are able to experiment with novel modelling techniques [9]. Accordingly, research efforts have been dedicated to ranking the content on online platforms [10], identifying particular defects online [11] and predicting future content [12]. Various degrees of precision, recall and F1-score have been reported by these works, pointing to the potential utility of the modelling approaches that are explored.

However, there is relatively little effort devoted to comparing the actual techniques used. In many cases, approaches are presented as fit for purpose, albeit only one dataset and one method are used for experimentation. While the outcomes presented by these studies help us to understand the usefulness of the approaches that are used and contribute to our understanding of the state of the art [13], there is need to properly evaluate the methods that are available for prediction. In the context of Stack Overflow code quality, we know that the quality of content on this portal varies [4], but there is little insights available into the specific variables that influence the errors that are observed. This lack of insights is derived due to limited variable range in datasets [10–12], however, it does not help that evaluation experiments are scarce. In fact, as noted above, limited experimentation is performed to understand how various methods perform. This work addresses this gap in providing insights into Stack Overflow code quality and the performance of multiple regression models, before further evaluating the best model against DeepNN. The novelty of the framework that we propose is generalizable to analyse code from other software repositories leveraging regression methods other than the ones described in this work.

Our contributions are summarized as follows: (1) we identify suitable existing regression approaches to predict the quality of Stack Overflow code, (2) we adopt hyper-parameter optimization to evaluate the relative performance of the candidate regression models to select the optimal regression models, (3) through rigorous statistical analysis we identify the better performing regression model and from it those variables/features which significantly influence Stack Overflow code quality, and (4) using two other unrelated regression data sets from different domains we show that the better performing regression model is able to outperform competing regression models and DeepNN and is generalizable to different regression datasets, thus validating its selection.

The remaining sections of this study are organized as follows. In the next section we review the literature and provide our methods in Section 3. Results are subsequently provided in Section 4. We discuss the findings and outline implications in Section 5, before providing concluding remarks, future work and limitations in Section 6.

2. Literature Review

Studies have revealed promising outcomes in the use of various approaches to model Stack Overflow data. For instance, on the backdrop that 29% of the questions on Stack Overflow do not have an answer [12] and 47% of the questions are not resolved [14], Mondal *et al.* [12] tried to detect questions that were likely to go unanswered using four machine learning models, where 79% accuracy was achieved using the Random Forest (RF) approach. Yazdaninia *et al.* [14] also proposed to predict questions that are not likely to attract answers at the time of submission where several machine learning models were evaluated. Of the approaches used, XGBoost achieved 71% accuracy [14]. Tavakoli *et al.* [15] proposed to automatically identify deficient Stack Overflow posts where the J48 decision tree algorithm was used to achieve 94.5% precision and 90.3% recall. Gyimóthy *et al.* [16] used deep learning and Natural Language Processing (NLP) techniques to train a model with linguistic and semantic features to detect Stack Overflow posts for closure, where 74% accuracy was reported. Similarly, classical machine learning methods have been used for quality assessment or prediction. Alikhashashneh *et al.* [17] compared the performance of Support Vector Machine (SVM), K-Nearest Neighbor (KNN), RF, and Repeated Incremental Pruning to Produce Error Reduction (RIPPER) to classify an error message generated by a Static Code Analysis tool. The best model was the RF which obtained F-measure values ranging from 83% to 98%. The RF was also found by Reddivari and Raman [18] to be the best performing machine learning model for defect and maintenance prediction obtaining an AUC of 0.8.

LLMs have also been used to assess code quality. Recent work conducted by da Silva Simões and Venson [19] compared the performance of GPT 3.5 Turbo and GPT 4o with SonarQube on two Open Source Java projects. Findings from this study highlighted differences between the results obtained by the two LLMs compared with the results of SonarQube in terms of the quality attributes measured. Mohajer *et al.* [20] proposed SkipAnalyser which not only adopt a LLM for static code analysis but used other modules to reduce the false positive rate for bugs and automatically generated patches of code for the detected bugs. Lu *et al.* [21] investigated how a fine-tuned and smaller LLM for code review could be adopted equaling the performance of much larger LLMs. Others have explored the optimal LLM parameter settings [22] and quality of code generated by LLMs when compared to humans [23].

While the community is keenly interested in such insights, it would be equally beneficial to learn about the most accurate methods for predicting outcomes given specific datasets. In the context of code quality, this is a regression-based problem, where little is known about how the simplest or most complex approaches perform [13, 24], and when these approaches provide the most value for the computational resource that they use. If the pattern of evidence is repeated across problem domains, this will strengthen claims for generalizability. In terms of the actual features that predict code quality, while the quality of software code is generally assessed along high level ISO9001:2008 dimensions (e.g., functionality) [25], evidence shows that publicly available code as those considered in this work tends to be assessed against other snippet-specific dimensions too (e.g., reusability) [6]. More granular structural elements of code quality are also held to be critical for evaluating software quality (e.g., lines of code) [26, 27], albeit this issue is not typically examined for Stack Overflow code.

We exploit this opportunity in this study in bridging the gap. Instead of creating an approach for predicting Stack Overflow code quality, we first explore the variables that predict code quality before investigating the regression approach that provides the best predictive power. In the process we also examine those regression approaches that reported poorer performance. We then perform follow-up evaluations using two independent datasets on Electrical Grid Stability and USA Cancer Mortality to explore the generalizability of the patterns observed. We contribute to the community's understanding of the quality of content on Stack Overflow. In addition, we add to the academic and practitioner communities' understanding of regression approaches, in terms of the value they provide given the availability of particular data. The research questions that supported our analyses are:

RQ1. *What variables predict Stack Overflow code quality?*

RQ2. *Which regression approach provides the best predictive power when modelling Stack Overflow code quality?*

3. Method

3.1. Data Acquisition and Processing

Applying Stack Overflow's data explorer[1] data was selected from the website for the years 2014, 2015, and 2016 using standard SQL statements as these years had the highest number of questions and answers on Stack Overflow [13]. Prior work [28] confirmed frequent reuse of these code snippets by the software engineering community. Moreover, according to Lotter *et al.* [28], Stack Overflow data are generalizable across languages and time, therefore Java code were sampled due to its popularity under the assumption that

[1] https://data.stackexchange.com/stackoverflow/query/new

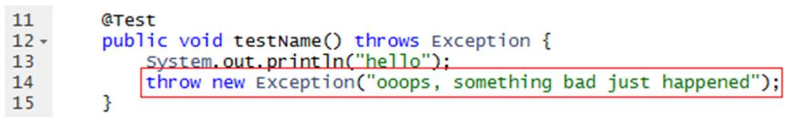
the sample would provide transferable findings for the software engineering community. Attributes relating to questions that generated these answers were extracted including answer posts to these questions. Answer posts to questions on Stack Overflow were structured using HTML where code snippets are included between code tags (i.e., <code> </code>) indicating presence of a code snippet. Those answer posts from questions tagged as Java and which contained at least one “<code>” tag were identified and sampled. The resulting dataset comprised of 117,523 answers (46,103 accepted answers and 71,423 unaccepted answers), which was then imported into a Microsoft SQL server database.

Each code snippet between the tags “<code>…</code>” was extracted from the *answerBody* and saved separately using a small Java program. Subsequent analysis of this Stack Overflow code revealed both in-line code snippets (surrounded by <code> tags) which were commonly added as part of text answers when contributors needed to provide explanation and code blocks (surrounded by <pre><code> tags) which provided more substantial implementation of specific solutions. However, Lotter *et al.* [28] argued against analyzing in-line code snippets for quality primarily due to the short length of this code, thus these were removed from our sample. Thus, 404,779 code snippets remained from the 117,523 answers (191,556 snippets from accepted answers and 213,223 snippets from unaccepted answers.

Next, in accordance with Lotter *et al.* [28], only code snippets that contained one or more lines of code were analyzed for quality. Using a Java program snippets from the database were checked for the presence of ‘import’, ‘package’, or ‘class’ and then we saved the file unmodified. Code snippets in which these words were absent were wrapped in a public class structure. To trace each code snippet, the file was saved as a .java file with a unique filename (e.g., C4567.java represented code snippet 4567). Minimal fixes to the code snippets were undertaken to mitigate the risk of confounding the results and analysis. The individual .java files are available as part of our replication package here: https://tinyurl.com/uv8jtfq.

For analyses, 66,389 snippets from accepted answers and 85,565 from unaccepted answers produced a total of 151,954 code snippets from 94,279 answers (i.e., 37.5% of the code snippets). The data set comprised of ten attributes, which were evaluated as relevant for studying the properties of Stack Overflow code [4, 28]: answer identification number (answerId), question identification number (questionId), answer score (answerScore), answer creation date (answerCreationDate), answer body (answerBody), question date (questionDate), question score (questionScore), view count (ViewCount), answer count (AnswerCount), and comment count (CommentCount).

LOC[2], Code Length[3], Code Space[4], and SPA[5] were also computed to aid the analyses, on the premise that these structural elements are also critical for evaluating code quality [26, 27]. The majority of code snippets were between 4 and 60 LOC.



```
@Test
public void testName() throws Exception {
    System.out.println("hello");
    throw new Exception("ooops, something bad just happened");
}
```

Fig. 1. Sample PMD reliability error from Stack Overflow code.

Measuring Code Quality: We used a combination of PMD[6], CheckStyle[7], and FindBugs[8] for checking code

[2] Refers to the number of lines of code.

[3] Refers to the length of the code snippet in terms of number of characters.

[4] Refers to the number of space characters (i.e., ‘ ’) in the code snippet provided as a part of answers.

[5] Refers to the number of code snippets per answer associated with each code snippet.

[6] https://pmd.github .io.

[7] http://checkstyle.sourceforge .net.

[8] http://findbugs.sourceforge .net/index.html.

quality as these tools have been effective in analyzing code snippets in the past [2, 27]. Moreover, each tool has complementary categories of checks which allow for greater quality coverage of the code snippets than using one tool alone [4]. FindBugs was able to check for security and malicious code vulnerabilities. Both PMD and CheckStyle were applied to check for reliability (and conformance to programming rules) and performance. Readability and compliance with Google coding standards were checked using CheckStyle. We configured the full list of code quality checks following Meldrum *et al.* [4]. For instance, Fig. 1 provides a sample reliability error as detected by PMD, where code at line 14 should be replaced with a subclass exception to fix the error. It was common for snippets to return multiple violations, at times even for a single code quality dimension. Our raw snippet dataset and processed code quality outcomes are available as part of the replication package here: https://tinyurl.com/uv8jtfq. Combining the outputs from these analyses formed the data set where the input variables comprised the code attributes above, and the output variable was the total number of code violations. The main task therefore was to adopt an appropriate regression algorithm that would result in the best estimation of the relationship between the input variables and total number of code violations.

3.2. Competing Regression Algorithms and Measures

A common problem is to adequately approximate a function of several variables requiring an approach to model a response variable, y, on one or more predictor variables $x_1, \dots, x_n$ given a set of data. The underlying assumption is that the system that generated the data is of the form $y = f(x_1, \dots, x_n) + \varepsilon$, where ε is an additive stochastic component which is neither controlled for nor observed [29]. Approaches to construct this function fall into the domain of regression analysis where the data is used to construct a function (model) $\hat{f}(x_1, \dots, x_n)$ which would act as a reasonable approximation of $f(x_1, \dots, x_n)$ [29]. Instead of adopting a single regression approach to develop a model, six different regression approaches were used to generate models based on the output of the tools that were used to analyze the code snippets in following recommended guidelines [30]. It is important to study these approaches as some of these have a long history of adoption and others are based on more recent approaches for generating better approximations for the models. Furthermore, some regression approaches deal with non-normal distributions of the response variable better than other methods.

The simplest regression approach is that of **Linear Regression** which uses ordinary least squares to estimate the parameters of the model resulting in a Linear Model (LM). As it is a classical approach, it is considered to be a baseline model that other sometimes more complex approaches are compared with [24]. However, adopting a linear model is contingent on the following assumptions. That for each value of the predictor variable, the distribution of the response variable must be normal. All observations should be independent and there is a linear relationship between the response variable and the predictor variables [24]. Finally, all values of the predictor variable should be constant with respect to the variance of the distribution of the response variable [24].

When the distribution of the response variable is not normal, a different approach, **Generalized Linear Models (GLM)**, can be applied. This linear model is augmented with a link function that allows its predicted value to be based on distributions other than the normal distribution [31]. The default method proposed in Nelder and Wedderburn [31] instead employs an iteratively reweighted least squares method to estimate the model's parameters for maximum likelihood. This results in generating a model with a high probability of it fitting the data [31].

An alternative approach is to use **Classification and Regression Trees (CART)** [32] as opposed to a least squares method to estimate the parameters of the model for the entire data space. In CART the data space is recursively partitioned into subregions, fitting a constant estimate of y within each subregion [32]. Once the tree has been generated, it can be depicted as a decision tree. Unlike Classification Trees where the response variable can take a finite set of unordered labels, Regression Trees specifically are for response variables that take continuous values [32]. However, the choice around the resulting depth or size of the tree impacts on the performance of it. Too shallow and the tree underfits the training data, and so does a poor job at generalizing to new data, whereas large trees tend to overfit the data [33]. Over the years different tree pruning strategies have been proposed to overcome these weaknesses [33].

Another main criticism of CART is that the product of recursively partitioning the data space results in the approximating function being discontinuous at the boundaries of the subregions that limits its effectiveness [29]. To overcome this limitation, Friedman [29] proposed a combination of Generalized Additive Models where another function is used to capture (smooth) the important parts of the data and Regression Trees. Friedman [29] refers to this as the **Multivariate Adaptive Regression Spline (MARS)** which adopts an expansion of product spline functions that models non-linear data and interactions among inputs. MARS (implemented as **Earth** due to trademark considerations) automatically determines the spline number and parameters from the data using recursive partitioning but distinguishes between the additive contributions of each input and interactions among them [34]. Iteratively adding the functions reduces the residual until a stopping criterion is met [34].

However, a key assumption underlying the approaches described above is that one model is generated but there are advantages to generating an ensemble of weak models which when combined forms the prediction model. This has the benefit of increasing predictive performance through diversity in the models which are typically decision trees [35]. The **Gradient Boosting Machine (GBM)** uses ensemble learning with each successive tree, learning and improving on the previous tree and generating gradient boosted trees which finds the optimal model to make predictions with the given data by minimizing a loss function [36]. More recently, Xtreme Gradient Boosting (**XGBoost or xgbTree)**, a popular GBM-based algorithm offers extensions to GBM to reduce the model overfitting to the data and lowering the complexity of the functions generated by each tree whilst still preserving the predictive accuracy of the model [37].

Finally, **Deep Neural Networks (DeepNN)s** which extend the single hidden layer in conventional perceptron-based neural networks to have multiple hidden layers have been proposed to model more complex patterns in data, and can be considered state-of-the-art for both classification and regression tasks [38], so this approach offers a good surface to evaluate the best performing regression models above.

Our intention to adopt the algorithms used in the experiments was to leverage diverse methods for building the regression models. LM was used as a baseline method as it suffers from modelling non-parametric distributions, therefore GLM was adopted to overcome this issue [31]. Earth and CART are already examples of non-parametric methods, so do not make any assumptions about the distribution of the data from which the model is built in addition to a better handling of outliers in the data set [29, 32]. The boosting algorithms (GBM and XGBoost) also can model non-parametric distributions [36, 37], resulting in less complex models compared with GBM. DeepNNs and other non-parametric methods are also considered as candidate models for our experiments. Although DeepNNs can model complex non-linear interactions between the variables and are more robust to noisy data sets, there are significantly more hyper-parameters which must be tuned to obtain highly accurate DeepNN models [38]. In addition, DeepNN-based models require large data sets with which to train on [38]. Finally, DeepNNs are subject to overfitting which can be the case with the regression methods described above, but this issue can be mitigated by adopting n-fold cross validation [39]. That said, we have assessed our regression outcomes against the DeepNN method to see if our pattern of outcomes is maintained.

Measuring Predictive Power: In this work, we used both simple models (LM, GLM, and CART) and complex models (xgbTree, Earth, GBM, and DeepNN—for further evaluation). This classification is based on the complexity of the modeling techniques used and their underlying algorithms [24]. There is also the need to evaluate the regression approaches applied in our work for predictive power. That is, how well is the discrepancy between the observed values and those produced by the model which is also known as the goodness-of-fit. Widely used metrics are the coefficient of determination (R^2) or the residual variance (error mean square σ_ε^2) [40]. Not doing this means it is difficult to make any claim as to how confident we are as to whether a model can generate reasonable predictions or not. More importantly, such performance metrics allow for more measurable and objective means for assessing the predictive power of a regression approach [41]. Finally, appropriate performance metrics should be selected to assess an individual model's performance and to compare different models to establish the best performing model. The Root Mean Square Error (RMSE) [42], for example, is a frequently used metric to compare the predictive performance across models and we use this metric in addition to the R-squared (R^2) value [42] and Mean Absolute Error (MAE) [38] in the work reported in this study.

3.3. Research Reproducibility

For implementations of the regression approaches, xgbTree and cart functions were used from the R *caret* package, the earth function was used from the R *earth* package which is an implementation of the MARS method, and the R *gbm* library provisioned the gbm function. The other two approaches, linear (lm) and glm functions, were part of the R standard library. The DeepNN was adopted from the R *ANN2* package. In facilitating easy reproducibility of our outcomes, we note that hyper-parameter optimization was used to establish the best learning parameters of the R implementations for the regression algorithms. More specifically, the same random seed was used for all folds for cross validation to ensure consistency across the models generated. For each regression approach we generated 20 candidate models with extensive hyperparameter optimization, including DeepNN:

- GBM: num_trees (range: 100–2000), interaction depth (3–10), shrinkage (0.01–0.2)
- XGBoost: max_depth (3–10), eta (0.01–0.3), nrounds (100–1000)
- CART: maxdepth (3–20), minsplit (5–50)
- DeepNN: hidden layers (3), no. hidden nodes per layer (2–50), learning rate (0.001–0.1)

Selection of the 20 candidate models was based on retaining those models which had a high probability of being optimal. For hyper-parameter optimization an adaptive resampling strategy [39] is used to find acceptable values of the hyper-parameters with fewer models generated than other schemes. The model that obtained the lowest RMSE was selected. We then repeated 10-fold cross validation on each of these optimal models twice, producing 20 results per model. Different random seeds were used to generate the folds for cross validation. For example, the optimal learning parameters for gbm were num_trees = 1843, interaction_depth = 6, shrinkage = 0.1223 and min_observations_in_node = 5. All datasets, generated data and scripts are provided in the replication package available here: https://tinyurl.com/uv8jtfq. The computer used for our experiments had an Intel Core i9 processor, 32GB of RAM with 1TB storage capacity. We will provide a permanent link to our replication package on Zenodo after the manuscript is reviewed.

4. Results

We recorded 244,266 related to reliability and conformance to programming rules errors (mean = 4.82), along with 530,521 readability errors (mean = 10.51), 3,949 performance errors (mean = 0.49) and 75 security errors (mean = 0.01). These errors were found in 50,717 Java code snippets. We see here that Stack Overflow code contained violations, and particularly those related to reliability and conformance to programming rules, readability and performance. Given our goal to examine the variables that predict Stack Overflow code quality, and that some of our data violated normality, we performed Kendall Rank correlation analysis to understand how the variables interacted and informed our modelling. In terms of high effect size, *code length* correlated with *code spaces* and *lines of code*, and this variable also correlated with Reliability and Conformance to Programming Rules and Readability violations. A similar pattern of outcome was observed for *code spaces* and *lines of code*. In addition, code snippets that suffered Reliability and Conformance to Programming Rules violations also returned many Readability violations. Other notable correlations (medium effect size) include *question score* and *view count*, and *answer score* and *accepted answer*. These were all statistically significant results ($p < 0.05$), refer to Table 1.

In **answering RQ1**, we next examine the specific variables that predict Stack Overflow code quality. Given that the outcome variable ('Total Violations') was numeric, we used regression modelling. We checked each column for the violation of normality assumption (via skewness) and used a log transformation if the data was not normal. Whenever this procedure did not result in normality, we instead used a square root transformation. We then performed a multiple regression, where the code attributes were regressed against the total violations, with outcomes showing that the model explained 81% of the variance in code snippet quality and was significant F(10,50706) = 22,042.17, $p < 0.01$. In terms of the weight of the code attributes, Table 2 shows that answer count (Answer Count), code length (Code Length), code spaces (Code Spaces) and Lines of Code (LOC) were most influential (with coefficients 0.138480, 0.382147, 0.099801 and 1.958790 respectively).

In **answering RQ2**, we next investigate xgbTree, earth, gbm, linear (lm), glm, and cart for their predictive power; with DeepNN outcomes validating the best of these approaches. This analysis is done to explore the regression approach that provides the best predictive power. To establish optimal learning parameters for the models we adopted 10-fold cross validation using the appropriate R package (refer to Section 3.3 for details). We plot the RMSE, R^2, and MAE, which reveal how data points are clustered around the line of best fit, the proportion of variance in the dependent variable that can be predicted, and the average magnitude of errors in the predictions, respectively. A lower RMSE, higher R^2 value, and lower MAE are favored, and signal better performing models. Fig. 2 shows that gbm recorded the best mean performance (RMSE = 2.77, R^2 = 0.99, and MAE = 0.79), while lm recorded the worst (RMSE = 40.55, R^2 = 0.09, and MAE = 24.92). We observed that while DeepNN performed well, outcomes for this approach were generally worse than gbm, xgbTree and cart (refer to Fig. 2).

Table 1. Kendall rank correlation (t) analysis for code attributes and violations

	Measure	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Code Attributes	1. Question Score^n	1	_*0.33_	*0.14	*–0.03	_*0._28	*0.03	*0.03	*0.03	*0.08	*–0.03	0.00	0.00	0.01	0.00
	2. View Count^l		1	*0.20	*0.10	*0.12	*0.15	*0.15	*0.14	0.00	*–0.06	*0.12	*0.12	*0.03	–0.02
	3. Answer Count^l			1	*0.11	*–0.05	0.01	*0.03	*0.02	*–0.09	*–0.27	*0.04	*0.03	0.00	–0.02
	4. Comment Count^s				1	0.00	*0.07	*0.09	*0.09	0.00	–0.02	*0.08	*0.08	0.01	0.00
	5. Answer Score^n					1	*–0.10	*–0.09	*–0.09	*0.28	_*0.34_	*–0.09	*–0.13	*0.03	–0.01
	6. Code Length^l						1	*__0.96__	*__0.94__	*–0.27	0.00	*__0.83__	*__0.79__	*0.10	*–0.03
	7. Code Spaces^l							1	*__0.95__	*–0.26	0.00	*__0.82__	*__0.82__	*0.10	*–0.04
	8. LOC^l								1	*–0.25	0.01	*__0.83__	*__0.82__	*0.09	*–0.04
	9. SPA^l									1	*0.24	*–0.25	*–0.27	0.00	0.02
	10. Accepted (1) /Unaccepted (0)^n										1	–0.01	–0.01	0.02	–0.01
Number of Violations	11. Reliability and Conformance to Programming Rules^s											1	*__0.73__	*0.16	*–0.04
	12. Readability^s												1	*0.11	*–0.04
	13. Performance^s													1	–0.02
	14. Security^s														1

Keys: * indicates significance ($p < 0.05$), italics indicates medium effect size using Cohen's [43] classification (0.3 ≤ rho ≤ 0.49), bold indicates high effect size using Cohen's classification (rho ≥ 0.50), n=normal distribution, l=log transformation, s=square root transformation

Table 2. Regression coefficients of significant code attributes

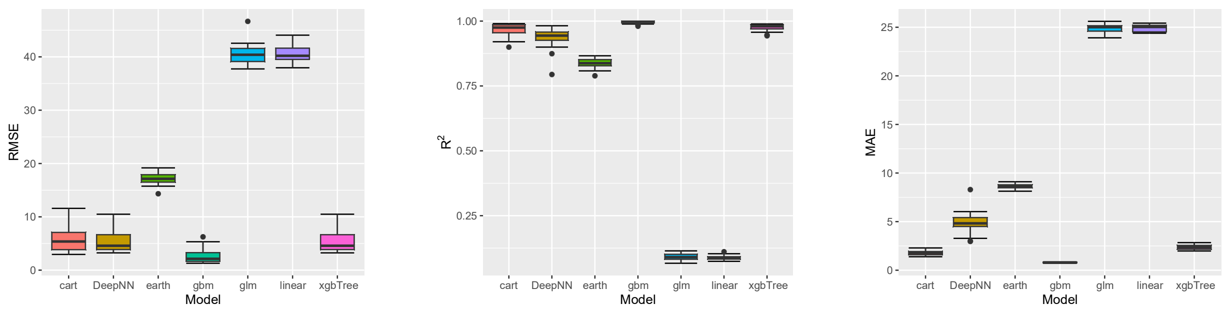
Code Attributes	Coefficient
Question Score	−0.001257
View Count	−0.037481
Answer Count	0.138480
Comment Count	0.041386
Answer Score	−0.001242
Code Length	0.382147
Code Spaces	0.099801
LOC	1.958790
SPA	−0.063379
Accepted	−0.032957

To probe the consistency of these findings, we further examine the RMSE outcomes for statistical differences using a Kruskal-Wallis H test with Bonferroni adjustments for post hoc pairwise comparisons. Findings confirmed statistical differences across the seven modelling approaches, ($X^2(6) = 125.03$, $p < 0.01$), with pairwise comparisons results in Table 3 showing that gbm, xgbTree, DeepNN, and cart performed significantly better than the three other approaches, with gbm returning the best performance overall. The linear model performed similar to glm. As can be seen in Table 3 not all patterns of outcomes are statistically significant ($p < 0.01$), but it is clear that glm and linear (lm) are the least performing models compared with gbm, xgbTree, DeepNN, and cart, with earth exhibiting performance between that of all the other models. We validate these outcomes in Section 5.3.

5. Discussion and Implications

5.1. Variables that Predict Stack Overflow Code Quality (Answering RQ1)

Our model only explained 81% of the variance in code snippet quality, so we concede that there are other important features missing from our dataset. However, it is important to note that longer Stack Overflow code tended to have more violations. It seems that with the opportunity to provide more insights for the community, contributors on Stack Overflow seem to attempt to do so without quality considerations at times. For instance, it was common to see a range of reliability and conformance to programming rules errors that were both serious and trivial. A specific serious error evident in longer code was the throwing of 'Raw Exception Types'. This error causes confusion for programmers, where it is unclear to understand the reason for faults, making it hard to debug the code.



(a) (b) (c)

Fig. 2. Performance metric outcomes for Code Quality regression models.

Table 3. Kruskal-Wallis results for pairwise comparisons

Sample 1-Sample 2	Test Statistic	Bonferroni Adjusted Significance
cart-DeepNN	2.1715	0.6279
cart-earth	**3.7659**	**0.0035**
cart-gbm	−1.9609	1.0000
cart-glm	**6.1556**	**0.0000**
cart-linear	**6.2647**	**0.0000**
cart-xgbTree	−0.2144	1.0000
DeepNN-earth	1.5945	1.0000
DeepNN-gbm	**−4.1324**	**0.0008**
DeepNN-glm	**3.9842**	**0.0014**
DeepNN-linear	**4.0933**	**0.0009**
DeepNN-xgbTree	−2.3858	0.3579
earth-gbm	**−5.7268**	**0.0000**
earth-glm	2.3897	0.3514
earth-linear	2.4989	0.2616
earth-xgbTree	**−3.9804**	**0.0014**
gbm-glm	**8.1165**	**0.0000**
gbm-linear	**8.2256**	**0.0000**
gbm-xgbTree	1.7465	1.0000
glm-linear	0.1092	1.0000
glm-xgbTree	**−6.3700**	**0.0000**
linear-xgbTree	**−6.4792**	**0.0000**

We observed that questions that were scored higher also attracted more views. In fact, the community also tended to use accepted answers as a basis to judge quality, as our outcomes show that accepted Stack Overflow answers tended to score higher than those that were not accepted. We observe that the more answers that are added to questions the more errors were typically observed in the code that was provided in these successive answers. This outcome goes against the value of online portals and open communities more generally, where the power of the crowd is said to act to enforce quality. Here we see that quality reduced as answers are added. Could it be that those providing new solutions are less prudent knowing that others have already tackled the issue identified/requested in the question? If this is the case, the motivation to add additional solutions/answers with poor quality seems unusual. Contributors' behavior seems to be at odds here, and points to code quality issues potentially being influenced by more than the lack of explanation, tending towards bad habits being disseminated in the code contributed itself. While mechanisms that allow others to provide corrections could be helpful [5] (including in future comments provided for posts), at the core of quality improvement would need to be a cultural shift in behavior. This cultural shift is especially necessary due to the widespread use of publicly available code, even for training LLMs.

5.2. Regression Approach with Best Predictive Power (Answering RQ2)

In terms of the regression models, we observed that Gradient Boosting Machine (GBM) performs the best of all regression approaches that were investigated in this study (including lm, xgbTree, glm, earth, DeepNN, and cart) for their predictive power. Gradient Boosting Machine uses ensemble learning where each successive tree learns and improves on the previous tree, generating gradient boosted trees which finds the optimal model to make predictions with the given data by minimizing a loss function [36]. The low RMSE and MAE, and high R^2 of gbm could be explained by the fact that the advantage of gradient boosting is to improve model accuracy whilst at the same time performing variable selection and model choice. Likewise, as gbm is based on a greedy additive algorithm, it will not select at any step any variable that is redundant, thereby reducing the effect of collinearity [36]. Our outcomes here show this learning process enhanced the prediction accuracy of the model. In fact, XGBoost previously achieved 71% accuracy in predicting questions that are not likely to attract answers at the time of submission, which was superior to other approaches used [14]. XGBoost is a popular GBM-based algorithm which offers extensions to GBM to reduce the model overfitting of the data and lowering the complexity of the functions generated by each tree whilst still preserving the predictive accuracy of the model [37]. The lower performance of XGBoost compared with gbm could have been attributed to the level of regularization specified to prevent overfitting of the model [37]. Finally, it should be noted that gbm also outperformed DeepNN, which we believe is because gbm uses ensemble learning, whereas DeepNN consisted of one multi-layer network.

We observed that, generally, cart and XGBoost also returned better performance than DeepNN in Fig. 2 (refer to R^2, and MAE outcomes). It is possible that the performance of cart was linked to the choice around the resulting depth or size of the tree [33]. The lower performance of earth may be explained by the procedure used for stepwise selection of the variables, where it is anticipated that these approaches would not guarantee the best set of variables for a given size. Furthermore, earth is likely to overfit the data when there are many predictor variables and there is no precise information about how these variables are related exactly. We observe that the outcomes from the other approaches (linear and glm) were unremarkable, pointing to their inadequacy when compared to gbm, cart, XGBoost and DeepNN.

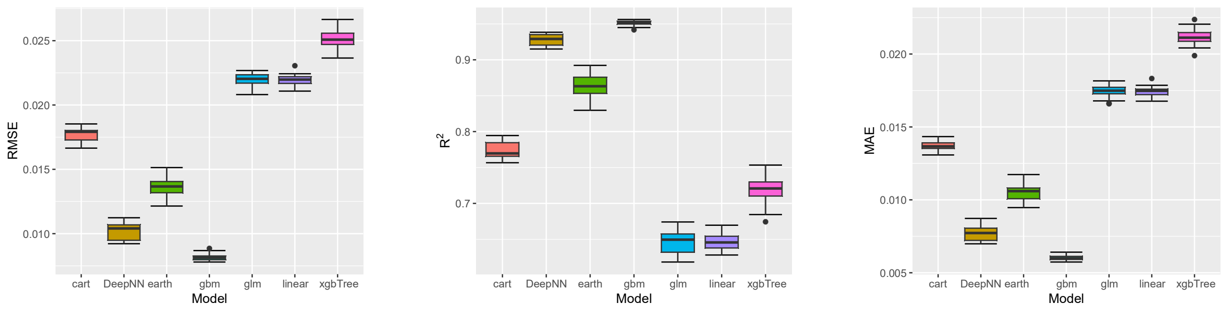
5.3. Regression Approaches Validation

We validate the pattern of outcomes by applying our methodology to two different independent regression datasets for two different domains to assess the six models' performance (via outcomes for RMSE, R^2, and MAE). The identical process was followed, with all datasets, generated data and scripts provided in the replication package available here: https://tinyurl.com/uv8jtfq. The Electrical Grid Stability Simulated dataset[9] contained 14 attributes with 10,000 instances and the Cancer Mortality dataset for USA Counties[10] contained 34 attributes with 3,048 instances. The former dataset was selected because it is challenging for a regression method to model the complex dynamics of electrical grid stability. Similarly, the latter dataset has many attributes which means the regression method must overcome potential collinearity in the data set to generate a model that performs well. Also, with the latter dataset, two columns were removed as these contained categorical values.

As can be seen in Figs. 3 and 4, gbm was the best performing model in both instances. Outcomes here support the position that our methodology is generalizable to different regression datasets and gbm continues to maintain superior performance over the other six approaches. The pattern of outcomes in Figs. 3 and 4 are not consistent for the other regression approaches, however. Although the RMSE, R^2, and MAE for DeepNN and xgbTree recorded among the top three outcomes at times in Figs. 3 and 4 respectively, earth also saw slight improvement (3rd best performance) when compared to its 5th place performance in Fig. 2. It is worth noting that within their respective groups, the relatively consistent rankings of complex models (DeepNN, xgbTree, earth, and gbm) across different datasets indicate robustness, whereas the varying rankings of the simple models (lm, glm, and cart) indicate dataset dependency and generalisation limits. Overall, however, gbm is consistently the best performing regression approach, regardless of dataset.

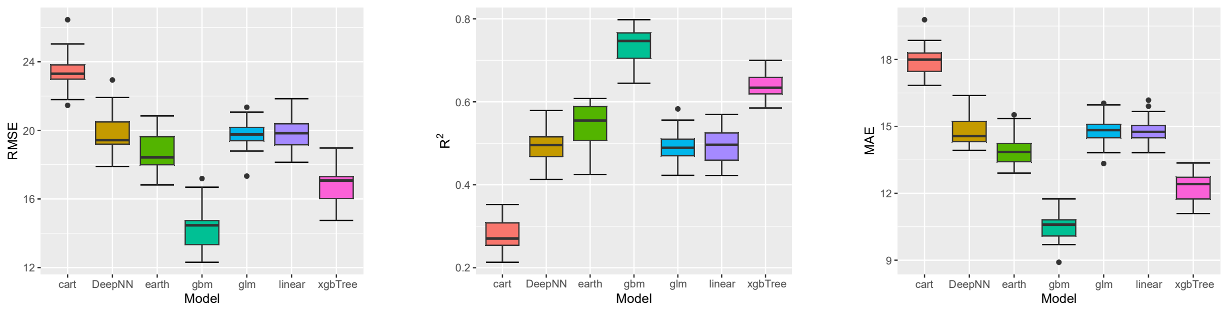
[9] https://archive.ics.uci.edu/dataset/471/electrical+grid+stability+simulated+data

[10] https://github.com/DeepanshuManchanda/Predict-cancer-mortality-rates-for-US-counties.-Exp-1-2



(a) (b) (c)

Fig. 3. Performance metric outcomes for the Electrical Grid Stability Simulated dataset.



(a) (b) (c)

Fig. 4. Performance metric outcomes for the cancer mortality for USA counties dataset.

6. Conclusion, Future Work and Limitations

In adding to the body of evidence around software code quality on online portals and other publicly available software code repositories and the evaluation of approaches that facilitate modelling and prediction of various software engineering issues, we examined the variables that predict Stack Overflow code quality, and the regression approach that provides the best predictive power. We observed that longer Stack Overflow code tended to have more violations, questions that were scored higher also attracted more views and the more answers that are added to questions the more errors were typically observed in the code that was provided in these answers. In addition, Gradient Boosting Machine (GBM) performed the best of all regression approaches that were investigated in this study (including linear regression, DeepNN, xgbTree, glm, earth and cart). These outcomes show that there is scope for the Stack Overflow community to improve and adopt a cultural shift in their behavior towards sharing high quality artifacts. Further, the mechanisms used by gbm (e.g., gradient boosting) are noteworthy in the way they improve prediction outcomes when compared to other regression approaches. The practical implications of our work demonstrate that combining multiple regression approaches with hyperparameter tuning creates more robust code quality assessment frameworks, as evidenced by GBM's superior performance across different domains. Our findings provide actionable metrics (answer count, code length, spaces, and LOC) that can be immediately implemented in automated quality monitoring systems for Stack Overflow and similar platforms, while offering more explainable results compared to LLM-based approaches.

Our future work will aim to understand the specific reasons for the way code quality on Stack Overflow degrades when code was longer or more snippets were added by the community (e.g., is it because longer code tended to use more third-party libraries, or such code attracts more mistakes?). In addition, we hope to investigate mechanisms for further optimizing the regression models to achieve better predictive performance. For example, with cart there are a number of existing pruning strategies that can be applied [32]. In addition, examining the regression models to establish variable importance through the likes of Shapley Values will provide other insight into consistency in specific variables that significantly influence Stack Overflow code quality, and how the patterns of outcomes align with model complexity. Finally, we seek to understand if and how variable importance changes over time and what effect it has on the regression models' performance.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Sherlock Licorish led the study, developing the first draft of the manuscript for review. Brendon Woodford and Lakmal Kiyaduwa Vithanage help to develop the methodology and results. Osayande Omondiagbe reviewed the manuscript, offering suggestions for improvement and enhancing the literature review. All authors approved the final version of the manuscript.

Acknowledgment

The authors are thankful for access to the Stack Overflow, Electrical Grid Stability and Cancer Mortality for USA datasets.

References

[1] Tahir, A., Dietrich, J., Counsell, S., Licorish, S., & Yamashita, A. (2020). A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites. *Information and Software Technology, 125,* 106333. doi: 10.1016/j.infsof.2020.106333

[2] Meldrum, S., Licorish, S. A., & Savarimuthu, B. T. R. (2017). Crowdsourced knowledge on stack overflow: A systematic mapping study, *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering* (pp. 180–185). New York, USA: Association for Computing Machinery. https://doi.org/10.1145/3084226.3084267

[3] Feng, Y., Vanam, S., Cherukupally, M., Zheng, W., Qiu, M., & Chen, H. (2023). Investigating code generation performance of ChatGPT with crowdsourcing social data," *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 876–885), IEEE. doi: 10.1109/COMPSAC57700.2023.00117

[4] Meldrum, S., Licorish, S. A., Owen, C. A., & Savarimuthu, B. T. R. (2020). Understanding stack overflow code quality: A recommendation of caution. *Science of Computer Programming, 199*, 102516. doi: 10.1016/j.scico.2020.102516

[5] Tavakoli, M., Heydarnoori, A., & Ghafari, M. (2016). Improving the quality of code snippets in stack overflow, *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (pp. 1492–1497). New York, USA: Association for Computing Machinery. https://doi.org/10.1145/2851613.285178

[6] Ndukwe, I. G., Licorish, S. A., & MacDonell, S. G. (2023). Perceptions on the utility of community question and answer websites like stack overflow to software developers. *IEEE Transactions on Software Engineering, 49(4)*, 2413–2425. doi: 10.1109/TSE.2022.3220236

[7] Licorish S. A. & Nishatharan, T. (2021). Contextual profiling of stack overflow java code security vulnerabilities initial insights from a pilot study, *Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (pp. 1060–1068). doi: 10.1109/QRS-C55045.2021.00160

[8] Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., & Counsell, S. (2018). Can you tell me if it smells? A study on how developers discuss code smells and anti-patterns in Stack Overflow, *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (pp. 68–78). New York, USA: Association for Computing Machinery. https://doi.org/10.1145/3210459.3210466

[9] Malgonde, O., & Chari, K. (2019). An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering, 24(2)*, 1017–1055.

[10] Amancio, L., Dorneles, C. F., & Dalip, D. H. (2021). Recency and quality-based ranking question in CQAs: A stack overflow case study. *Information Processing & Management, 58(4)*, 102552.

[11] Shcherban, S., Liang, P., Tahir, A., & Li, X. (2020). Automatic identification of code smell discussions on stack overflow, *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1–6). New York, USA: Association for Computing Machinery. https://doi.org/10.1145/3382494.3422161

[12] Mondal, S., Saifullah, C. M. K., Bhattacharjee, A., Rahman, M. M., & Roy, C. K. (2021). Early detection and guidelines to improve unanswered questions on stack overflow, *Proceedings of the 14th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference)* (pp. 1–11). New York, USA: Association for Computing Machinery. https://doi.org/10.1145/3452383.3452392

[13] Omondiagbe, O. P., Licorish, S. A., & Macdonell, S. G. (2022). Evaluating simple and complex models' performance when predicting accepted answers on stack overflow, *Proceedings of the 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 29–38). doi: 10.1109/SEAA56994.2022.00014

[14] Yazdaninia, M., Lo, D., & Sami, A. (2021). Characterization and prediction of questions without accepted answers on stack overflow, *arXiv preprint,* arXiv:2103.11386.

[15] Tavakoli, M., Izadi, M., & Heydarnoori, A. (2020). Improving quality of a post's set of answers in stack overflow, *Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 504–512). https://doi.org/10.1109/SEAA51224.2020.00084

[16] Gyimóthy, T., Vidács, L., Janthó, D., Nagy, B., & Tóth, L. (2019). Towards an accurate prediction of the question quality on stack overflow using a deep-learning-based NLP approach, *Proceedings of the 14th International Conference on Software Technologies* (pp. 631–639). Portugal: SCITEPRESS. https://doi.org/10.5220/0007971306310639

[17] Alikhashashneh, E. A., Raje, R. R., & Hill, J. H. (2018). Using machine learning techniques to classify and predict static code analysis tool warnings, *Proceedings of the 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)* (pp. 1–8). doi: 10.1109/AICCSA.2018.8612819

[18] Reddivari, S., & Raman, J. (2019). Software quality prediction: An investigation based on machine learning, *Proceedings of the 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)* (pp. 115–122). doi: 10.1109/IRI.2019.00030.

[19] Simões, I. R. d. S., & Venson, E. (2024). Evaluating source code quality with large language models: A comparative study, *Proceedings of the XXIII Brazilian Symposium on Software Quality* (pp. 103–113). New York, USA: Association for Computing Machinery. https://doi.org/10.1145/3701625.3701650

[20] Mohajer, M. M., *et al.* (2023). SkipAnalyzer: An embodied agent for code analysis with large language models. *ArXiv, abs/2310.18532*.

[21] Lu, J., Yu, L., Li, X., Yang, L., & Zuo, C. (2023). LLaMA-Reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning, *Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 647–658). https://doi.org/10.1109/ISSRE59848.2023.00026

[22] Arora, C., Sayeed, A. I., Licorish, S., Wang, F., & Treude, C. (2024). Optimizing large language model hyperparameters for code generation. *ArXiv, abs/2408.10577.*

[23] Licorish, S. A., Bajpai, A., Arora, C., Wang, F., & Tantithamthavorn, K. (2025). Comparing human and LLM generated code: The jury is still out! *ArXiv, abs/2501.16857.*

[24] Alexopoulos, E. C. (2010). Introduction to multivariate regression analysis (in eng). *Hippokratia, 14(Suppl 1),* 23–28. https://pubmed.ncbi.nlm.nih.gov/21487487

[25] Ndukwe, I. G., Licorish, S. A., Tahir, A., & MacDonell, S. G. (2023). How have views on Software quality differed over time? Research and practice viewpoints. *Journal of Systems and Software, 195*, 111524. doi: https://doi.org/10.1016/j.jss.2022.111524

[26] Börstler, J., *et al.* (2023). Developers talking about code quality. *Empirical Software Engineering, 28(6)*, 128. doi: 10.1007/s10664-023-10381-0

[27] Wimalasooriya, C., Licorish, S. A., da Costa, D. A., and MacDonell, S. G. (2024). Just-in-time crash prediction for mobile apps. *Empirical Software Engineering, 29(3)*, 68. doi: 10.1007/s10664-024-10455-7

[28] Lotter, A., Licorish, S. A., Savarimuthu, B. T. R., & Meldrum, S. (2018). Code reuse in stack overflow and popular open source java projects, *Proceedings of the 2018 25th Australasian Software Engineering Conference (ASWEC)* (pp. 141–150). doi: 10.1109/ASWEC.2018.00027

[29] Jerome, H. F. (1991). Multivariate Adaptive Regression Splines. *The Annals of Statistics, 19(1)*, 1–67. doi: 10.1214/aos/1176347963

[30] Calefato, F., Lanubile, F., & Novielli, N. (2019). An empirical assessment of best-answer prediction models in technical Q&A sites. *Empirical Softw. Engg., 24(2)*, 854–901. doi: 10.1007/s10664-018-9642-5

[31] Nelder, J. A. & Wedderburn, R. W. M. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society: Series A (General),* 135(3), 370–384. https://doi.org/10.2307/2344614

[32] Loh, W.-Y. (2011). Classification and regression trees. *WIREs Data Mining and Knowledge Discovery, 1(1)*, 14–23. doi: https://doi.org/10.1002/widm.8

[33] Rokach, L., & Maimon, O. (2005). Decision Trees. In O. Maimon, & L. Rokach (Eds), *Data Mining and Knowledge Discovery Handbook* (pp. 165–192). Boston, MA: Springer US, 2005,.

[34] Fernández-Delgado, M., Sirsat, M. S., Cernadas, E., Alawadi, S., Barro, S., & Febrero-Bande, M. (2019). An extensive experimental survey of regression methods. *Neural Networks, 111*, 11–34. https://doi.org/10.1016/j.neunet.2018.12.010

[35] Tin Kam, H. (1995). Random decision forests, *Proceedings of 3rd International Conference on Document Analysis and Recognition* (pp. 278–282). doi: 10.1109/ICDAR.1995.598994

[36] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics, 29*, 1189–1232.

[37] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). https://doi.org/10.1145/2939672.2939785

[38] Hong, T., Xie, S., Liu, X., Wu, J., & Chen, G. (2025). Do machine learning approaches perform better than regression models in mapping studies? A systematic review. *Value in Health.* doi: https://doi.org/10.1016/j.jval.2024.12.010.

[39] Kuhn, M. (2014). Futility analysis in the cross-validation of machine learning models, *ArXiv, abs/1405.6974.*

[40] Prairie, Y. T. (1996). Evaluating the predictive power of regression models. *Canadian Journal of Fisheries, 53(3)*, 490–492.

[41] Song, K. Y., Chang, I. H., & Pham, H. (2019). A testing coverage model based on NHPP software reliability considering the software operating environment and the sensitivity analysis. *Mathematics, 7(5)*, 450. https://doi.org/10.3390/math7050450

[42] Pham, H. (2019). A New Criterion for Model Selection. *Mathematics, 7(12)*, 1215.

[43] Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. *The SAGE Encyclopedia of Research Design.* New York, USA: Routledge.