Indonesian Journal of Electrical Engineering and Computer Science
Vol. 34, No. 1, April 2024, pp. 508~519
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v34.i1.pp508-519

Comparing machine learning techniques for software requirements risk prediction

Yasiel Pérez Vera, Álvaro Fernández Del Carpio
Department of Software Engineering, Universidad La Salle, Arequipa, Perú

Article Info

Article history:

Received Dec 13, 2023
Revised Jan 14, 2024
Accepted Jan 16, 2024

Keywords:

Classification techniques
Comparative analysis
Evaluation metrics
Machine learning
Risk prediction
Software requirements

ABSTRACT

Software requirements are the most critical phase focused on documenting, eliciting, and maintaining the stakeholders' requirements. Risk identification and analysis are preemptive actions designed to anticipate and prepare for potential issues. Usually, this classification of risks is done manually, a practice that the personal judgment of the risk analyst or the project manager might influence. Machine learning (ML) techniques were proposed to predict the risk level in software requirements. The techniques used were logistic regression (LR), multilayer perceptron (MLP) neural network, support vector machine (SVM), decision tree (DT), naive bayes, and random forest (RF). Each model was trained and tested using cross-validation with k-folds, each with its respective parameters, to provide optimal results. Finally, they were compared based on precision, accuracy, and recall metrics. Statistical tests were performed to determine if there were significant differences between the different ML techniques used to classify risks. The results concluded that the DT and RF are the techniques that best predict the risk level in software requirements.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Yasiel Pérez Vera
Department of Software Engineering, Universidad La Salle
Alfonso Ugarte Avenue, Arequipa, Perú
Email: yperez@ulasalle.edu.pe

1. INTRODUCTION

Software development involves a group of interconnected activities ranging from planning where the objectives and scope of the project are established, specification of requirements to determine the needs and expectations of the users, conceptualization of the solution through the design, writing the source code, and finally deployment and maintenance to ensure the availability of the solution, up-to-date and free of errors. Software requirements are the most critical phase focused on documenting, eliciting, and keeping the stakeholders' necessities [1]. Frequently, successfully discussing and securing stakeholders' main requisites is a critical factor in creating a high-quality software system [2]. However, in the software development life cycle (SDLC), there is a risk of imprecise processes that can lead to the potential failure of the software. These alarming processes are called software risks. If these risks are not identified in time, it may lead to project failure [3].

The standish group report [4] has continually pointed out the software industry's frequent high failure rates. The report said that only 16.2% of projects were successful, completed on time and budget, and had all expected features. Most projects (approximately 52.7%) exceeded budget, missed deadlines, and missed some promised features. This results in 31.1% of projects being categorized as failures, indicating they were either abandoned or canceled. Some factors influencing whether a project succeeds or fails include executive engagement, precise requirement definitions, efficient project management, and active involvement of the end user.

Risk identification and analysis are preemptive actions designed to anticipate and prepare for potential issues. It implies identifying potential risks, evaluating their likely impact and frequency, and formulating strategies for mitigation or contingency plans to tackle them proactively [5]. Risks can be categorized using different criteria, including their origin, type, or impact on the project [6]. Usually, this classification of risks is done manually, a practice that the personal judgment of the risk analyst or the project manager might influence [7]. That process can introduce biases and variability in risk management, leading to inconsistent risk evaluations and unpreparedness for critical issues. An inappropriate identification of risks can have a relevant significance, especially in software requirements, because misunderstood or poorly managed requirements can lead to software failing to meet its intended use, causing cost overruns, delays, or even project failure [8].

With the vast possibilities offered by artificial intelligence (AI), machine learning (ML) techniques can provide good support to improve risk classification. With the analysis of enormous amounts of data to uncover patterns and trends, these techniques offer a more unbiased and systematic approach to risk evaluation. These techniques can be trained with historical data from software projects to recognize risk indicators and thus prevent future issues with better precision than manual methods. Also, they can adapt and learn over time, and as more data is processed, more abilities will be acquired to predict and classify risks, becoming a fundamental tool for project teams.

This research aims to conduct a comparative analysis of several ML techniques to determine the best solution that automates the classification of risks in software development projects, reducing subjectivity and improving consistency in risk assessment. A solution implemented with ML techniques can provide early warnings and actionable recommendations, enhancing decision-making and contingency planning in software project management. This paper is organized as follows: Section 2 introduces related work; section 3 details the materials and methods used in this study. Then, section 4 describes the results and discussions. Finally, section 5 presents conclusions and future work.

2. MATERIALS AND METHODS

The materials and methods used to conduct this research are presented below. Works related are presented to analyze other authors on the prediction of risks associated with software requirements. The theoretical foundations of ML and the classification methods that will be used are presented. In addition, the tools, programming languages, and code libraries used in the development of the comparison of risk classification methods associated with software requirements are presented.

2.1. Related work

Ensuring and meeting software requirements is critical to delivering a high-quality software system. Failures in software projects often arise when stakeholders' needs, budgets, or timelines are unmet. A significant characteristic of most software project developments is their high failure rate. Such failure rates can be traced back to unforeseen events during the SDLC process, leading to substantial software losses in numerous organizations. The term "risks" refers to these unknown occurrences that arise from different risk variables that are present in the numerous activities that make up the software development lifecycle. A successful software development process depends on early risk prediction in the requirements, as late risk recognition can have a negative impact on the project's quality, schedule, and money. We have identified that this problem is broadly mentioned in the studies analyzed in this section [3], [9]-[13]. As the initial phase of the SDLC, accurately forecasting risks in software requirements can enhance the software's productivity and quality.

The objective presented in [9] focused on assessing various ML models and proposing an ensemble classifier named ABMJ. This classifier merges AdaBoostM1 and J48 to predict risks in software requirements. The intention behind this model was to enhance the precision of risk prediction over other existing ML methods. Thus, the model developed was compared with several ML algorithms, including the random forest (RF), average one dependency estimator (A1DE), multilayer perceptron (MLP), Naive Bayes (NB), cost-sensitive decision forest (CSF), support vector machine (SVM), and J48 decision tree (J48). An available risk dataset from the Zenodo repository, comprising 13 attributes and 253 instances, was utilized for its evaluation. Evaluation techniques such as 10-fold cross-validation and diverse performance indicators like recall, precision, Matthew's correlation coefficient (MCC), F-measure, and accuracy were employed.

The ABMJ model outperformed others, achieving an impressive 97.63% accuracy, unlike the MLP, which showed the least effectiveness with an accuracy of just 62.06%. These findings indicate that the ABMJ model is significantly more efficient in predicting software requirements risks than the other tested techniques. Using a combined ensemble model featuring AdaBoostM1 and J48 represented a novel approach to risk prediction for software requirements. Moreover, the proposed model's high accuracy underscored its viability for real-world applications. A limitation mentioned in the research is the threats to internal and external validity, especially regarding the applicability of the results to different datasets or contexts.

Regarding the main objective in [10], the requirement risk dataset served as the basis for a new approach that the authors presented and assessed, utilizing tree-family machine learning (TF-ML) techniques for predicting software requirement risks. This study employed various TF-ML techniques, such as forest by penalizing attributes (Forest-PA), J48, decision stump (DS), REP-Tree (REP-T), RF, hoeffding tree (HT), credal decision tree (CDT), CS-Forest, random tree (RT), and logistic model tree (LMT). A risk dataset from the Zenodo repository contains 13 characteristics and 299 occurrences. The study implemented test scenarios assessment measures (like, root mean square error (RMSE), mean absolute error (MAE), root relative squared error (RRSE), relative absolute error (RAE), recall, precision, MCC, F-measure, and accuracy). Moreover, each model calculates incorrectly classified instances (ICI) and correctly classified instances (CCI).

The findings indicated that specific techniques outperformed others in various test scenarios, mainly the CDT and Forest-PA. The metrics with values of 4.498% for RAE, 0.0126 for MAE, 23.741% for RRSE, and 0.0888 for RMSE highlighted the superiority of specific TF-ML techniques in risk prediction. Furthermore, each recall, accuracy, and F-measure accomplished 0.980 results. The paper comprehensively compared different TF-ML techniques, offering valuable insights into their effectiveness in risk prediction in software requirements. The results were specific to the dataset used from the Zenodo repository, which may limit the generalization of the findings to other datasets or real-world scenarios. Some techniques did not perform well in specific test scenarios, which might indicate a need for further optimization or modification of these approaches for particular contexts.

Naseem *et al.* [11], the primary aim of the study was to evaluate several ML methods for forecasting risks in software specifications. The paper aimed to identify the most effective classifier for this purpose. The study utilized a set of ML classifiers, including A1DE, NB, K-nearest neighbor (KNN), decision table, composite hypercube on iterated random projection (CHIRP), cost-sensitive decision forest, decision table/Naive Bayes hybrid classifier, J48 DT, CDT, and RF. These techniques were evaluated using the risk dataset from Zenodo, applying various evaluation metrics such as RRSE, RMSE, CCI, RAE, MAE, recall, precision, MCC, F-measure, precision-recall curves area (PRC area), and receiver operating characteristic area (ROC area).

The model consisted of four main components: risk identification, risk analysis, risk prioritization, and requirement risk dataset. The study revealed that CDT and the Decision Table/Naive Bayes Hybrid Classifier techniques performed exceptionally well regarding accuracy and error rates. Specifically, CDT showed remarkable performance with the lowest error rates across various metrics. CDT demonstrated strong performance with notable overall accuracy, a higher CCI, and lower MAE and RMSE. The experimental data showed the results of MAE at 0.0126, RMSE at 0.089, RAE at 4.498%, and RRSE at 23.741%. The precision, recall, and F-measure each reached 0.980, with MCC at 0.975 and an overall accuracy of 97.993%. Additionally, in terms of CCI, CDT was the most effective.

The extensive comparison of various ML techniques provided valuable insights into risk prediction in software development. A comprehensive set of evaluation metrics ensured a thorough and multi-faceted assessment of the classifiers. However, the study’s applicability was limited to the dataset used, which may affect the generalizability of the findings to other datasets or real-world scenarios. Specific techniques did not perform as well in some test scenarios, indicating potential limitations in their adaptability or effectiveness in particular contexts. Other assessment measures, such as mean magnitude of relative error (MMRE) and pairwise relative distance (PRED), can be used for validation.

Mamman *et al.* [12] underscores the increasing complexity and duration of modern software systems, indicating that escalating failure rates in software projects are a significant concern. These failures lead to substantial revenue losses for software organizations. Just 9% of software projects in worldwide organizations are finished on schedule and under budget, making this problem even more severe. These projects frequently fall short of fulfilling all of the original specifications. Thus, US-based technology businesses realize about 42% of the functional criteria defined in software projects. These risk concerns can be resolved with the use of the subjective risk management (SRM) technique. Its usefulness is constrained, nevertheless, by its dependence on human judgment, which increases the possibility of errors because it depends on the knowledge and expertise of an expert. Furthermore, because knowledge cannot be simply transferred between several teams working on a software project, the SRM approach is not repeatable.

As a result, the authors presented rule-based methods for estimating risks in software specifications. In particular, they created the nested dichotomy-based fuzzy unordered rule induction algorithm (FURIA) and its variations. Fuzzy rules are combined with an effective rule-stretching method for classification in FURIA, a rule-learning system. Because these fuzzy rules provide dynamic limits in categorization tasks, they are more flexible than standard rules. The authors examined models for their predictive capabilities in existing studies of software risk prediction and other ML techniques. This included rule-based models like partial decision tree (PART), rough set (RS), and RIPPER, along with ML-based models such as SVM, KNN, decision tree (DT), A1DE, NB, and RF. For the evaluation, a benchmark dataset was used to compare these proposed models with pre-existing ML and rule-based models.

The outcomes showed that the FURIA models performed better than other rule-based and ML models, as well as the basic FURIA model, particularly the ones with nested dichotomy variants. In various testing conditions, these models showed improved accuracy, F-measure, and MCC levels. With an accuracy of 97.99%, an F-measure of 0.980, and an MCC of 0.975, FURIA produced the greatest results in the performance study when compared to rule- and ML-based models. Furthermore, the FURIA-FCS version achieved the greatest results with an accuracy of 98.62%, an F-measure of 0.987, and an MCC of 0.983 when compared to its nested subset selection dichotomy counterparts. Rules and ML-based models were surpassed by the suggested fuzzy induction models. Furthermore, datasets with fuzziness and imprecise values may be handled by these models. The study's reliance on a specific dataset, however, might limit how broadly the findings can be applied to other datasets or situations. Furthermore, there may be challenges with the interpretability and practical use of the presented models due to their complexity.

Akumba *et al.* [13], the main objective was to use supervised ML methods, namely the Naive Bayes classifier, to create a predictive risk model. The purpose of this approach was to forecast the degree of risk in software projects and to pinpoint the essential components that raise that risk. Using a dataset from Zenodo, the Naive Bayes classifier approach was used and assessed with an emphasis on software requirement risk prediction. This dataset was employed in the requirement-gathering stage to estimate risks. There were 299 data instances total, with one target variable and 12 variable characteristics.

The model's confusion matrix revealed accurate predictions across different risk levels: four (4) data points correctly identified as Catastrophic, eleven (11) as High, eighteen (18) as Moderate, thirty-three (33) as Low, and seven (7) as Insignificant. The model achieved an accuracy rate of 98%, with a 95% confidence interval and a Kappa value of 97%. This high degree of accuracy in classifying various risk levels was a significant accomplishment of the model. In conclusion, the software development lifecycle's requirement-gathering phase saw accurate risk level prediction from the model. It offers the ability to forecast outcomes for further stages, such as design, coding, testing, and maintenance. This skill would be useful in determining the main factors influencing risk in different software projects. However, the study was constrained by the dataset, which might have an impact on how well it applies to various software projects with various features.

In addition to the problem described at the beginning of this section, a significant point raised in [3] is that many datasets used in research are focused on effort estimation. However, these datasets lack specific risk attributes such as risk category, magnitude, impact, dimensions of risk, probability, and priority. To date, no dataset includes these risk attributes specifically for software requirements. The study's objective was to develop a detailed risk dataset encompassing attributes of software requirements and risks, along with their interrelations, essential for forecasting risks in future software projects. This dataset aimed to aid in predicting risks associated with new software requirements through data mining classification methods. Rather than creating a new predictive model, the research concentrated on building and validating a dataset for such predictions.

The technique was divided into three primary stages: filtering, validation of the dataset by IT specialists, and risk-oriented data collecting. The dataset included risk factors that were determined by IT professionals and literature, as well as specifications from software requirement specification (SRS) papers of several open-source projects. Project category, requirement category, risk category, impact, dimension, likelihood, priority, influencing the number of modules, fixing length, fix cost, and risk level were the requirements risk attributes. The experimental results indicated that the use of normalized and standardized filters did not enhance the accuracy of the KNN classification technique on the proposed dataset. In contrast, the discretize filter achieved a correct classification rate of 76.25% for the instances, 17% higher than other filters. Additionally, this filter demonstrated a relatively low MAE of 0.11 and an RMSE of 0.280.

Creating a specialized dataset for software requirement risk prediction is a significant contribution, as such datasets are scarce in this field. The proposed dataset comprised categorical data, leading to unchanged accuracy levels when subjected to normalization and standardization. Conversely, the discretize method transforms numeric attributes into ordinal ranges, thereby enhancing the precision of classification in terms of accuracy. A limitation of the study is that it focuses on dataset development and does not provide an in-depth analysis of the predictive models or their performance using this dataset, which limits its scope. The reliance on open-source project data and expert opinions may introduce biases or limitations in the dataset, affecting its generalizability.

2.2. Theoretical background

The theoretical foundations of this research are framed in AI, ML, and different ML methods for classification. The main concepts are presented below, and the automatic learning techniques will be described. The main characteristics of the techniques LR, DT, SVM, RF, and MLP as neural networks are described.

2.2.1. Machine learning

Also known as Expert Systems, it is a branch of AI [14] that aims to develop techniques that enable computers to learn and generalize from previously provided data sets. These techniques are used when there is a large volume of data, making it challenging for a human to analyze and draw conclusions. It involves the development of algorithms that enable machines to iteratively learn from data, adapting their behavior to evolving patterns and information.

2.2.2. Logistic regression

It is a statistical technique involving multiple variables [15] that allows us to analyze the relationship between a group of independent variables and a dichotomous dependent variable. It can only take two values, typically zero and one, where zero represents absence, and one represents presence. In other words, the dependent variable is discrete, in contrast to the group of independent variables, which can be both quantitative and qualitative. The underlying function is exponential, which helps better represent certain phenomena.

2.2.3. Decision tree

This tree-based model [16] allows for the classification of cases into groups or the prediction of the value of a dependent variable based on a set of independent variables. It provides a more graphical way to represent the alternatives or events that arise from choosing an option, ultimately providing the most accurate probabilistic outcome. All of this can be observed visually through a process in which it is assumed that the participating sets are disjoint, meaning that all input objects are directed to a single membership group.

2.2.4. Support vector machine

These [17] belong to the class of linear classifiers, using hyperplanes to create a separation in the original space if the input cases are linearly separable or in a transformed space. They differ from other ML methods for generating separation or classification because they focus on minimizing structural risk. The goal is to find a hyperplane equidistant from the closest examples belonging to separate classes. The training elements located on the boundary of the separation are called support vectors. SVMs often exhibit high generalization capacity and help prevent overfitting.

2.2.5. Random forest

It is an ensemble ML method [18] that allows for regression and classification. It consists of a set of DTs constructed using bagging, whose variation is controlled based on the training data. In the classification process, each tree individually casts its vote for the predicted class, and the final decision is the class with the majority of votes.

2.2.6. Neural networks - multilayer perceptron

It is a neural network [19] that can have one or many hidden layers between its output and input layers, in which the connections always keep the direction of going from a lower layer to a higher one. The neurons in the same layer are not connected. Additionally, the number of features for the issue pattern is equal to the number of neurons in the input layer. The number of classes is equal to the number of neurons in the output layer. The primary objective is to optimize the network until it has sufficient parameters and strong generalization for the classification or regression job, taking into account the number of layers, neurons in each layer, and connections.

2.2.7. Naive Bayes

The Naive Bayes classifier [20] is a statistical classifier based on probabilities that depend on frequencies of values in a given dataset. Probability is an intrinsic characteristic of the data based on this dataset. This algorithm is based on Bayes' Theorem and assumes independence between input variables.

2.3. Tools

ML practitioners commonly leverage powerful tools to streamline the development and deployment of models. Utilizing tools in ML enhances efficiency by providing standardized frameworks, collaborative environments, and streamlined processes, empowering practitioners to focus on model development, experimentation, and deployment without being encumbered by manual complexities. The tools used in this work are presented below.

2.3.1. Python

Python is a high-level, general-purpose programming language with great ease of use. Verdhan [21], it is detailed that it has a great variety of useful libraries for data loading, processing, and visualization, in addition to libraries capable of executing ML algorithms for classification, prediction, and clustering. It's a versatile and widely adopted programming language due to its extensive libraries and frameworks tailored for data manipulation, analysis, and model implementation.

2.3.2. Google Colab

Google Colab is a cloud-based Jupyter Notebook environment in various web browsers. As mentioned in [22], Colab allows us to write and execute Python or R code for applications in AI, enabling tasks such as developing and training neural networks and other algorithms. It also facilitates code sharing with multiple users, ensuring simultaneous code editing.

2.3.3. Scikit Learn

Scikit-Learn is an open-source library that is under constant development and improvement. The authors in [23] say that this library provides multiple ML algorithms, for example, for supervised and unsupervised learning and other multiple scientific tools that have made it an essential tool for developing ML applications. It simplifies the implementation of various algorithms for tasks such as classification, regression, and clustering, fostering accessibility and ease of experimentation in the field.

2.4. Proposed research framework

The present work has two main stages: the first is the risk level prediction model, and the second is the comparative analysis of classification methods. The proposed research framework is demonstrated in Figure 1. The first stage is developing the risk classification model associated with software requirements. The following methods were chosen for classification: DT, LR, RF, SVM, Naive Bayes, and Neural Network - MLP.

The second stage refers to the comparative analysis of the results obtained from each classification method. Each classification method is evaluated using the risk dataset to compare and choose the most suitable classifier in this stage. The comparison uses precision, recall, and accuracy metrics to determine the quality of each classification method. Finally, rigorous statistical tests are carried out to determine if there are significant differences between the classification methods used. It allows for choosing the best risk classification methods associated with software requirements.

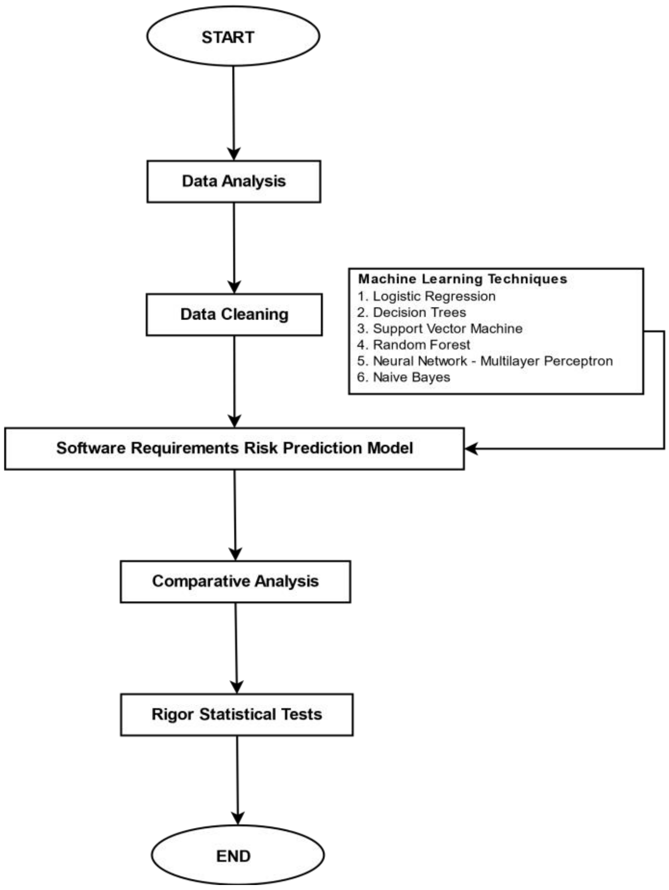

Figure 1. Proposed research framework

3. RESULTS AND DISCUSSION

In this section, the process followed by applying the six ML techniques to predict the risk level of software requirements is presented. The analysis of the data and the source of the dataset are shown. Then, the data-cleaning process is carried out. The parameters of each of the ML techniques used are presented. Finally, the results obtained are compared.

3.1. Data analysis

The dataset analyzed for risk level prediction in software requirements was obtained from the Zenodo platform [24]. The dataset authors presented the process of obtaining, validating, and using it [3]. The dataset has 13 columns representing 12 input features and one output feature, which in this case is the level of risk to be predicted. It has 299 rows or instances of risks associated with software requirements. Table 1 shows the description of the features of the dataset used.

Table 1. Description of the dataset features

Feature	Description	Type
Requirements	Textual description of the software requirement.	Text
Project Category	The category of the software project can take the following values: enterprise system, management information system, safety-critical system, and transaction processing system.	Categorical
Type	The category of the software requirement can take the following values: functional, not functional.	Categorical
Requirement Category	The category of the software requirement can take the following values: usability, functional, performance, reliability & availability, supportability, security, interfaces, constraints, safety, and standard.	Categorical
Risk Target Category	The target category of the risk can take the following values: quality, budget, personal, schedule, functional validity, performance, project complexity, people, team, planning & control, availability, requirement, resource user, time dimension, cost, organizational environment, business, design, overdrawn budget, unrealistic requirements, process, and software.	Categorical
Probability	Percentage that indicates the probability of occurrence of a risk.	Numeric
Magnitude of Risk	The magnitude of the risk for the project can take the following values: extreme, very high, high, medium, low, very low, and negligible.	Categorical
Impact	The impact of the risk for the project can take the following values: high, catastrophic, moderate, low, and insignificant.	Categorical
Dimension of Risk	The risk dimension for the project can take the following values: organizational requirements, project cost, complexity, schedule, planning and control, software requirement, estimations, requirements, user, project complexity, planning and control, organizational environment, and team.	Categorical
Affecting No of Modules	The number of software modules that affect the occurrence of a risk.	Numeric
Fixing Duration	Number of days required to fix the software if a risk occurs.	Numeric
Fix Cost	Percent of project cost needed to fix the software if a risk occurs.	Numeric
Priority	Priority with which a risk must be considered on a scale from 0 to 100.	Numeric
Risk Level	Risk levels are grouped into five categories on an ascending scale from 1 to 5. That is the output variable.	Numeric

3.2. Data cleaning

Once we have loaded the corresponding data, we analyze the influence between the features. To do this, we verify through the correlation matrix, as shown in Figure 2. The correlation matrix indicates, for example, that variables such as “Probability” and “Priority” have the highest degree of correlation with our target variable “Risk level”, while, conversely, variables like “Type” or “Requirement Category” are among those with the lowest correlation.

We continue with the verification and treatment of missing data and outliers. In the dataset, no empty data was found. We used a parametric method that validates anomalous data using the interquartile range [25], thus excluding highly atypical data, except for categorical fields, as shown in Figure 3. As a result of the adjustment, we went from initially having a total of 299 rows to 221 rows. Finally, we finish with the data transformation phase, transforming the categorical variables into numeric data from the following features: project category, type, requirement category, risk target category, magnitude of risk, impact, and dimension of risk. The Python’s map function was used to accomplish that.

3.3. Application of algorithms

We evaluated the algorithms described in the following table and their assigned parameters for the comparative analysis, as shown in Table 2. A training process with cross-validation was carried out where five folds were considered. Cross-validation is crucial in ML as it assesses the generalizability of a model by systematically validating its performance across multiple subsets of the dataset, mitigating the risk of overfitting and providing a more robust estimate of its true predictive power.

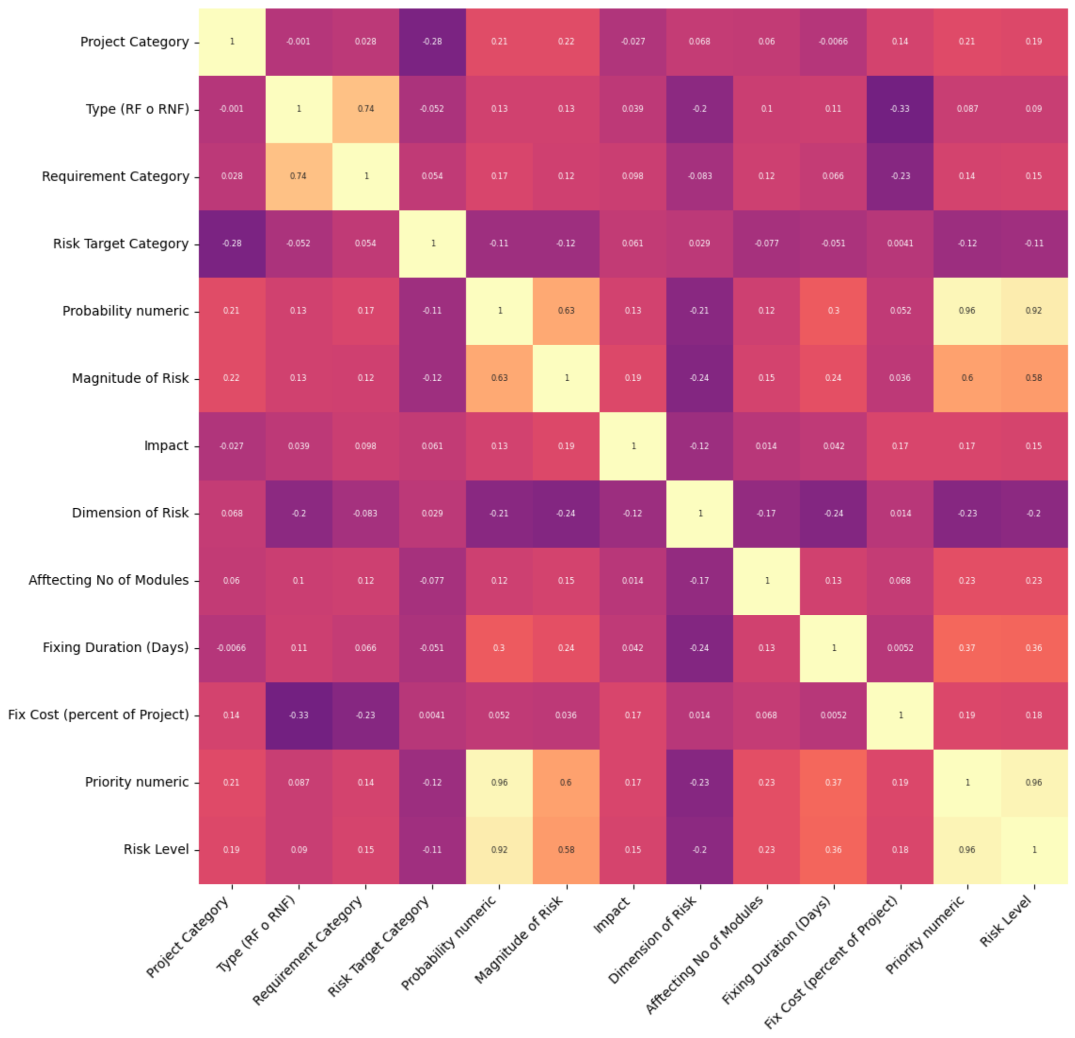

Figure 2. Correlation matrix

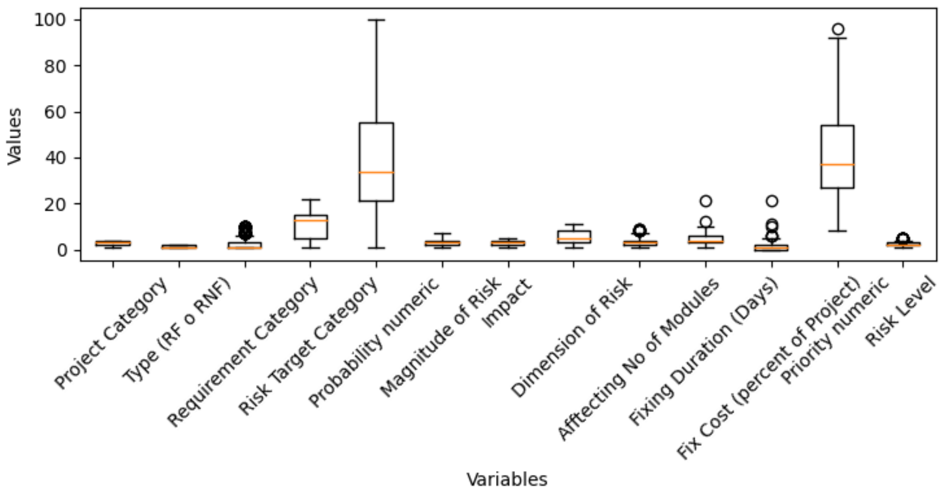

Figure 3. Box plot chart for outliers' detection

Table 2. Description of the parameters assigned to each algorithm

Algorithm	Parameters
Logistic regression	penalty: “12”, solver: “newton-cholesky”
Decision trees	criterion = “gini", min_samples_split=2, splitter=“best”
Support vector machine	decision_function_shape=“ovr”, C=1.0, kernel=“linear”
Random forest	n_estimators=6, criterion=“gini”
Neural network - multilayer perceptron	learning_rate=“adaptive”, max_iter=500, activation=“relu”, hidden_layer_sizes=(50), alpha=1e-5, solver=“sgd”
Naive Bayes	priors=None, var_smoothing= 1e-9

3.4. Comparison of results

The results of evaluating the five folds in the six models for precision, accuracy, and recall metrics are shown below. The models’ performances for the precision metric are shown in Figure 4. The DT model achieves the maximum at fold three. In contrast, the MLP model obtains the minimum at fold four.

The models’ performances for the recall metric are shown in Figure 5. The DT model achieves the maximum at folds one, two and three. In contrast, the MLP model obtains the minimum at fold one. The models’ performances for the accuracy metric are shown in Figure 6. The DT model achieves the maximum at folds three. In contrast, the MLP model obtains the minimum at fold one.

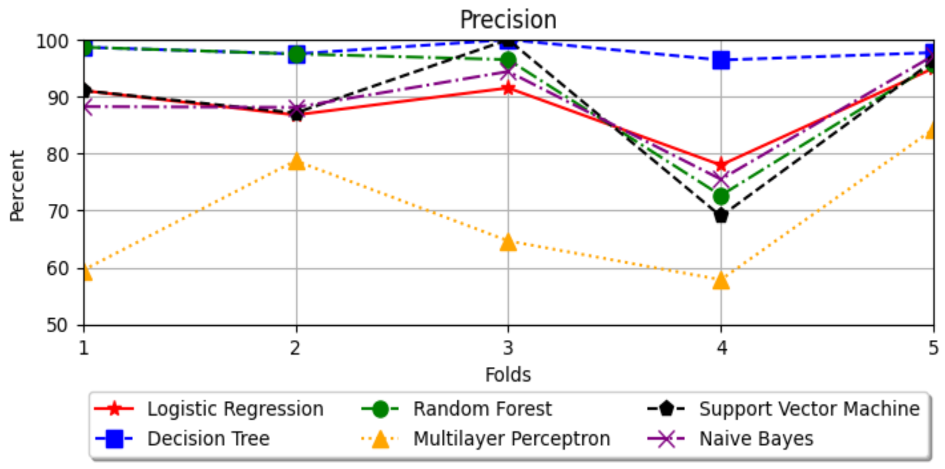

Figure 4. Chart of the precision metric in the five folds

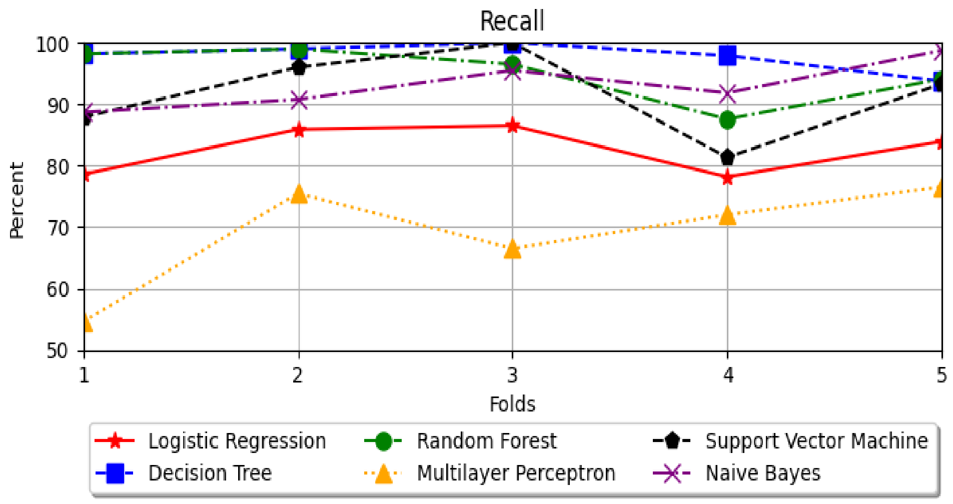

Figure 5. Chart of the recall metric in the five folds

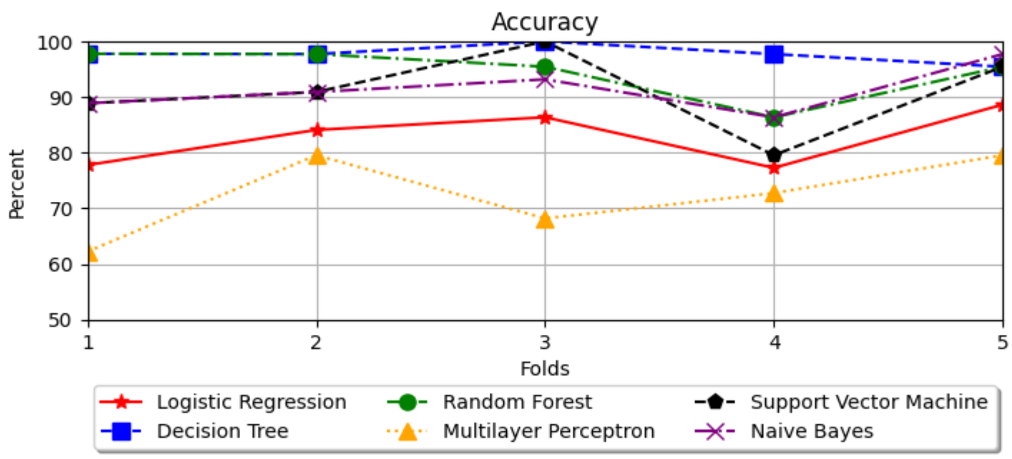

Figure 6. Chart of the accuracy metric in the five folds

The Shapiro-Wilk test was applied to the results obtained in each metric, where normality was verified in each case. Then, the Levene test was performed to confirm that the data had equal variances. In both, a tendency to pass the tests was reflected in all cases. With these two conditions met, we applied the Repeated Measures ANOVA test.

It was determined that there is a significant difference in at least two means within the group of results for the three metrics. However, deciding where these differences are located is still necessary. Therefore, we apply the Student's T-Test for paired samples with a 99% confidence interval, comparing each model pairwise and grouping those techniques that do not have significant differences. This forms groups ranging from the best results (Group 1) to the worst, as shown in Table 3. The group that yielded the highest results for the precision metric comprised the algorithms DT and RF. For the recall metric, the best algorithms are also DT and RF. Finally, for the accuracy metric, the best group is again formed by the algorithms DT and RF.

From the results obtained, it can be seen that almost all models achieve good results. However, when compared, the DT and RF algorithms consistently appear as the best results for the three metrics. It indicates that both algorithms are good ML models for predicting the risk level of software requirements. The result is consistent with the findings in [11], where DT achieved the best results among other ML techniques proposed for software requirements risk prediction.

Table 3. Description of the parameters assigned to each algorithm

Algorithm / Metric	Precision		Recall		Accuracy	
	Group	Average	Group	Average	Group	Average
Logistic regression	2	88.47%	3	82.61%	3	82.82%
Decision trees	1	96.26%	1	95.63%	1	96.36%
Support vector machine	2	88.69%	2	91.70%	2	90.95%
Random forest	1	95.08%	1	94.52%	1	95.02%
Multilayer perceptron	3	71.04%	4	67.72%	4	73.36%
Naive Bayes	2	88.69%	2	93.10%	2	91.41%

4. CONCLUSION

This research conducted a comparative analysis of six ML techniques to predict the risk level in software requirements. ML techniques are valuable tools in various fields, including predicting risk levels. The proposed techniques perform the classification or prediction of the risk level in software requirements with good results. Despite the effectiveness of ML techniques, some yielded lower results than others. Precision, accuracy, and recall metrics were used to compare the techniques. The DT and RF techniques obtained the best results, and MLP showed the worst results. The DT technique is the one that provides the best results with a confidence interval of 99%, compared to the other techniques considered in the analysis. Given the positive results obtained with various ML techniques, in future work, we intend to conduct studies that delve more deeply into other branches of AI, such as deep learning.

REFERENCES

[1] F. Hujainah, R. B. A. Bakar, M. A. Abdulgabber, and K. Z. Zamli, "Software requirements prioritisation: a systematic literature review on significance, stakeholders, techniques and challenges," *IEEE Access*, vol. 6, pp. 71497–71523, 2018, doi: 10.1109/ACCESS.2018.2881755.

[2] I. M. Keshta, M. Niazi, and M. Alshayeb, "Towards the implementation of requirements management specific practices (SP 1.1 and SP 1.2) for small- and medium-sized software development organisations," *IET Software*, vol. 14, no. 3, pp. 308–317, Jun. 2020, doi: 10.1049/iet-sen.2019.0180.

[3] Z. S. Shaukat, R. Naseem, and M. Zubair, "A dataset for software requirements risk prediction," in *2018 IEEE International Conference on Computational Science and Engineering (CSE)*, Oct. 2018, pp. 112–118, doi: 10.1109/CSE.2018.00022.

[4] C. Bilir and E. Yafez, "Project success/failure rates in Turkey," *International Journal of Information Systems and Project Management's*, vol. 9, no. 4, pp. 24–40, 2021, doi: 10.12821/ijispm090402.

[5] T. Hussain, "Risk management in software engineering: what still needs to be done," in *Intelligent Computing*, K. Arai, S. Kapoor, and R. Bhatia, Eds., Cham: Springer International Publishing, 2019, pp. 515–526, doi: 10.1007/978-3-030-01177-2_37.

[6] A. S. Abbas, A. Alabaichi, and M. A. A. Khodher, "Review Study on software risk factors," *Journal of Physics: Conference Series*, vol. 1530, p. 012007, May 2020, doi: 10.1088/1742-6596/1530/1/012007.

[7] Y. Assefa, E. Alemneh, S. Nibret, and A. Worku, "Software risk prediction at requirement and design phase : an ensemble machine learning approach," in *2023 International Conference on Information and Communication Technology for Development for Africa (ICT4DA)*, Bahir Dar, Ethiopia: IEEE, Oct. 2023, pp. 19–24, doi: 10.1109/ICT4DA59526.2023.10302250.

[8] M. Roy, N. Deb, A. Cortesi, R. Chaki, and N. Chaki, "Requirement-oriented risk management for incremental software development," *Innovations in Systems and Software Engineering*, vol. 17, no. 3, pp. 187–204, Sep. 2021, doi: 10.1007/s11334-021-00406-6.

[9] M. M. Otoom, "ABMJ: an ensemble model for risk prediction in software requirements," *International Journal of Computer Science and Network Security*, vol. 22, no. 3, pp. 710–718, Mar. 2022, doi: 10.22937/IJCSNS.2022.22.3.93.

[10] B. Khan, R. Naseem, I. Alam, I. Khan, H. Alasmary, and T. Rahman, "Analysis of tree-family machine learning techniques for risk prediction in software requirements," *IEEE Access*, vol. 10, pp. 98220–98231, 2022, doi: 10.1109/ACCESS.2022.3206382.

[11] R. Naseem *et al.*, "Empirical assessment of machine learning techniques for software requirements risk prediction," *Electronics*, vol. 10, no. 2, Art. no. 2, Jan. 2021, doi: 10.3390/electronics10020168.

[12] H. Mamman *et al.*, "Software requirement risk prediction using enhanced fuzzy induction models," *Electronics*, vol. 12, no. 18, p. 3805, Sep. 2023, doi: 10.3390/electronics12183805.

[13] B. O. Akumba, S. U. Otor, I. Agaji, and B. T. Akumba, "A predictive risk model for software projects'requirement gathering phase," *International Journal of Innovative Science and Research Technology*, vol. 5, no. 6, pp. 231–236, Jun. 2020, doi: 10.38124/IJISRT20JUN066.

[14] T. Freiesleben and T. Grote, "Beyond generalization: a theory of robustness in machine learning," *Synthese*, vol. 202, no. 4, 2023, doi: 10.1007/s11229-023-04334-9.

[15] J. J. Wimhurst, J. S. Greene, and J. Koch, "Predicting commercial wind farm site suitability in the conterminous United States using a logistic regression model," *Applied Energy*, vol. 352, 2023, doi: 10.1016/j.apenergy.2023.121880.

[16] S. Ma and J. Zhai, "Big data decision tree for continuous-valued attributes based on unbalanced cut points," *Journal Big Data*, vol. 10, no. 1, 2023, doi: 10.1186/s40537-023-00816-2.

[17] Q. Wang *et al.*, "A novel method for petroleum and natural gas resource potential evaluation and prediction by support vector machines (SVM)," *Applied Energy*, vol. 351, 2023, doi: 10.1016/j.apenergy.2023.121836.

[18] Q. Yun, X. Wang, C. Yao, and H. Wang, "Random forest method for estimation of brake specific fuel consumption," *Scientific Reports*, vol. 13, no. 1, 2023, doi: 10.1038/s41598-023-45026-1.

[19] P. Mohapatra *et al.*, "Artificial neural network based prediction and optimization of centelloside content in Centella asiatica: A comparison between multilayer perceptron (MLP) and radial basis function (RBF) algorithms for soil and climatic parameter," *South African Journal of Botany*, vol. 160, pp. 571–585, 2023, doi: 10.1016/j.sajb.2023.07.019.

[20] A. Rawal and B. Lal, "Predictive model for admission uncertainty in high education using Naïve Bayes classifier," *Journal of Indian Business Research*, vol. 15, no. 2, pp. 262–277, 2023, doi: 10.1108/JIBR-08-2022-0209.

[21] V. Verdhan, "*Supervised Learning with Python: Concepts and Practical Implementation Using Python*," in Supervised Learning with Python: Concepts and Practical Implementation Using Python, 2020, p. 372, doi: 10.1007/978-1-4842-6156-9.

[22] D. Paper, "*State-of-the-Art deep learning models in TensorFlow: Modern machine learning in the google colab ecosystem*," in State-of-the-Art Deep Learning Models in TensorFlow: Modern Machine Learning in the Google Colab Ecosystem, 2021, p. 374, doi: 10.1007/978-1-4842-7341-8.

[23] O. Kramer, "Scikit-learn," in Machine Learning for Evolution Strategies, O. Kramer, Ed., *in Studies in Big Data, Cham: Springer International Publishing*, 2016, pp. 45–53. doi: 10.1007/978-3-319-33383-0_5.

[24] Z. Shaukat, R. Naseem, and M. Zubair, "Software requirement risk prediction dataset," *Zenodo*, Mar. 29, 2018, doi: 10.5281/zenodo.1209601.

[25] Y. H. Dovoedo and S. Chakraborti, "Boxplot-based outlier detection for the location-scale family," *Communications in Statistics - Simulation and Computation*, vol. 44, no. 6, pp. 1492–1513, 2015, doi: 10.1080/03610918.2013.813037.

BIOGRAPHIES OF AUTHORS

Yasiel Pérez Vera is a Lecturer Professor at the School of Software Engineering, Universidad La Salle, Arequipa Peru. He received a B.Sc. degree in Informatics Science from the University of Informatics Science, La Habana, Cuba, an M.Sc. degree in Project Management from the University of Informatics Science, La Habana, Cuba, and a Ph.D. degree in Business Administration (DBA). His research areas are artificial intelligence, project management, ML, and pattern recognition. He used to hold several administrative posts with the School of Software Engineering, Universidad La Salle, Arequipa Peru, including the Head of Program of Software Engineering, and now is the Dean of Postgraduate Studies at Universidad La Salle, Arequipa, Peru. He can be contacted at email: yperez@ulasalle.edu.pe.

Álvaro Fernández del Carpio received a Ph.D. degree in Computer Science and Technology from the Carlos III University of Madrid, Spain, and an Official Master in Computer Science and Technology from the Carlos III University of Madrid, Spain. He obtained a B.Sc. degree in Systems Engineering from the Santa María Catholic University of Arequipa, Peru. He completed Master's in Software Engineering at the Universidad Católica Santa María de Arequipa, Peru. General Director of the Latin American Institute for Research, Innovation and Technological Studies ILIIET, a company dedicated to research, training, and development of innovation in software Research professor at the La Salle University and Catholic University of Santa María. He can be contacted at email: alfernandez@ulasalle.edu.pe.