Research Article

IoT-based group size prediction and recommendation system using machine learning and deep learning techniques

Deepti Chopra[1] · Arvinder Kaur[1]

Received: 28 July 2020 / Accepted: 5 January 2021 / Published online: 20 January 2021

Abstract
In an open source software development environment, it is hard to decide the number of group members required for resolving software issues. Developers generally reply to issues based totally on their domain knowledge and interest, and there are no predetermined groups. The developers openly collaborate on resolving the issues based on many factors, such as their interest, domain expertise, and availability. This study compares eight different algorithms employing machine learning and deep learning, namely—Convolutional Neural Network, Multilayer Perceptron, Classification and Regression Trees, Generalized Linear Model, Bayesian Additive Regression Trees, Gaussian Process, Random Forest and Conditional Inference Tree for predicting group size in five open source software projects developed and managed using an open source development framework GitHub. The social information foraging model has also been extended to predict group size in software issues, and its results compared to those obtained using machine learning and deep learning algorithms. The prediction results suggest that deep learning and machine learning models predict better than the extended social information foraging model, while the best-ranked model is a deep multilayer perceptron((R.M.S.E. sequelize—1.21, opencv—1.17, bitcoin—1.05, aseprite—1.01, electron—1.16). Also it was observed that issue labels helped improve the prediction performance of the machine learning and deep learning models. The prediction results of these models have been used to build an Issue Group Recommendation System as an Internet of Things application that recommends and alerts additional developers to help resolve an open issue.

Keywords Internet of things · Machine learning · Deep learning · Software repositories · Open source software development · Edge computing

1 Introduction

Open source software development [1, 2] is different from proprietary software development. While proprietary software does not provide source code to the users, open source software, on the other hand, makes the source code available to the world, allowing them to redistribute original or modified versions. Open source software (O.S.S.) not only makes free software available to the world but also changed the way software development worked until then. It allowed a collaborative and distributed software development environment [3], which allowed software developers to collaborate with developers from across the globe.

The team size in a software organization is mostly determined by project managers through careful planning using various effort estimation methods. Effort estimation can be done using various methods such as analogies with past projects [4, 5] and machine learning [6–8], among others. Managers use effort estimates to determine the number of people on a team and assign people with the required skills to the relevant teams. In an

✉ Deepti Chopra, dchopra27@gmail.com; Arvinder Kaur, arvinder70@gmail.com | [1]University School of Information and Communication Technology (U.S.I.C.T.), Guru Gobind Singh Indraprastha University (G.G.S.I.P.U.), New Delhi, India.

O.S.S. development environment, estimating group size for a task is not that simple. The developers of open source software work in a collaborative environment, and why they decide to contribute to a particular task is still not clear. A lot of research has been attempted to determine patterns in the participation of O.S.S. developers [9–11]. However, since the developers undertake the responsibility of a particular task or issue at their own will and interests, hence in O.S.S. development group size cannot be fixed by a single person. Bhowmik et al. [12] used the social information foraging model [13] to predict optimal group size in software change tasks. The optimal group size was determined for software change tasks by associating productivity with group size. In this paper, the group size prediction has been made for software issues reported in an open source development environment. Issues are any task, feature enhancement requests, and bugs that are reported for the software. Unlike change tasks, issues may or may not involve changes to the software. Change tasks are mostly carried out to add features, bug resolution, and maintenance activities after thorough analysis. In contrast, issues are reported and resolved by the open source development community and do not necessarily require any changes in the software. Some issues may be resolved just by providing required guidance to the initiator of the issue. It is essential to predict the group size for software issues so that they can be resolved quickly and efficiently. Prediction of group size may help get the required number of people to work on the issue and thus minimize the issue's resolution time. An estimation of the group size helps in planning for faster resolution of the software issue at hand. For instance, if the actual group size is less than the predicted group size and there are many pending tasks (i.e., tasks which have no developer assigned) for issue resolution, this estimation will suggest to the project members that more developers are required for resolution of the issue. Thus group size prediction is essential even in the O.S.S. development environment for better planning and resource utilization. We extend the social information foraging approach used to predict optimal group size in software change tasks given by Bhowmik et al. [12] and apply it for prediction of group size in software issues. We also apply eight algorithms employing machine learning and deep learning, namely, Convolutional Neural Network (CNN), Multilayer Perceptron (M.L.P.), Classification and Regression Trees (CaRT), Generalized Linear Model, Bayesian Additive Regression Trees, Gaussian Process, Random Forest, and Conditional Inference Tree to predict group size based on past issues in the software project. Employing machine learning and deep learning methods not only helps in faster and automated decision making but also have the capability to continuously improve the results as more historical data becomes available for learning.

We further compare the extended social information foraging model results to those obtained using machine learning and deep learning algorithms. Predicting group size is only beneficial if we can use it to recommend and alert the developers that may help resolve the issue efficiently. For this reason, we propose I.G.R.S., an IoT-based recommendation system that uses the prediction done by a machine learning or deep learning model to recommend/not recommend additional developers on the issue. This IoT based application can use platforms like cloud and edge computing to perform the analysis. An IoT-based I.G.R.S. will not only recommend additional developers for quick resolution of software issues but also alert them on their IoT devices. This way, an unresolved issue can be brought to the attention of developers who have resolved similar issues in the past, who may then choose to join the issue resolution group. Thus an IoT-based I.G.R.S. would help speed up issue resolution by alerting potential resolvers, rather than just waiting for a developer to notice the issue on its own and picking it up for resolution.

The background and related work for our research is described in Sect. 2. The research approach is presented in Sect. 3, and the analysis of results is done in Sect. 4. Section 5 discusses the threats to the validity of the proposed model. Section 6 proposes the Issue Group Recommendation System (I.G.R.S.), and finally, Sect. 7 concludes this study.

2 Background and related work

Group size for handling software issues is generally predetermined and fixed for proprietary software. In the case of O.S.S., which is developed in a collaborative community-based approach, this group size is not fixed and predetermined by a manager, i.e., anyone can contribute on topics of their interest [14]. While group size prediction is made using various effort estimation models [15–17] in the case of proprietary software, for O.S.S., such an approach cannot be used. In this study, we extend the social information foraging model of predicting optimal group size for software change tasks given by Bhowmik et al. [12] for prediction of group size in software issues. Further, we also apply eight different techniques employing machine learning/deep learning algorithms namely, Convolutional Neural Network (CNN), Multilayer Perceptron (M.L.P.), Classification and Regression Trees (CaRT), Generalized Linear Model, Bayesian Additive Regression Trees, Gaussian Process, Random Forest and Conditional Inference Tree for predicting group size in five open source software projects. In Sect. 2.1, an overview of social information foraging [13] model has been provided, while Sect. 2.2–2.9 describe Convolutional Neural Network (CNN), Multilayer

Perceptron (M.L.P.), Classification and Regression Trees (CaRT), Generalized Linear Model, Bayesian Additive Regression Trees, Gaussian Process, Random Forest and Conditional Inference Tree techniques.

2.1 Social information foraging

Information foraging theory was given by Pirolli [18], and it attempts to model the information-seeking pattern of users on the Web, analogous to optimal foraging theory in biology [19]. Optimal foraging in the context of information seeking aims at maximizing the information gain per unit of foraging. If each valuable information site is taken as a patch, a web user is either collecting information from a relevant patch or searching for a valuable patch. Let the time that is spent collecting information from a valuable patch be called inside-patch search time (denoted by t_{IS}), and the time that is spent searching for a valuable patch be called the outside-patch search time (denoted by t_{OS}). The information foraging environment can be illustrated, as shown in Fig. 1. The Information Gain (denoted by I) can thus be depicted as in (1), where G denotes the expected net gain.

$$I = \frac{G}{t_{IS} + t_{OS}} \tag{1}$$

Pirolli [13] augmented the information foraging theory to a social environment such as an O.S.S. development environment with multiple users to formulate the social information foraging theory. The major hypothesis in social information foraging is that hints are shared regarding the potential location of valuable data. Apart from the signs obvious in the environment, foragers also profit from the hints shared by the community. Information Gain for an individual in a group on n foragers can be depicted as follows: [12]. Let the time taken by an individual forager for processing a patch in a group consisting of n foragers be denoted by $\tau(n) = cn^z$, where $0 < z < 1$ is the rate parameter, and c depicts the time spent foraging for a patch in a solo environment. The information gain for a single group member is then given by G/n. Similarly, let $\lambda(n)$ represent the individual search rate. Thus the search rate for a group of n foragers becomes $n \cdot \lambda(n)$. Hence the expected time for n foragers required for finding a valuable information patch will be $t_V = 1/[n\ \lambda(n)]$. If $\lambda(H)$ depicts the rate of discovering valuable patches of information with H distinct hints. Then the outside-patch search time and inside-patch search time for n foragers is $t_{OS} = \lambda(H)/[n \cdot \lambda(n)]$ and $t_{IS} = \tau(n)/[n \cdot \lambda(n)]$ respectively. Thus the information gain for an individual member of a group of n foragers can be given by (2).

$$I(n,H) = \frac{G/n}{t_V + t_{OS} + t_{IS}} = \frac{G/n}{\frac{1}{n.\lambda(n)} + \frac{\lambda(H)}{n.\lambda(n)} + \frac{\tau(n)}{n.\lambda(n)}} = \frac{\lambda(n).G}{1 + \lambda(H) + \tau(n)} \tag{2}$$

In this paper, the social information foraging theory model has been extended for predicting group size for software issues from five different software projects on the GitHub repository. The extended social information foraging model required for predicting group size of software issues in O.S.S. development environments is described in Sect. 3.2.1.

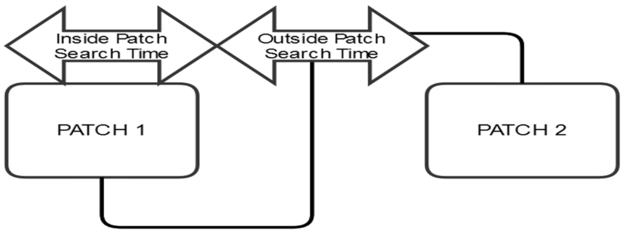

Fig. 1 Information foraging environment

2.2 Convolutional neural network (CNN)

A Convolutional Neural Network (CNN) [20] is a deep learning technique that consists of an input layer, an output layer, and multiple hidden layers. These hidden layers are generally composed of a sequence of convolutional layers. A convolutional layer simply applies a filter to an input, which results in activation. When there are multiple convolutional layers, repeatedly applying the same filter to an input returns a feature map. The feature map suggests the strength and location of a detected feature in an input. The novelty of CNN is its capability to automatically learn not one but multiple filters in parallel for a particular training dataset and prediction problem. CNN is a quite popular technique for image and video classification [21–25]. Apart from that, it has also been used for medical diagnosis [26], computer vision [27], and weather analysis [28], among other applications.

2.3 Multilayer perceptron (M.L.P.)

A multilayer perceptron (M.L.P.) [29, 30] is an artificial neural network used for deep learning. An M.L.P. consists of an input layer to obtain the input, an output layer that returns the prediction result about the input, and several hidden layers acting as computational engines of the M.L.P. In fact, M.L.P.s are so powerful that an M.L.P. with only one hidden layer can approximate all continuous functions. M.L.P. is used for supervised learning problems, where the neural network trains on a training dataset. The training process adjusts the parameters, including weights and biases, such that the resulting model minimizes the error probably using backpropagation. An M.L.P. generally works in two passes:

- *Forward pass*—In this, the input moves from the input layer through the hidden layers to finally the output layer, and the prediction made by the output layer is measured against the actual labels.
- *Backward pass*—This employs backpropagation. Backpropagation calculates the gradient of the loss function with respect to the weights of the network for each sample. The backpropagation algorithm calculates the gradient of the loss function with respect to each weight, calculating the gradient one layer at a time, iterating backward from the last layer. The weights were updated to minimize loss until changing the weights has no impact.

M.L.P. is used extensively for classification [31, 32] and pattern recognition [33, 34] in various fields such as medical science [31, 32], communication systems and networks [35, 36], and software maintainability [37, 38].

2.4 Classification and regression trees (CaRT)

CaRT is a machine-learning model that constructs a prediction tree using a dataset [39, 40]. The results of the model are determined by recursively dividing the dataset and then fitting a straightforward prediction model for each division of the dataset. These divisions can be represented as a decision tree [41]. Decision trees in machine learning have been used for both classification and regression. Classification trees are generally intended for predicting variables that can take a value belonging to a finite set of unordered values, and the error in prediction is measured as miss-classification cost. Regression trees are used for predicting variables that can take continuous or ordered values, with the error in prediction being commonly estimated by measures like mean absolute error (M.A.E.) and root mean square error (R.M.S.E.). There have been many applications of CaRT in areas such as finance [42], health care [43, 44], computer networks [45], remote sensing [46], and software engineering [6, 47, 48].

2.5 Generalized linear model

Generalized Linear Model (G.L.M.) [49] is a universal generalization of standard linear regression that considers predictors that have error distribution other than a normal distribution. The G.L.M. generalizes the linear model by permitting the linear model to be identified with the predictor variable through a link function and permitting every measure's variance to be a function of its predicted value.

The generalized linear model unifies various other statistical models such as linear regression, logistic regression, and Poisson regression. Unlike linear regression, which works only in case of normal distribution, G.L.M. works for all types of distribution. Hence there are many applications of G.L.M. such as for prediction [50, 51], pattern recognition [52], and trend analysis [53].

2.6 Bayesian additive regression trees

Bayesian additive regression trees (B.A.R.T.) [54] is a flexible machine learning algorithm. It is considered flexible since it is able to handle nonlinear predictors and multi-way interactions. It relies on an underlying Bayesian probability model. In fact, B.A.R.T. provides a Bayesian approach for nonparametric function estimation using regression trees. Regression trees carry out a recursive binary partitioning of predictor space for approximating the value of some unknown function, say *f*. The predictor space dimension is equal to the number of variables used for prediction, say *p*.

B.A.R.T. is a sum-of-trees model, whose estimation approach relies on a Bayesian probability model. The B.A.R.T. model can be expressed as given in (3).

$$Y = f(X) + E \tag{3}$$

where Y represents the n × 1 output vector of predicted values, X represents the n × p predictors matrix, and E represents the n × 1 noise vector. The value of f(X) is calculated using the sum of trees approach. There is a wide range of applications for the B.A.R.T. model, such as prediction of avalanches on mountain area roads [55], prediction of interaction of transcription factors with D.N.A. [56], and rain forecasting [57].

2.7 Gaussian process

The Gaussian process is nothing but a stochastic process such that the finite collection of random variables has a multivariate normal distribution. A Gaussian process machine-learning algorithm [58] employs lazy learning and a measure that determines the similarity between points (known as kernel function) for predicting the value of unseen data.

The prediction not only gives an estimate for that data point but also provides uncertainty information. For simple kernel functions, matrix algebra is utilized to calculate the predicted values using the kriging technique [59]. For a more sophisticated kernel, optimization approaches are utilized for fitting a Gaussian process model. There are various applications of Gaussian process machine learning, including slope stability evaluation [60], traffic flow prediction [61], and black-box modeling of bio-systems [62].

2.8 Random forest

Random forests [63] are a popular machine learning model used for classification, regression, and other tasks. Random forests construct a large number of decision trees using a training dataset and while predicting it took the mode of the classifications or mean value (in case of regression). In this way, Random forests try to correct the overfitting to the training dataset done by individual decision trees.

The basic principle of Random forest is that the decision made by a group of unrelated models is going to be better than the decision of a single tree alone. The advantage of having multiple decision trees or, as we call it, a forest of decision trees is that while some of the trees may predict wrong and have large errors, but as a group, we get a prediction in the correct direction and mostly better than that given by a single tree alone. As a result, there are many applications of random forest, such as fault prediction [64, 65], anomaly detection [66], and cancer diagnosis [67].

2.9 Conditional inference tree

Conditional Inference tree [68] is a nonparametric decision tree approach that employs unbiased recursive partitioning. It selects the predictor variables using permutation-based significance tests instead of selecting a predictor that maximizes information measures like information gain. It thus eliminates the biasness that other decision trees have towards the variable that maximizes the information measure. It uses multiple test procedures to decide when no significant correlation exists between any of the predictor variables and the predicted variable and then decides to stop the recursion and state the prediction. The conditional inference trees have been used in many applications like reliability analysis of automobile engines [69], crash severity analysis of asteroid corridors [70], among others.

3 Research methodology

One of the primary objectives of this study is to propose a model for predicting group size for software issues in an O.S.S. development environment, which in turn feeds the I.G.R.S. that recommends and alerts the developers that may be helpful for quick and efficient resolution of the software issue. The research methodology is depicted in Fig. 2. The first step involves project selection and data extraction, which is described in Sect. 3.1. In the second step, the extended social information foraging model and the different machine learning/ deep learning algorithms are applied to predict the group size of software issues. These prediction approaches are summarized in Sect. 3.2. Finally, the results of the prediction approaches are compared based on the evaluation measures described in Sect. 3.3. The predictions of machine learning/ deep

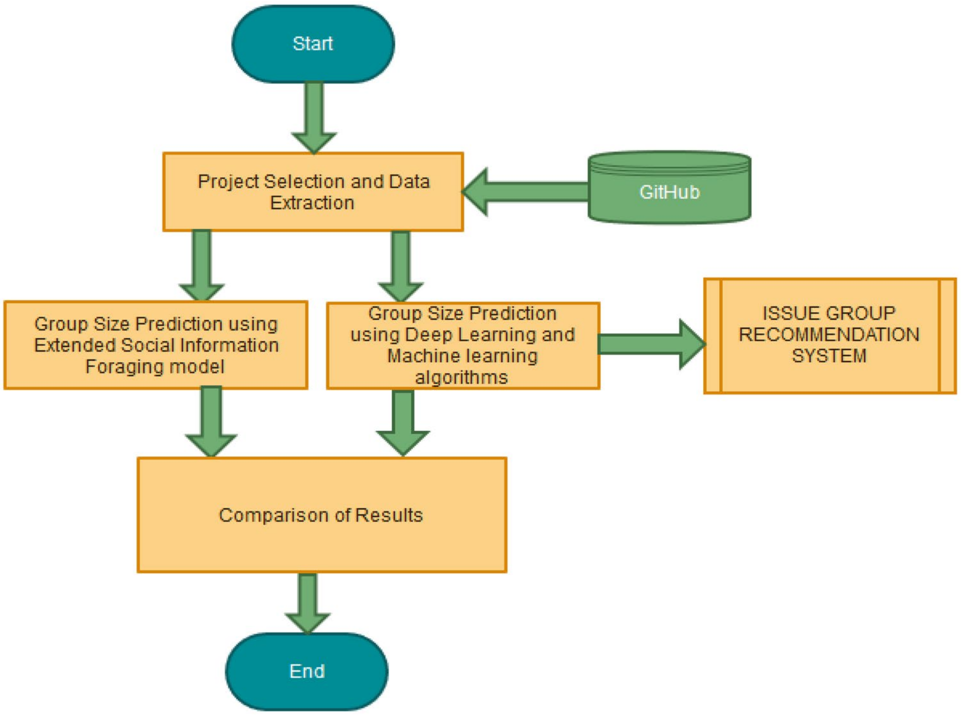
Start
Project Selection and Data Extraction
GitHub
Group Size Prediction using Extended Social Information Foraging model
Group Size Prediction using Deep Learning and Machine learning algorithms
ISSUE GROUP RECOMMENDATION SYSTEM
Comparison of Results
End

Fig. 2 Research methodology

learning algorithms are also fed to the IoT-based I.G.R.S., which is proposed in Sect. 6.

3.1 Project selection and data extraction

The machine learning and deep learning models and extended social information foraging model are applied to software issues data of five different software projects: sequelize, opencv, bitcoin, aseprite, and electron. All these five software projects are developed and managed in an open source environment. More specifically, it uses GitHub, which is a Web based community of open source developers and helps developers collaborate around the globe. Table 1 provides a brief description of the five software projects that we selected for our analysis.

Issues can be classified as either open issue or closed issue. A closed issue is an issue that has been resolved, while an open issue is an issue that has not yet been resolved and is currently under discussion. While collecting the data for our prediction models, only the closed issues were considered since the group size may be unstable for open issues. The data collected for each issue includes the following fields:

- Issue number—is used for uniquely identifying an issue.
- Open date—represents the date on which the issue was raised.
- Close date—represents the date on which the issue was marked as resolved.
- Group size—represents the total number of participants that contribute towards issue resolution.
- Number of Comments (N.O.C.)—is the total comments made by participants while discussing the issue.
- Issue Label—used for describing the issue type, category, location, etc.
- Duration—is the number of days between issue close Date and issue open Date.

The data extraction process was performed using R programming with the help of the rvest package to scrape the relevant data from GitHub. It included two significant steps. First, the URLs of closed issues were extracted and stored in a CSV file. Second, for each issue using the URL from the CSV file, the above data fields were extracted using appropriate CSS selectors and regular expressions.

3.2 Prediction models

The extended social information foraging and the parameters for machine learning and deep learning models are described in the subsections below, and their results are analyzed in Sect. 4.

3.2.1 Extended social information foraging model

The predictions of various machine learning and deep learning models are compared with the group size prediction done by modifying the model given by Bhowmik et al. [12] for optimal group size prediction of software change tasks. Optimal group size prediction for software issues is made by setting up the parameters in (2) as:

(a) Every issue is viewed as a patch wherein social or solo information foraging can happen. An issue is taken to be a solo patch if just a single individual handles the issue; else, it is viewed as a social patch.
(b) Similar to Pirolli [13] let n (group size) = H, and the in-patch information gain G be equivalent to the quantity of hints (denoted by H). Subsequently $n = G = H$ [12].
(c) The group rate of discovering significant information $\lambda(H) = \lambda(n)$ = duration of the issue [12], which is calculated using issue open and close time.
(d) The time taken by an individual forager to process a patch in a group comprising of n foragers, i.e., $\tau(n) = cn^z$, is determined by setting aside c (solo foraging effort) to be equivalent to the average dura-

Table 1 Selected software projects

Project	Sequelize	Opencv	Bitcoin	Aseprite	Electron
Issues URL	https://github.com/sequelize/sequelize/issues	https://github.com/opencv/opencv/issues	https://github.com/bitcoin/bitcoin/issues	https://github.com/aseprite/aseprite/issues	https://github.com/electron/electron/issues
Programming language	JavaSript	C++	C++	C++	C++
Number of contributors	683	1114	688	42	932
Analysis begin date	20-Aug-10	27-Jul-15	22-Jan-13	20-Aug-14	05-Jun-13
Analysis end date	11-May-18	16-Mar-20	16-Mar-20	16-Mar-20	16-Mar-20
Number of issues	6317	4625	4512	1320	11,420

tion of solo patches in the considered time window. Note that a time window of three months is taken to predict the ideal group size for issues in the window. The window depends on the close time of the issue and not on the open time. The rate parameter *z* is aligned to get the best lognormal curve for information gain (I(n, H), as depicted in (2)) [13]. For our study, *z* is equivalent to 0.3.

(e) I(n, H) is then used to decide the ideal group size for the issues.

3.2.2 Parameters of machine learning and deep learning models

Machine learning and deep learning models analyzed in this study have been described in Sect. 2 already. In this section, the parameters of the machine learning and deep learning models are described. Firstly, group size is taken to be the predicted variable (also known as the dependent variable). N.O.C., issue label, and duration are set up as predictor variables (also known as independent variables). The models are also built excluding issue label as one of the predictor variables since the extended social information foraging model does not take issue label into account. The machine learning models are built using the caret library in R Studio, whereas the deep learning models are implemented with the help of the keras library. The configuration parameters of the machine learning and deep learning models are specified in Table 2.

3.3 Evaluation measures

An assessment of the prediction models is fundamental for figuring out which model ought to be favored over others in real-time prediction. The data gathered is partitioned into training data (about 80%) and testing data (about 20%). The predicted and actual values of group size for the test data are utilized to assess the models. A well-known error metric, i.e., Root Mean Square Error (R.M.S.E.) [71], is utilized for analyzing the prediction performance of the models. R.M.S.E. is determined utilizing the formula given in (4), where p_i is the predicted value of group size, o_i is the actual value of group size for the ith issue, and *t* is the total number of predictions done.

$$R.M.S.E. = \sqrt{\frac{\sum_{i=1}^{n}(p_i - o_i)^2}{t}} \tag{4}$$

R.M.S.E. is chosen as error measure over Mean Absolute Error (M.A.E.) for assessment of models as it gives more weight to large errors. While predicting group size, we do not wish the model to make an incredibly colossal error. Thus a model with lower R.M.S.E. is chosen.

4 Analysis of results

In this section, the results of the prediction models are compared. The eight machine learning/ deep learning models, i.e., Convolutional Neural Network (CNN), Multilayer Perceptron (M.L.P.), Classification and Regression Trees (CaRT), Generalized Linear Model, Bayesian Additive Regression Trees, Gaussian Process, Random Forest and Conditional Inference Tree, are trained on software issues data from five software projects, i.e., sequelize, opencv, bitcoin, aseprite and electron. The models are trained in two ways, once excluding issue label as one of the predictors and once including it as one of the predictors. Since the extended social information foraging model does not take issue label into account for predicting optimal group

Table 2 Configuration parameters for prediction models

Prediction Model	Configuration parameters
CNN	No. of hidden layers = 3 Hidden layer 1—nodes = 10, activation function = "relu" Hidden layer 2—nodes = 5, activation function = "relu" Hidden layer 3—nodes = 1
Multilayer perceptron	No. of hidden layers = 3 Hidden layer 1—nodes = 5, activation function = "relu" Hidden layer 2—nodes = 2, activation function = "relu" Hidden layer 3—nodes = 1, activation function = "linear"
CART	Method = rpart, cp = 0.01
GLM	Method = glm
BART	Method = bartMachine, num_trees = 50, k = 2, alpha = 0.95, beta = 2, nu = 3
Gaussian process	Method = gaussprLinear
Random forest	Method = rf, mtry = 2
Conditional inference tree	Method = ctree, mincriterion = 0.95

size, its prediction results are compared with models built excluding issue label as a predictor.

4.1 Results excluding issue label as a predictor

Firstly, let us consider the performance of machine learning and deep learning models, excluding issue label as one of the predictors, and compare the results with those of the extended social information foraging model. The R.M.S.E. values are depicted in Table 3. Figure 3, 4, 5, 6, 7 display these results graphically.

Figure 3 depicts the results for sequelize project. It can be clearly seen that machine learning and deep learning models perform better than the Extended social information foraging model (R.M.S.E. = 3.13). Also amongst the machine learning and deep learning models the minimum prediction error was obtained for CNN (RMSE = 1.18), followed by M.L.P. (RMSE = 1.21), Random Forest (RMSE = 1.38), B.A.R.T. (RMSE = 1.65), Gaussian process (RMSE = 1.65), CART (RMSE = 1.69), G.L.M. (RMSE = 1.71) and Conditional Inference Tree (RMSE = 1.89).

Figure 4 depicts the performance of the models on opencv project. It is noticed that all the machine learning and deep learning model perform better than the extended social information foraging model (RMSE = 2.94). Also MLP (RMSE = 1.17) model gives the best results, followed by CNN (RMSE = 1.19), Random Forest (RMSE = 1.23), BART (RMSE = 1.37), CART (RMSE = 1.54), Gaussian Process (RMSE = 1.62), GLM (RMSE = 1.65) and Conditional Inference Tree (RMSE = 1.73).

Figure 5 displays the results for bitcoin project. It can be clearly seen that all the machine learning and deep learning models have a lower prediction error than the social information foraging model (RMSE = 2.67). The minimum error is obtained using CNN (RMSE = 1.02) model, followed by MLP (RMSE = 1.05), BART (RMSE = 1.12), Random Forest (RMSE = 1.21),

Table 3 Model results excluding issue label as a predictor

Project Model	Sequelize RMSE	Opencv	Bitcoin	Aseprite	Electron
CNN	1.18	1.19	1.02	1.02	1.22
Multilayer perceptron	1.21	1.17	1.05	1.01	1.16
CART	1.69	1.54	1.56	1.49	1.78
GLM	1.71	1.65	1.72	1.63	1.89
BART	1.65	1.37	1.12	1.28	1.59
Gaussian process	1.65	1.62	1.58	1.34	1.67
Random forest	1.38	1.23	1.21	1.08	1.35
Conditional inference tree	1.89	1.73	1.67	1.59	1.45
Extended social information foraging	3.13	2.94	2.67	2.12	3.65

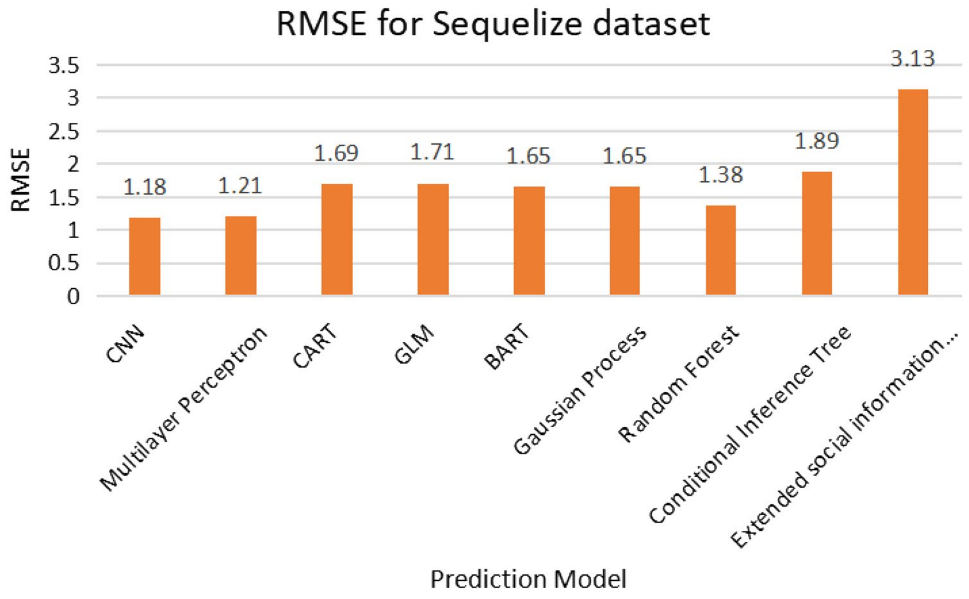

Fig. 3 Results for sequelize project (excluding issue label)

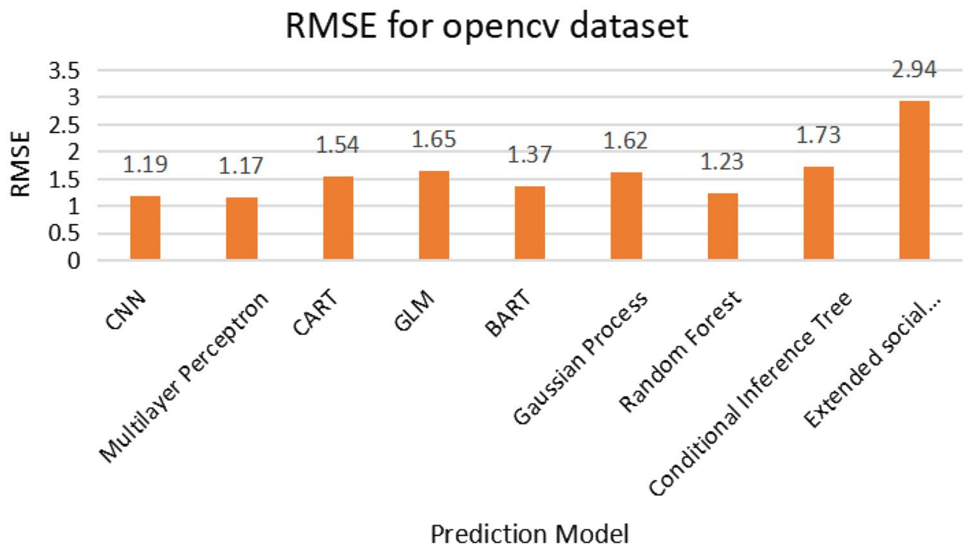

Fig. 4 Results for opencv project (excluding issue label)

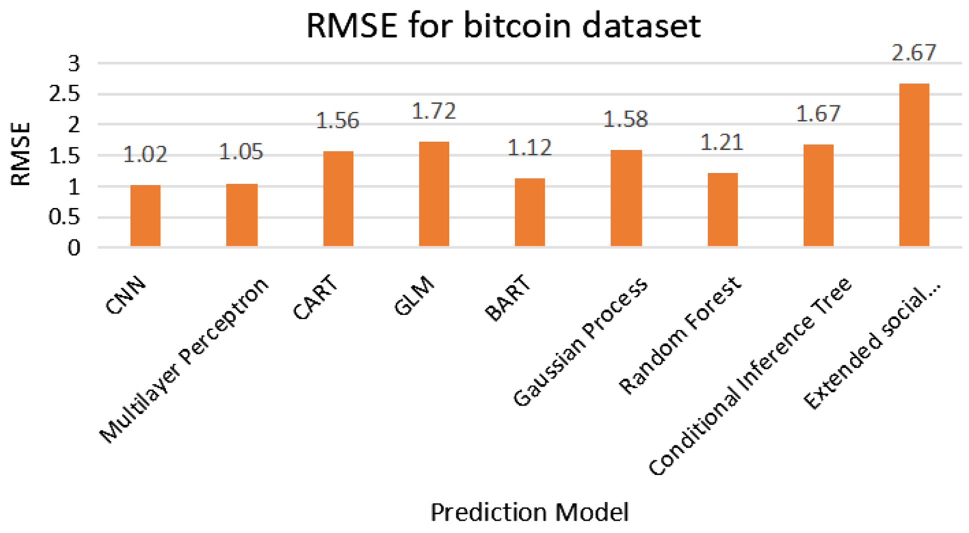

Fig. 5 Results for bitcoin project (excluding issue label)

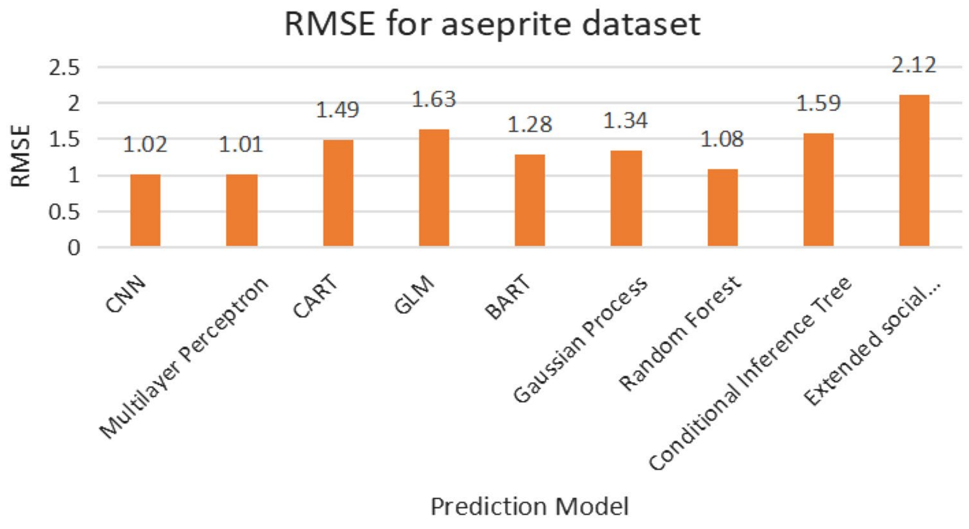

Fig. 6 Results for aseprite project (excluding issue label)

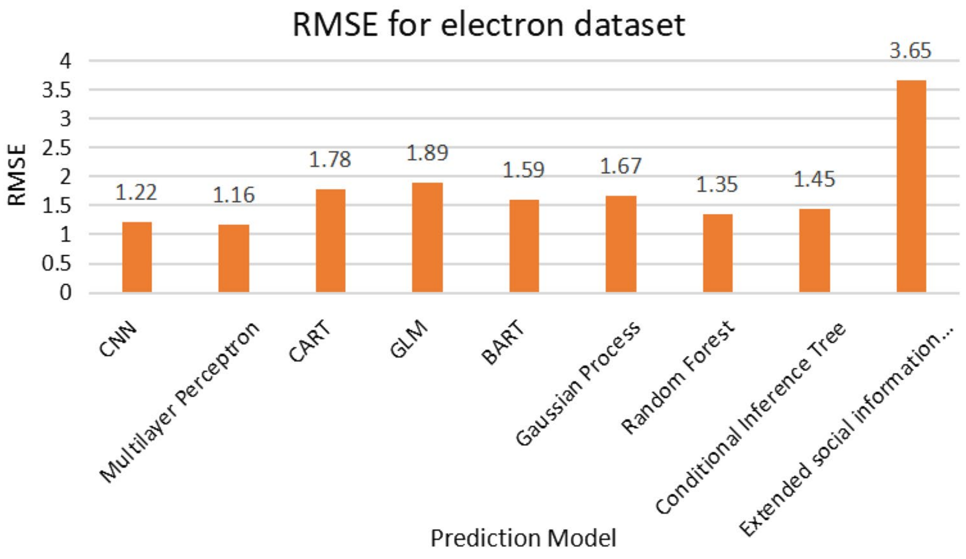

Fig. 7 Results for electron project (excluding issue label)

CART (RMSE = 1.56), Gaussian Process (RMSE = 1.58), Conditional Inference Tree (RMSE = 1.67) and GLM (RMSE = 1.72).

Figure 6 depicts the performance of models for the issues of aseprite project. It is seen that extended social information foraging model (RMSE = 2.12) gives the maximum prediction error. MLP (RMSE = 1.01) model gives the minimum error, followed by CNN (RMSE = 1.01), Random Forest (RMSE = 1.08), BART (RMSE = 1.28), Gaussian Process (RMSE = 1.34), CART (RMSE = 1.49), Conditional Inference Tree (RMSE = 1.59) and GLM (RMSE = 1.63).

Figure 7 depicts the results for electron project. It is clearly seen that all the machine learning and deep learning models outperform the extended social information foraging model (RMSE = 3.65). The lowest prediction error is noticed for MLP (RMSE = 1.16) model, followed by CNN (RMSE = 1.22), Random Forest (RMSE = 1.35), Conditional Inference Tree (RMSE = 1.45), BART (RMSE = 1.59), Gaussian Process (RMSE = 1.67), CART (RMSE = 1.78) and GLM (RMSE = 1.89).

The above results clearly show that the Extended social information foraging model gives the maximum error for all five software projects. However, there is no single model that gives the best result in all cases. Therefore in order to compare the performance of the algorithms, the Friedman Test is applied to the results. There was a statistically significant difference in the prediction error depending on the algorithm used for prediction, $\chi^2(8) = 37.035$, $p = 0.000$. According to the Friedman test, we get the average ranks for all models, as given in Table 4. It was observed that Multilayer Perceptron (M.L.P.) gets Rank 1 and can thus be considered the best performing model amongst all nine models. It was further noticed that both deep learning algorithms have a better rank than machine learning models.

Table 4 Friedman test average ranks

Model	Average rank (obtained using Friedman Test)	Assigned rank
CNN	1.60	2
Multilayer perceptron	1.40	1
CART	5.80	6
GLM	7.60	8
BART	4.10	4
Gaussian process	5.50	5
Random forest	3.20	3
Conditional inference tree	6.80	7
Extended social information foraging	9.00	9

Post hoc analysis using Wilcoxon signed-rank tests was also conducted with a Bonferroni correction applied, resulting in a significance level set at $p < 0.0014$. There were no significant differences between any of the two algorithms taken at a time, and for all cases, it was noticed that $p > 0.0014$.

4.2 Results including issue label as a predictor

Secondly, in this section, we analyze the results of the eight machine learning/ deep learning models, i.e., Convolutional Neural Network (CNN), Multilayer Perceptron (M.L.P.), Classification and Regression Trees (CaRT), Generalized Linear Model, Bayesian Additive Regression Trees, Gaussian Process, Random Forest and Conditional Inference Tree including issue label as one of the predictor variables. Since the extended social information foraging model does not consider issue label for prediction, its result will be the same as those depicted in Sect. 4.1.

Table 5 displays the R.M.S.E. values obtained by machine learning and deep learning models when the issue label is included as one of the predictors.

Figure 8 depicts the RMSE obtained by each of the eight machine learning and deep learning models for sequelize project. The minimum error is obtained using CNN (RMSE = 1.11), followed by MLP (RMSE = 1.15), Random Forest (RMSE = 1.21), BART (RMSE = 1.59), CART (RMSE = 1.61), Gaussian Process (RMSE = 1.65), GLM (RMSE = 1.67) and Conditional Inference Tree (RMSE = 1.77).

Figure 9 displays the results for opencv project for machine learning and deep learning models, including issue label as one of the predictors. It is observed that minimum prediction error is reported by MLP (RMSE = 1.06), followed by CNN (RMSE = 1.09), Random Forest (RMSE = 1.15), BART (RMSE = 1.31), CART (RMSE = 1.38), GLM (RMSE = 1.56), Gaussian Process (RMSE = 1.62) and Conditional Inference Tree (RMSE = 1.68).

Figure 10 depicts the results of machine learning and deep learning models, including the issue label as one of the predictors for bitcoin software project. The model that gives minimum prediction error is CNN (RMSE = 1.01), followed by MLP (RMSE = 1.03), BART (RMSE = 1.12), Random Forest (RMSE = 1.16), CART (RMSE = 1.48), Gaussian Process (RMSE = 1.48), Conditional Inference Tree (RMSE = 1.54) and GLM (RMSE = 1.68).

The results of machine learning and deep learning models, including issue label as one of the predictors for aseprite project, are depicted in Fig. 11. The best results are obtained using MLP (RMSE = 0.94) model, followed by CNN (RMSE = 0.98), Random Forest (RMSE = 1.01), BART (RMSE = 1.17), Gaussian Process (RMSE = 1.26), CART (RMSE = 1.34), Conditional Inference Tree (RMSE = 1.52) and GLM (RMSE = 1.54).

The R.M.S.E. values for electron project for the eight machine learning and deep learning models using issue label as one of the predictors are depicted in Fig. 12. MLP (RMSE = 1.12) model gives the minimum prediction error, followed by CNN (RMSE = 1.18), Random Forest (RMSE = 1.35), Conditional Inference Tree (RMSE = 1.43), BART (RMSE = 1.48), Gaussian Process (RMSE = 1.62), CART (RMSE = 1.67) and GLM (RMSE = 1.82).

In order to compare the prediction performance of the models, the Friedman test is applied. There was

Table 5 Model results including issue label as a predictor

Project	Sequelize	Opencv	Bitcoin	Aseprite	Electron
Model	RMSE				
CNN	1.11	1.09	1.01	0.98	1.18
Multilayer perceptron	1.15	1.06	1.03	0.94	1.12
CART	1.61	1.38	1.48	1.34	1.67
GLM	1.67	1.56	1.68	1.54	1.82
BART	1.59	1.31	1.12	1.17	1.48
Gaussian process	1.65	1.62	1.48	1.26	1.62
Random forest	1.21	1.15	1.16	1.01	1.35
Conditional inference tree	1.77	1.68	1.54	1.52	1.43

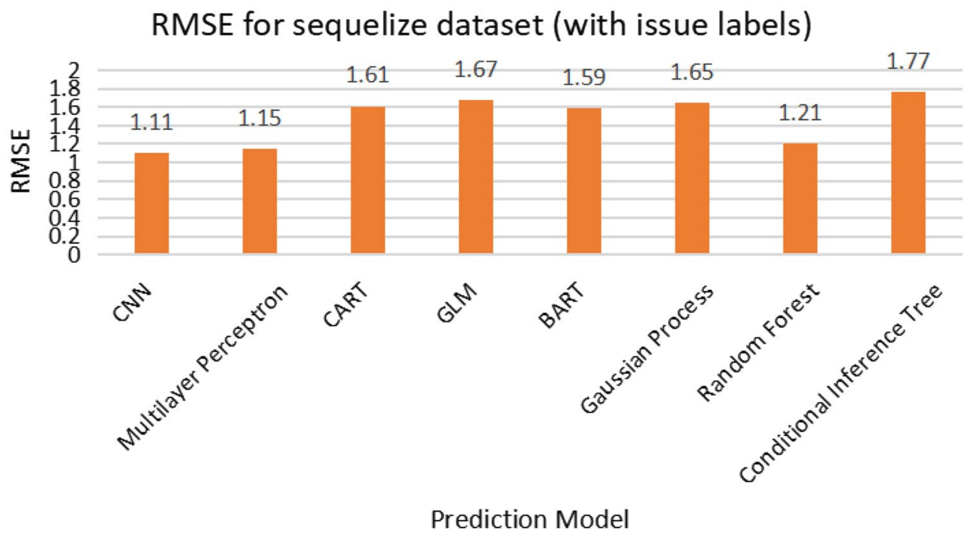

Fig. 8 Results for sequelize project (including issue label)

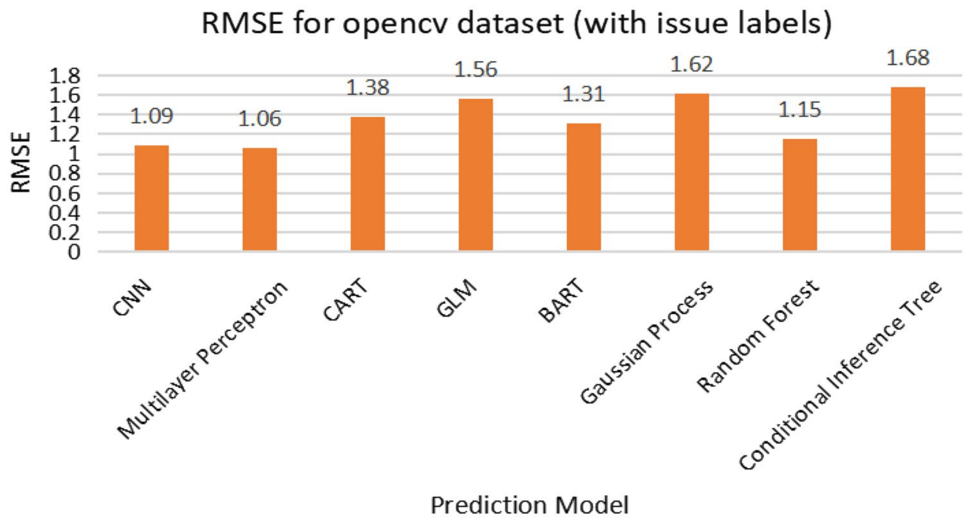

Fig. 9 Results for opencv project (including issue label)

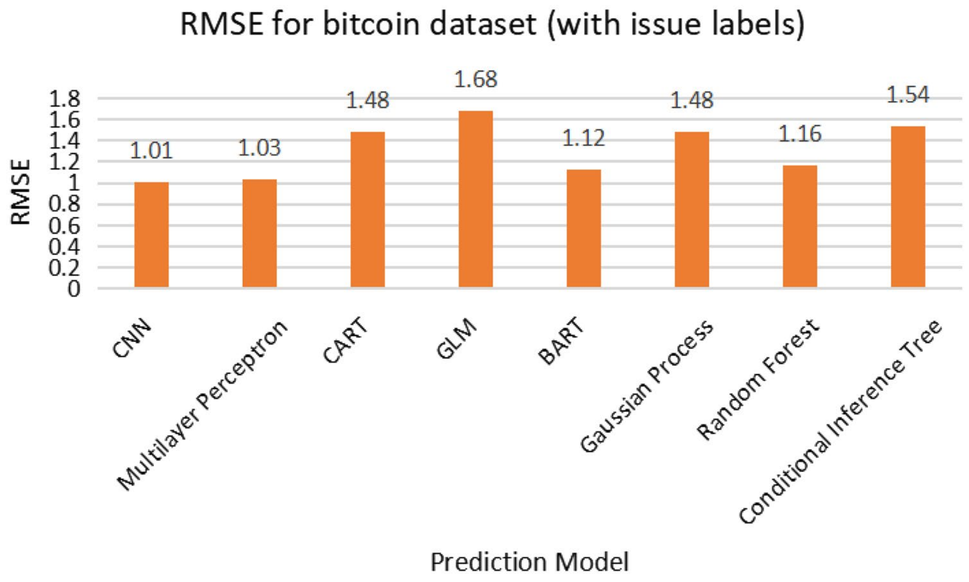

Fig. 10 Results for bitcoin project (including issue label)

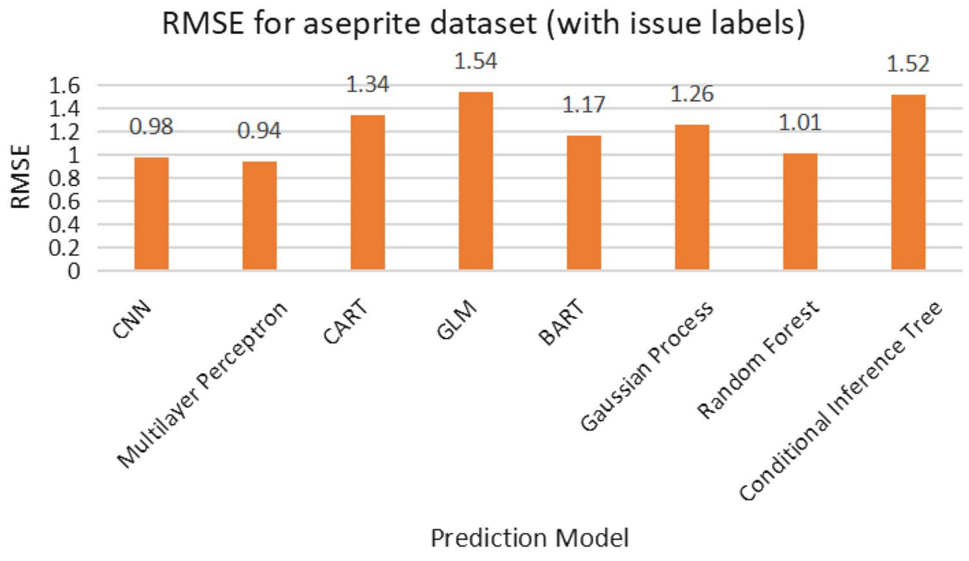

Fig. 11 Results for aseprite project (including issue label)

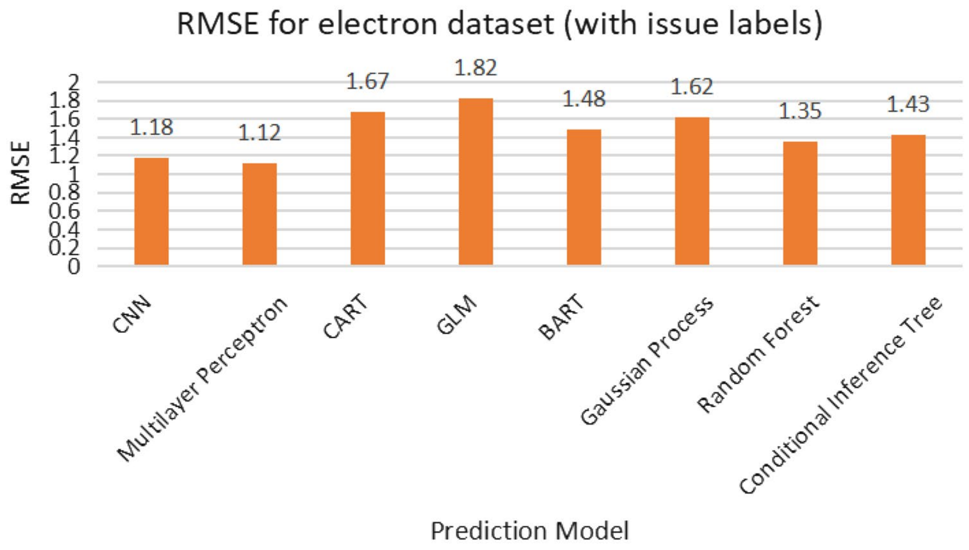

Fig. 12 Results for electron project (including issue label)

Table 6 Friedman test average ranks (including issue label)

Model	Average rank (obtained using Friedman Test)	Assigned rank
CNN	1.60	2
Multilayer perceptron	1.40	1
CART	5.70	5
GLM	7.40	8
BART	4.00	4
Gaussian process	5.90	6
Random forest	3.20	3
Conditional inference tree	6.80	7

a statistically significant difference in the prediction error depending on the algorithm used for prediction, $\chi^2(7) = 30.957$, $p = 0.000$. According to the Friedman test, we get the average ranks for all models, as given in Table 6. It is observed that Multilayer Perceptron (M.L.P.) is ranked first among all models, while G.L.M. is ranked last. Also, both the deep learning models, i.e., M.L.P. and CNN, are better ranked than machine learning models.

Post hoc analysis using Wilcoxon signed-rank tests was also conducted with a Bonferroni correction applied, resulting in a significance level set at $p < 0.0018$. There were no significant differences between any of the two algorithms taken at a time, as for all cases, it was observed that $p > 0.0018$.

Finally, we compare the results of the machine learning and deep learning models obtained using the issue label as one of the predictors to those obtained without using the issue label as a predictor. The comparison is depicted in Fig. 13. It was observed that all the machine learning and deep learning models, including issue label as one of the predictors, performed better than or equivalent to models that did not use the issue label as one of the predictors. Issue labels help improve the prediction performance of the models as they provide vital information about the type of issue. It is generally based on the labels and issue description that O.S.S. developers decide whether or not they can contribute towards issue resolution.

A quick summarization of the observed results highlights the following:

- All the machine learning and deep learning prediction models performed better than the extended social information foraging model.
- M.L.P. (R.M.S.E. sequelize—1.21, opencv—1.17, bitcoin—1.05, aseprite—1.01, electron—1.16) was the best performing model amongst all nine models for prediction of group size when issue label was excluded as one of the predictors.
- M.L.P. (R.M.S.E. sequelize—1.15, opencv—1.06, bitcoin—1.03, aseprite—0.94, electron—1.12) was the best performing model among all eight models for prediction of group size when issue label was included as one of the predictors.
- The deep learning models, i.e., M.L.P. and CNN, ranked better than machine learning models in the Friedman Test.
- The best ranked deep learning model provided an improvement of 61.34%, 60.2%, 61.8%, 52.36% and 68.22% over the extended social information foraging model, for sequelize, opencv, bitcoin, aseprite and electron datasets respectively.
- The machine learning and deep learning models, including issue label as one of the predictors, performed better than or equivalent to models that did not use issue label as one of the predictors.

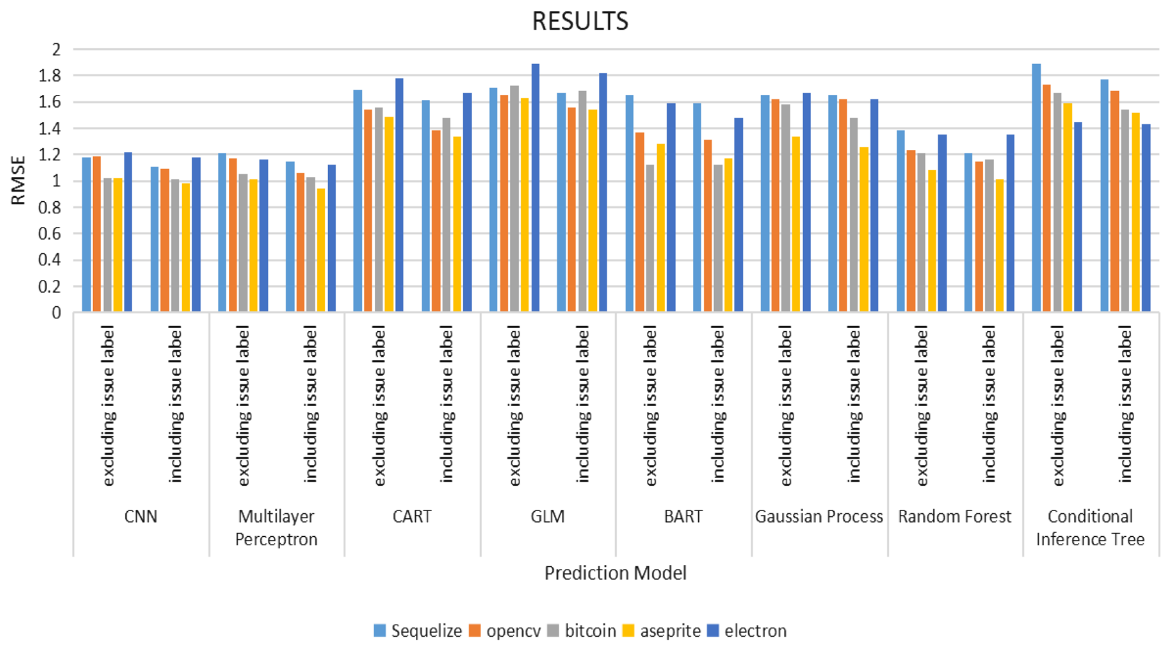

Fig. 13 Comparison of model results including and excluding issue label

- The best ranked model when included issue label as one of the predictors provided an improvement of 8.26%, 9.4%, 0.98%, 6.93% and 3.45% for sequelize, opencv, bitcoin, aseprite and electron datasets respectively, over the prediction of best ranked model excluding issue label as one of the predictors.

5 Threats to validity

Threats to the validity of a study are generally classified into internal and external. Threats to internal validity arise when the true facts and foundations on which the experimental results are based are misinterpreted. Threats to external validity are threats that revolve around the validation of results in different settings or the generalizability of the results.

5.1 Threats to internal validity

One of the significant threats to the internal validity of any study is the misinterpretation of underlying data and facts. This threat is eliminated by extracting data directly from GitHub, where all the O.S.S. development data is maintained and updated in real-time. So the data collected for building the prediction models is collected from a reliable source.

Another limitation that may impact the study is that in the social information foraging model, we take into account a time window of three months. The reason for selecting three months time window is that it has been shown to provide meaningful and reliable results in previous studies [12, 72]. Also, the issues are classified in a three month window based on the close date alone. The reason for not using both the close date and open date to classify an issue in a three month time window is that certain issues might open in one time window and close in another time window. Thereby making them not wholly fall within one time window and leaving them out of consideration. To avoid this, issues are classified in a time window based on the close date alone.

5.2 Threats to external validity

The external validity of a study relates to the generalizability of the results of the study. Since the analysis is performed on data from GitHub, which has one of the biggest open source community of developers working on software projects, the results of our study should hold for other O.S.S. systems. Also, the five projects selected for the study are established projects and have been under active development; hence the community participation on issues is also similar to other active open source projects.

6 Issue group recommendation system (I.G.R.S.)

We also propose an IoT-based I.G.R.S. that will input from one of the machine learning/ deep learning models to predict the group size of a software issue and, based on the prediction, will either recommend addition of other developers working on the project to the software issue or recommend no addition of software developer. Figure 14 depicts the flow diagram for the proposed I.G.R.S. This system uses data from the closed issues of a project to build the prediction model for predicting the group size of an open issue.

The predicted group size is then compared with the current group size of the open issue. If the predicted group size is greater than the current group size, then the I.G.R.S. recommends additional developers for the issue. I.G.R.S. will use the past participation data of the developers working on the project. It associates the issue labels of closed issues that a developer has previously worked on with the developer. Then it matches the issue label of the open issue with the issue labels associated with the developers. If the issue label of the open issue matches any of the issue labels associated with a developer, then that developer is alerted about the issue, and his expertise is solicited on the issue at hand. In case an open issue has a new label not previously used for other issues, then the developers associated with issues having no label (represented as Blank) are alerted. If the predicted group size is less than or equal to the current group size, the system recommends no additional developers for the issue, and no alert needs to be sent to them.

It should be noted that not all developers who are alerted may participate in the issue; therefore, it is necessary to alert all developers with matching issue labels and not just the number of developers required to make the group size equal to the predicted group size. Also, no additional developers are recommended for the issue when the group size already exceeds the predicted group size; this does not mean that additional developers cannot join the issue. It only means that I.G.R.S. will not send alerts to additional developers, but developers may join the issue depending on their interests and other factors. The purpose of the system is to reach out to the O.S.S. developers with relevant experience and expertise who may help to resolve the issue efficiently and timely; for this reason, the system alerts only the developers having matching issue labels and not all the developers who have been working on the project.

Internet of Things (IoT) has been changing how data sharing and communication used to take place. IoT enables things, i.e., objects of interest, to process, communicate, and transmit information on their own [73, 74]. There are many IoT applications in areas such as transport systems [75], traffic management [76], healthcare [77], and military [78], among others. This system can be built in such a way that with just internet connectivity on the server and an IoT device, the recommended developers

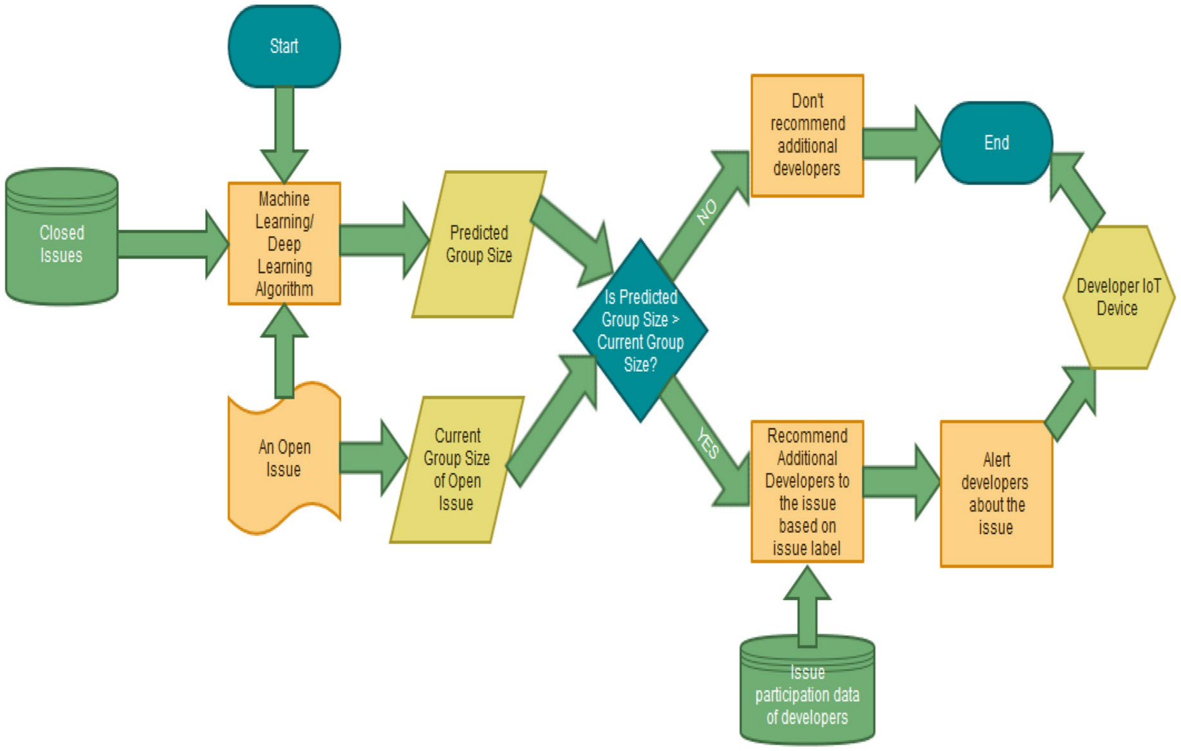
Start
Closed Issues
Machine Learning/ Deep Learning Algorithm
Predicted Group Size
An Open Issue
Current Group Size of Open Issue
Is Predicted Group Size > Current Group Size?
NO
YES
Don't recommend additional developers
End
Developer IoT Device
Recommend Additional Developers to the issue based on issue label
Alert developers about the issue
Issue participation data of developers

Fig. 14 Issue group recommendation system

can be alerted on the fly. Thus, making it a useful IoT application [79, 80], which at periodic intervals (which can be set by an administrator) can be used to alert the developers regarding the software issue automatically they may help resolve. This way, the developers need not browse manually through issues; they will be alerted if an open issue exists similar to an issue they resolved in the past. Each developer is assigned a unique identifier (UID) [81] and can be recommended on open issues using the recommendation system. I.G.R.S. can be implemented with all the analyses being done using a cloud or edge computing infrastructure [82–84].

7 Conclusion and Future Work

After careful analysis of the results, the use of deep learning and machine learning models, most preferably the best-ranked model, i.e., Multilayer Perceptron, is prescribed for group size prediction of software issues in O.S.S. development. The R.M.S.E. values using M.L.P. when issue label was excluded as a predictor were 1.21, 1.17, 1.05, 1.01, 1.16 for sequelize, opencv, bitcoin, aseprite, and electron datasets, respectively. It helps us determine a group size in a development environment that allows developers across the world to collaborate. In comparison to the social information foraging model, all the eight deep learning and machine learning models provided better prediction results. The best ranked deep learning model provided an improvement of 61.34%, 60.2%, 61.8%, 52.36% and 68.22% over the extended social information foraging model, for sequelize, opencv, bitcoin, aseprite and electron datasets respectively. It is additionally seen that issue labels gives significant information about the issue and improves the prediction performance. The best results were obtained using M.L.P. (R.M.S.E. sequelize—1.15, opencv—1.06, bitcoin—1.03, aseprite—0.94, electron—1.12) when the issue label was included as one of the predictors. In fact using issue label as one of the predictors provided an improvement of 8.26%, 9.4%, 0.98%, 6.93% and 3.45% for sequelize, opencv, bitcoin, aseprite and electron datasets respectively, compared to when issue label was excluded as one of the predictors. Hence, it is recommended to use issue label for prediction of group size using machine learning and deep learning models. This paper also proposed an IoT-based I.G.R.S. to recommend and alert additional developers that may help resolve the software issue timely and efficiently. I.G.R.S. uses the prediction results of a machine learning/deep learning model (suggested M.L.P.) to determine whether or not additional developers should be recommended for the issue. It also recommends preferable developers for resolving the issue and alerts them. I.G.R.S. is proposed to be used as an IoT application that alerts developers on the fly about software issues they may help resolve and can be implemented using cloud and edge computing framework. Our future work incorporates studying and predicting the behavior of developers in an O.S.S. development environment, implementing and testing the efficiency of I.G.R.S. in an open source development environment.

Acknowledgements I would like to acknowledge the "Visvesvaraya Ph.D. Scheme for Electronics & I.T." instituted by the Ministry of Electronics and Information Technology, Government of India for promoting research in Indian universities.

Author contributions All authors contributed to the study conception and design. Data collection and analysis were performed by [DC]. The first draft of the manuscript was written by [DC], which was reviewed and edited by [AK]. All authors read and approved the final manuscript.

Data availability The data is extracted from the GitHub repository, which is an open source repository. The links are provided in the Project Selection and Data Extraction Section.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Von Krogh G (2003) Open-source software development. M.I.T Sloan Manag Rev 44(3):14
2. Feller J, Fitzgerald B (2002) Understanding open source software development. Addison-Wesley, London, pp 143–159
3. Herbsleb JD, Mockus A (2003) An empirical study of speed and communication in globally distributed software development. IEEE Trans Software Eng 29(6):481–494
4. Shepperd M, Schofield C (1997) Estimating software project effort using analogies. IEEE Trans Software Eng 23(11):736–743
5. Benala TR, Mall R (2018) D.A.B.E.: dDifferential evolution in analogy-based software development effort estimation. Swarm Evolut Comput 38:158–172
6. Srinivasan K, Fisher D (1995) Machine learning approaches to estimating software development effort. IEEE Trans Softw Eng 21(2):126–137

7. García-Floriano A, López-Martín C, Yáñez-Márquez C, Abran A (2018) Support vector regression for predicting software enhancement effort. Inf Softw Technol 97:99–109
8. Pospieszny P, Czarnacka-Chrobot B, Kobylinski A (2018) An effective approach for software project effort and duration estimation with machine learning algorithms. J Syst Softw 137:184–196
9. Chełkowski T, Gloor P, Jemielniak D (2016) Inequalities in open source software development: Analysis of contributor's commits in Apache Software Foundation projects. PLoS ONE 11(4):e0152976
10. Iskoujina Z, Roberts J (2015) Knowledge sharing in open source software communities: motivations and management. J Knowl Manag 19(4):791–813
11. Ahmed, F., Campbell, P., Jaffar, A., & Capretz, L. F. (2015). Myths and realities about online forums in open source software development: an empirical study. *arXiv preprint* arXiv:1507.06927.
12. Bhowmik T, Niu N, Wang W, Cheng JRC, Li L, Cao X (2016) Optimal group size for software change tasks: a social information foraging perspective. IEEE Trans Cybern 46(8):1784–1795
13. Pirolli, P. (2009, April). An elementary social information foraging model. In *Proceedings of the S.I.G.C.H.I. Conference on Human Factors in Computing Systems* (pp. 605–614). A.C.M.
14. Kogut B, Metiu A (2001) Open-source software development and distributed innovation. Oxford Rev Econ Policy 17(2):248–264
15. Amasaki S, Lokan C (2015) On the effectiveness of weighted moving windows: experiment on linear regression based software effort estimation. J Softw: Evol Process 27(7):488–507
16. Ahn Y, Suh J, Kim S, Kim H (2003) The software maintenance project effort estimation model based on function points. J Softw: Evol Process 15(2):71–85
17. Haapio T, Eerola A (2010) Software project effort assessment. J Softw: Evol Process 22(8):629–652
18. Pirolli P (2007) Information foraging theory: adaptive interaction with information. Oxford University Press, Oxford
19. Stephens DW, Krebs JR (1986) Foraging theory. Princeton University Press, Princeton
20. Liu W, Wang Z, Liu X, Zeng N, Liu Y, Alsaadi FE (2017) A survey of deep neural network architectures and their applications. Neurocomputing 234:11–26
21. Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D. D., & Chen, M. (2014, December). Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)* (pp. 844–848). IEEE.
22. Mallick PK, Ryu SH, Satapathy SK, Mishra S, Nguyen GN, Tiwari P (2019) Brain M.R.I. image classification for cancer detection using deep wavelet autoencoder-based deep neural network. IEEE Access 7:46278–46287
23. Rajmohan G, Chinnappan CV, John William AD, Chandrakrishan Balakrishnan S, Anand Muthu B, Manogaran G (2020) Revamping land coverage analysis using aerial satellite image mapping. Trans Emerg Telecommun Technol. https://doi.org/10.1002/ett.3927
24. Kong F, Li J, Jiang B, Zhang T, Song H (2019) Big data-driven machine learning-enabled traffic flow prediction. Trans Emerg Telecommun Technol 30(9):e3482
25. Medapati PK, Murthy PHST, Sridhar KP (2020) LAMSTAR: For IoT-based face recognition system to manage the safety factor in smart cities. Trans Emerg Telecommun Technol 31:e3843. https://doi.org/10.1002/ett.3843
26. Acharya UR, Fujita H, Oh SL, Hagiwara Y, Tan JH, Adam M (2017) Application of deep convolutional neural network for automated detection of myocardial infarction using E.C.G. signals. Inf Sci 415:190–198
27. Hongtao L, Qinchuan Z (2016) Applications of deep convolutional neural network in computer vision. J Data Acquis Process 31(1):1–17
28. Liu Y, Racah E, Correa J, Khosrowshahi A, Lavers D, Kunkel K, Wehner M, Collins W (2016) Application of deep convolutional neural networks for detecting extreme weather in climate datasets. arXiv preprint. arXiv:1605.01156
29. Pal SK, Mitra S (1992) Multilayer perceptron, fuzzy sets, and classification. Trans Neur Netw 3(5):683–697
30. Tang J, Deng C, Huang GB (2015) Extreme learning machine for multilayer perceptron. IEEE Trans Neural Netw Learn Syst 27(4):809–821
31. Antal P, Fannes G, Timmerman D, Moreau Y, De Moor B (2003) Bayesian applications of belief networks and multilayer perceptrons for ovarian tumor classification with rejection. Artif Intell Med 29(1–2):39–60
32. Hu YH, Tompkins WJ, Urrusti JL, Afonso VX (1993) Applications of artificial neural networks for E.C.G. signal detection and classification. J Electrocardiol 26:66–73
33. Emmerson MD, Damper RI (1993) Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. IEEE Trans Neural Networks 4(5):788–793
34. Verma, B. K. (1995, November). Handwritten Hindi character recognition using multilayer perceptron and radial basis function neural networks. In *Proceedings of ICNN'95-International Conference on Neural Networks* (Vol. 4, pp. 2111–2115). IEEE.
35. Gibson, G. J., Siu, S., & Cowan, C. F. (1990). Application of multilayer perceptrons as adaptive channel equalisers. In Adaptive Systems in Control and Signal Processing. Pergamon. (pp. 573–578)
36. Meyer, M., & Pfeiffer, G. (1993). Multilayer perceptron based decision feedback equalisers for channels with intersymbol interference. I.E.E. Proceedings I (Communications, Speech and Vision), 140(6), 420–424.
37. Dubey SK, Rana A, Dash Y (2012) Maintainability prediction of object-oriented software system by multilayer perceptron model. ACM SIGSOFT Softw Eng Notes 37(5):1–4
38. Gayathri M, Sudha A (2014) Software defect prediction system using multilayer perceptron neural network with data mining. Int J Recent Technol Eng 3(2):54–59
39. Breiman L, Friedman JH, Olshen RA (2017) Classification and regression trees. Chapman and Hall CRC, Routledge
40. Loh WY (2011) Classification and regression trees. Wiley Interdiscip Rev: Data Mining Knowled Discov 1(1):14–23
41. Safavian SR, Landgrebe D (1991) A survey of decision tree classifier methodology. IEEE Trans Syst, Man, Cybern 21(3):660–674
42. Lee TS, Chiu CC, Chou YC, Lu CJ (2006) Mining the customer credit using classification and regression tree and multivariate adaptive regression splines. Comput Stat Data Anal 50(4):1113–1130
43. Marshall RJ (2001) The use of classification and regression trees in clinical epidemiology. J Clin Epidemiol 54(6):603–609
44. Fonarow GC, Adams KF, Abraham WT, Yancy CW, Boscardin WJ, ADHERE Scientific Advisory Committee (2005) Risk stratification for in-hospital mortality in acutely decompensated heart failure: classification and regression tree analysis. JAMA 293(5):572–580
45. Hecker, A., & Kurner, T. (2007, April). Application of classification and regression trees for paging traffic prediction in L.A.C. planning. In Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th (pp. 874–878). IEEE.
46. Bittencourt, H. R., & Clarke, R. T. (2003, July). Use of classification and regression trees (CART) to classify remotely-sensed digital images. In Geoscience and Remote Sensing Symposium, 2003. IGARSS'03. Proceedings. 2003 IEEE International (Vol. 6, pp. 3751–3753). IEEE.

47. Khoshgoftaar TM, Allen EB, Deng J (2002) Using regression trees to classify fault-prone software modules. IEEE Trans Reliab 51(4):455–462
48. Khoshgoftaar, T. M., Allen, E. B., Jones, W. D., & Hudepohl, J. I. (1999). Classification tree models of software quality over multiple releases. In Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on (pp. 116–125). IEEE.
49. Nelder JA, Wedderburn RW (1972) Generalized linear models. J Royal Stat Soc: Series A (General) 135(3):370–384
50. Gotway CA, Stroup WW (1997) A generalized linear model approach to spatial data analysis and prediction. J Agric Biol Environ Statist 157–178
51. Lan CJ, Miaou SP (1999) Real-time prediction of traffic flows using dynamic generalized linear models. Transp Res Rec 1678(1):168–178
52. Guisan A, Edwards TC Jr, Hastie T (2002) Generalized linear and generalized additive models in studies of species distributions: setting the scene. Ecol Model 157(2–3):89–100
53. Chandler RE, Wheater HS (2002) Analysis of rainfall variability using generalized linear models: a case study from the west of Ireland. Water Resour Res 38(10):10–11
54. Chipman HA, George EI, McCulloch RE (2010) B.A.R.T: bayesian additive regression trees. Annals Appl Stat 4(1):266–298
55. Blattenberger G, Fowles R (2014) Avalanche forecasting: using bayesian additive regression trees (BART). In Demand for Communications Services–Insights and Perspectives. Springer, Boston, MA, pp 211–227
56. Zhou Q, Liu JS (2008) Extracting sequence features to predict protein–D.N.A. interactions: a comparative study. Nucleic Acids Res 36(12):4137–4148
57. Wu, J., Huang, L., & Pan, X. (2010, May). A novel bayesian additive regression trees ensemble model based on linear regression and nonlinear regression for torrential rain forecasting. In 2010 Third International Joint Conference on Computational Science and Optimization (Vol. 2, pp. 466–470). IEEE.
58. Shi JQ, Choi T (2011) Gaussian process regression analysis for functional data. CRC Press, London
59. Lataniotis, C., Marelli, S., & Sudret, B. (2017). UQLab user manual–Kriging (Gaussian process modelling). *Report UQLab-V0*, 9–105.
60. Kang F, Han S, Salgado R, Li J (2015) System probabilistic stability analysis of soil slopes using Gaussian process regression with Latin hypercube sampling. Comput Geotech 63:13–25
61. Sun S, Xu X (2010) Variational inference for infinite mixtures of Gaussian processes with applications to traffic flow prediction. IEEE Trans Intell Transp Syst 12(2):466–475
62. Ažman K, Kocijan J (2007) Application of Gaussian processes for black-box modelling of biosystems. ISA Trans 46(4):443–457
63. Liaw A, Wiener M (2002) Classification and regression by random forest. R news 2(3):18–22
64. Kaur, A., & Malhotra, R. (2008, December). Application of random forest in predicting fault-prone classes. In 2008 International Conference on Advanced Computer Theory and Engineering (pp. 37–43). IEEE.
65. Weyuker EJ, Ostrand TJ, Bell RM (2010) Comparing the effectiveness of several modeling methods for fault prediction. Emp Softw Eng 15(3):277–295
66. Primartha, R., & Tama, B. A. (2017, November). Anomaly detection using random forest: A performance revisited. In 2017 International conference on data and software engineering (ICoDSE) (pp. 1–6). IEEE.
67. Sun G, Li S, Cao Y, Lang F (2017) Cervical cancer diagnosis based on random forest. Int J Performability Eng 13(4):446–457
68. Hothorn, T., Hornik, K., & Zeileis, A. (2015). ctree: Conditional inference trees. The Comprehensive R Archive Network, 1–34.
69. Wang, S., Liu, Y., Di Cairano-Gilfedder, C., Titmus, S., Naim, M. M., & Syntetos, A. A. (2018). Reliability analysis for automobile engines: Conditional inference trees. Procedia C.I.R.P., 72, 1392–1397.
70. Das A, Abdel-Aty M, Pande A (2009) Using conditional inference forests to identify the factors affecting crash severity on arterial corridors. J Safety Res 40(4):317–327
71. Ian HW, Eibe F (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann Publishers
72. Bird, C., Pattison, D., D'Souza, R., Filkov, V., & Devanbu, P. (2008, November). Latent social structure in open source projects. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 24–35). A.C.M.
73. Ray PP (2018) A survey on internet of things architectures. J King Saud Univ-Comput Inf Sci 30(3):291–319
74. Chen J, Hu K, Wang Q, Sun Y, Shi Z, He S (2017) Narrowband internet of things: implementations and applications. IEEE Internet Things J 4(6):2309–2314
75. Chavhan S, Gupta D, Chandana BN, Khanna A, Rodrigues JJPC (2019) IoT-based context-aware intelligent public transport system in a metropolitan area. IEEE Internet Things J 7(7):6023–6034. https://doi.org/10.1109/JIOT.2019.2955102
76. Chavhan S, Venkataram P (2020) Prediction based traffic management in a metropolitan area. J Traffic Trans Eng (English edition) 7(4):447–466
77. Pace P, Aloi G, Gravina R, Caliciuri G, Fortino G, Liotta A (2018) An edge-based architecture to support efficient applications for healthcare industry 4.0. IEEE Trans Indust Inform 15(1):481–489
78. Johnsen, F. T., Zieliński, Z., Wrona, K., Suri, N., Fuchs, C., Pradhan, M., ... & Marks, M. (2018, May). Application of IoT in military operations in a smart city. In 2018 International Conference on Military Communications and Information Systems (I.C.M.C.I.S.) (pp. 1–8). IEEE.
79. Gubbi J, Buyya R, Marusic S, Palaniswami M (2013) Internet of Things (IoT): a vision, architectural elements, and future directions. Future Generat Comput Syst 29(7):1645–1660
80. Lee I, Lee K (2015) The Internet of Things (IoT): applications, investments, and challenges for enterprises. Bus Horiz 58(4):431–440
81. Saravanan, K., & Saraniya, S. (2018). Cloud I.O.T. based novel livestock monitoring and identification system using UID. Sensor Review.
82. De Donno M, Tange K, Dragoni N (2019) Foundations and evolution of modern computing paradigms: cloud, IoT, edge, and fog. IEEE Access 7:150936–150948
83. Calheiros, R. N. (2020). Fog and Edge Computing: Challenges and Emerging Trends (Invited Talk). In 2nd Workshop on Fog Computing and the IoT (Fog-IoT 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
84. Goudarzi M, Wu H, Palaniswami MS, Buyya R (2020) An application placement technique for concurrent IoT applications in edge and fog computing environments. IEEE Trans Mobile Comput. https://doi.org/10.1109/TMC.2020.2967041