Diverse Bagging Effort Estimation Model for Software Development Project

Mohammad Haris, Fang-Fang Chua(✉), and Amy Hui-Lan Lim

Faculty of Computing and Informatics, Multimedia University, Cyberjaya, Malaysia
1221400087@student.mmu.edu.my, {ffchua,amy.lim}@mmu.edu.my

Abstract. Creating successful projects is challenging and estimation of software development efforts is thus an important activity of the software engineering community. It enables project managers to organize and manage project quality, cost, resources, and timelines. However, standard techniques for estimating development effort struggle due to the increased complexity, dynamic requirements, multifaceted nature, non-linear relationship, and greater interdependencies of modern software. Various machine learning models have been created periodically to tackle the deficiencies of standard estimation techniques. Nevertheless, the deployment is limited due to inefficient model-constructing approaches and inconclusive results. By meticulously optimizing preprocessing and hyperparameter tuning steps, this research presents a Diverse Bagging Effort ESTimation (DBEEST) model for more reliable and accurate software development effort estimation. To accomplish this, six homogeneous ensembles through bagging were applied to the USP05-FT and SEERA datasets. Subsequently, the predictions of each homogeneous ensemble were combined through averaging to generate a more reliable and accurate prediction with improved robustness against inconsistencies and errors. The results demonstrate the DBEEST model outperformed all individual bagging ensembles and produced consistent results by delivering an overall average of low Mean Square Error, Root Mean Square Error, Mean Absolute Error, and Mean Magnitude Relative Error values and an overall average of high Coefficient of determination values across both diverse datasets. Moreover, the proposed model can improve efficiency in handling software development projects, resource optimization, facilitating informed decision-making, and on-time project completion.

Keywords: Software Project Effort Estimation · Ensemble Learning · Ensemble Bagging · Machine Learning Techniques

1 Introduction

As the need for software increases, software development companies must build the software within the agreed budget and timeframe while meeting the client's expectations [1]. This can be achieved by estimating the effort needed to execute a project, as it provides methods to calculate the project's duration and cost for software development. The amount of developer force required to fulfill the requirement for a certain task is known

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024, LNCS 14814, pp. 293–310, 2024.
https://doi.org/10.1007/978-3-031-64608-9_19

as software effort, and it is typically quantified in units of "person-hours" [1]. Commonly used algorithmic effort estimation techniques such as the Constructive Cost Estimation Model (COCOMO) [2], Putnam's [3], and Function Point Analysis (FPA) [4] which rely on mathematical and statistical work [5], struggle to provide accurate estimations of the effort needed for software project development [6, 7] because they cannot perform early estimations, and the properties utilized could only be calculated after project completion [8, 9]. Furthermore, in the rapidly evolving field of software engineering, these estimation techniques struggle to keep updated particularly with the rise of code reuse and customizable software deployment [10]. For early estimation, expert-based estimation can be a useful non-algorithmic technique for complex or innovative projects, particularly in situations where historical data is restricted [11, 12]. Expert Judgment, Delphi, and planning poker are the most common estimation techniques that rely on Experts [13, 14]. In the Expert judgment, the estimations are done by the Expert, or a group of Experts, based on their own knowledge and expertise [11]. The Delphi technique is a structured interaction technique. It entails privately collecting consensus estimates from a group of experts through iterative rounds of surveys and comments [15], and the Wideband Delphi technique [10] is an extension of the Delphi technique that includes group discussions between rounds. Moreover, Planning Poker is a gamified technique that involves a group determining development effort using a deck of cards, privately assigning values, then disclosing and discussing their choices (especially outliers) until each estimator reaches a consensus [13]. However, overreliance on these expert-based techniques can lead to human bias and subjectivity in the estimation process [2, 14]. Project slippage is a persistent challenge for software project development companies [16]. Therefore, accurate software project effort estimation is essential before developing a project, as it paves the way for successful software development projects, while inaccurate or incorrect estimations can harm companies' sales and reputation, resulting in financial losses [14]. As the procedure of estimating software development effort evolves promptly, it enables ML techniques superior to other standard effort estimation techniques [8, 17]. Hence, the purpose of this research is to propose an ensemble learning-based **D**iverse **B**agging **E**ffort **EST**imation (DBEEST) model by utilizing ML algorithms including Linear Regression (LR), Support Vector Regression (SVR), Decision Tree Regressor (DTR), Multi-Layer Perceptron (MLP), Random Forest Regressor (RFR), and K-Nearest Neighbors (KNN). The USP05-FT [6, 17–19], and SEERA [5, 20] datasets were utilized independently for training and testing the proposed model. Therefore, an extensive approach that predicts software project effort more accurately is presented that starts from smart data pre-processing and optimal parameter tuning to the construction and integration of diverse homogeneous ensembles.

2 Literature Review

The standard effort estimation techniques, including algorithmic methods, are designed to provide mathematical equations for estimating the effort required for a software project development [8, 14, 21]. These algorithmic methods are unable to perform early estimates because the variables on which they rely can only be determined after the project is completed [8]. For early estimation, the expert-based non-algorithmic technique is

widely used in practice [11, 12]. However, overreliance on expert-based techniques may result in inaccurate and usually overoptimistic estimates due to human bias and subjectivity [2, 10, 14]. Therefore, to overcome the deficiencies of these standard techniques and improve the process of estimating software development effort, a range of ML-based techniques have been periodically proposed over the last two decades [2, 7, 15, 22]. In [23], they proposed a ML-based method to estimate effort for agile-based projects, utilizing an SVM from eight different public-domain project issue appraisals as independent features. The study [24] contributed by developing a new effort estimation dataset that takes into consideration the complexity and dynamic nature of modern software. DT, KNN, and SVM were employed on this dataset to estimate the effort, and the DT performed better than the other two. To assess a project's level of risk, the research [25] has proposed a neural network model with three layers, utilizing the backpropagation method, to accurately predict hazardous software projects. To further enhance the estimation process, an ensemble stacking incorporating RF has been introduced, and the study utilizes seven distinct public datasets for training [15]. The paper [26] used nine regression base estimators within the ensemble model to predict the effort, along with hyperparameter tuning to enhance the model's predictive capability. The authors [10] have presented an ensemble averaging model to estimate the software project duration and effort. In the paper [27], the researchers utilized four distinct public datasets to compare the two ML algorithms (SVM and RF) with two deep learning algorithms (Neuralnet and Deepnet). The results of the experiment showed that RF outperformed all algorithms. The study [1] also plays a part in the building of an ideal ensemble-learning-based model by using DT. Based on predictive accuracy, the proposed DT ensemble model surpasses a single DT algorithm. In [5], MLP, LR, and SVR were contrasted with one another to predict software effort, and a relatively new public dataset (SEERA) was used to train and test the models. Based on the evaluation results, LR performed better by having the highest R-squared value.

In [28], researchers have presented the "COSENS" framework, which integrates Cost-Sensitive learning with an Ensemble technique to predict the risk of outsourcing projects. Moreover, the authors [22] proposed bagging for estimating software effort, employing six base estimators and two variable selection methods, the Genetic Algorithm and BestFit. Paper [29] has developed multiple models. One regression and four ML models are built for comparative evaluation. In the study [17] thirteen ML models are used to identify the best model for effort prediction. RF performs best on the "USP05-FT" dataset, while KStar, Additive Regression, and Reduce Error Pruning Tree excel on the "USP05" dataset. The study [30] used training datasets from public and non-public domains to evaluate and compare individual and ensemble approaches. The results revealed that ensemble methods consistently outperform individual techniques. The paper by [6] has used Naive Bayes (NB) and Artificial Neural Networks (ANN) for estimating effort, and a rough set analysis approach has been considered for selecting independent variables. In [16], the authors used historical data from two substantial governmental ICT projects to train seven base learning classifiers and aggregate their prediction results using the ensemble voting approach to generate more reliable software cost predictions. The paper [31] proposed the AdaBoost ensemble model by utilizing two base learners, SVM and KNN. Based on the findings, the model predicts software effort

effectively. In [32], the authors propose a novel approach that integrates ANN with the "Cuckoo Optimization Algorithm (COA)" to improve software project estimation. Further, the researchers in [33] employed K-means clustering throughout the three stages of software development with six classifiers such as DT (J48), RF, DT (C4.5), AdaBoost, NB, and Bagging to examine the efficiency of defect prediction. Another paper [13] proposed an autonomous estimation method for agile methodology story cards, together with supervised learning. As a result of their findings, the researchers exhibited that the DT (J48), Logistic Model Tree (LMT) and Planning Poker, as well as DT (J48) and Planning Poker all outperformed manual Planning Poker.

Upon thoroughly exploring a diverse range of literature related to software project estimation, the ML has the potential to effectively overcome the deficiencies of standard effort estimation techniques, which were observed to be insufficient and outdated [34–36] for estimating today's complex and advanced software project whereas those who do provide early estimates are susceptible to bias and subjective decisions during the estimation process [8, 35–37]. Furthermore, the literature reviews also demonstrated that the ensemble-learning method is notably effective in improving prediction accuracy and robustness [5, 9, 10, 15, 22, 26, 28, 30, 31, 34, 38–40] because it combines the strength of multiple ML algorithms. Ensemble-based models can be constructed as heterogeneous ensembles (train diverse base learners on the same dataset) [41] by utilizing stacking or boosting, or as homogeneous ensembles (train a single base learner on multiple training sets) [10] by utilizing bagging, each with its own distinct strengths and limitations in terms of performance. Therefore, a diverse bagging model that incorporates the strengths of both homogenous and heterogeneous ensembles by integrating six distinctive bagging ensembles is proposed in this study to achieve more accurate and reliable effort estimation for software development projects.

3 Methodology

This section discusses the methodology in stages for constructing a machine learning model by utilizing ensemble bagging for software development effort estimation.

3.1 Stage 1: Data Collection

In this research, two PROMISE [42] repository datasets, namely USP05-FT and SEERA, were used. The USP05-FT [6, 17, 18, 43] is a widely adopted dataset within the field of software project effort estimation. It comprises 76 distinct client/server architecture and web development projects that were completed at the "University of Calgary" in 2005. Each project is described by 15 variables, of which the dependent variable is "Effort," quantified in "person-hours (unit)," and the remaining 14 are independent variables, of which 4 are numerical variables and 10 are categorical variables. Whereas SEERA (**S**oftware engin**EER**ing in Sud**A**n) [5, 20] is a relatively new dataset. It aggregates information from 120 diverse software development projects conducted by 42 software development companies in Sudan. This dataset encompasses a total of 76 variables, where "Actual effort," which is quantified in "person-hours (unit)," serves as the dependent variable, and the other 75 are independent variables, of which nearly 80% of

these independent variables are categorical, which contain information about different software applications across different domains such as Bespoke applications, Enterprise Resource Planning (ERP), Financial and Managerial applications, Banking system applications, Web applications, and Mobile applications. Both datasets are ideally suited for conducting software engineering experiments, which encompass all the necessary variables for predicting software development effort and will be crucial for the training process of the proposed DBEEST model. Table 1 summarizes the proposed datasets based on source repository information, number of attributes, number of projects, and unit of dependent variables.

Table 1. Summary of the proposed datasets

Dataset	Source	No. of Projects	No. of Variables	Unit of Dependent Variable
USP05-FT	PROMISE	76	15	Person-hours
SEERA	PROMISE	120	76	Person-hours

3.2 Stage 2: Data Preprocessing

Effective data preprocessing stands as a crucial stage in the construction of machine learning models [10, 44]. In this stage, considering insights from the previous related studies, four different data preprocessing techniques were implemented.

Detection and Handling of Missing Values. In the data preprocessing stages, instances with missing values in the proposed datasets were addressed by applying imputation techniques such as mean, median, and k-nearest neighbors (KNN) imputation. These imputation techniques were chosen because missing values in both datasets followed a pattern of Missing Completely at Random (MCAR) [45, 46]. Little's test was conducted to validate this pattern. It is a statistical test that evaluates the Null Hypothesis (HO) that the missing values are missing completely at random. It assists in determining whether the missing values occur completely at random or if there is a systematic pattern to the missingness [45, 47]. This test's result demonstrated that the p-value exceeded the predetermined significance level of 0.05, indicating the null hypothesis was not successfully rejected. This implies that the missingness in the datasets is MCAR. In the USP0F-FT dataset, there were a total of 37 missing values across 8 columns (Number of External Outputs, Unadjusted Function Point, Programming Language, Tools, Tool Experience, Database Management System, Method, and Architecture). For columns containing fewer than 5 missing values, numerical columns were imputed using the mean imputation technique, and categorical columns were imputed using the median imputation technique. This involved substituting the values that are missing with the respective median or mean of the observed values [45, 47, 48]. Additionally, the KNN-Imputation technique was employed for the column (Method) with 14 missing values. KNN-Imputation imputes missing values based on the values of their closest neighboring data points. It considers how similar these nearby data points are to the one with the missing value and uses that information to make an estimate for the missing value [48]. In the

second SEERA dataset, 42 columns were identified as containing missing values. The 3 columns (Estimated Size, Outsourcing Impact, Degree of Standards Usage) that have more than 90 missing values were simply dropped. The remaining 39 columns (Object Points, Government Policy Impact, Management Structure Clarity, Developer Hiring Policy, Developer Incentives Policy, Developer Training, Development Team Management, Clarity of Manual System, Project Manager Experience, Consultant Availability, Precedentedness, Software Tool Experience, Programming Language Experience, Programmers Capability, Analysts Capability, Team Selection, Team Contracts, Team Continuity, Team Cohesion, Income Satisfaction, Schedule Quality, Development Environment Adequacy, Tool Availability, Technical Stability, Level of Outsourcing, Degree of Software Reuse, Degree of Risk Management, Use of Standards, Process Reengineering, Requirement Accuracy Level, Technical Documentation, Comments in the Code, User Manual, Required Reusability, Performance Requirements, Product Complexity, Security Requirements, Reliability Requirements, and Specified Hardware) have less than 4 missing values, and the mean imputation was applied to numerical columns, while the median imputation was used for categorical columns to impute the missing values.

Detection and Handling of Outlier Data Points. Handling outliers ensures model robustness, enhances generalization, and prevents biased predictions by addressing data points exhibiting substantial deviations from the remaining values [9, 49]. In this stage, outliers were detected through statistical measures, particularly leveraging the Interquartile Range (IQR). The IQR describes the range of the dataset, and the formula to calculate it is mentioned below.

$$\text{IQR} = \text{Q3} - \text{Q1} \tag{1}$$

In the formula, the IQR is the Interquartile Range, Q3 is the value at which 75% of the distribution is obtained, and Q1 is the value at which 25% of the distribution is obtained. In IQR, outliers can usually be detected with a threshold of 1.5 times the IQR [49, 50]. The upper fence is calculated as "$\text{Q3} + 1.5 \times \text{IQR}$," while the lower fence is calculated as "$\text{Q1} - 1.5 \times \text{IQR}$." Therefore, in the USP05-FT dataset, 2 variables, namely Number of External Inputs and Unadjusted Function Point, were identified that have outliers with the highest values. The calculated upper fence values of these variables are 14 and 25. In order to preserve meaningful information, only those outliers that were close to their calculated upper fence values are converted to their respective upper fence values. In IQR, this process is known as capping. On the other hand, in the SEERA dataset, 7 variables with outliers were identified, namely Estimated Duration, Actual Duration, Object Point, Estimated Effort, Actual Effort, Team Size, Dedicated Team Members. The calculated upper fence values of these variables are 13, 22, 778, 13921, 21912, 10, and 11, respectively. First, the IQR trimming method was applied to those outliers that were extremely high compared to their upper fence values. Then, the IQR capping method was utilized for other outliers that were close to their calculated upper fence values.

Feature Selection. The variables obtained from the proposed datasets were sequentially analyzed to assess their suitability for feature extraction and to uncover the potential relationships or dependencies among them. In this study, Karl Pearson's product-moment correlation coefficient was initially used as a statistical metric to ascertain the association between numerical independent and numerical dependent variables [16]. The coefficient falls within the range of + 1 to −1. A positive coefficient value denotes a direct correlation. A negative value denotes a negative correlation. A coefficient of zero ("0") denotes that there is no correlation observed between the variables [16, 26]. After employing the correlation coefficient to examine the relationships between numerical dependent and numerical independent variables in both datasets, it was observed that in the USP05-FT dataset, 2 independent numerical variables, Number of External Inputs and Number of Data Files, exhibited very weak negative correlations, approaching zero ("0"), with values of −0.0061 and −0.0512, respectively. Meanwhile, in the SEERA dataset, 2 independent numerical variables, Daily Working Hours and Team Continuity, exhibited weak positive correlations, and 1 independent numerical variable, Multiple Programming Languages, exhibited weak negative correlation, approaching zero ("0") with values of 0.0197, 0.0806, and −0.0487, respectively. The crucial categorical independent variables across both datasets were identified by employing Analysis of Variance (ANOVA) [51]. It is typically utilized when the independent variables are categorical in nature and the dependent variable is numerical. In ANOVA, variables with high F-statistic values and low p-values (generally less than 0.05) are usually considered more important variables. Conversely, low F-statistic values and high p-values (generally greater than 0.05) denote that the independent categorical variable may not have a significant impact on the dependent numerical variable [51, 52]. The ANOVA test identified a total of 17 insignificant independent categorical variables by analyzing their associated F-values and p-values. Specifically, 2 variables, Project ID and Method, from the USP05-FT dataset, exhibited low F-values of 0.0000 and 0.0006 respectively, and high p-values (>0.05) of 0.9965, and 0.9796, respectively, and 15 variables from the SEERA dataset such as Management Structure Clarity, User Manual, Software Tool Experience, Degree of Software Reuse, Performance Requirements, Requirements Flexibility, Use of Standards, Tool Availability, Reliability Requirements, Economic Instability Impact, Users Stability, Organization ID, Project ID, Technical Stability, and Schedule Quality, exhibited low F-values of 0.0017, 0.0048, 0.0049, 0.0060, 0.0061, 0.0097, 0.0145, 0.0423, 0.0435, 0.0789, 0.0930, 0.1426, 0.1521, 0.1610, and 0.1654, respectively, and high p-values (>0.05) of 0.9671, 0.9446, 0.9442, 0.9380, 0.9375, 0.9216, 0.9040, 0.8374, 0.8352, 0.7793, 0.7609, 0.7063, 0.7432, 0.6890, and 0.6850, respectively. Therefore, irrelevant numerical variables identified through Pearson's correlation and irrelevant categorical variables identified through ANOVA were eliminated from both datasets to enhance the proposed model's performance, interpretability, and computational efficiency [51, 52].

Feature Scaling. The purpose of feature scaling is to modify the range of independent numerical variables, making them more compatible with one another while preventing certain variables from dominating the learning process [26]. In this stage, the standardization method for feature scaling was utilized for both proposed datasets [1]. This was necessary because certain proposed models, such as SVR, KNN, and MLP, are sensitive to variations in the scale of input variables [26]. The standardization centers

the data on a mean of "0" and scales it to a standard deviation of "1." The following formula is used to calculate the standard scale for the variable 'p' [43]:

$$S_{standardized} = \frac{p - \mu}{\sigma} \quad (2)$$

where "$S_{standardized}$" is the standardized values of the variables, "p" is the initial variable values, "μ" is the mean of the considered training samples, "σ" is the standard deviation of the variable values.

The data preprocessing entails cleaning, standardizing numerical variables, reducing irrelevant variables, and splitting the datasets into an 80–20 ratio, 80% training sets, and 20% test sets. This sets the stage for effective model training and testing.

3.3 Stage 3: Hyperparameter Tunning

The hyperparameters are the set of parameters that are not explicitly assigned or learned by the estimators or models [26]. These parameters are not predefined as defaults by any estimator. However, hyperparameter tuning is the technique of systematically searching for the optimum setting of these hyperparameters, which govern the performance of ML models [26, 28, 53, 54]. In this stage, the Grid Search Cross-Validation (GridSearchCV) hyperparameter tuning technique [55] is implemented. As it systematically explores all potential parameter combinations, this technique is deemed to be more effective [26, 55]. In the context of GridSearchCV, Cross Validation (CV) is an essential step in the hyperparameter tuning process. The training data serves as the input, and the cross-validation technique handles the dynamic creation of validation sets while examining various hyperparameter settings [55]. It performed K-fold cross-validation at each point on the predefined hyperparameter grid to evaluate performance [26, 53, 54]. Moreover, as GridSearchCV handles cross-validation internally, there is no need to explicitly specify a separate validation set [54]. The optimum parameter settings for the proposed base estimators, as well as their corresponding ensemble bagging models, were determined on both proposed datasets through this GridSearchCV hyperparameter tuning technique. In each dataset, 10-fold cross-validation was applied. This entailed splitting the respective training set into 10 subsets (folds), 9 for training and 1 for validation in each iteration. This process was repeated 10 times to ensure that each subset was utilized as a validation set at least once. Furthermore, these parameter settings were determined based on the set of hyperparameters that produced optimal 10-fold cross-validated results.

3.4 Stage 4: Construction of Proposed Model (DBEEST)

To construct the proposed DBEEST model, six base estimators, MLP, SVR, LR, DTR, RFR, and KNN, are utilized to construct six separate homogenous ensembles through the bagging method. Each homogeneous ensemble is constructed on a distinct estimator that leverages the diverse learning capabilities of these models or estimators to identify a broad spectrum of patterns and relationships within the data. To train each of these six separate homogeneous ensembles, the training sets (80%) acquired from the proposed datasets are utilized independently. Bagging is implemented on all six homogeneous

ensembles, yielding six distinct prediction outcomes. The final prediction was derived by combining the predictions from all six homogeneous ensembles using the averaging method. Therefore, the estimate will be more reliable and accurate. The six base estimators and bagging method proposed for constructing the DBEEST model have been successfully employed in multiple prior studies. The estimators and bagging method were chosen for their proven effectiveness in software estimation and their ability to provide adequate predictions. The descriptions of the base estimators and ensemble bagging are presented in the following sections.

Base Estimators. MLP is a supervised algorithm that gains insights from a training dataset using a function $f(a) : S^m \rightarrow S^q$ in which 'm' denotes the quantity of variables with regard to input and 'q' denotes the quantity of variables with regard to output [56]. The MLP is capable of learning a non-linear function for a given set of variables signified as W = w1, w2, …, wm and a dependent variable d through neural layers that are related to one another [17]. This allows the MLP to address regression or classification problems effectively [26]. SVR is another supervised learning Support Vector Machine (SVM) method for regression problems [26]. It utilizes the use of support vectors to generate a hyperplane that depicts the relationship between independent variables and continuous dependent variables [26, 34]. LR is a parametric model that is utilized in regression scenarios when the dependent variable is assumed to be a combination of other variables in a linear manner [22]. It seeks to identify the best-fitting linear line that minimizes the difference between the actual and predicted values [26]. While DT is a non-parametric method designed to develop a model using decision rules derived from data variables [16]. It is a binary tree, with its root positioned at the top, and the data is arranged into a tree structure to handle different ML-related problems. DT utilizes a collection of predictor variables to demonstrate how a target variable is predicted [2]. In dealing with regression problems with DT, the Decision Tree Regressor (DTR) is used [26]. Furthermore, RF is an ensemble meta-estimator composed of several decision trees [9]. It works by constructing numerous Decision Trees (DTs) and then aggregating their predicted results to make ultimate predictions. Every decision tree is trained on a randomly generated portion of the training data and attributes [17]. To generate a prediction, the RF takes a set of attributes and distributes them under each tree in the forest. Random Forest Regressor (RFR) is implemented when dealing with regression problems [13, 26]. Lastly, KNN is another non-parametric method based on proximity that generates predictions by locating the K nearest observations to a provided input and averaging their dependent values (for regression) or choosing the majority class (for classification) [17]. The fundamental concept of this method is case-based reasoning, which includes processes like detecting cases, collecting related cases, and adopting cases. Regression-based neighboring methods are employed when dealing with continuous data [26].

Ensemble Bagging. Bagging, short for "Bootstrap Aggregation", is an ensemble learning method that assists ML models to increase their stability and accuracy [33]. It is well-effective for managing the balance between underfitting and overfitting, reducing the variance of the predictive model [22]. Bagging generates several bootstrap samples from the training set, each of which is used to train multiple base estimators of same

type. To obtain a final prediction, all base estimator predictions are aggregated [28]. Mathematically, this bagging process can be described as follow.

$$\hat{f} = \frac{1}{E} \sum_{a=1}^{E} t_a(d) \tag{3}$$

where $\hat{f}$ is the final prediction, **E** is the total no of base estimators, t_a denotes the a-th base estimator, and **d** denotes the individual data points. Each base estimator t_a is trained on a distinct bootstrap sample, and their predictions are aggregated by using averaging method for regression problem or majority voting method for classification problem.

3.5 Stage 5: Proposed Model Evaluation Criteria

In this stage, five established and reliable evaluation metrics were used for evaluation. The brief explanations of the proposed evaluation metrics are discussed below:

Mean Squared Error (MSE). It is utilized to evaluate quality by computing the average of squared errors between actual values and predictions [26]. MSE is non-negative; therefore, a value of zero demonstrates a high-quality estimator [36]. MSE is calculated for each individual observation 'a' across 'm' samples as the sum of squared differences between actual and predicted values, divided by the total no of samples [26, 36].

$$\text{MSE} = \frac{1}{m} \sum_{a=0}^{m-1} (\text{Actual value}_a - \text{Predicted value}_a)^2 \tag{4}$$

Root Mean Squared Error (RMSE). It provides the sample's standard deviation for the variations between actual values and predicted values [10, 36]. This evaluation is used to address a limitation in which MAE's sensitivity to moderate errors understates the impact of large errors. Evaluating larger errors is important as they signify critical deviations from predicted model performance. It is computed by finding the square root of the mean of the squared differences between actual and predicted values [15]. The equation is mentioned as follows:

$$\text{RMSE} = \sqrt{\frac{1}{m} \sum_{a=0}^{m-1} (\text{Actual value}_a - \text{Predicted value}_a)^2} \tag{5}$$

Mean Absolute Error (MAE). It serves as a risk evaluation metric that evaluates the predicted absolute error rate between actual values and predictions [6]. It is calculated for each individual data point 'a' among 'm' no of samples using the following formula [26]:

$$\text{MAE} = \frac{1}{m} \sum_{a=0}^{m-1} |\text{Actual value}_a - \text{Predicted value}_a| \tag{6}$$

Mean Magnitude of Relative Error (MMRE). It calculates the mean of the magnitude of the relative errors between actual and predicted values in a set of data points [1, 10]. The MMRE for every individual observation 'a' can be calculated using the following formula [6, 26, 29]:

$$\text{MMRE} = \frac{1}{m} \sum_{a=0}^{m-1} \left| \frac{\text{Actual value}_a - \text{Predicted value}_a}{\text{Actual value}_a} \right| \tag{7}$$

Coefficient of Determination (R-Square/R^2). It represents the proportion of the dependent variable's variance explained by the independent variable(s) in a regression model [6, 36]. This proportion is used to determine how accurately the actual values are predicted. A R^2 value close to 1 is regarded as the finest achievable score.

The R^2 for each individual data point 'a' can be computed over the total amount of samples 'm' as follows [26].

$$R_a^2 = 1 - \frac{\sum_{a=1}^{m} (\text{Actual value}_a - \text{Predicted value}_a)^2}{\sum_{a=1}^{m} (\text{Actual value}_a - \bar{y})^2} \tag{8}$$

where, $\bar{y} = \sum_{a=1}^{m} \text{Actual value}_a$

Figure 1 depicts the architecture of the DBEEST. The initial section of the figure depicts the processes of smart data preparation and hyperparameter optimization whereas the second section depicts the construction and training of six separate homogeneous ensembles through bagging (or Bootstrap Aggregation). The third section of the figure depicts the average method for integrating all six homogeneous ensemble predictions, and the figure's fourth and final section depicts the evaluation of the proposed model.

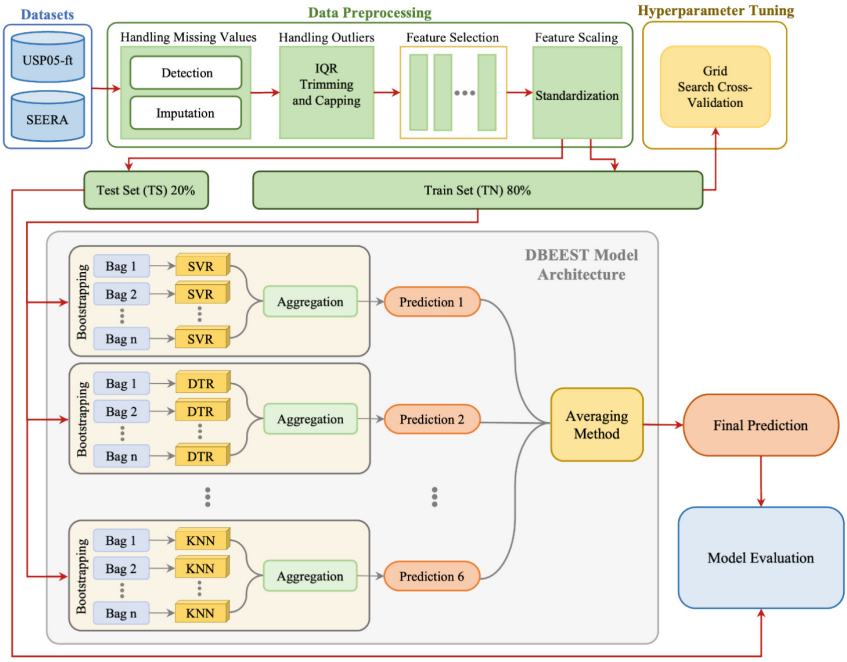

Fig. 1. Process of constructing the proposed model (DBEEST) to estimate software effort.

4 Results and Discussion

In this section, the experimental results of the proposed model (DBEEST) along with the results of six separate homogeneous ensembles based on bagging on both proposed datasets are presented individually. It was discovered that the smaller values of the MSE, RMSE, MAE, and MMRE values, along with an R^2 value nearer to 1, indicated a better-performing model. All experiments were performed using Jupyter Notebook (version 6.5.2) with the most recent version of Python 3.10.12.

4.1 Performance of Models

Tables 2 and 3 present the performance of the models on both proposed datasets. In Table 2, the DBEEST model exhibits the lowest values for MSE, RMSE, MAE, and MMRE, which are 0.02903, 0.17039, 0.13239, and 0.39513, respectively. Additionally, it achieves a higher R^2 value of 0.94554 compared to individual bagging ensembles on the USP05-FT dataset. Table 3 presents the performance of the models on the SEERA dataset, and the results demonstrate that the proposed model again performed well on this dataset, yielding a higher R^2 value of 0.93685 and lower values of 0.07322, 0.27059, 0.20363, and 0.29162 for MSE, RMSE, MAE, and MMRE, respectively.

Table 2. Evaluation results of models using the USP05-FT dataset.

Models	MSE	RMSE	MAE	MMRE	R^2
Bagging SVR	0.10648	0.32632	0.22532	0.66101	0.80027
Bagging DTR	0.05017	0.24042	0.23565	0.59417	0.90465
Bagging LR	0.15432	0.39284	0.31307	0.75264	0.71053
Bagging MLP	0.07768	0.27872	0.21129	0.59373	0.85429
Bagging RFR	0.04094	0.20235	0.24374	0.53931	0.91320
Bagging KNN	0.08173	0.28589	0.18244	0.69922	0.84669
DBEEST Model	**0.02903**	**0.17039**	**0.13239**	**0.39513**	**0.94554**

After a comparative evaluation of the models, it was determined that the DBEEST model outperformed, achieving low values for MSE, RMSE, MAE, MMRE, and high R^2 values on both datasets. In ML, every algorithm functions uniquely, with distinct methodologies and mechanisms for each dataset [10], a fact evident when observing the performance of bagging SVR and bagging KNN in Table 2 and 3. Bagging SVR exhibits good performance on the USP05-FT dataset by providing low error measures of MSE = 0.10648, RMSE = 0.32632, MAE = 0.22532, MMRE = 0.66101, and a high R^2 value of 0.80027, while on SEERA datasets, it appears slightly less accurate by providing marginally higher error measures of MSE = 0.55993, RMSE = 0.74828, MAE = 0.52431, MMRE = 0.74175, and a low R^2 value of 0.51716. Bagging KNN, on the other hand, performs poorly on the SEERA dataset by providing high error measures

Table 3. Evaluation results of models using the SEERA dataset.

Models	MSE	RMSE	MAE	MMRE	R^2
Bagging SVR	0.55993	0.74828	0.52431	0.74175	0.51716
Bagging DTR	0.09561	0.30596	0.24005	0.90759	0.90927
Bagging LR	0.49503	0.70358	0.60785	1.52982	0.57312
Bagging MLP	0.19033	0.43627	0.34446	1.23672	0.83587
Bagging RFR	0.09957	0.31555	0.23740	0.84309	0.91413
Bagging KNN	0.65338	0.80832	0.62331	1.28796	0.43658
DBEEST Model	**0.07322**	**0.27059**	**0.20363**	**0.29162**	**0.93685**

of MSE = 0.65338, RMSE = 0.80832, MAE = 0.62331, MMRE = 1.28796, and a low R^2 value of 0.43658, but performs well on the USP05-FT dataset by providing low error measures of MSE = 0.08173, RMSE = 0.28589, MAE = 0.18244, MMRE = 0.69922, and a high R^2 value of 0.84669. The diversity of estimators in the proposed DBEEST model can effectively mitigate this issue by leveraging the strengths of multiple algorithms. This can be proved by examining the evaluation metric results for the proposed model in Tables 2 and 3 across diverse datasets. The performance of the proposed model was thoroughly evaluated across these two diverse datasets, each representing a wide variety of projects with different characteristics and requirements. The SEERA dataset encompasses projects with varying software sizes, development methodologies, complexities, programming languages, application domains, and extensive feature sets, while the USP05-FT dataset focuses on smaller-scale projects from different domains with distinct characteristics and requirements. These projects represented scenarios with distinct challenges, such as the complexity of internal calculations, resource constraints, and intricate software architectures. The DBEEST model has demonstrated consistent and robust performance across these diverse datasets, showcasing its ability to predict software effort effectively in various project scenarios. Furthermore, the DBEEST model structure, which includes training base estimators on diverse subsets (bags) and integrating different bagging ensembles, enhances robustness and generalization while reducing individual model variability. Table 4 provides the DBEEST models' average overall prediction performance across both datasets for assessing their generalization ability, determined by taking the average of the evaluating metrics results. Additionally, Figs. 2 and 3 depict the graphical representation of actual effort values and the predicted effort values of a DBEEST model on the USP05-FT and SEERA datasets, respectively. The green grid lines with green square datapoints represent the actual effort values, whereas the blue grid lines with red circle datapoints represent the predicted effort values. The closer the actual and predicted effort values are, the lower the error and the better the model. The graphs demonstrate that the actual effort and predicted effort values are close to each other.

Table 4. Average overall performance of the DBEEST model on both proposed datasets.

Datasets	MSE	RMSE	MAE	MMRE	R^2
USP05-FT	0.02903	0.17039	0.13239	0.39513	0.94554
SEERA	0.07322	0.27059	0.20363	0.29162	0.93685
Overall Performance	**0.05112**	**0.22099**	**0.16801**	**0.39337**	**0.94120**

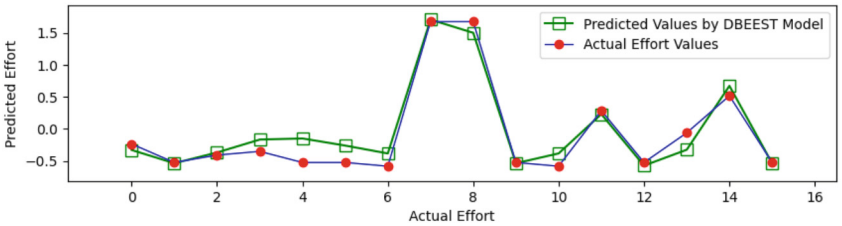

Fig. 2. Actual vs predicted effort using the DBEEST Model for the USP05-FT dataset.

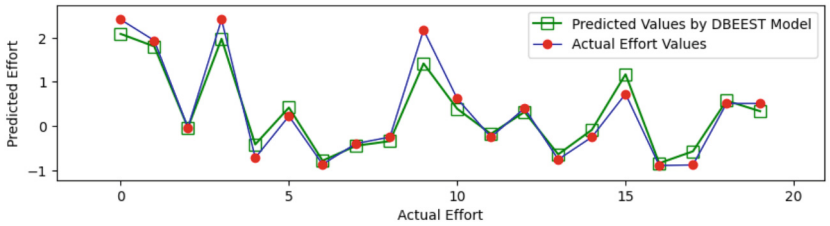

Fig. 3. Actual vs predicted effort using the DBEEST Model for the SEERA dataset.

5 Conclusion and Future Work

This study proposed a **D**iverse **B**agging **E**ffort **EST**imation (DBEEST) model to enhance software project effort estimation. Six separate homogenous bagging ensembles are constructed by using multiple diverse base estimators such as SVR, DTR, LR, MLP, RFR, and KNN. They are then combined through an averaging method to produce the final prediction. To further enhance the model's performance, data pre-processing techniques and hyperparameter tuning are employed on the USP05-FT and SEERA datasets. According to the evaluation results, the proposed DBEEST model provides consistent and good results with an overall average of low MSE, RMSE, MAE, and MMRE values (0.05112, 0.22099, 0.16801, and 0.39337, respectively) and an overall average of high R^2 values (0.94120) across both proposed datasets and outperformed all individual bagging ensembles. This research will significantly help software development companies enhance their software project management capabilities, resulting in heightened success and effectiveness in their software projects. The future prospects are to integrate the expert estimation technique with this proposed model (DBEEST). By integrating expert estimations, the

machine learning model is able to capture the intuition and human experience for highly innovative projects.

References

1. Bakmeedeniya, T.: Estimation of software development effort based on decision tree approach (2021)
2. Ritu, Garg, Y.: Comparative analysis of machine learning techniques in effort estimation. In: 2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON). IEEE, pp. 401–405 (2022)
3. Issa, A., Odeh, M., Coward, D.: Software cost estimation using use-case models: a critical evaluation. In: Proceedings - 2006 International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA 2006 (2006)
4. Shrivastava, A.K.: Comparative study of different project size estimation technique for the development of software. Int. J. Sci. Res. Sci. Technol., 231–236 (2018). https://doi.org/10.32628/IJSRST18401142
5. Assefa, Y., Berhanu, F., Tilahun, A., Alemneh, E.: Software effort estimation using machine learning algorithm. In: 2022 International Conference on Information and Communication Technology for Development for Africa, ICT4DA 2022 (2022)
6. Shivhare, J., Rath, S.K.: Software effort estimation using machine learning techniques. In: ACM International Conference Proceeding Series (2014)
7. Zakaria, N.A., Ismail, A.R., Ali, A.Y., et al.: Software project estimation with machine learning. Int. J. Adv. Comput. Sci. Appl., **12** (2021). https://doi.org/10.14569/IJACSA.2021.0120685
8. Marapelli, B.: Software development effort duration and cost estimation using linear regression and K-nearest neighbors machine learning algorithms. Int. J. Innov. Technol. Exploring Eng. (2019)
9. Satapathy, S.M., Acharya, B.P., Rath, S.K.: Early stage software effort estimation using random forest technique based on use case points. IET Softw. **10** (2016). https://doi.org/10.1049/iet-sen.2014.0122
10. Pospieszny, P., Czarnacka-Chrobot, B., Kobylinski, A.: An effective approach for software project effort and duration estimation with machine learning algorithms. J. Syst. Softw. **137** (2018). https://doi.org/10.1016/j.jss.2017.11.066
11. Kapoor, D., Gupta, R.K.: Software cost estimation techniques– a review of literature. Int. J. Res. Dev. Appl. Sci. Eng. (IJRDASE), 9359 (2014)
12. Sarro, F., Moussa, R., Petrozziello, A., Harman, M.: Learning from mistakes: machine learning enhanced human expert effort estimates. IEEE Trans. Softw. Eng. **48** (2022). https://doi.org/10.1109/TSE.2020.3040793
13. Moharreri, K., Sapre, A.V., Ramanathan, J., Ramnath, R.: Cost-effective supervised learning models for software effort estimation in agile environments. In: Proceedings - International Computer Software and Applications Conference (2016)
14. Fernández-Diego, M., Méndez, E.R., González-Ladrón-De-Guevara, F., et al.: An update on effort estimation in agile software development: a systematic literature review. IEEE Access **8** (2020)
15. Priya Varshini, A.G., Anitha Kumari, K., Varadarajan, V.: Estimating software development efforts using a random forest-based stacked ensemble approach. Electronics (Switzerland) **10** (2021). https://doi.org/10.3390/electronics10101195
16. Makris, C., Vikatos, P., Visser, J.: Classification model for predicting cost slippage in governmental ICT projects. In: Proceedings of the ACM Symposium on Applied Computing (2015)

17. Al Asheeri, M.M., Hammad, M.: Machine learning models for software cost estimation. In: 2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies, 3ICT 2019 (2019)
18. Amazal, F.A., Idri, A., Abran, A.: Analysis of cluster center initialization of 2FA-kprototypes analogy-based software effort estimation. J. Softw. Evol. Process **31** (2019). https://doi.org/10.1002/smr.2180
19. Hammad, M.: Software cost estimation using stacked ensemble classifier and feature selection. Int. J. Adv. Comput. Sci. Appl. **14** (2023). https://doi.org/10.14569/IJACSA.2023.0140621
20. Mustafa, E.I., Osman, R.: SEERA: a software cost estimation dataset for constrained environments. In: PROMISE 2020 - Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Co-located with ESEC/FSE 2020 (2020)
21. Bilal, K., Wahab, K., Muhammad, A., Nazir, J.: Software cost estimation: algorithmic and non-algorithmic approaches. Int. J. Data Sci. Adv. Anal. **2** (2020)
22. Alhazmi, O.H., Khan, M.Z.: Software effort prediction using ensemble learning methods. J. Softw. Eng. Appl. **13** (2020). https://doi.org/10.4236/jsea.2020.137010
23. Scott, E., Pfahl, D.: Using developers' features to estimate story points. In: ACM International Conference Proceeding Series (2018)
24. Rahman, M., Roy, P.P., Ali, M., et al.: Software effort estimation using machine learning technique. Int. J. Adv. Comput. Sci. Appl. **14** (2023). https://doi.org/10.14569/IJACSA.2023.0140491
25. Han, W.M.: Discriminating risky software project using neural networks. Comput. Stand. Interfaces **40** (2015)
26. Kanneganti, A.: Using ensemble machine learning methods in estimating software development effort (2020)
27. Priya Varshini, A.G., Anitha Kumari, K., Janani, D., Soundariya, S.: Comparative analysis of machine learning and deep learning algorithms for software effort estimation. J. Phys. Conf. Ser. (2021)
28. Hu, Y., Feng, B., Mo, X., et al.: Cost-sensitive and ensemble-based prediction model for outsourced software project risk prediction. Decis. Support Syst. **72** (2015). https://doi.org/10.1016/j.dss.2015.02.003
29. Malhotra, R., Jain, A.: Software effort prediction using statistical and machine learning methods. Int. J. Adv. Comput. Sci. Appl. **2** (2011). https://doi.org/10.14569/ijacsa.2011.020122
30. Mahmood, Y., Kama, N., Azmi, A., et al.: Software effort estimation accuracy prediction of machine learning techniques: a systematic performance evaluation. Softw. Pract. Exp. **52** (2022). https://doi.org/10.1002/spe.3009
31. Hidmi, E.: Software development effort estimation using ensemble machine learning. Int. J. Comput. Commun. Instrum. Eng. **4** (2017). https://doi.org/10.15242/IJCCIE.E0317026
32. Desai, V.S., Mohanty, R.: ANN-cuckoo optimization technique to predict software cost estimation. In: 2018 Conference on Information and Communication Technology, CICT 2018. Institute of Electrical and Electronics Engineers Inc. (2018)
33. Ramaswamy, V., Suma, V., Pushphavathi, T.P.: An approach to predict software project success by cascading clustering and classification. IET Semin. Dig. (2012)
34. Ghotra, B., McIntosh, S., Hassan, A.E.: Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings - International Conference on Software Engineering (2015)
35. Rekha, T., Rai, P.K.: Machine learning methods of effort estimation and it's performance evaluation criteria. Int. J. Comput. Sci. Mob. Comput. **6** (2017)

36. Dada, E.G., Oyewola, D.O., Joseph, S.B., Duada, A.B.: Ensemble machine learning model for software defect prediction. Adv. Mach. Learn. Artif. Intell. **2** (2021). https://doi.org/10.33140/amlai.02.01.03
37. Mahdi, M.N., Mohamed, M.Z., Yusof, A., et al.: Design and development of machine learning technique for software project risk assessment - a review. In: 2020 8th International Conference on Information Technology and Multimedia, ICIMU 2020 (2020)
38. Sehra, S.K., Brar, Y.S., Kaur, N., Sehra, S.S.: Research patterns and trends in software effort estimation. Inf. Softw. Technol. **91** (2017)
39. Singh, R.: Machine learning algorithms and ensemble technique to improve prediction of students performance. Int. J. Adv. Trends Comput. Sci. Eng. **9** (2020). https://doi.org/10.30534/ijatcse/2020/221932020
40. Sagi, O., Rokach, L.: Ensemble learning: a survey. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. **8** (2018)
41. Luong, A.V., et al.: A homogeneous-heterogeneous ensemble of classifiers. In: Yang, H., Pasupa, K., Leung, A.C.-S., Kwok, J.T., Chan, J.H., King, I. (eds.) Neural Information Processing. CCIS, vol. 1333, pp. 251–259. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63823-8_30
42. Wang, H., Zhuang, W., Zhang, X.: Software defect prediction based on gated hierarchical LSTMs. IEEE Trans. Reliab. **70** (2021). https://doi.org/10.1109/TR.2020.3047396
43. Idri, A., Amazal, F.A., Abran, A.: Accuracy comparison of analogy-based software development effort estimation techniques. Int. J. Intell. Syst. **31** (2016). https://doi.org/10.1002/int.21748
44. Sathesh, A., Hamdan, Y.B.: Analysis of software sizing and project estimation prediction by machine learning classification. J. Ubiquit. Comput. Commun. Technol. **3** (2022). https://doi.org/10.36548/jucct.2021.4.006
45. Nakai, M., Chen, D.-G., Nishimura, K., Miyamoto, Y.: Comparative study of four methods in missing value imputations under missing completely at random mechanism. Open J. Stat. **04** (2014). https://doi.org/10.4236/ojs.2014.41004
46. Pereira Barata, A., Takes, F.W., Van Den Herik, H.J., Veenman, C.J.: Imputation methods outperform missing-indicator for data missing completely at random. In: IEEE International Conference on Data Mining Workshops, ICDMW (2019)
47. Abdelhadi, S., Elbahnasy, K., Abdelsalam, M.: A proposed model to predict auto insurance claims using machine learning techniques. J. Theor. Appl. Inf. Technol. **98** (2020)
48. Jadhav, A., Pramod, D., Ramanathan, K.: Comparison of performance of data imputation methods for numeric dataset. Appl. Artif. Intell. **33** (2019). https://doi.org/10.1080/08839514.2019.1637138
49. Iwata, K., Nakashima, T., Anan, Y., Ishii, N.: Applying machine learning classification to determining outliers in effort for embedded software development projects. In: Proceedings - 2019 6th International Conference on Computational Science/Intelligence and Applied Informatics, CSII 2019 (2019)
50. Dovoedo, Y.H., Chakraborti, S.: Boxplot-based outlier detection for the location-scale family. Commun. Stat. Simul. Comput. **44**, 1492–1513 (2015). https://doi.org/10.1080/03610918.2013.813037
51. Elssied, N.O.F., Ibrahim, O., Osman, A.H.: A novel feature selection based on one-way ANOVA F-test for e-mail spam classification. Res. J. Appl. Sci. Eng. Technol. **7** (2014). https://doi.org/10.19026/rjaset.7.299
52. Gao, K., Khoshgoftaar, T.M., Wang, H., Seliya, N.: Choosing software metrics for defect prediction: an investigation on feature selection techniques. Softw. Pract. Exp. **41** (2011). https://doi.org/10.1002/spe.1043
53. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12** (2011)

54. Tsamardinos, I., Rakhshani, A., Lagani, V.: Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. Int. J. Artif. Intell. Tools (2015)
55. Adnan, M., Alarood, A.A.S., Uddin, M.I., ur Rehman, I.: Utilizing grid search cross-validation with adaptive boosting for augmenting performance of machine learning models. PeerJ Comput. Sci. **8** (2022). https://doi.org/10.7717/PEERJ-CS.803
56. Popescu, M., Balas, V.E., Mastorakis, N.: Multilayer perceptron and neural networks LILIANA PERESCU-POPESCU. WSEAS Trans. Circuits Syst. **8**, 579–588 (2009)