International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V71I9P216

Volume 71 Issue 9, 170-177, September 2023

Original Article

Defect Prediction Model for Software Projects using Naïve Bayesian Classifier

K. Suresh[1], K. Jayasakthi Velmurugan[2], S. Hemavathi[3], V. Kavitha[4]

[1]*Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology, Kattankulathur, India.*
[2]*Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Chennai, India.*
[3]*Department of Computer Science and Engineering, Sri Sai Ram Engineering College, Tambaram, India.*
[4]*Department of Computational Intelligence, School of Computing, SRM Institute of Science and Technology, Kattankulathur, India.*

[1]*Corresponding Author : mailsureshkrish@gmail.com*

Received: 03 May 2023 Revised: 29 July 2023 Accepted: 15 August 2023 Published: 03 September 2023

Abstract *- The objective of this paper is to examine how effective supervised learning mechanisms are in classifying the defective and non-defected software modules during the software development process by means of applying a Naïve Bayesian (NB) classifier. Defect in software modules is the main cause of crucial software project risks. In other words, high-quality software products can be achieved by applying the most significant risk management process. However, an organization's environment or the development of projects is severely affected by the presence of risk events. Some of the critical constraints such as resources, time or budget are damaged due to risk factors or risk. Major steps included in risk assessment techniques are i) identifying, ii) analyzing, iii) planning, and iv) controlling events that are affecting the project environment. In this work, a model can be developed using Machine Learning (ML) methods and its metric data for predicting the defective modules in the software project. The NB classifier used in this work classifies the predicted and non-predicted data based on the parameters to best suit complex real-time situations.*

Keywords *- Classification, Fuzzy decision-making trial and evaluation laboratory, Machine Learning, Naïve bayesian classifier, Support Vector Machine.*

1. Introduction

Requiring the commitment of various skills and resources in one shot is the crucial goal of software management experts. Involving multiple parties, a plan, a budget, defined responsibilities and goals in a temporal activity having a starting and ending date is defined as a project. Software product delivery and development goals are achieved with different project categories in software development projects. Software products are developed through maintenance, re-engineering, re-using, and modifying modules in the software engineering project.

A project completed on a given time schedule, meeting the desired requirements and consuming only minimum expense cost is called a successful project. Most of the previous studies have reported the mismanagement of software development projects. According to the study, it was observed that the outcomes of information system development efforts were not delivered properly, not widely utilized and so on. Prior to the completion of projects, about 32% of defective modules affecting the project were removed, as suggested by the Standish group research. The normal estimation of the project cost was determined as 190%, but the outcomes obtained after removing defective modules showed that the project cost exceeded 52.4%.

Also, this research group has evidently proved that, considering the scheduled cost and time, there appears only a few successful projects (i.e. about 16.3% of software projects). Nonetheless, according to the Standish survey, it appears none of the software projects exhibit even a shadow for the incomplete project. The failure of software projects with competitors drastically affects the company's marketing position. The software development process is done based on the resources and time invested by the software companies.

When the needs for software projects change, risk factors related to those projects may also change. It has been determined that the machine learning methodology is the most effective method for accommodating changes in risk

factors in software projects. Machine learning approaches offer accurate outcomes by managing the programs based on the gained experience. The major objective of this study is to create an effective framework for risk management that will enable software project risk to be more effectively identified and prioritized. This framework can help software professionals manage software projects for bettering software performance without any problems.

1.1. Literature Survey

Some of the effective strategies created by the researchers have been used to perform robust and effective risk management activities in software projects. This module thoroughly reviews novel strategies to assess their shortcomings and potency. However, the theory of soft computing approaches is motivated by the drawbacks of current methodologies created for enhancing risk management activities in software project life cycles.

A project performed following the agile and waterfall model demands the time schedule and cost of expense required for a business in its initial lifecycle stage. Therefore, expert knowledge-based software estimation techniques are developed. However, some commonly used simple and effortless methods in software project development are planning poker, wideband Delphi, decomposition and so on [1].

However, the main limitation found of these methods is that they do not have much error tolerance; hence, function points-FP and Lines of Code (SLOC) are developed alternatives to these approaches. New technical changes in software development methodologies, software architecture, and programs are adapted by frequently updating the FP models and SLOC. [2] have stated that those techniques may struggle a lot with software project development during the deployment of configurable software, code reusability and so on.

Over the past two decades, researchers have tried to tackle the aforementioned issues by using data mining techniques and Machine Learning (ML) methods can be used for software estimation [3, 4]; uncertainty in software projects is better handled in the initial stage of project lifecycle itself using these machine learning algorithms. This is done by predicting the efforts and estimating duration at an appropriate time [5, 6, 7].

Furthermore, historical information is used by these ML-based techniques to perform the automated predictive process; thereby, it acquires the ability to minimize the political or psychological influences as well as the human biases and so on. However, only a smaller number of implementations is seen with ML algorithms on up-to-date research works. The reason for this fact is that finding an accurate data mining algorithm remained an open, challenging research issue.

However, projects performed using traditional ML algorithms have used difficult ensemble approaches and outdated data sets; hence, data is overfitted in the software programs [8, 9]. Software development project risk is defined as uncertainty in software products developed due to potential magnitude loss causing the project failure. [10] stated that factors threatening the project's success cause uncertainty in the software development project environment.

In other words, incompleteness in software development projects is generated by the factors named risk factors. In order to maintain risk management in software development projects, the researchers [11, 12] have approached risk assessment and risk control techniques (i.e. two-step forwarding approaches). The success of the project compromised through identifying, analyzing and prioritizing the risk factors is referred to as risk assessment; eliminating or controlling each risk factor by means of acting on it is called risk control [12].

Before the risk assessment is completed, it is impossible to perform the risk control activity. The development projects pursued by the managers create project failure because the risk involved is not sufficiently handled using these IS projects. When the managers wrongly recognize the associated risks, then risky decisions are made unknowingly [13]. Using a risk factor checklist for risk identification is the most commonly used method in developing software projects [14, 12].

A project manager can obtain a list of all potential risks using these checklists. Then, the type of risk factors that suit the project are verified and decided. Most of the past research works have analyzed and found that risk factors are combined to obtain the list of software project risks.

To deal with the complexity of SE activities, numerous objective functions have not been required to be formulated in any of the prior literal efforts. In other words, these difficulties make it a perennial research issue to employ soft computing-based methods during various life cycle stages of software project development. However, due to the existence of improbable elements, imprecision, and complexity introduced during carrying out the intermediate phases, wrong answers will be generated when employing basic algorithms to handle these problems. Soft computing-based techniques best handle the aforementioned issues and several real-time applications. Due to an unlikely component, the majority of current approaches have not been able to resolve all the issues generated at the intermediate stage fully.

1.2. Benefits of Risk Management

Identifying, analyzing and creating mechanisms to control and mitigate software project risks thoroughly from a system or process by complete study is called the risk

management process. Complete removal or elimination of software project risks does not imply the successful completion of the risk mitigation process; rather, it implies that the risk has been reduced to an acceptable level. It is important to gain knowledge about the severity of risk factors, all the possible risks and their potential consequences before determining risks in a software system. The mitigation or control steps have been developed to remove crucial risks based on the knowledge obtained on all risks.

Software developers can complete their projects within the scheduled time and cost of expense (budget) by applying the preventive approach to risk management. Better quality outputs are produced by the risks-managed software projects with minimum time and costs. Risk management techniques assess risks in all the stages of software project development by combining the software development process and risk management practices.

The importance of risk management has been realized by the organization with increased software development complexity. The reason for this fact is that risk management techniques have decreased the probability of project failures and uncertainties included in software development.

Notably, it is important that software developers should have knowledge about the causes of risks. Instead of solving risks in a single part of a software system, the risks should be managed throughout the whole system process. Software developers should follow the below-given criteria provided during the training program conducted for risk management as follows:

- Risk identification (integrity, process and business).
- Quantitative assessments are made through computing the risk probabilities.
- Qualitative risk impacts are determined with the computation of quantitative.
- Determining accurately the time for qualitative and quantitative risk assessment.
- Analyzed the hazard and safety measures of a software product.
- Applying management strategies to monitor and mitigate the risks.
- In all processes of the software project, the risks are identified. Risks can be identified and classified using various risk models. Business risks are managed using these models better than product risks.

In the software development process, software defect prediction methods have played a major role [15-19]. Software testing resources are allocated perfectly by accurately predicting the defective software modules. In the current trend, software metrics from the software modules were extracted by various researchers to form many of the training samples.

Then, software defect distribution prediction models are constructed by applying machine learning methods. Successful software defect prediction can be made using machine learning-based technology [20]. Most commonly used machine learning-based algorithms such as Linear Classifier (LC), Naïve Bayes (NB), Decision Tree (DT), and Particle Swarm Optimization (PSO) have been analyzed in the work of [21-25]. Nevertheless, defect classification is predicted by applying popular learning algorithms, namely, imbalanced learning, semi-supervised learning, unsupervised learning, and migration learning. Software quality attributes are predominantly ascertained with the support of search-based techniques having better predictive abilities [26].

Furthermore, error-prone modules were found to be difficult to predict due to complexities and difficulties in measuring software modules. The machine learning method [27] was applied in the software defect prediction model to satisfy the requirement of a large number of actual complexity measurement attributes. Curse of dimensionality issues are faced by many of the complex metric attributes [28].

Furthermore, data redundancy is produced by the software defect prediction method while introducing a greater number of software metric attributes. To withstand these issues, a model for ML-Based Software Risk Assessment Using Naive Bayes has been recommended to eliminate software defect modules.

The following software project hazards are identified, predicted, and classified using a novel supervised learning and classification technique described in this dissertation:

- To better accurately describe software risk by implementing Naive Bayesian Classification.
- Machine learning techniques are used to build a model that predicts possibly defective modules to the metrics of a given set of software modules.
- By using machine learning techniques and considering their individual metrics, potentially infected modules in the software project are predicted.
- The classification of complicated and non-complex software risks is done using a Naive Bayesian technique.

2. Classification Model

The working procedure of this system is shown in Figure 1. A process of handling, monitoring, analyzing and identifying risk in a specific project is called risk management. For the project's entire lifecycle, the risks are analyzed, handled, and assessed (prioritizing the risks), and front planning is made continuously using the risk management process. Prior to the damage to the whole project, the risk management process largely focused on identifying, communicating, analyzing and controlling the risks causing software failures. A core functionality of the

risk management process is to maintain the applicable principles by means of analyzing the independencies between the success of the project and risk occurrence.

Step 1: Input Selection
The first stage involves choosing the right input. The dataset's data are examined during this input selection procedure. The dataset used in this study is defined by a single statistical data matrix. Rows and tables in the matrix indicate the variable and data members. Each member's height and weight are elucidated in the data set lists.

Step 2: Data Processing
The duplication and missing attributes are rearranged and normalized in the data processing step to improve the software risk prediction process further.

Step 3: Attribute classification based on Naive Bayes classifier
Based on the Naive Bayes classifier, a supervised ML approach is created to categorize the predictable software risks. Initially, precise monitoring of risk factors is accomplished by combining data from various sources. Risk factors have a significant impact on some important factors, including time, resources, and expense budgets. Risk management primarily addresses risks that arise from project-related activities like event control, planning, analysis, and identification. Monitoring risk factors like geographic distribution, temporal trends, and intensity is crucial.

Step 4: Probability Estimation
In Naive Bayes classification, the risk can be defined by probability functions, which are used to describe the fundamental relationships between variables that can accept input in the form of a set of parent node values and compute the given node probability and can be placed in node probability tables.

Step 5: Risk Classification
Based on the probability score and machine learning method, the software risks will be classified.

Step 6: Risk Evaluation
Once the risk is classified, the risk evaluation is done by F-measure (i.e. precision and recall's harmonic mean values), recall, precision and accuracy.

Step 7: Predict Risk Factors
Based on the risk evaluation process, the risk factor will be predicted.

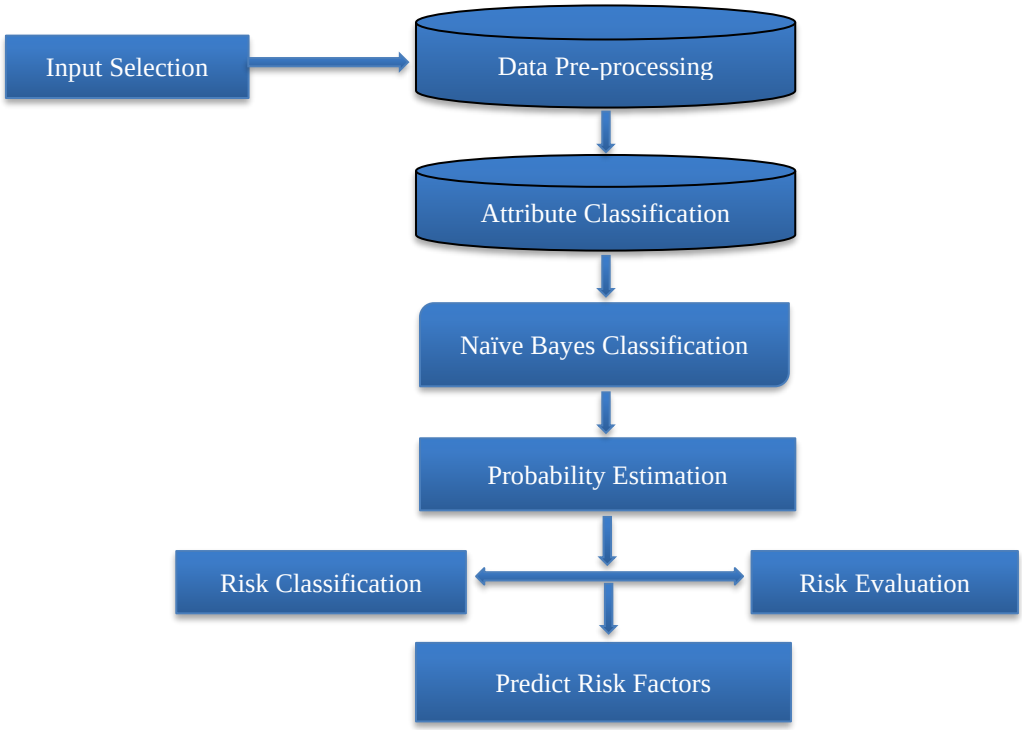

Fig. 1 Defect prediction model

3. Results and Analysis

3.1. Dataset Selection

Experiments were performed on the NASA metric data program (MDP) repository [30], which was applied successfully to predict the software defects included in 13 data sets. Table 1 indicates the datasets used in this study. Different samples included in each dataset are associated with each software module. In each software module, a number of attributes associated are identified by considering the number of static code attributes comprised in each software module. [31] Halstead attribute [15] and Line of Code (LoC) are the static code attributes determined in each code. Experiments are verified using MW1, MC2, PC4, PC1, KC3, and CM1 of NASA.

3.2. Evaluation Metrics

The predictive ability of the system model is evaluated using certain well-known system metrics, namely, F-measure (i.e. precision and recall's harmonic mean values), recall, precision and accuracy . The recall measure is the number of detected defective modules in relation to the total number of defective modules in a software system. Conversely, the number of defects predicted to a system's total number of defective modules is called precision rate.

A better trade-off maintained between precision and recall rate is called F-measure. Parameter optimization can be described clearly using a sample CMI data set focused on the appropriate order of importance for software risk prioritization in developing software fields. Providing a good solution is the primary challenge. The fuzzy DEMATEL approach has considerably increased the prioritization's effectiveness.

3.2.1. Accuracy (ACC)

The "correct classification rate" is another name for accuracy, which is calculated by dividing the number of predictions produced properly by the software defect prediction model by the number of predictions made overall.

$$ACC = \frac{T_{posi} + T_{nega}}{T_{posi} + T_{nega} + F_{posi} + F_{nega}} \quad (1)$$

3.2.2. Sensitivity (SEN)

Sensitivity, also called true positive rate, is calculated by determining the proportion of correctly detected not-defective software modules and is expressed as:

$$SEN = \frac{T_{posi}}{T_{posi} + F_{nega}} \quad (2)$$

3.2.3. Specificity (SPE)

Specificity, also known as true negative rate, is expressed as the percentage of correctly identified defective modules and is calculated as follows:

$$SPE = \frac{T_{nega}}{T_{nega} + F_{posi}} \quad (3)$$

3.2.4. Precision (PRE)

It can be determined by adding the total predicted not-defective software modules and the amount of correctly detected defect-free software modules, resulting in the formula shown below:

$$PRE = \frac{T_{posi}}{T_{posi} + F_{posi}} \quad (4)$$

3.2.5. F-Measure (F)

It is determined by computing the harmonic mean of sensitivity and precision.

$$F = \frac{2 * SEN * PRE}{SEN + PRE} \quad (5)$$

3.2.6. Recall (REC)

An accurate model can be used to define the majority of true positives.

$$REC = \frac{T_{posi}}{T_{posi} + F_{nega}} \quad (6)$$

Table 1. NASA dataset

Dataset	Attributes	Modules	Defective	Non-Defective	Defective (%)
CM1	38	327	42	285	12.8
JM1	22	7,720	1,612	6,108	20.8
KC1	22	1,162	294	868	25.3
KC3	40	194	36	158	18.5
MC1	39	1952	36	1916	1.8
MC2	40	124	44	80	35.4
MW1	38	250	25	225	10
PC1	38	679	55	624	8.1
PC2	37	722	16	706	2.2
PC3	38	1,053	130	923	12.3
PC4	38	1,270	176	1094	13.8
PC5	39	1694	458	1236	27.0

3.3. Analysis of Parameter Optimization

The experiment is performed by adopting the ten-fold cross-validation technique for verifying the predictive capability of the naive bayes model. About ten subsets are formed by dividing the training data sets. All 9 subsets are considered the training sets, and the residual one is the testing subset. Using one test set, a total of ten experiments are performed. An average of ten experiments is conducted to evaluate the performance of the naive bayes system. Four evaluation indicators are applied to select synthetically the parameter d. The system's accuracy is influenced greatly by the parameter k, which means different accuracies are produced by using the same dataset for selecting different sample sizes. For CM1 (software defect dataset), the intrinsic dimension is computed by means of considering the maximum probability occurrence of the algorithm (i.e. d=4). Figure 1 indicates the software defect prediction estimated using the naive bayes model on applying different dimensions d, and the size of the samples varied from 0 to 100. From the figure, it is observed that the naive bayes machine learning-based Naive Bayes classification algorithm has achieved better classification accuracy on varying the dimensional to 10 (i.e. d=10).

3.4. Comparative Analysis of Different Prediction Algorithms

Four evaluation indexes with the same feature are considered to compare the efficiency of the naive bayes classification algorithm with conventional classifiers, namely, KNN, SVM, and Neural Networks. Tables 2 to 7 indicate the obtained prediction results under different datasets. The SVM classifier is chosen because it adheres to the Structural Risk Minimization (SRM) principle, which lowers the likelihood of risk during the training phase and improves its capacity to generalize. With the aid of already identified data samples, also known as nearest neighbors, KNN discovers new or unidentified data samples and assigns the class to data samples using a voting mechanism. The classification of data samples includes participation from more than one nearest neighbor. Sensitive analysis was used by the NN model, which was trained using previous data, to identify significant metrics.Separate Neural Network models were created using the identified metrics to predict the problematic modules. However, the Naïve Bayesian classifier considers data samples' discrete, posterior, and prior probability distributions and is a probabilistic classifier. NB is efficiently used in software defect prediction because of its excellent performance and simpler computation technique. Additionally, this probabilistic Naive Bayes classifier does a good job of handling nonlinear issues. From the results, it is evident that the machine learning-based Naïve Bayes algorithm has outperformed all three conventional classifiers in terms of accuracy, F-measure, precision and recall. The reason for this fact is that nonlinear problems in the dataset cannot be solved using conventional algorithms.

Tables 2 to 7 indicate the obtained prediction results under different datasets. As can be seen from the results, the proposed machine learning-based Naive Bayes algorithm outperformed all three conventional classifiers in terms of accuracy, F-measure, precision, and recall (For MW1, the average of predicting defective software modules is 98% and non-defective is 96%; For MC2; For PC4, the average of predicting defective software modules is 95% and non-defective is 94%; For PC1, the average of predicting defective software modules is 88%). KC3's average for predicting defective software modules is 92%, while CM1's average for predicting defective software modules is 99%, and KC3's average for predicting non-defective software modules is 95%. Because the dataset contains nonlinear problems, traditional algorithms cannot be used to tackle them.

Table 2. Prediction results obtained on the MW1 dataset

Classifier	F-measure	Recall	Precision	Accuracy
SVM	87.65%	91.0%	84.43%	83.3%
KNN	89.05%	94.13%	86.25%	85.6%
Neural Networks	91.45%	95.3%	86.37%	87.85%
Naive Bayes	95.3%	98.7%	88.98%	96.5%

Table 3. Prediction results obtained on the MC2 dataset

Classifier	F-measure	Recall	Precision	Accuracy
SVM	80.66%	91.05%	72.43%	72.67%
KNN	86.05%	94.82%	82.26%	85.66%
Neural Networks	89.45%	92.3%	87.37%	87.52%
Naive Bayes	94.3%	93.7%	89.95%	97.53%

Table 4. Prediction results obtained on the PC4 dataset

Classifier	F-measure	Recall	Precision	Accuracy
SVM	73.66%	89.05%	72.43%	73.67%
KNN	80.06%	93.82%	72.35%	86.66%
Neural Networks	95.45%	90.3%	77.47%	88.52%
Naive Bayes	96.3%	92.7%	90.95%	98.53%

Table 5. Prediction results obtained on the PC1 dataset

Classifier	F-measure	Recall	Precision	Accuracy
SVM	80.65%	88.05%	77.43%	76.67%
KNN	85.07%	92.82%	75.35%	87.66%
Neural Networks	92.45%	89.3%	79.47%	89.57%
Naive Bayes	97.3%	92.75%	95.95%	95.53%

Table 6. Prediction results obtained on the KC3 dataset

Classifier	F-measure	Recall	Precision	Accuracy
SVM	87.65%	87.05%	78.43%	77.77%
KNN	89.07%	93.82%	76.35%	88.68%
Neural Networks	94.45%	90.3%	80.47%	94.57%
Naive Bayes	97.45%	93.75%	96.95%	98.53%

Table 7. Prediction results obtained on the CM1 dataset

Classifier	F-measure	Recall	Precision	Accuracy
SVM	90.65%	88.05%	82.43%	92.56%
KNN	92.76%	93.82%	75.35%	97.83%
Neural Networks	95.45%	92.3%	83.47%	95.58%
Naive Bayes	98.45%	95.65%	96.99%	99.43%

4. Conclusion

This paper examined software projects to increase the system's prediction rate by efficiently detecting and removing defective modules in software projects using the machine learning-based naïve bayes classification algorithm. This model selects inputs from the dataset by navigating through it. The dataset used in this study is defined by a single statistical data matrix. Rows and tables in the matrix indicate the variable and data members. Each member's height and weight are elucidated in the data set lists. Potentially Defective modules in the software project are predicted by applying the machine learning methods by means of considering their respective metric. Also, the Naïve Bayesian classification approach is applied to classify the complex and non-complex software risks. However, the software prediction problem is solved alone using two classification cases.

Also, this work does not elucidate defect severity using a software complexity metric dataset. In the future, cost computation should be reduced while training many different samples to optimize the parameters.

Abbreviations

SVM	-	Support Vector Machine
KNN	-	K-nearest neighbor
DT	-	Decision Tree
PSO	-	Particle Swarm Optimization
T_{POSI}	-	True Positive
T_{NEGA}	-	True Negative
F_{POSI}	-	False Positive
F_{NEGA}	-	False Negative
NN	-	Neural Network
NB	-	Naïve Bayes

References

[1] Robert K. Wysocki, *Effective Project Management: Traditional, Agile, Extreme*, John Wiley & Sons, 2013. [Google Scholar] [Publisher Link]

[2] Daniel D. Galorath, and Michael W. Evans, *Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves*, 1st Edition, Auerbach Publications, 2006. [Google Scholar] [Publisher Link]

[3] Sumeet Kaur Sehra et al., "Research Patterns and Trends in Software Effort Estimation," *Information and Software Technology*, vol. 91, pp. 1–21, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[4] Jianfeng Wen et al., "Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models," *Information and Software Technology*, vol. 54, no. 1, pp. 41–59, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[5] Cuauhtemoc Lopez-Martin, Claudia Isaza, and Arturo Chavoya, "Software Development Effort Prediction of Industrial Projects Applying A General Regression Neural Network," *Empirical Software Engineering*, vol. 17, pp.738–756, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[6] Iris Fabiana de BarcelosTronto, José DemísioSimões da Silva, and NilsonSant'Anna, "An Investigation of Artificial Neural Networks Based Prediction Systems in Software Project Management," *Journal of Systems and Software*, vol. 81, pp. 356–367, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[7] Stanislav Berlin et al., "Comparison of Estimation Methods of Cost and Duration in IT Projects," *Information and Software Technology*, vol. 51, no. 4, pp. 738–748, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[8] Jacky Keung, Ekrem Kocaguneli, and Tim Menzies, "Finding Conclusion Stability for Selecting the Best Effort Predictor in Software Effort Estimation," *Automated Software Engineering*, vol. 20, pp. 543–567, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[9] Dinesh R. Pai, Kevin S. McFall, and Girish H. Subramanian, "Software Effort Estimation using a Neural Network Ensemble," *Journal of Computer Information System*, vol. 53, no. 4, pp. 49–58, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[10] Devon K. Barrow, and Sven F. Crone, "Cross-Validation Aggregation for Combining Autoregressive Neural Network Forecasts," *International Journal of Forecasting*, vol. 32, no. 4, pp. 1120–1137, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[11] Shai Ben-David, and Shai Shalev-Shwartz, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014. [Google Scholar] [Publisher Link]

[12] Barry W. Boehm et al., *Software Cost Estimation with Cocomo II*, Prentice Hall PTR, 2000. [Publisher Link]

[13] Jianglin Huang, Yan-Fu Li, and Min Xie, "An Empirical Analysis of Data Pre-Processing for Machine Learning-Based Software Cost Estimation," *Information and Software Technology*, vol. 67, pp. 108–127, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[14] Ali Idri, Mohamed Hosni, and Alain Abran, "Improved Estimation of Software Development Effort using Classical and Fuzzy Analogy Ensembles," *Applied Soft Computing*, vol. 49, pp. 990–1019, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[15] Venkata Udaya B. Challagulla et al., "Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques," *International Journal on Artificial Intelligence Tools*, vol. 17, no. 2, pp. 389-400, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[16] Khaled El Emam et al., "Comparing Case-Based Reasoning Classifiers for Predicting High-Risk Software Components," *Journal of System Software*, vol. 55, no. 3, pp. 301–320, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[17] K. Ganesan, Taghi M. Khoshgoftaar, and Edward B. Allen, "Case-Based Software Quality Prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 2, pp. 139–152, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[18] P. Sivasankaran, "Quality Concepts in Industrial Systems using QFD (Quality Function Deployment) – Survey," *SSRG International Journal of Industrial Engineering*, vol. 8, no. 1, pp. 7-13, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[19] Jonathan D. Strate, and Phillip A. Laplante, "A Literature Review of Research in Software Defect Reporting," *IEEE Transaction on Reliability*, vol. 62, no. 2, pp. 444–454, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[20] E. Kokiopoulou, and Y. Saad, "Orthogonal Neighborhood Preserving Projections," *Fifth IEEE International Conference on Data Mining*, pp. 234–241, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[21] Wang Xu-hui et al., "A ROC Curve Method for Performance Evaluation of Support Vector Machine with Optimization Strategy," *International Forum on Computer Science-Technology and Applications, Chongqing, China*, pp. 117–120, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[22] Liu Yang, Yong Xiang, and DezhongPeng, "Precoding-Based Blind Separation of MIMO FIR Mixtures," *IEEE Access*, vol. 5, pp. 12417–12427,2017.[CrossRef] [Google Scholar][Publisher Link]

[23] Changshui Zhang et al., "Reconstruction and Analysis of Multi-Pose Face Images Based on Nonlinear Dimensionality Reduction," *Pattern Recognition*, vol. 37, no. 2, pp. 325–336, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[24] Praman Deep Singh, and Anuradha Chug, "Software Defect Prediction Analysis Using Machine Learning Algorithms," *7th International Conference on Cloud Computing, Data Science Engineering - Confluence, Vancouver, Canada*, pp. 775–781, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[25] Manvi Chahar, and Savita, "Implementation and Classification of Anomalous Detection with Varying Parameters," *SSRG International Journal of Computer Science and Engineering*, vol. 6, no. 4, pp. 16-18, 2019. [CrossRef] [Publisher Link]

[26] Mohammad A. Alsmirat et al., "Accelerating Compute Intensive Medical Imaging Segmentation Algorithms Using Hybrid CPU-GPU Implementations," *Multimedia Tools and Application*, vol. 76, no. 3, pp. 3537–3555, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[27] Nagappan Nachiappan, and Ball Thomas, "Static Analysis Tools as Early Indicators of Pre-Release Defect Density," *Proceedings of the 27th International Conference on Software Engineering*, pp. 580–586, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[28] Zhitao Guan et al., "Privacy-Preserving and Efficient Aggregation based on Blockchain for Power Grid Communications in Smart Communities," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 82-88, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[29] Nitu Bhardwaj, and A.S Bhattacharya, "Survey on General Classification Techniques for Effective Bug Triage," *SSRG International Journal of Computer Science and Engineering*, vol. 2, no. 11, pp. 6-10, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[30] S. SathiyaKeerthi, and Chih-Jen Lin, "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel," *Neural Computing*, vol. 15, no. 7, pp. 1667-1689, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[31] T.J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, 1976. [CrossRef] [Google Scholar] [Publisher Link]