[1]Sonali Idate

[2]T. Srinivasa Rao

[3]Milind Gayakwad

[4]Priyanka Paygude

[5]Prashant Chavan

[6]Rajendra Pawar

[7]Kalyani Kadam

Performance analysis of Machine Learning Algorithms to classify Software Requirements

Abstract: - Building a software system requires a clear understanding of its purpose and its operational characteristics. Functional requirements establish the system's purpose, while non-functional requirements define its operational performance aspects. It is essential to identify and classify requirements accurately to develop reliable software. In this research paper, we aim to classify functional and non-functional software requirements using different Machine Learning algorithms and techniques. We utilized four popular classification models, including Logistic Regression, Support Vector Machines, Decision Tree, and Random Forest Multi-layer Perceptron Neural Network to classify the requirements. To enhance the accuracy of the classification, we also applied a technique based on cosine similarity to verify if the custom string provided as input is related to software requirements. The addition of cosine similarity improved the accuracy of classification and reduced the misclassification of non-requirements.

Keywords: Functional requirement, Non-functional requirement, machine learning.

I. INTRODUCTION

Software requirements classification is a crucial aspect of software engineering, where the requirements are divided into two main categories, i.e., functional, and non-functional [1].

Functional requirements are the core functionality that a software system must provide. These requirements specify what the system should do and how the system should do it. For example, a functional requirement for an online shopping website might be the ability to search for products, add them to a shopping cart, and check out [2][3].

Non-functional requirements, on the other hand, are the qualities and characteristics of the system that are not directly related to the functionality of the system [4][5]. These requirements describe how well the system must perform, how easy it is to use, and how reliable it must be. For example, a non-functional requirement for an online shopping website might be that it must be able to handle at least 1000 concurrent users [6][7].

Classifying the requirements accurately is essential for developing a reliable software system. However, manually classifying the requirements can be time-consuming and error-prone [8][9]. Therefore, researchers have proposed various approaches to automate this process, including machine learning algorithms. In this research paper, we aim to classify functional and non-functional software requirements using different algorithms: Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), and Multilayer Perceptron (MLP) Neural Networks to test how classification algorithms perform and how it can improve [10][11]. We present an effective methodology for pre-processing the text data and applying machine learning algorithms to classify the requirements [12][13]. We compare the performance of the algorithms and provide insights into the accuracy and limitations of each algorithm [14][15]. Our research contributes to the development of automated

[1,3,4,5] Department of Information Technology, Bharati Vidyapeeth (Deemed to be University) College of Engineering, Pune

[2]Associate Professor, Department of Computer Science, Gitam Institute of Technology, GITAM University, India

[6]Associate Professor, Department of computer science and engineering, School of computing MIT Art Design and Technology University, Loni Kalbhor Pune

[7]Assistant Professor, Artificial Intelligence and Machine Learning Department, Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, India

[4*]Corresponding author: pspaygude@bvucoep.edu.in

approaches for software requirements classification, which can lead to more efficient and reliable software engineering practices [16][17].

In conclusion, the classification of software requirements as functional and non-functional is a necessary and fundamental step in the software development process [18][19]. It helps to ensure that all the requirements are captured and prioritized, and that the system is built to meet the needs of the stakeholders [20][21]. This classification also helps to guarantee the quality of the system, such as its ease of use and reliability, making it an essential process for all software development projects [22][23][25].

II. LITERATURE SURVEY

Software requirements are essential for developing software products that meet users' needs and expectations. Therefore, software requirements classification is a critical step in the software development process. Several studies have been conducted to automate the software requirements classification process using various machine learning techniques [26][27]. This section presents a review of the existing literature on software requirements classification. One of the earliest works in this field was conducted by [28], who used decision trees and Naive Bayes algorithms to classify software requirements into functional and non-functional requirements [29]. They achieved an accuracy of 86.4% using the Naive Bayes algorithm [30].

In another study, [31] used a Support Vector Machine (SVM) algorithm to classify software requirements. They evaluated the performance of the algorithm with three different feature extraction techniques: Term Frequency-Inverse Document Frequency (TF-IDF), Latent Semantic Analysis (LSA), and Latent Dirichlet Allocation (LDA). Their results showed that the SVM algorithm combined with the TF-IDF feature extraction technique outperformed the other two techniques in terms of accuracy. More recently, [32] proposed a method for software requirements classification based on ensemble learning. They used a combination of Naive Bayes, SVM, and Random Forest algorithms to classify software requirements into functional and non-functional requirements. They achieved an accuracy of 92.55% using the ensemble learning approach [33][34][35].

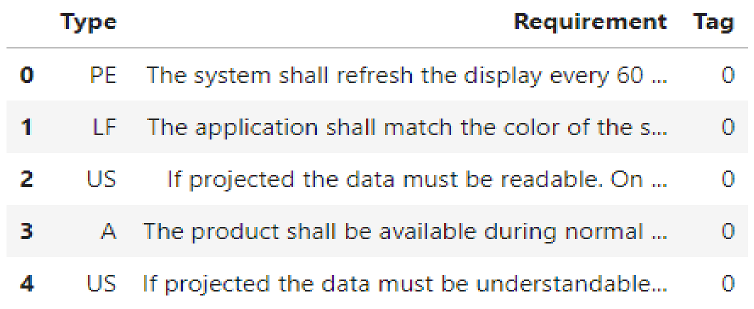
Beyond traditional machine learning methods, recent research has explored using deep learning for software requirements classification. For instance, [36][37][38] proposed a method based on using Convolutional Neural Networks (CNNs), a type of deep learning, to automatically classify software requirements as functional or non-functional. Their findings suggest that this approach is more effective than traditional machine learning methods for this task**,** achieving an accuracy of 97.4%. The study [39][40][41] investigates an approach that leverages both Natural Language Processing (NLP) techniques for analyzing text and deep learning algorithms for pattern recognition to classify software requirements [42][43]. It utilizes word embedding, which captures the semantic meaning of words, to represent requirements as numerical data [44][45][46]. This data is then fed into a Convolutional Neural Network (CNN), a powerful deep learning architecture, to automatically categorize requirements as functional or non-functional [46][47]. Their findings demonstrate that this method surpasses the performance of traditional machine learning algorithms in this task, achieving an accuracy of 91.2%. In a recent paper [48][49][50], the classification of nonfunctional requirements was the priority, and the feature extraction technique employed was Bag of Word (BoW)[51][52][53].

In the proposed research work, we implemented our method using scikit-learn [54][55], a well-established Python library for machine learning tasks [56][57]. We collected a dataset of software requirements from a public source and preprocessed the dataset to remove any missing values [58][59]. We then used the CountVectorizer and Tfidf Transformer classes to convert the textual data into numerical data [60][61]. We prepared the models for training by separating the dataset into two groups. The first group, the training set, was used to train the models [62][63]. The second group, the testing set, was used later to assess how well the trained models performed [64][65][66]. We then evaluated the performance of our models on the testing set [67][68].

III. METHODOLOGY

The methodology used in this project is a combination of natural language processing and supervised machine learning [69][70]. The data used during the research is a dataset of labelled software requirements of PROMISE and from requirement documents of PURE repository consisting primarily of two columns: requirement type and requirement text [71][72][73].

The first step in this methodology is data pre-processing and cleaning[74][75]. This includes removing any missing values and converting the text data into a numerical representation. A new column, “Tag”, is created, which is used to track the functional and non-function requirements, i.e. if a requirement is functional, it is given a tag of 1; otherwise, a tag of 0. Since the machine only understands numerical data, the textual data (requirement text column) of the dataset is vectorized in the next step. This is done using the CountVectorizer and TfidfTransformer methods, which are popular NLP techniques and are provided by the scikit-learn library.



	Type	Requirement	Tag
0	PE	The system shall refresh the display every 60 ...	0
1	LF	The application shall match the color of the s...	0
2	US	If projected the data must be readable. On ...	0
3	A	The product shall be available during normal ...	0
4	US	If projected the data must be understandable...	0

Figure. 1: Dataset after introducing the tag column

After cleaning and preparing the dataset, we divided it into training and testing sets using the scikit-learn'strain_test_split function. The training data acts as a teaching tool for the machine learning models, while the testing data helps us assess how accurate each model is.

The next step is to apply machine learning algorithms to classify the software requirements as functional or non-functional. The proposed method explored a range of machine-learning techniques to tackle the classification problem, including Support Vector Machines, Logistic Regression, Random Forests, Decision Trees, and culminating in Neural Networks**.** We implemented machine learning algorithms from the scikit-learn library to tackle this classification task. Notably, neural networks were implemented using TensorFlow. To evaluate their effectiveness, each algorithm was trained on a designated training dataset. Subsequently, their performance was assessed using the 'score' function on a separate testing dataset. This approach allowed us to compare the accuracy. Finally, the results are evaluated and compared to determine which algorithm is the most accurate. The accuracy of each algorithm is calculated using the score function provided by scikit-learn. The algorithm with the highest accuracy is considered the most suitable for classifying software requirements as functional or non-functional (at least for this dataset).

In summary, the methodology used is a combination of natural language processing and supervised machine learning. The data pre-processing involves cleaning and processing the data before splitting it into training and testing sets. Subsequently, various machine learning algorithms are employed to classify the software requirements, and the accuracy of each algorithm is evaluated on the testing data. Our evaluation focuses on identifying the most accurate algorithm for categorizing software requirements to improve clarity by distinguishing between functionalities and overall qualities. The algorithm that demonstrates the highest performance in correctly distinguishing between these requirement types will be chosen for this task. The methodology followed in this research is simple and easy to replicate for other similar tasks and datasets, providing an efficient way to classify software requirements.

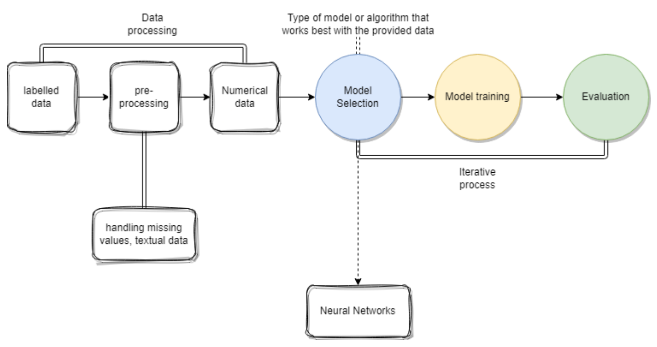


Figure 2: Supervised learning framework

Algorithms used are as follows:

3.1. Logistic Regression

Among statistical algorithms, Logistic Regression stands out for its effectiveness in binary classification. This means the algorithm is designed to predict outcomes that can be strictly categorized into two distinct classes.

In our case, the output variable is either 0 (non-functional requirement) or 1 (functional requirement). LR is a powerful technique for estimating the likelihood of an event happening. It doesn't simply predict a yes or no outcome, but rather calculates the probability of that outcome falling within one of two categories, i.e., the probability that a requirement is functional or non-functional.

The mathematical formula for logistic regression is as follows:

$$logit\left(\pi(x)\right) = log\ log\ \left(\frac{\pi(x)}{1-\pi(x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \quad (1)$$

Here, logit($\pi(x)$) is the log odds of the probability that a requirement is functional. $\pi(x)$ is the probability that a requirement is functional, and β0, β1, … βp are the model coefficients.

The logistic function is used to transform the log odds into the predicted probability:

$$\pi(x) = \frac{1}{1 + e^{-logit(\pi(x))}} \qquad (2)$$

In our code, we use the LogisticRegression() function from the scikit-learn library. This function implements the logistic regression algorithm using maximum likelihood estimation. The regularization parameter C is used to control the amount of regularization applied to the model. The regularization term helps prevent overfitting by penalizing large coefficient values.

We fit the logistic regression model to the training data using the fit() function and then use the predict() function to make predictions on the test data. To assess how well the model performs, we track its accuracy score. This score reflects the percentage of software requirements the model categorized correctly as functional or non-functional.

In addition, we use the term frequency-inverse document frequency (TF-IDF) technique to preprocess the requirement text data. The CountVectorizer () function is used to convert the text data into a matrix of term frequencies, and the TfidfTransformer() function is used to transform the term frequency matrix into a TF-IDF matrix, which is used as the input to the logistic regression model.

3.2. Support Vector Machine

SVMs are powerful models for categorizing data. Primarily used for classification tasks, SVMs are a type of supervised learning algorithm, meaning they learn from labeled data to make predictions. In the code, we used a linear kernel for the SVM model, which means that the algorithm tries to separate two classes by seeking a linear decision boundary line between them in two-dimensional space, while in a higher-dimensional space, it can be a hyperplane.

The algorithm strategically plots this hyperplane to maximize the margin; This metric indicates how far apart the most difficult points to classify are from the decision boundary. This emphasis on maximizing the margin allows SVM to establish the most robust separation possible between the two data classes. By maximizing the margin, the SVM algorithm ensures that the decision boundary is as far away from the data points as possible, which makes the classifier more robust to noise and outliers.

Mathematically, the SVM algorithm tries to solve the following optimization problem:

$$min_{w,b}\frac{1}{2}||w||^2 \qquad (3)$$

subject to $y_i(w^{\top x_i} + b) \geq 1\ for\ i = 1, \dots, n.$

Where x_i is the ith data point, $y_i \in -1{,}1$ is its corresponding label, w and b are the hyperplane parameters, and n is the number of data points.

The optimization problem tries to find the hyperplane with the largest margin, However, it must also ensure that all data points are assigned to the correct category. The margin is given by $\frac{2}{|w|}$, which means that we want to minimize $|w|^2$ in order to maximize the margin.

To implement the SVM algorithm in our code, we leveraged the SVC class readily available within the scikit-learn library. This class specifically handles SVMs with a linear kernel. The kernel parameter can be set to 'linear' to use a linear kernel. The C parameter controls how much emphasis to put on creating a wide margin between the two data categories and how much to prioritize correctly classifying all the data points. A higher C value prioritizes accuracy, potentially leading to a slightly smaller margin. Conversely, a lower C value focuses on a wider margin, but this might come at the cost of some misclassified data points.

3.3. Decision Tree

DT is a supervised learning algorithm applicable to both classification and regression. They function like a flowchart, with internal nodes posing questions about specific data attributes. Each branch represents a possible answer to that question, ultimately leading to a leaf node that reveals the final classification or predicted value.

The decision tree algorithm starts with the root node, which contains the entire training dataset. The algorithm selects an attribute to split the dataset into two or more subsets based on the information gain or Gini index. Information gain measures the reduction in entropy or randomness after splitting the dataset on the attribute, while the Gini index measures the impurity of the dataset.

The attribute with the highest information gain or lowest Gini index is selected to split the dataset. The splitting process continues recursively until the leaf nodes are pure, meaning that they contain only one class label or a majority of one class label. The decision tree algorithm has several advantages, such as easy interpretation, the ability to handle both categorical and numerical data, and robustness to outliers and noise. However, it is susceptible to overfitting. This occurs when the model becomes overly intricate and starts to capture irrelevant details or "noise" within the training data. This can negatively impact the model's ability to generalize well to unseen data.

The mathematical formulas used in decision tree algorithms are entropy and information gain. Entropy measures the randomness or impurity of a dataset, while information gain measures the reduction in entropy after splitting the dataset on an attribute.

The entropy formula is as follows:

$$Entropy(S) = -\sum_{i=1}^{c} \quad p_i(p_i) \qquad (4)$$

where S is the dataset, c is the number of classes, and p_i is the proportion of instances in class i

The information gain formula is as follows:

$$Gain(S,A) = Entropy(S) - \sum_{v\in values(A)} \quad \frac{|S_v|}{|S|} Entropy(S_v) \qquad (5)$$

where A is an attribute, $values(A)$ is the set of values for attribute A, S_v is the subset of S for which attribute A has value v, and |S| and |Sv| are the sizes of the datasets.

In the code, the decision tree algorithm is implemented using the Decision Tree Classifier class from the sklearn library. The algorithm uses the Gini index as thecriterion for splitting and can handle both categorical and numerical data. The algorithm also allows for pruning to prevent overfitting.

3.4. Random Forest

Random Forest is an ensemble-based learning algorithm, it builds upon the strengths of decision trees by combining multiple trees into a powerful ensemble. This ensemble approach leverages the predictions from each individual tree to arrive at a more accurate and robust final prediction. This technique helps to mitigate the overfitting issue that can sometimes plague single decision trees. Each tree in the forest is trained on a different subset of data points randomly sampled from the overall training set. Additionally, for each split within a tree, only a random subset of features is considered. This randomness helps prevent the trees from becoming overly reliant on specific features or data points in the training data, ultimately leading to a model that generalizes better to unseen data.

Following are the key steps involved in training a RF model:

1. A random subset of the training data is sampled with replacement (bootstrap sample).
2. Each tree in the Random Forest is built using a random selection of features and a random sample of the data points, to prevent the trees from becoming too focused on specific features or patterns in the training data.
3. Repeat Steps 1 and 2 to create multiple decision trees.

During prediction in a Random Forest model, each individual decision tree casts its own vote for the class a new instance belongs to. The final classification for that instance is determined by the most popular vote (the mode) amongst all the trees in the forest. This ensemble approach leverages the collective wisdom of the forest to deliver a more robust and accurate prediction.

The RF algorithm uses the following mathematical formula to calculate the impurity of a node in a decision tree:

$$I(node) = \sum_{i=1}^{c} p_i log_2(p_i) \qquad (6)$$

a decision tree node that contains data points, eachbelonging to one of several categories (c).

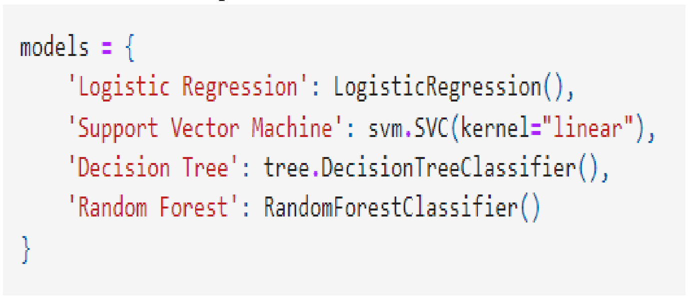
This formula considers the proportion of data points within that node that fall into each category p_i . By calculating a value called entropy, it helps us understand how mixed up the data points in that node are in terms of their categories.

Random Forests make strategic choices when splitting data within a decision tree. To achieve this, they rely on a metric called the Gini index or entropy. These metrics assess how well-mixed the data points are at a particular node in the tree. The algorithm aims to minimize this impurity, essentially creating a split that leads to the most homogenous child nodes possible. The Gini index is measure of impurity that is calculated using the following formula:

$$\mathrm{I_G(node)} = \sum_{\mathrm{i=1}}^{\mathrm{c}} \mathrm{p_i(1\text{-}p_i)} \qquad (7)$$

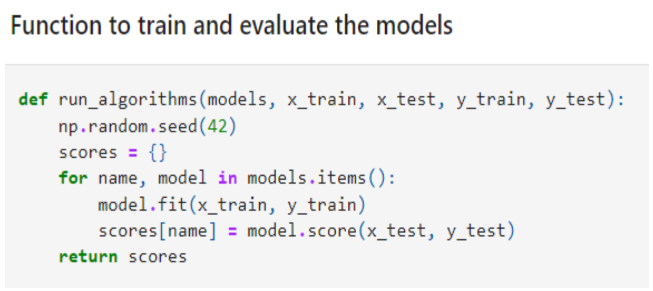
where c is the number of categories, and p_i is the proportion of samples in the node that belong to class i.

The Random Forest algorithm builds a team of decision trees, each making its own prediction. The final prediction is determined by the most popular choice among all the trees, resulting in a more robust outcome. For better performance out of the Random Forest model, we can fine-tune certain settings. These settings, called hyper parameters, include the number of trees in the forest, the maximum complexity allowed for each tree and the number of features considered at each decision point. By adjusting these parameters, we can optimize the model's ability to identify patterns and make accurate predictions.



```python
models = {
    'Logistic Regression': LogisticRegression(),
    'Support Vector Machine': svm.SVC(kernel="linear"),
    'Decision Tree': tree.DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier()
}
```

Figure 3: Python dictionary to store all the models



Function to train and evaluate the models

```python
def run_algorithms(models, x_train, x_test, y_train, y_test):
    np.random.seed(42)
    scores = {}
    for name, model in models.items():
        model.fit(x_train, y_train)
        scores[name] = model.score(x_test, y_test)
    return scores
```

Figure 4: Function to run all the models stored in the dictionary

3.i. Neural Networks

In addition to exploring the performance of the four aforementioned models, we also investigated the effectiveness of neural networks for this task. Neural networks have garnered significant attention due to their impressive results in various NLP applications, including text classification In this research, we used neuralnetworks to classify functional and non-functional software requirements and compared the results with four other machine learning algorithms: logistic regression, support vector machine, decision tree, and random forest.

In the research, a multi-layer perceptron (MLP) neural network is used to classify the software requirements. The MLP is a feedforward neural network, where the neurons are arranged in layers and the input flows from the input layer to the output layer. MLPs are like advanced learning machines. With their multiple hidden layers, they can discover subtle patterns and connections between the starting information and the final outcome, even when those connections are not simple or obvious.

To build our neural network model, we first transformed the textual data of software requirements into numerical vectors using the same CountVectorizer and TfidfTransformer techniques used in the other machine learning models. We then converted the transformed data to numpy arrays and built a sequential neural network model using the Keras library in TensorFlow. Our model consisted of three fully connected layers, each with 64 units and a ReLU activation function. We included a dropout layer with a rate of 0.5 after each fully connected layer to reduce over fitting. The final layer of the MLP contained just one element. This element used a sigmoid activation function to translate the processed data into a single probability score between 0 and 1. This score reflected the likelihood of a given requirement being classifiedas functional.

While training, a Multilayer Perceptron (MLP) fine-tunes the weights between neurons using a technique called backpropagation. Backpropagation calculates how sensitive the network's overall error is to each weight. It then adjusts these weight in the opposite direction of the error, effectively helping the MLP learn and minimize its loss. In a MLP, hidden layers use a ReLU function to activate neurons, while the final layer uses a Sigmoid function to turn the output into a probability-like value. The ReLU activation function is defined as:

$$f(x) = max(0, x) \qquad (8)$$

where x is the input to the neuron. The ReLU function sets any negative values of x to zero, which helps the network learn non-linear relationships. The Sigmoid activation function is defined as:

$$f(x) = \frac{1}{(1 + e^{-x})} \qquad (9)$$

where x is the input to the neuron. The Sigmoid function squashes the output between 0 and 1, which makes it suitable for binary classification problems.

During training, the model utilizes the binary cross-entropy loss function to measure prediction errors for binary classification tasks. The Adam optimizer facilitates efficient weight updates, and the accuracy metric allows us to monitor the model's ability to make correct predictions. The model underwent training for five epochs, where each epoch involved processing the entire dataset in batches of 32 samples.

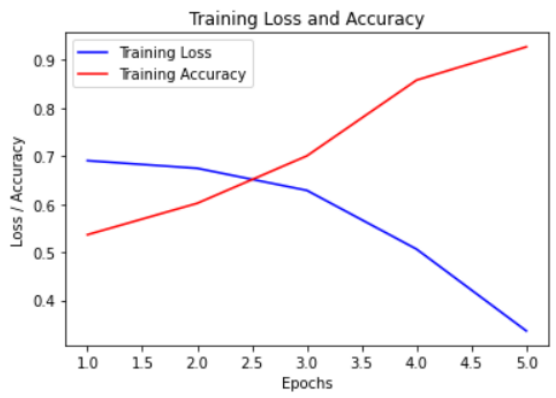


Figure 5: Loss and Accuracy plot for each epoch.

The most often used metric is, cosine similarity quantifies how similar two vectors are. It can be used to represent the degree of similarity between two documents inside the text categorization domain. The range of values it accepts is 0 to 1, where 0 denotes no similarity between the documents and 1 denotes exact correspondence [18].

In requirement classification, MLP classifiers benefit from considering both a category's estimated value and its similarity to a test requirement. This combined approach, as shown in Figure 5, leads to improved performance with increasing accuracy and decreasing loss as the number of training epochs grows.

After training, we evaluated the model on the test data and obtained an accuracy of 0.9285, which is slightly higher than the best performing machine learning algorithm, SVM. The results suggest that neural networks can be a viable approach for classifying functional and non-functional software requirements and may outperform other traditional machine learning algorithms.

Table 1 Performance of classifier.

Classifier	Feature Extraction Technique	Accuracy
Logistic Regression (LR)	Count Vectors	0.785714
	Word Level TF-IDF	0.833333
SVM	Count Vectors	0.916667
	Word Level TF-IDF	0.892857
Decision Tree (DT)	Count Vectors	0.757143
	Word Level TF-IDF	0.780952
Random Forest (RF)	Count Vectors	0.790142
	Word Level TF-IDF	0.824877
MLP Classifier	Count Vectors	0.928571
	Word Level TF-IDF	0.9166670

As shown in table 1, we have demonstrated the effectiveness of neural networks in software requirement classification and how they can be used as an alternative to traditional machine learning algorithms.

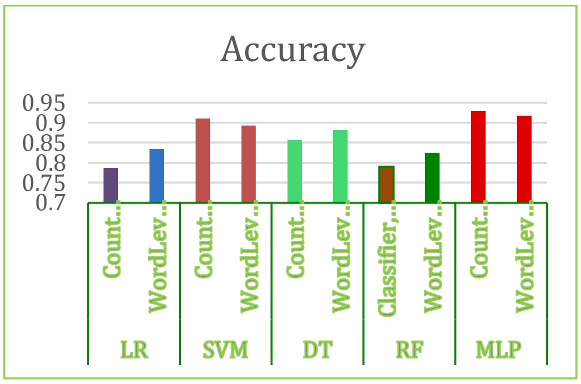


Figure 6: Accuracy of classifiers with respect to word embedding techniques

Further studies can investigate the potential of employing different neural network structures and fine-tune settings to enhance its performance.

IV. DISCUSSION

In this research, we aimed to classify software requirements as functional or non-functional using machine learning algorithms. In the survey of research papers [19-30] following gaps has been identified 1. Application machine learning models have been performed either for classification of functional requirement or non-functional requirement. 2. Single word embedding, or feature extraction techniques are used 3. Performance measures used are accuracy, recall and Score. However, in the proposed work four algorithms: Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest. Along with that we also tested the performance of the Neural Networks in comparison with the best of other four algorithms that is SVM.

Among the four algorithms SVM and MLP neural network were able to achieve a high accuracy in classifying software requirements as functional or non-functional. Neural Networks and SVM performed best with an accuracy of 0.928 and 0.9166 respectively. Logistic Regression performed with an accuracy of 0.83. Random Forest also competed well, reaching an accuracy of 82.4. Although Decision Tree achieved the lowest accuracy, it still had a good performance in classifying the requirements. The reason for the different accuracy rates for each algorithm is due to their specific strengths and weaknesses, the complexity of the data and the suitability of the algorithms for the task at hand.

Further SVM performance is compared with the MLP neural network based on cosine similarity.

While traditional algorithms like SVM, DT, RF, and LR rely on various methods for classification, Multilayer Perceptron (MLP) neural networks excel in this task. This study demonstrates that MLPs achieve superior performance.

Overall, the findings of this research and the existing literature demonstrate that machine learning algorithms can be effectively used to classify software requirements as functional or non-functional. The results of this project indicate that the Random Forest, SVM and Logistic regression algorithms performed well in terms of accuracy, and they can be considered as suitable algorithms for this task. However, it is important to note that the suitability of the algorithm will depend on the complexity of the data, the size of the dataset and the specific problem at hand.

Funding

This research was funded by “Research Support Fund of Symbiosis International (Deemed University), Pune, Maharashtra, India”.

REFERENCES

[1] E.D. Canedo, B.C. Mendes, “Software Requirements Classification Using Machine Learning Algorithms”, Entropy, 2020, vol. 22(9):1057.

[2] Z. Kurtanovic, W. Maalej, “Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning”, *IEEE Intl. Requirements Engineering Conf.*, 2017.

[3] Y. Li and D. Rine, "Automatic classification of software requirements using machine learning techniques," in Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering, 2010, pp. 379-384.

[4] X. Liu and Y. Zhang, "Support vector machines-based software requirements classification with feature extraction techniques," in Proceedings of the 11th IEEE International Conference on Computer and Information Technology, 2011, pp. 18-23.

[5] S. Srivastava and S. Singh, "Ensemble learning approach for software requirements classification," in Proceedings of the 2nd International Conference on Information Management in the Knowledge Economy, 2014, pp. 1-8.

[6] M. Samadzadeh, E. Fazl-Ersi, and M. Asadpour, "Software requirements classification using convolutional neural networks," Journal of Intelligent Information Systems, vol. 52, no. 2, pp. 237-249, 2019.

[7] S. H. Shahbaz Kia and M. A. Zamani, "Software requirements classification using hybrid natural language processing.

[8] Lin, H., Liu, Y., & Chen, G. (2019). A machine learning approach to functional and non-functional requirements classification. Proceedings of the 2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), 428-433.

[9] Chen, Y., Li, Z., Li, Y., & Li, M. (2018). Using machine learning to classify software requirements based on requirement attributes. Journal of Ambient Intelligence and Humanized Computing, 9(3), 559-566.

[10] Ferrari, Alessio, Giorgio Oronzo Spagnolo, and Stefania Gnesi. "Pure: A dataset of public requirements documents." 2017 IEEE 25th International Requirements Engineering Conference (RE). IEEE, 2017.

[11] Hosmer, D. E., &Lemeshow, S. (1989). Applied logistic regression. New York: Wiley.

[12] Gayakwad, Milind, Suhas Patil, Rahul Joshi, Sudhanshu Gonge, and Sandeep Dwarkanath Pande. “Credibility Evaluation of User-Generated Content Using Novel Multinomial Classification Technique.” International Journal on Recent and Innovation Trends in Computing and Communication 10 (2s): 151–57.

[13] Rajendra Pawar et al.,“ Farmer Buddy-Plant Leaf Disease Detection on Android Phone” In International Journal of Research and Analytical Reviews. Vol 6 (2), 874-879

[14] Gayakwad, Milind, Suhas Patil, Amol Kadam, Shashank Joshi, Ketan Kotecha, Rahul Joshi, Sharnil Pandya, et al. 2022. “Credibility Analysis of User-Designed Content Using Machine Learning Techniques.” Applied System Innovation 5 (2): 43.

[15] Harane, Swati T., Gajanan Bhole, and Milind Gayakwad. 2017. “SECURE SEARCH OVER ENCRYPTED DATA TECHNIQUES: SURVEY.” International Journal of Advanced Research in Computer Science 8 (7).

[16] Kavita Shevale, Gajanan Bhole, Milind Gayakwad. 2017. “Literature Review on Probabilistic Threshold Query on Uncertain Data.” International Journal of Current Research and Review 9 (6): 52482–84

[17] Mahamat Adam Boukhari, Milind Gayakwad. 2019. “An Experimental Technique on Fake News Detection in Online Social Media.” International Journal of Innovative Technology and Exploring Engineering (IJITEE) 8 (8S3): 526–30.

[18] Maurya, Maruti, and Milind Gayakwad. 2020. “People, Technologies, and Organizations Interactions in a Social Commerce Era.” In Proceeding of the International Conference on Computer Networks, Big Data and IoT (ICCBI-2018), 836–49. Springer International Publishing.

[19] Milind Gayakwad, B. D. Phulpagar. 2013. “Requirement Specific Search.” IJARCSSE 3 (11): 121.

[20] Panicker, Aishwarya, Milind Gayakwad, Sandeep Vanjale, Pramod Jadhav, Prakash Devale, and Suhas Patil. n.d. “Fake News Detection Using Machine Learning Framework.”

[21] [21] Gonge, S. et al. (2023). A Comparative Study of DWT and DCT Along with AES Techniques for Safety Transmission of Digital Bank Cheque Image. In: Chaubey, N., Thampi, S.M., Jhanjhi, N.Z., Parikh, S., Amin, K. (eds) Computing Science, Communication and Security. COMS2 2023. Communications in Computer and Information Science, vol 1861. Springer, Cham. https://doi.org/10.1007/978-3-031-40564-8_6

[22] Self-Driving Electrical Car Simulation using Mutation and DNN Paygude, P. Idate, S. Gayakwad, M. Kadam, K. Shinde, A.

[23] SSRG International Journal of Electronics and Communication Engineering, 2023, 10(6), pp. 27–34

[24] Probing to Reduce Operational Losses in NRW by using IoT Hingmire, S. Paygude, P. Gayakwad, M. Devale, P. SSRG International Journal of Electronics and Communication Engineering, 2023, 10(6), pp. 23–32

[25] Paygude, P.., Singh, A.., Tripathi, E.., Priya, S.., Gayakwad, M.., Chavan, P.., Chaudhary, S.., Joshi, R.., & Kotecha, K.. (2023).

[26] A Parameter-Based Comparative Study of Deep Learning Algorithms for Stock Price Prediction. International Journal on Recent and Innovation Trends in Computing and Communication, 11(7s), 138–146. https://doi.org/10.17762/ijritcc.v11i7s.6985

[27] Dixit, B., Pawar, R. G., Gayakwad, M., Joshi, R., & Mahajan, A. (2023). Challenges and a Novel Approach for Image Captioning Using Neural Network and Searching Techniques. International Journal of Intelligent Systems and Applications in Engineering, 11(3), 712-720.

[28] Godse, D. ., Mulla, N. ., Jadhav, R. ., Gayakwad, M. ., Joshi, R. ., Kadam, K. ., & Jadhav, J. . (2023). Automated Video and Audio-based Stress Detection using Deep Learning Techniques. International Journal on Recent and Innovation Trends in Computing and Communication, 11(11s), 487–492. https://doi.org/10.17762/ijritcc.v11i11s.8178

[29] Paygude, P. ., Chavan, P. ., Gayakwad, M. ., Gupta, K. ., Joshi, S. ., Gopika, G., Joshi, R., Gonge, S., & Kotecha, K. . (2023). Optimizing Hyperparameters for Enhanced LSTM-Based Prediction System Performance. International Journal on Recent and Innovation Trends in Computing and Communication, 11(10s), 203–213. https://doi.org/10.17762/ijritcc.v11i10s.7620

[30] Bhole, G. V., et al. "Implementation of Virtual Mouse Control System Using Hand Gestures for Web Service Discovery." International Journal of Intelligent Systems and Applications in Engineering 12.13s (2024): 663-672.

[31] Pawar, R., Ghumbre, S., Deshmukh, R. (2018). Developing an Improvised E-Menu Recommendation System for Customer. In: Sa, P., Bakshi, S., Hatzilygeroudis, I., Sahoo, M. (eds) Recent Findings in Intelligent Computing Techniques. Advances in Intelligent Systems and Computing, vol 708. Springer, Singapore. https://doi.org/10.1007/978-981-10-8636-6_35

[32] Pawar, R., Ghumbre, S., & Deshmukh, R. (2019). Visual Similarity Using Convolution Neural Network over Textual Similarity in Content- Based Recommender System. International Journal of Advanced Science and Technology, 27, 137 - 147.

[33] Pawar, R., Ghumbre, S., & Deshmukh, R. (2020). A Hybrid Approach towards Improving Performance of Recommender System Using Matrix Factorization Techniques. International Journal of Future Generation Communication and Networking, Vol. 13, No. 4, (2020), pp. 467–477

[34] Dr. Dhanashree Wategaonkar, Dr. Rajendra Pawar , Prathamesh Jadhav , Tanaya Patole , Rohit R. Jadhav , Saisrijan Gupta. (2022). SIGN GESTURE INTERPRETER FOR BETTER COMMUNICATION BETWEEN A NORMAL AND DEAF PERSON. Journal of Pharmaceutical Negative Results, 5990–6000. https://doi.org/10.47750/pnr.2022.13.S07.731

[35] [35] Garg, A. R.G.Pawar et al. (2023). Slider-Crank Four-Bar Mechanism-Based Ornithopter: Design and Simulation. In: Tuba, M., Akashe, S., Joshi, A. (eds) ICT Systems and Sustainability. Lecture Notes in Networks and Systems, vol 516. Springer, Singapore. https://doi.org/10.1007/978-981-19-5221-0_26

[36] N. Cristianini and J. Shawe-Taylor. An Introduction to Support Vector Machines. Cambridge University Press, 2000.

[37] Priyanka and D. Kumar, “Decision tree classifyer: a detailed survey,” International Journal of Information and Decision Sciences, vol. 12, no. 3, pp. 246–269, 2020

[38] Gama, K. D., Oliveira, K. R. C., & Oliveira, G. H. (2020). Comparison of machine learning algorithms for software requirement classification. Journal of Software Engineering and Applications, 13(10), 559-571.

[39] Breiman, L. Random forests. Mach. Learn. 2001, 45, 5–32

[40] Hosseini, R., Parsa, S., & Shafie-Khah, M. (2019). A comparative study of machine learning algorithms for software requirement classification. Proceedings of the 2019 5th International Conference on Web Research (ICWR), 84-89.

[41] Aitkin, Murray, and Rob Foxall. "Statistical modelling of artificial neural networks using the multi-layer perceptron." *Statistics and Computing* 13 (2003): 227-239.

[42] Kocher, M., and J. Savoy. 2017. Distance measures in author profiling. Information Processing and Management 53:1103–19. doi:10.1016/j.ipm.2017.04.004.

[43] Liu, L., Gao, H., Sun, J., & Li, X. (2020). A comparative study of machine learning techniques for software requirement classification. Proceedings of the 2020 6th International Conference on Computer and Technology Applications (ICCTA), 108-112. 2011.

[44] A comparative study of machine learning techniques for software requirements classification by A. Rosado et al. (2019)

[45] Automated Requirements Classification: A Systematic Literature Review by C. Franzke et al. (2020)

[46] Software Requirements Classification using Deep Learning Techniques by H. R. Bakhshi et al. (2021)

[47] A novel approach to software requirements classification based on word embeddings by S. M. R. Bahrani et al. (2021)

[48] Software Requirements Classification with Word Embedding and Convolutional Neural Networks by Y. Liu et al. (2021)

[49] Automated Classification of Software Requirements into Functional and Non-Functional Categories by M. Goyal and A. Bhatnagar (2017)

[50] A comparative study of supervised learning algorithms for software requirement classification by N. Pandey et al. (2018)

[51] Software Requirement Classification: A Survey by M. R. Khan and M. A. Memon (2020)

[52] Software Requirement Classification Using Word2Vec and Decision Tree Algorithm by P. Mishra and A. Shukla (2018)

[53] Multi-Label Software Requirement Classification Based on Textual Feature Extraction by C. Wu et al. (2021)

[54] Classifying software requirements using hybrid approach of deep learning and machine learning" by S. Srivastava and N. Aggarwal (2021)

[55] Shreda, Q. A., & Hanani, A. A. (2021). Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches. *IEEE Access*, *4*, 1–22. https://doi.org/10.1109/ACCESS.2021.3052921

[56] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan.2012. "The PROMISE Repository of empirical software engineering data," http://promisedata.googlecode.com, West Virginia University, Department of Computer Science (last accessed March 2014).

[57] Khadikar, S., P. Sharma, and P. Paygude. n.d. "Compassion Driven Conversational Chatbot Aimed for Better Mental Health." https://scholar.google.ca/scholar?cluster=14383174570134551860&hl=en&as_sdt=0,5&sciodt=0,5.

Mishra, S., P. Paygude, and S. Chaudhary. 2018. "Use of Data Mining in Crop Yield Prediction." *2018 2nd International*. https://ieeexplore.ieee.org/abstract/document/8398908/.

Modak, A., S. D. Chaudhary, and P. S. Paygude. 2018. "Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes?" *2018 Second*. https://ieeexplore.ieee.org/abstract/document/8473104/.

Paygude, P., R. Garg, P. Pathak, A. Trivedi, and A. Raj. n.d. "IMAGE PROCESSING USING MACHINE LEARNING,=." *Academia.edu*. https://www.academia.edu/download/84231706/IJSDR2009078.pdf.

Paygude, P., D. S. D. Joshi, and D. M. Joshi. 2020. "Fault Aware Test Case Prioritization in Regression Testing Using Genetic Algorithm." *International Journal of Emerging Trends in*. https://scholar.google.ca/scholar?cluster=11372764101768453854&hl=en&as_sdt=0,5&sciodt=0,5.

Paygude, P., and S. D. Joshi. 2020. "Use of Evolutionary Algorithm in Regression Test Case Prioritization: A Review." *Proceeding of the International Conference on*. https://link.springer.com/chapter/10.1007/978-3-030-24643-3_6.

Paygude, Priyanka, and Shashank Joshi. 2020. "Optimization of Test Case Prioritization through Calibration of Genetic Algorithm." *Solid State Technology* 63 (1s): 2377–86.

Paygude, Priyanka, and Aatmic Tiwari. 2021. "Comparative Analysis on Different Generations of Cryptocurrency." *International Journal of Future Generation Communication and Networking* 14 (1): 3016–26.

Paygude, Priyanka, Aatmic Tiwari, Bhavya Goel, and Akshat Kabra. 2023. "Comparative Analysis of Stock Prices by Regression Analysis and FB Prophet Models." In *Data Intelligence and Cognitive Informatics*, 295–307. Springer Nature Singapore.

Priyanka, P. 2019. "Comparative Analysis of Test Case Prioritization Approaches in Regression Testing." *International*

Journal of Advanced Trends in Computer. https://scholar.google.ca/scholar?cluster=11774303718873317097&hl=en&as_sdt=0,5&sciodt=0,5.

Sachdeva, S., A. Arya, and P. Paygude. 2018. “Prioritizing User Requirements for Agile Software Development.” *On Advances in* …. https://ieeexplore.ieee.org/abstract/document/8529454/.

Tyagi, Shubham, Rishabh Solanki, Adarsh Tiwari, Rohit Ray, and Priyanka Paygude. 2021. “Analysis of Various Machine Learning Models for Detecting Depression in Twitter Tweets.” *Turkish Online Journal of Qualitative Inquiry* 12 (7): 6616–25.

Varshney, Abhishek, Saumya Asthana, Chetan Sharma, Swapnil Patil, Ankur Kashyap, and Priyanka Paygude. 2020. “Study of Solving Knapsack Problem Using Genetic Algorithm.” *Solid State Technology*, February, 93–99.

Verma, Rauhil, Priyanka Paygude, Snehal Chaudhary, and Sonali Idate. 2018. “Real Time Traffic Control Using Big Data Analytics.” In *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, 637–41.

[58] Patil, T. B., Kashif Jamal, Keshav Anand, S. T. Sawant-Patil, Aditya Bhateja, and Priyanka Paygude. 2023. “Real-Time Clickstream Data Processing and Visualization Using Apache Tools.” In , 1–5.

[59] Tyagi, S., R. Solanki, A. Tiwari, and R. Ray. 2021. “Analysis of Various Machine Learning Models for Detecting Depression in Twitter Tweets.” *Online Journal of* …. https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=13096591&AN=161812024&h=FCpdmxyAldRm93M3QBUtP1h98n%2B%2FkkFFQ%2BziwZ6MaOgebc%2BSiN33H8r6SXVUXEsF1swUFu1rxiDHiXWVoUjVaw%3D%3D&crl=c.

[60] Tyagi, S., R. Solanki, A. Tiwari, and R. Ray. 2021. “Analysis of Various Machine Learning Models for Detecting Depression in Twitter Tweets.” *Online Journal of* …. https://search.ebscohost.com/login.aspx?direct=true&profile=ehost&scope=site&authtype=crawler&jrnl=13096591&AN=161812024&h=FCpdmxyAldRm93M3QBUtP1h98n%2B%2FkkFFQ%2BziwZ6MaOgebc%2BSiN33H8r6SXVUXEsF1swUFu1rxiDHiXWVoUjVaw%3D%3D&crl=c.

[61] Kadam, A., Garg, B., Gayakwad, M., Kotecha, K., & Joshi, R. (2024). Novel DSIDS-Deep Sniffer Intrusion Detection System. International Journal of Intelligent Systems and Applications in Engineering, 12(16s), 400-407.

[62] Khatik, I., Kadam, S., Gayakwad, M., Joshi, R., & Kotecha, K. (2024). Automatic Diagnosis of Fracture using Deep Learning and External Validation: A Systematic Review and Meta-Analysis. International Journal of Intelligent Systems and Applications in Engineering, 12(16s), 41-48.