Bulletin of Electrical Engineering and Informatics
Vol. 13, No. 2, April 2024, pp. 1090~1102
ISSN: 2302-9285, DOI: 10.11591/eei.v13i2.5531

Feature importance for software development effort estimation using multi level ensemble approaches

K. Eswara Rao[1], Pandu Ranga Vital Terlapu[1], Paidi Annan Naidu[1], Tammineni Ravi Kumar[1], Bala Murali Pydi[2]
[1]Department of Computer Science and Engineering, Aditya Institute of Technology and Management, Tekkali, India
[2]Department of Electrical and Electronics Engineering, Aditya Institute of Technology and Management, Tekkali, India

Article Info

Article history:

Received Dec 12, 2022
Revised Jun 17, 2023
Accepted Oct 5, 2023

Keywords:

Boosting approaches
Ensemble technique
Feature ranking
Machine learning
Software development effort estimation
Stacked ensemble

ABSTRACT

Feature importance strategy that substantially impacts software development effort estimation (SDEE) can help lower the dimensionality of dataset size. SDEE models developed to estimate effort, time, and wealth required to accomplish a software product on a limited budget are used more frequently by project managers as decision-support tool effort estimation algorithms trained on a dataset containing essential elements to improve their estimation accuracy. Earlier research worked on creating and testing various estimation methods to get accurate. On the other hand, ensemble produces superior prediction accuracy than single approaches. Therefore, this study aims to identify, develop, and deploy an ensemble approach feasible and practical for forecasting software development activities with limited time and minimum effort. This paper proposed a collaborative system containing a multi-level ensemble approach. The first level grabs the optimal features by adopting boosting techniques that impact the decided target; this subset features forward to the second level developed by a stacked ensemble to compute the product development effort concerning lines of code (LOC) and actual. The proposed model yields high accuracy and is more accurate than distinct models.

Corresponding Author:

K. Eswara Rao
Department of Computer Science and Engineering
Aditya Institute of Technology and Management, Tekkali, Andhra Pradesh, India
Email: eswarkoppala@gmail.com

1. INTRODUCTION

One of the most significant tasks in the software industry is to develop a quality software product with minimal components depending on how accurately it estimates software development effort [1]. The challenge is evaluating those metrics early in the project lifecycle when each effort's limits must be determined, and there are significant uncertainties about the end product's functionality. It was defined as "estimating the effort and time required to develop a software product. The accuracy of its effort estimates primarily determines the success of any software product. Kumar *et al.* [2] demonstrates that the reasons for a software product failure are idealistic or inarticulate project goals, erroneous resource estimates, and inability to handle product difficulty. Perfect effort estimates are critical for project success. In papers [3]–[5] defines a reasonable estimation as providing a clear enough view of the product reality to allow project management to make sound decisions about overseeing the product to meet its objectives.

Software effort estimating (SEE) approaches of various types have been presented [6]. Among the suggested techniques, machine learning (ML) based effort estimators such as support vector machines (SVM), decision tree function (DTF) networks, and random forest trees (RFTs) have drawn more attention [7]. Making

no or few assumptions about the function being modelled and the training data is the driving force behind deploying such techniques. Such methods are preferred since they don't or lightly assume things about the modelled function and the training data. For instance, Rao and Rao [8] demonstrated that ensemble techniques outperform single classification models in SEE because the voting classifier in the ensemble model reduces any residual effect related to feature insignificance and redundancy. To mitigate this, higher weights are given to specific classifiers that excel on the tested datasets. The prediction performance is undoubtedly improved by the robustness of irrelevant and redundant features. In its simplest form of averaging, the voting model assures the reduction of noise property, which improves the overall prediction performance.

This paper proposes a multilevel ensemble (MLE) learning module for the software development effort estimation method. The proposed MLE system incorporates adaptive boost gradient tree boost in the first level and uses seven individual classifiers in the second, including the proposed stacked ensemble. The research sequel states that the base classifiers have been chosen following thorough simulation validation. Some of these classifiers, including the RF and SVM models, are considered in the literature [8]. The basis classifiers' diverse classification abilities also allow them to distinguish between various statistical properties of the underlying data, which adds value to the proposed ensemble learning approach.

The proposed ensemble model needs effective feature selection models to perform better overall. The enhancement's final effects will determine how well the redundant and unnecessary features in software product datasets are handled. This research aims to show how feature selection improves effort estimation performance and suggest a multilevel ensemble learning technique that is resistant to data imbalance and feature redundancy. The proposed multilevel ensemble technique has additionally demonstrated enhanced resistance to redundant and irrelevant characteristics, substantially contributing to this research. This research aims to show how feature importance improves effort estimation and suggests a multilevel ensemble learning technique resistant to feature redundancy and data imbalance. Another significant contribution credited to this research is the improved robustness of the suggested MLE to redundant and irrelevant information.

This paper has contained two innovative discoveries. Section 2 describes the literature-related research on the estimation methods for software development effort estimation. The ML models neural networks (NN 30-30), linear regression (LR), k-nearest neighbor (K-NN), SVM radial basis function (SVM RBF), naive bayes (NB), SVM polynomial (SVM poly), have to consider combining some of the best features of the suggested method are discussed with experimental setup in section 3, proposed multilevel ensemble learning model and summarizes the results of the studies and demonstrates the experimental design in section 4. The research is concluded in section 5, along with its future scope.

2. RELATED WORK

This section describes a summary of ML strategies offered after a study of general effort estimation algorithms, concludes with a review of various classification methods and methodologies, as well as a comparison of ways that can be used to estimate software development effort.

2.1. Single classifier for software development effort estimation

Quality development has evolved into a critical activity for professional companies. Indeed, developed software's prominence, cost, and suitability are frequently decisive elements in an organization's success. The complete analysis of ML techniques used for effort estimation was carried out by [9]. According to researchers' analyzed work, the researchers mainly focused on customizing specific algorithms, particularly artificial neural networks, case-based reasoning models, and decision trees, for the most outstanding performance. The machine's precision with mean magnitude relative errors (MMRE) ranging from 35 to 55%, percentage close error deviations (PRED(25)) of 47 to 75%, and median magnitude relative errors (MdMRE) of 30 to 55%, learning models were of an acceptable level and outperformed statistical ones. According to the researchers, ML algorithms may produce disparate findings due to outliers, missing variables, and the chance of over fitting problems. To estimate the early stages of the software life cycle initiatives, LR, and NN [10]. Shahpar *et al.* [11] investigated several data sets and obtained encouraging findings for software development effort assessment. When estimating software maintenance effort using particle swarm optimization, Singh *et al.* [12] proposed a successful swarm intelligence-based method. Regardless of the approach used to develop ML, valuable recommendations for effort and duration estimation at early project stages can be retrieved. Because ML is sensitive to noise in data sets, models should not rely on unique algorithms but should be employed in tandem, which improves prediction accuracy [13]. Boosting, bagging, and complex random sampling techniques [14] were proposed by researchers, generally for the same sort of ML algorithms. However, if used excessively, ensemble methods can cause significant performance overhead [15]. As a result, for developing ML effort and duration models, a limited selection of algorithms and a simple ensemble method, such as averaging of acquired estimates, should be employed.

2.2. Evolutionary strategy

A hybrid method to estimate work using the use case point methodology has been put out by [16]. Numerous observations were made based on college student projects and industrial projects. The authors of this paper gave the environmental elements of the UCP approach significant weight. The researchers used feed-forward algorithms like radial basis feed forward neural (RBFNN) to predict the effort and productivity feed-forward algorithm. This project concluded that the UCP method's environmental considerations are ideal for software system productivity forecasting.

More recently, [17]–[19] examined the application of learning machine ensembles for SEE. Ensembles of learning machines are groups of learners trained to complete the same job and are put together to enhance prediction performance [8]. It is generally accepted that learners should act differently when combined to obtain more accurate predictions. Otherwise, the total forecast won't be more accurate than the individual guesses. Therefore, several ensemble learning strategies can be viewed as various ways to create variation among the base learners. The authors tried to estimate effort with a low failure ratio and cost.

None of the publications compares the outcomes of other easily accessible methods for ensemble learning from the ML literature and the issues raised above. Reseachers in [20], [21] provide data from a few ensemble approaches. However, the research does not statistically compare these methods and single learners. Different ensemble approaches can be more or less suitable for SEE and should be included in the comparisons. The papers also need to examine how the results were obtained.

2.3. Other approaches

According to particular research in the literature, the properties of the data set substantially impact how well various models perform. However, as previously indicated, existing research on ensemble models suggests that they perform better than distinct models even when numerous different data sets are employed. The results of the research methods, as presented in Table 1 (in Appendix) [18], [22]-[40] demonstrated that the optimal and significantly improved SDEE estimate performance was obtained by combining their two strategies. According to their projections, hybrid approaches may produce satisfactory results for varying sizes.

3. RESEARCH METHODOLOGY

This proposed research includes MLE approach to grab the optimal features as a precise step in the first level approach for choosing SDEE models to calculate effort in level 2. Figure 1 shows the overall feature selection process and software development effort estimation. The suggested method efficiently assigns ranks for features while simultaneously dealing with the imbalanced data problem in a software quality dataset to avoid bias problems. The following feature selection phases are defined to meet the objective.

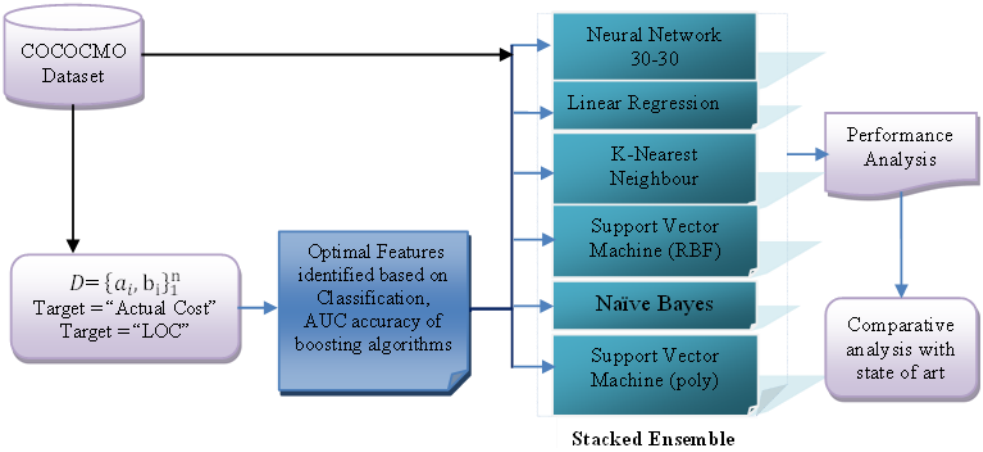

Figure 1. Prediction of SDEE using proposed MLE

3.1. Feature subset selection

A compelling feature selection system requires finding the primitive features that will be used to train the models [41]. The relevance or correlation between the characteristic and the class label serves as the

selection process's core guiding principle and is frequently applied to classification problems. Relevance measurements can be used to assess the significance of qualities like dimensionality reduction with five cross-fold validations. The model seeks to identify the best feature subset $\hat{f}\,(|f|) = k$ that maximises classification accuracy for a given dataset D=(a_i, b_i), using a feature set and class label b. To this, we proposed an adaptive boosting [42] and gradient tree boosting [43] ensemble model that takes into account the base ensemble classifiers trained using classification and AUC accuracy of each feature and increased to three permutations produced by feature selection is known as ensemble feature selection aims to limit the impact of high dimensions on learning algorithms while conducting classifier accuracy and developing successful ensemble learning systems for classification difficulties.

The general process of ensemble feature selection is shown in Figure 1. The fundamental concept is to use loss (Ψ) of accuracy as shown in (1) of individual feature weight based on the feature subset is divided randomly and diversity across the chosen feature subsets is ensured, and finally the mean of all loss of all classification methods has been computed based on (2) to perform consistently and how effectively each single feature separates the given dataset D=(a_i, b_i) to all 15 features they can distinguish between examples of proposed classifier models.

$$\Psi_{a_i}^{\mathrm{M_i}} = ab\,(m_j\,.accuracy\,(D) - m_j\,.accuracy\,(D - a_i)) \quad (1)$$

In (1) Ψ represents loss of boosting model on each attribute a_i and $m_j\,.accuracy\,(D)$ denotes accuracy measure of boosting model $\mathrm{m_i}$.

$$mean_{loss}^{a_i} = \sum_{f=1}^{N} ls_{a_i}^{\mathrm{m_i}} \quad (2)$$

The proposed approach has two ensemble methods, in this approach both ensembles choose the correct class label, resulting in the correct conclusion have assigned a class label to each unique occurrence of the dataset, and the final class label is selected by a frequently occurrence in both methods as optimal. To support the first phase, the following algorithms 1 and 2 are taken.

Algorithm 1. Pseudo code of Ada-Boost for feature selection

Input:

Training Dataset $D=\{a_i, \mathrm{b_i}\}_1^{\mathrm{n}}$ where $a_i \in R^p$ *and* $b_i \in \{-1, +1\}$
Set of α features $F_e = \{a_1, a_2, \ldots, a_\alpha\}$

Output:

Optimal features based on ranks $R = (D, \{r_{a_1}, r_{a_2} \ldots r_{a_\alpha}\})$

Begin :

1: Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $b_i = 0,1$ respectively, for $k = 1,2,..,K$
2: Normalize weights $w_{k,i} = \frac{w_{k,i}}{\sum_{j=1}^{n} w_{k,j}}$
3: Each feature j train a base classifier h_j which is classified to using a distinct feature.
4: Calculate error w_k $\epsilon_k = \sum_i w_i \left|h_j(a_i) - b_i\right|$
5: opt the classifier h_j, with the lowest error ϵ_k
6: revise weights $w_{k+1,i} = w_{k,i}\,\beta_k^{1-x_i}$
7: Repeat until weights upto get final $h(x) = \begin{cases} 1 & \sum_{k=1}^{K} \alpha_k\, h_k(a) \geq \frac{1}{2}\sum_{k=1}^{K} \alpha_k \\ 0 & otherwise \end{cases}$
where $\alpha_k = \log \frac{1}{\beta_k}$
8: **end for**
The resulting model outputs are used as the final forecast for test cases.
Note: Ranking assigned from 1 to 10.

End

Algorithm 2. Pseudo code of gradient boost feature selection

Input:

Training Set $D==\{a_i, \mathrm{b_i}\}_1^{\mathrm{n}}$ where $a_i \in R^p$ and $b_i \in \{0,1\}$
Set of α features $F_e = \{a_1, a_2, \ldots, a_\alpha\}$

Output:

To assign rankings of features $R = (D, \{r_{a_1}, r_{a_2} \ldots r_{a_\alpha}\})$

Begin :

1: $\{h_1, h_2, \ldots, h_m,\} \leftarrow train\ GBT$
2: $\hat{\mathrm{I}} \leftarrow [0, \ldots, 0]$
3: for each $h_m\ in\ \{h_1, h_2, \ldots, h_m,\}\ do$
4: for j = 1 to d do
5: $\hat{\mathrm{I}}_j = \hat{\mathrm{I}}_j + \frac{1}{M}\,.I_j(h_m)$

$\hat{I} \leftarrow \frac{\hat{I}}{\sum_{j=1}^{M} \hat{I}_j}$
$for\ i = 1\ to\ n\ do$
  $a_{i,j} \leftarrow \left[\hat{I}_j a_{i,j}\right]_{\hat{I}_j \geq th}$ where threshold $th \in (0,1)$
Return $\{(a_1(i), b_i), i = 1,2,\ldots n\}$
The dataset D is returned preserving only the selected features
The resulting model outputs features importance according to weights.
Note: Ranking assigned from 1 to 10.
End

To reach the proposed approach, this research used the COCOMO-81 dataset, as shown in Table 2**,** which accomplishes the selection of weighted features by rating their classification accuracy and AUC value in an initial model that includes all predictors. The gradient tree boost model uses a greedy optimisation strategy to identify the top-performing subset of elements based on the proposed Ada-Boost [35]. The dataset in Table 2 contains 17 features from $\boldsymbol{F_1}$ to $\boldsymbol{F_{15}}$ to as contributing, and the dependent as "LOC," in $\boldsymbol{F_{16}}$ and $\boldsymbol{F_{17}}$ as another target was "actual cost".

Table 2. Dataset information for COCOMO-81

Feature. No.		Description of feature	Code	Value	GT Boost		Ada-Boost	
					Mean	Std	Mean	Std
F_1	Features participated	Required software reliability	rely	Numeric	0.01131	0.002986	0.00267857	0.00087617
F_2		Data base size	data		0.027381	0.002393	0.100645	0.0172055
F_3		Process complexity	cplx		0.001091	0.000982	0.0168651	0.00505271
F_4		Time constraint for cpu modern	time		0.009921	0.00271	0.126438	0.0119631
F_5		Main memory constraint	stor		0.00377	0.001964	0.0274802	0.0165147
F_6		Machine volatility	virt		0.000794	0.00014	0.00124008	0.00059936
F_7		Turnaround time	turn		0.003373	0.000612	0.00763889	0.00092
F_8		Analysts capability	acap		0.001687	0.001148	0.003125	0.00107994
F_9		Application experience	aexp		0.028373	0.00231	0.0198413	0.00546806
F_{10}		Programmers capability	pcap		0.002679	0.000643	0.00128968	0.00101171
F_{11}		Virtual machine experience	vexp		-9.92E-05	0.00014	1.98E-04	0.0001403
F_{12}		Language experience	lexp		0.000198	0.00014	0.0014881	0.00084179
F_{13}		Programming practices	modp		0.000595	0.000643	0.018502	0.00633835
F_{14}		Use of software tools	tool		0.001984	0.000982	0.00138889	0.0002806
F_{15}		Schedule constraint	sced		0.000496	0.000506	0.00530754	0.00401939
F_{16}	Target	Lines of code	LOC					
F_{17}		Actual cost	Actual					

From the above, collected sufficient number of features based on their ranks with respect to two targets. Then these sets of data forwarded to seven classifiers and calculate the loss Ψ of each one as shown in (3):

$$\Psi_{M_i}^{M_n} = ab\left(M_n.acc(T_r)\right) \quad (3)$$

Where $mean_{loss}^{a_i}$ calculates mean loss of each permutation to three permutations for all individual classifiers. When working on a specific learning set, the stacked model can be thought of as a method of calculating all base classifier losses $\sum_{i=1}^{N} \Psi$ and then correcting prediction residuals using the level 1 model. The mean of all accuracy losses is derived using (5), which stands for the mean of all accuracy losses.

$$Mean_{\Psi}^{M_n} = \sum_{j=1}^{N} \Psi_{M_i}^{M_n} \quad (4)$$

In (4), $\Psi_{M_i}^{M_n}$ represents the loss of classifier M_n on selected feature f_i and $M_n.acc(T_r)$ denotes accuracy measure of classifierM_n. In (4), $Mean_{\varphi}^{M_n}$ represents mean loss upon ranked features M_i from all classifiers. The overall accuracy is produced in the order that optimal features are selected based on the ranking. The ensemble classifier was used to choose and consider the top features for inclusion in the classification model based on the output of the ranked features that were analyzed.

3.2. Experimental setup and simulation

The proposed research was executed on a system which contains Intel(R) i^5 – 6200 CPU 3.40 GHz, 8 GB (RAM), and a 64-bit latest Windows-10 (OS) GUI interface. Python Anaconda is an open-source programming language, and Spyder IDE is used for the simulation. Table 3, all the classifiers' parameters are selected using a trial-error method.

Table 3. Parameter setup

Base models	Parameter setup
KNN	{Number of neighbours (k): 5}
Naïve bayes	{No hyperparameters to tune }
SVM poly	{C (regularization parameter): 1.0}; {kernel: polynomial};{degree of polynomial kernel: 3}
SVM RBF	{C (regularization parameter): 1.0}; {kernel: RBF}
NN 30-30	{Neurons per layer: 30}; {activation function: relu for hidden} {layers, softmax for output layer}; {learning rate: 0.001}; {training duration: 100 epochs}
LR	{Regularization type: l2 (ridge)}; {regularization parameter (c): 1.0} {training duration: 100 epochs}
Proposed stacked ensemble model	{Base models: nn 3030 classifier, SVM RBF, NB, SVM poly, LR Meta learner: logistic regression} {Number of base models: 6} {Hyperparameters for base models and meta learner tuned during stacking}

4. RESULT ANALYSIS

As stated in the first phase results in Table 2, using two boosting models, optimal features were identified, and ranks were assigned. Those two models gave good recognition to the elements in standard, prepared an optimal dataset with the optimal ranked features, and then calculated mean loss and rank. Figure 2 shows an analysis of the optimal dataset with ten features assigned positions for each element by applying GT Boosting classification accuracy (CA) and AUC score. Figure 3 represents classifier accuracy and AUC of the Ada-Boost classifier of optimal features. Calculated ranks for top ten features with the support of Ada-Boost classifier. Here, we can observe the positions for the top ten out of fifteen parameters based on their scores.

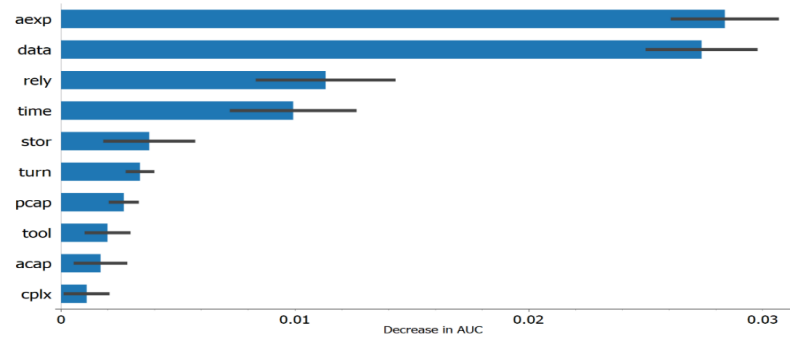

Figure 2. Gradient tree boosting classifier feature ranking

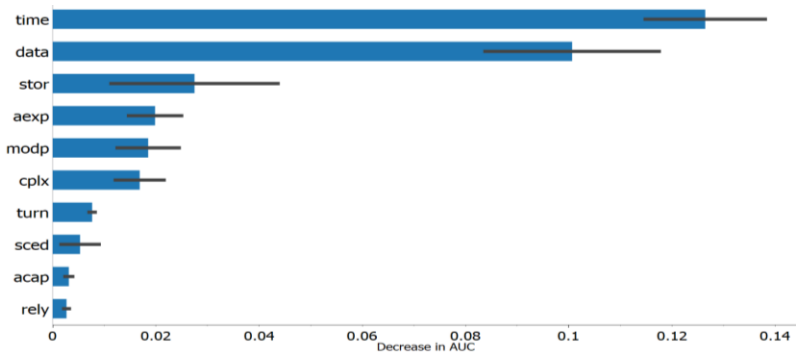

Figure 3. Ada-Boosting classifier feature ranking

According to algorithms 1 and 2, calculate CA and AUC scores each boosting algorithm assigns ranks as shown in Table 4. Positions are given to all fifteen features such as $\boldsymbol{F_{rely}, F_{data}, F_{cplx}, F_{time}, F_{stor}, F_{virt}, F_{turn}, F_{acap}, F_{aexp}, F_{pcap}, F_{vexp}, F_{lexp}, F_{modp}, F_{tool}, F_{sced}}$. Out of fifteen features GT Boosting CA is very low for $\boldsymbol{F_{virt}, F_{vexp}, F_{lexp}, F_{modp}, F_{sced}}$ features, and Ada-Boost CA is very low $\boldsymbol{F_{virt}, F_{pcap}, F_{vexp}, F_{lexp}, F_{tool}}$. So, which features are commonly identified and get low accuracies for both

algorithms are removed from the original dataset and a new subset of features with twelve features and this subset of features dataset forward to next level ensemble approach.

Table 4. Rankings of features for predicting boosting classifiers

Optimal features based on their ranks	
Adaptive boost classifier	GT boost classifier
$[F_1, F_8, F_{15}, F_7, F_3, F_{13}, F_9, F_5, F_2, F_4]$ required software reliability (rely), analysts capability (acap), schedule constraint (sced), turnaround time (turn), process complexity (cplx), programming practices (modp), application experience (aexp), data base size (data), main memory constraint (stor), time constraint for cpu modern (time).	$[F_3, F_8, F_{14}, F_{10}, F_7, F_5, F_9, F_4, F_1, F_2]$ process complexity (cplx), analysts capability (acap), programmers capability (pcap), use of software tools (tool), turnaround time (turn), main memory constraint (stor), application experience (aexp), required software reliability (rely), data base size (data), time constraint for cpu modern (time).
New optimal subset features total 12	$[F_1, F_2, F_3, F_4, F_5, F_7, F_8, F_9, F_{10}, F_{13}, F_{14}, F_{15}]$

4.1. Discussion on level 1 results

As per MLE proposed model the first level conducting experiments for the original dataset and the performance metrics of all six classifiers (NN 30-30, NB, SVM RBF, SVM poly, K-NN, and LR) concerning the LOC and actual cost as targets reside in the original dataset, based on the experiments calculate the CA, AUC, F1, precision and recall to all classifiers and outcomes showed in Table 5 for each model. All models have shown relatively good performance in predicting the performance metrics. NB scored 97%, and K-NN scored 99%, indeed a better performance compared to all other models, as the process's objective was to predict the actual effort and LOC in the target dataset for developing the SDEE. SVM poly stands good in CA with 95% precision and a low error rate, and the NN 30-30 classifier stands at an accuracy of 54% on other models to predict targets.

Table 5. Classifiers performance was observed with 15 features

S.NO	Base model	Performance measures with 15 features				
		AUC	CA	F1-score	Precision	Recall
1.	NN 30-30 classifier	0.831	0.476	0.438	0.545	0.476
2.	NB	0.973	0.667	0.694	0.846	0.667
3.	SVM RBF	0.541	0.825	0.833	0.851	0.825
4.	K-NN (EQUL)	0.993	0.873	0.870	0.883	0.873
5.	SVM poly	0.651	0.937	0.939	0.955	0.937
6.	LR	0.906	0.873	0.873	0.908	0.873

The ROC-AUC curve in Figure 4 shows the relationship between the true positive rate (TPR), which measures the model sensitivity and the false positive rate (FPR) which measures model specificity for the original dataset, which participates in fifteen features and finds the targets as LOC and actual cost, both are proportional.

According to the ROC curve, NB performs better than the remaining individual classifiers, all indicating respective colours, as shown in Figure 4. A lift curve analysis helps evaluate the performance of different models, especially in classification tasks. Figure 5 shows the K-NN model achieves an exceptionally high AUC of 4.65 at a probability threshold of 0.0, indicating that it can make highly accurate predictions when selecting the nearest neighbors. The SVM model with a polynomial kernel has a lower AUC of 0.508 at a probability threshold 0.168. This suggests that it may not perform as well.

4.2. Discussion on stacking ensemble learning approach at level 2

Based on the research proposal in the second level, we used an effective ensemble learning algorithm that learns how to combine predictions from two or more base ML techniques, the stacked ensemble. It is used to train models and make predictions, and the advantage of stacking is that it can combine the abilities of six high-performing models on a classification and regression difficulties task to provide forecasts that perform better than any one model in the ensemble [44]. Using this method, the performances of multiple models are integrated to create a single, effectual output. This method uses level 1 as a base model fitting to the training data and whose predictions are generated and level 2 as a meta-model that learns how to best combine the base models' predictions. The results of the fundamental learners' developing features can be integrated using a weighted average. It grants the model dominance in prediction performance as well as reliability. Also, the high-impact features selected by RF and GTB can be seen in Table 4, with the optimal subset of features a new featured dataset prepared separately and given as input to

all classifiers, including proposed stacked ensemble and conduct experiments. For this experiment, actual cost and LOC are the target variables (both are proportional) to find the performance of the proposed model, including all the model's accuracies, as shown in Table 6. The model performance evaluation used the same metrics followed by level 1. The evaluation mechanism used in this study focuses on assessing the performance of the new optimal feature dataset in predicting the development effort.

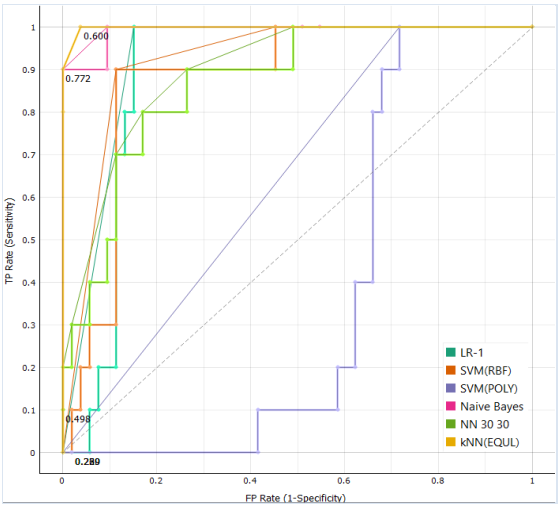

Figure 4. Performance of all models on fifteen features

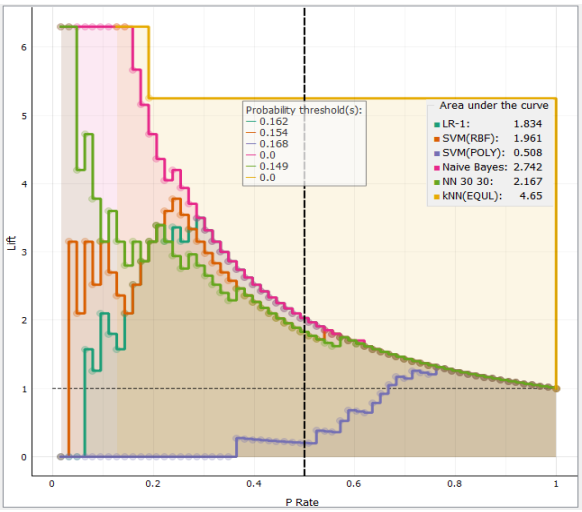

Figure 5. Lift curve of all models on original dataset

Table 6. Performance of ensemble model in comparison to that of classifiers for optimal features

S.No	Base model	Performance measures with 12 features				
		AUC	CA	F1-score	Precision	Recall
1.	NN 30-30 classifier	0.841	0.492	0.481	0.580	0.492
2.	Navie bayes	0.977	0.667	0.692	0.781	0.667
3.	SVM RBF	0.606	0.810	0.812	0.833	0.810
4.	K-NN (EQUL)	0.983	0.873	0.870	0.883	0.873
5.	SVM poly	0.965	0.921	0.925	0.947	0.921
6.	LR	0.851	0.794	0.791	0.807	0.794
7.	Stacked ensemble	0.989	0.990	0.990	0.991	0.996

After an experimental study with a prosed model conducting experiments with twelve features, all classifiers have shown good performance compared with level 1 results in predicting the performance metrics. The outcome of our research proposal stacked ensemble scored 99%, and K-NN scored 98%, indeed a better performance compared to all other models, as the objective of the process was to predict the actual

effort and LOC in the target dataset for developing the SDEE. The proposed stacked ensemble stands at 99% in CA, 99% precision and less error to predict the target.

The ROC-AUC curve in Figure 6 shows the relationship between TPR, which measures the model sensitivity, and FPR, which measures model specificity for the optimal feature subset of the dataset which participates in twelve features and finds the targets as LOC and actual cost, (both are proportional). According to the ROC curve, the proposed stacked ensemble reached 1 to perform all-time better than all remaining individual classifiers, all indicating respective colours, as shown in Figure 6.

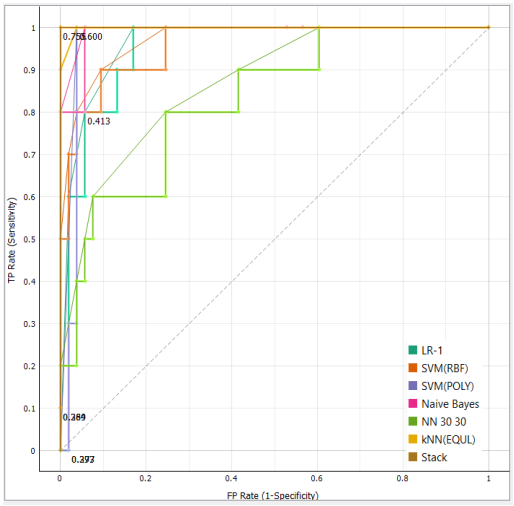

Figure 6. Performance of all models on optimal feature subset dataset

The probability of the threshold of the proposed stacked ensemble classifier for the optimal feature subset dataset (12 features) was also calculated to find the error in each effort category. Figure 7 shows the lift curve analysis for each model, including the stacked ensemble concerning LOC and Actual as targets. The proposed model achieves an exceptionally high AUC of 2.783 at a probability threshold of 0.025, indicating that it can make highly accurate predictions compared to other individual models. The proposed model has a lower high AUC at a probability threshold 0.0258. This suggests that our proposed model performs well, indicating that it can make highly accurate predictions comparatively with the state-of-the-art models.

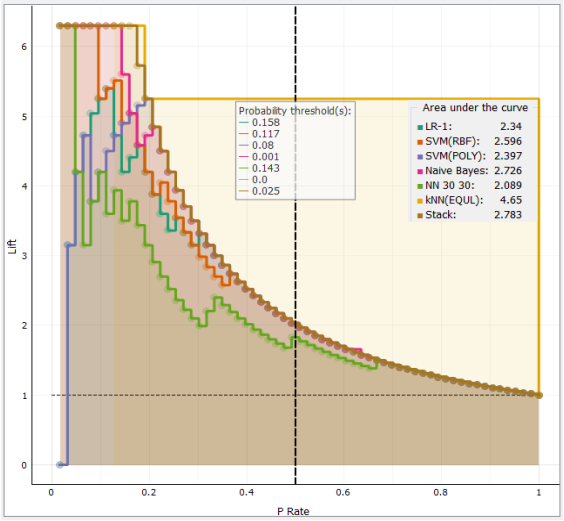

Figure 7. Lift curve of all models on optimal feature subset dataset

Compared to the feature ranking technique, the accuracy obtained by stacked ensemble learning for all classifiers as shown in Table 6 has been impossibly promising. This is partly due to the stacked ensemble-learning algorithm's dedicated targeting of the top features based on ranking. Compared to the ideal dataset, Figure 8 displays the classification accuracy and AUC values for the base learners and the proposed stacked ensemble approach. The proposed stacking ensemble produced the all-time highest accuracy in predicting SDEE and proves this research study's objective.

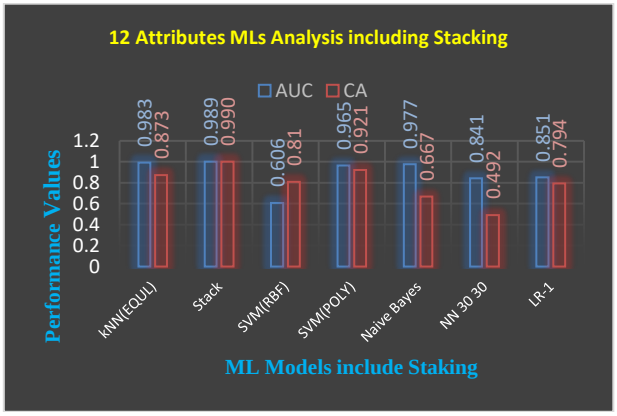

Figure 8. Model analysis of proposed model on optimal feature subset dataset

5. CONCLUSION

This research examined ways to estimate the software development effort with minimum time. We noticed that using a few outstanding features yields a considerably greater AUC than the alternative. In addition, NB and K-NN outperformed comparative with traditional techniques like NN 30-30, SVMs (poly, RBF), and LR on the COCOMO dataset, and we proved that they usually incorporate more essential features in achieving an acceptable level of accuracy and indicating that giving features weights may improve SDEE when employing individual classifiers. For this, we propose a multi-level ensemble model to predict outstanding features based on priority to estimate development effort by adopting a stacked ensemble with a group of six well-designed learners, which outperformed and higher AUC measurements over the more traditional techniques like NN 30-30, SVMs (poly, RBF), NB, K-NN, and LR. Based on the No Free Lunch theorem, according to “No Free Lunch hypothesis” No-one ML classifier model is the best on every situation, so when taking into account different ensembles, our work has revealed that it is improbable that there is a model that is always the best. The software product manager should ideally test several models while employing a guiding framework considering all the goals and projects they access. It makes it possible to pinpoint the model to deliver the behaviour that best suits the manager's requirements. In future, we plan to use further feature selection approaches to support our claim that many features in publicly available software product datasets are unnecessary or redundant. Investigating additional ensemble learners to contrast our system will also be part of future work.

APPENDIX

Table 1. Research on software effort estimation by adopting various appraochs

S.No	Techniques used	Data-set used	Problem name	State of art	Metrics used for study	Ref.
1.	RSA	USP05-FT USP05-RQ	Feature reduction	FFNN NB	– MMRE – RMSE – MAE	[22]
2.	DTF	ISBSG Desharnais	SEE	DT MLR	– MRE – MMRE – MdMRE – PRED	[23]
3.	COCOMO	NASA 93	SDEE	NB LR RF	– AUC – CA – Precision – Recall	[24]
4.	ANN	COCOMO II	Minimize predetermined error	-	– MMRE – MSE	[25]

Table 1. Research on software effort estimation by adopting various appraochs (*continued*)

S.No	Techniques used	Data-set used	Problem name	State of art	Metrics used for study	Ref.
5.	GLM	ISBSG	SEE	SVM MLP	– MAE – RMSE – MMER, etc	[26]
6.	RF	ISBSG COCOMO	SEE	-	– PRED – MRE – MMRE – PRED	[18]
7.	GA, PSO, FL, ACO, ABC	-	Predict reliability	-	-	[27]
8.	SEER-SEM	COCOMO	FPA	-	– MMRE – PRED	[28]
9.	OLS SWR RR	COCOMO MAXWELL CHINA	Regression based effort estimation	ML	– MAE – BMMRE	[29]
10.	ABEO-KN	Promise Repository datasets	Ranking of estimation methods	Analog based methods	– MMRE – MAR – MdAR – SD – RSD – LSD	[30]
11.	ASEE	Desharnais ISBSG Albrecht COCOMO Kemerer	SDEE	Analog based SDEE	– MMRE – PRED – MdMRE – MRE	[31]
12.	ANN	COCOMO	Estimating effort	-	– MMRE – PRED – RMSE	[32]
13.	Classical analogy Ensemble	ISBSG	SEE	Fuzzy analogy models	– MAE – LSD – MBRE – MIBRE	[33]
14.	GP, MOGP	Desharnais, Finnish Miyazaki	Accuracy	-	– MMRE – PRED – MdEMRE	[34]
15.	Multi layered feed forward neural network (MLFFANN)	COCOMOII	Prediction of software effort	-	– MSE – MMRE	[35]
16.	Fuzzy logic	-	SCE	Bailey Basili, Dotly, Halstead	– MRE,MF – MMRE	[36]
17.	ABE	ISBSG	Predict SEE	CART MLR CNN	– MRE – MMRE – PRED	[37]
18.	Ada Boost	Desharnais MAXWELL	LOC, actual cost	K-NN SVM	– Loss – Accuracy	[38]
19.	ML CBR	Infoway Diyatech Tsoft	SDE	-	– BRE – MRE	[39]
20.	Metaheuristic optimization	NASA		GA, PSO, FA	– MAE – MMRE – VAF	[40]

REFERENCES

[1] H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation," *Expert Systems with Applications*, vol. 35, no. 3, pp. 929–937, Oct. 2008, doi: 10.1016/j.eswa.2007.08.001.

[2] K. V. Kumar, V. Ravi, M. Carr, and N. R. Kiran, "Software development cost estimation using wavelet neural networks," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1853–1867, Nov. 2008, doi: 10.1016/j.jss.2007.12.793.

[3] A. Khalid, M. A. Latif, and M. Adnan, "An approach to estimate the duration of software project through machine learning techniques," *Gomal University Journal of Research*, vol. 33, no. 1, pp. 47–59, 2017.

[4] A. B. Nassif, L. F. Capretz, and D. Ho, "Estimating software effort based on use case point model using sugeno fuzzy inference system," in *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, Nov. 2011, pp. 393–398. doi: 10.1109/ICTAI.2011.64.

[5] I. C. Suherman, R. Sarno, and Sholiq, "Implementation of random forest regression for COCOMO II effort estimation," in *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*, Sep. 2020, pp. 476–481. doi: 10.1109/iSemantic50169.2020.9234269.

[6] H. Leung and Z. Fan, "Software cost estimation," in *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Company, 2002, pp. 307–324. doi: 10.1142/9789812389701_0014.

[7] J. Shivhare, "Effectiveness of feature selection and machine learning techniques for software effort estimation," National Institute of Technology Rourkela, 2014.

[8] K. E. Rao and G. A. Rao, "Ensemble learning with recursive feature elimination integrated software effort estimation: a novel approach," *Evolutionary Intelligence*, vol. 14, no. 1, pp. 151–162, 2021, doi: 10.1007/s12065-020-00360-5.
[9] Z. Polkowski, J. Vora, S. Tanwar, S. Tyagi, P. K. Singh, and Y. Singh, "Machine learning-based software effort estimation: an analysis," in *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Jun. 2019, pp. 1–6. doi: 10.1109/ECAI46879.2019.9042031.
[10] I. Attarzadeh, A. Mehranzadeh, and A. Barati, "Proposing an enhanced artificial neural network prediction model to improve the accuracy in software effort estimation," in *2012 Fourth International Conference on Computational Intelligence, Communication Systems and Networks*, Jul. 2012, pp. 167–172. doi: 10.1109/CICSyN.2012.39.
[11] Z. Shahpar, V. K. Bardsiri, and A. K. Bardsiri, "An evolutionary ensemble analogy-based software effort estimation," *Software: Practice and Experience*, vol. 52, no. 4, pp. 929–946, Apr. 2022, doi: 10.1002/spe.3040.
[12] C. Singh, N. Sharma, and N. Kumar, "An efficient approach for software maintenance effort estimation using particle swarm optimization technique," *International Journal of Recent Technology and Engineering*, vol. 7, no. 6C, pp. 1–6, 2019.
[13] L. L. Minku and X. Yao, "Ensembles and locality: insight on improving software effort estimation," *Information and Software Technology*, vol. 55, no. 8, pp. 1512–1528, Aug. 2013, doi: 10.1016/j.infsof.2012.09.012.
[14] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, Nov. 2011, doi: 10.1109/TSE.2011.111.
[15] D. Azhar, P. Riddle, E. Mendes, N. Mittas, and L. Angelis, "Using ensembles for web effort estimation," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Oct. 2013, pp. 173–182. doi: 10.1109/ESEM.2013.25.
[16] M. Azzeh, A. B. Nassif, and L. L. Minku, "An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation," *Journal of Systems and Software*, vol. 103, pp. 36–52, May 2015, doi: 10.1016/j.jss.2015.01.028.
[17] A. B. Nassif, M. Azzeh, A. Idri, and A. Abran, "Software development effort estimation using regression fuzzy models," *Computational Intelligence and Neuroscience*, vol. 2019, Feb. 2019, doi: 10.1155/2019/8367214.
[18] A. Idri, F. A. Amazal, and A. Abran, "Analogy-based software development effort estimation: a systematic mapping and review," *Information and Software Technology*, vol. 58, pp. 206–230, Feb. 2015, doi: 10.1016/j.infsof.2014.07.013.
[19] P. S. Kumar and H. S. Behera, "Estimating software effort using neural network: an experimental investigation," in *Advances in Intelligent Systems and Computing*, Springer Singapore, 2020, pp. 165–180. doi: 10.1007/978-981-15-2449-3_14.
[20] L. L. Minku and X. Yao, "A principled evaluation of ensembles of learning machines for software effort estimation," Sep. 2011. doi: 10.1145/2020390.2020399.
[21] M. Auer, A. Trendowicz, B. Graser, E. Haunschmid, and S. Biffl, "Optimal project feature weights in analogy-based cost estimation: improvement and limitations," *IEEE Transactions on Software Engineering*, vol. 32, no. 2, pp. 83–92, Feb. 2006, doi: 10.1109/TSE.2006.1599418.
[22] J. Shivhare and S. K. Rath, "Software effort estimation using machine learning techniques," Feb. 2014. doi: 10.1145/2590748.2590767.
[23] A. B. Nassif, M. Azzeh, L. F. Capretz, and D. Ho, "A comparison between decision trees and decision tree forest models for software development effort estimation," in *2013 Third International Conference on Communications and Information Technology (ICCIT)*, Jun. 2013, pp. 220–224. doi: 10.1109/ICCITechnology.2013.6579553.
[24] A. B. Mustafa, "Predicting software effort estimation using machine learning techniques," Jul. 2018. doi: 10.1109/CSIT.2018.8486222.
[25] P. Rijwani and S. Jain, "Enhanced software effort estimation using multi layered feed forward artificial neural network technique," *Procedia Computer Science*, vol. 89, pp. 307–312, 2016, doi: 10.1016/j.procs.2016.06.073.
[26] P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," *Journal of Systems and Software*, Mar. 2017, doi: 10.1016/j.jss.2017.11.066.
[27] C. Diwaker, P. Tomar, R. C. Poonia, and V. Singh, "Prediction of software reliability using bio inspired soft computing techniques," *Journal of Medical Systems*, vol. 42, no. 5, pp. 1–16, May 2018, doi: 10.1007/s10916-018-0952-3.
[28] W. L. Du, D. Ho, and L. F. Capretz, "A neuro-fuzzy model with SEER-SEM for software effort estimation," 2015.
[29] S. Mensah, J. Keung, M. F. Bosu, and K. E. Bennin, "Duplex output software effort estimation model with self-guided interpretation," *Information and Software Technology*, vol. 94, pp. 1–13, Feb. 2018, doi: 10.1016/j.infsof.2017.09.010.
[30] P. Phannachitta, J. Keung, A. Monden, and K. Matsumoto, "A stability assessment of solution adaptation techniques for analogy-based software effort estimation," *Empirical Software Engineering*, vol. 22, no. 1, pp. 474–504, Feb. 2017, doi: 10.1007/s10664-016-9434-8.
[31] A. Idri, F. A. Amazal, and A. Abran, "Analogy-Based software development effort estimation: a systematic mapping and review," *Information and Software Technology*, Feb. 2014, doi: 10.1016/j.infsof.2014.07.013.
[32] K. K. T. M, S. Aihole, and S. Putage, "Anticipation of software development effort using artificial neural network for NASA data sets," *International Journal of Engineering Science and Computing*, vol. 7, no. 5, p. 11228, 2017.
[33] A. Idri, M. Hosni, and A. Abran, "Improved estimation of software development effort using classical and fuzzy analogy ensembles," *Applied Soft Computing*, vol. 49, pp. 990–1019, Dec. 2016, doi: 10.1016/j.asoc.2016.08.012.
[34] F. Sarro, F. Ferrucci, and C. Gravino, "Single and multi objective genetic programming for software development effort estimation," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, Mar. 2012, pp. 1221–1226. doi: 10.1145/2245276.2231968.
[35] L. Friedman and O. V Komogortsev, "Assessment of the effectiveness of seven biometric feature normalization techniques," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 10, pp. 2528–2536, Oct. 2019, doi: 10.1109/TIFS.2019.2904844.
[36] A. Mittal, K. Parkash, and H. Mittal, "Software cost estimation using fuzzy logic," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 1, pp. 1–7, Jan. 2010, doi: 10.1145/1668862.1668866.
[37] E. Khatibi and V. K. Bardsiri, "Model to estimate the software development effort based on in-depth analysis of project attributes," *IET Software*, vol. 9, no. 4, pp. 109–118, Aug. 2015, doi: 10.1049/iet-sen.2014.0169.
[38] O. Hidmi and B. E. Sakar, "Software development effort estimation using ensemble machine learning," *International Journal of Computing, Communication and Instrumentation Engineering*, vol. 4, no. 1, pp. 143–147, Jun. 2017, doi: 10.15242/IJCCIE.E0317026.
[39] M. Usman, K. Petersen, J. Börstler, and P. S. Neto, "Developing and using checklists to improve software effort estimation: a multi-case study," *Journal of Systems and Software*, Dec. 2018, doi: 10.1016/j.jss.2018.09.054.
[40] N. Ghatasheh, H. Faris, I. Aljarah, and R. M. H. Al-Sayyed, "Optimizing software effort estimation models using firefly algorithm," *Journal of Software Engineering and Applications*, 2019, doi: 10.4236/jsea.2015.83014.
[41] P. Rani, R. Kumar, A. Jain, and S. K. Chawla, "A hybrid approach for feature selection based on genetic algorithm and recursive feature elimination," *International Journal of Information System Modeling and Design*, vol. 12, no. 2, pp. 17–38, Apr. 2021, doi: 10.4018/IJISMD.2021040102.

[42] B. Al-Salemi, M. Ayob, and S. A. M. Noah, "Feature ranking for enhancing boosting-based multi-label text categorization," *Expert Systems with Applications*, vol. 113, pp. 531–543, Dec. 2018, doi: 10.1016/j.eswa.2018.07.024.
[43] H. Rao *et al.*, "Feature selection based on artificial bee colony and gradient boosting decision tree," *Applied Soft Computing*, vol. 74, pp. 634–642, Jan. 2019, doi: 10.1016/j.asoc.2018.10.036.
[44] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, Feb. 2015, doi: 10.1016/j.infsof.2014.07.005.

BIOGRAPHIES OF AUTHORS

K. Eswara Rao received the Doctorate Degree in Computer Science and Engineering (CSE) from GITAM University, Visakhapatnam, AP, India, in 2023, and the Masters Degree in Neural Networks specialization in the CSE from JNT University, Kakinada, AP, India, in 2009. He is currently working as Associate Professor in Aditya Institute of Technology and Management (AITAM), Tekkali, Srikakulam. His research interests include machine learning, data mining, data analytics, and operating system. He has published numerous conference proceedings as well as published Scopus, Wos, Google Scholors indexed papers also published various international books. He can be contacted at email: eswarkoppala@gmail.com.

Pandu Ranga Vital Terlapu obtained his Degree in CS from Andhra University in A.P,. He pursued his M. Tech in CSE from ANU in A.P., and completed his Ph.D. in CSE from GITAM University. With a total of 24 years of teaching and 18 years of research experience, he currently holds the position of Professor in the Department of Computer science and Engineering at Aditya Institute of Technology and Management (AITAM), India. Dr. Terlapu has contributed to the field of computer science with over 50 research papers published in reputed international journals, including SCI, SCOPUS-indexed journals, and conferences published by Springer, Elsevier, and available online. He can be contacted at email: vital2927@gmail.com.

Paidi Annan Naidu working as an Associate Professor, Department of Computer Science and Engineering in Aditya Institute of Technology and Management (Autonomous), Tekkali, Srikakulam, AndhraPradesh, India. He has 15 years of teaching experience in Engineering Colleges/University and he has published many research papers in UGC/WoS/Scopus indexed journals, three patents, three book chapters and in the proceedings of several conferences. His area of research includes data mining, artificial intelligence, machine learning, and GAN. He is a life member of ISTE and various reputed computer science associations. He also served as reviewer, editorial board member and co-chair for various international journals and conferences respectively. He can be contacted at email: annanpaidi@gmail.com.

Tammineni Ravi Kumar received the Doctorate Degree in Computer Science and Engineering (CSE) from GITAM University, Visakhapatnam, AP, India, in 2023, and the Masters Degree in Computer Science and Engineering CSE from JNT University, Hyderabad, AP, India, in 2008. He is currently working as Associate Professor in Aditya Institute of Technology and Management (AITAM), Tekkali, Srikakulam. His research interests include machine learning, data mining, data analytics, software engineering, and network security. He has published numerous conference proceedings as well as papers in international journals. He can be contacted at email: ravi.4u@adityatekkali.edu.in.

Bala Murali Pydi received the Bachelor's Degree in Electrical and Electronics engineering from JNT University, Kakainada in 2001, the Master's Degree in Powersystem from NIT Jamshedpur in 2006, and the Completed philosophy of doctorate in Electrical and Electronics Engineering in NIT Jamshedpur, respectively. Now he is currently working as an Associate Professor at the Department of Electrical and Electronics Engineering, Aditya Institute of Technology and Management, Tekkali, Andhra Pradesh, India. His research areas include power systems, electrical machines, renewable energy sources, and control systems. He has been serving as a reviewer for many highly-respected journals. He can be contacted at email: balu_p4@yahoo.com.