# Rotation Forest in Software Defect Prediction

GORAN MAUŠA, Faculty of Engineering, University of Rijeka, Rijeka, Croatia
NIKOLA BOGUNOVIĆ, Faculty of Electrical Engineering and Computing, University of Zagreb
TIHANA GALINAC GRBAC, Faculty of Engineering, University of Rijeka
BOJANA DALBELO BAŠIĆ, Faculty of Electrical Engineering and Computing, University of Zagreb

Software Defect Prediction (SDP) deals with localization of potentially faulty areas of the source code. Classification models are the main tool for performing the prediction and the search for a model of utmost performance is an ongoing activity. This paper explores the performance of Rotation Forest classification algorithm in the SDP problem domain. Rotation Forest is a novel algorithm that exhibited excellent performance in several studies. However, it was not systematically used in the SDP. Furthermore, it is very important to perform the case studies in various contexts. This study uses 5 subsequent releases of Eclipse JDT as the objects of the analysis. The performance evaluation is based on comparison with two other, known classification models that exhibited very good performance so far. The results of our case study concur with other studies that recognize the Rotation forest to be the state of the art classification algorithm.

Categories and Subject Descriptors: **D.2.9 [Software Engineering]:** Management—*Software quality assurance (SQA)*; **H.2.8 [Information Systems]**: Database Applications—*Data mining*

Additional Key Words and Phrases: Rotation Forest, Random Forest, Logistic Regression, Software Defect Prediction

## 1. INTRODUCTION

Software Defect Prediction (SDP) is an evolving research area that aims to improve the software quality assurance activities. It is in search for an effective predictive model that could lead the testing resource allocation towards the software modules that are more likely to contain defects. Empirical studies proved that there is certain regularity in defect distribution. It follows the Pareto principle, meaning that minority of source code (20%) is responsible for majority of defects (80%) [Galinac Grbac et al., 2013]. Many classification models have been used for SDP, with various outcomes. It is important to emphasize that the context of data source may be the cause of inconsistent results in software engineering research [Galinac Grbac and Huljenić, 2014]. Therefore, we need to perform a large number of systematically defined case studies with as much data from various domains in order to achieve generalizability of results.

In this paper, we examine the potential of Rotation Forest (RttFor), a novel classification model. The RttFor achieved some promising results in classification problem domain [Rodríguez et al., 2006]. However, its potential is scarcely examined in the SDP research area. That is why we compare its performances with two known classification models, the Logistic Regression (LogReg) and the Random Forest (RndFor). LogReg is an example of a reliable classification model of good performance in many application domains. RndFor is another novel approach that showed promising results and in many cases outperformed other classification models [Lessmann et al., 2008]. RttFor is similar to RndFor because it uses a number of decision trees to make the prediction. But unlike RndFor, each decision tree uses all the features. Furthermore, it weights each feature using the principal component analysis (PCA) method upon randomly selected groups of input features. That way it maximizes the variance between features and achieves better performance [Amasyali and Ersoy, 2014].

The work presented in this paper is supported by the University of Rijeka research grant Grant 13.09.2.2.16.
Author's address: G. Mauša, Faculty of Engineering, Vukovarska, 58, 51000 Rijeka, Croatia; email: goran.mausa@riteh.hr; N. Bogunović, Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Coratia; email: nikola.bogunovic@fer.hr; T. Galinac Grbac, Faculty of Engineering, Vukovarska 58, 51000 Rijeka, Croatia; email: tihana.galinac@riteh.hr; B. Dalbelo Bašić, Faculty of Electrical Engineering and Computing Unska 3, 10000 Zagreb, Croatia; email: bojana.dalbelo@fer.hr

We perform the comparison in the context of five subsequent releases of an open source projects for java development, the Eclipse JDT. The obtained results are evaluated in terms of Accuracy, TPR, FPR, F-measure, Kappa statistics and AUC. The paired T-test indicated RttFor to perform equally good or significantly better than the other two classifiers in all the evaluation metrics, with the exception of TPR. Our findings are consistent with other few case studies that obtained favorable results when comparing RttFor to other classification and even regression models [Pardo et al., 2013].
The structure of this paper is as following: Section 2 provides the promising results achieved by RttFor in classification domain that motivated this study. Section 3 presents the algorithms of the 3 classifiers we compare in more details. The description of our case study is given in section 4. The results are presented and their threats to validity are examined in section 5. Section 6 finally gives the conclusion.

## 2. BACKGROUND

There are some case studies that achieved promising results when using RttFor algorithm. The authors of the algorithm compared it with Bagging, AdaBoost and RndFor on a random selection of 33 benchmark data sets from UCI machine learning repository [Rodríguez et al., 2006]. These datasets contained from 4 up to 69 features, from 57 up to 20,000 instances and from 2 to 26 distinct classes. Their results indicated that the RttFor outperformed other algorithms, achieving 84 wins and only 2 loses in paired comparison of significant differences between models' accuracy levels. Amasyali and Ersoy [Amasyali and Ersoy, 2014] performed the comparison of Bagging, Random Subspaces, RndFor and RttFor algorithm on 43 datasets from the same UCI repository in terms of accuracy. Each algorithm was used with and without the addition of new, artificially combined features. RttFor with the addition of new features outperformed all the other algorithms and RttFor without the addition of new features was the second best.
There are several studies that used the RttFor for different classification purposes, like image classification [Kuncheva et al., 2010], [Xia et al., 2014], [Zhang, 2013]. There, the RttFor was outperformed by Random Subspace with Support Vector Machine (SVM) and several other algorithms in classification of brain images obtained through functional magnetic resonance imaging [Kuncheva et al., 2010]. The impact of several feature transformation methods was analyzed in the RttFor: PCA that is the default method, maximum noise fraction, independent component analysis and local Fisher discriminant analysis [Xia et al., 2014]. Xia et al. also compared each of these variations of RndFor to CART, Bagging, AdaBoost, SVM and LogReg via Variable Splitting and Augmented Lagrangian (LORSAL) in hyperspectral remote sensing image classification. The default variant of RttFor with PCA outperformed others in terms of accuracy. An interesting cascade classifier ensemble is created by Zhang [Zhang, 2013]. He combined k-Nearest Neighbor (kNN), SVM, Multi-Layer Perceptrons (MLP) and RndFor as the first cascade and RttFor with MLP as the second cascade. Each stage gives a majority vote that is supposed to be above a predefined threshold value. The second cascade is targeting the rejected instances from previous cascade, for which the majority vote was not above that threshold, to further insure the confidence. They achieved a great improvement in reducing the rate of rejected instances and, therefore, minimizing the misclassification cost. All of these studies used the RttFor because its performance was reported to be very good comparing to other classifiers.
To the best of our knowledge, the only use of RttFor in the SDP domain was done by [Palivela et al., 2013]. However, it remained unexplained what is their source of data, what information is stored in it, how many features and how many instances does it contain. They compared several classification algorithms: C4.5, SMO, RttFor, Bagging, AdaBoost M1, RndFor and DBScan, evaluated the performance in terms of 8 different evaluation metrics: Accuracy, True Positive rate, False Positive rate, Recall, F-measure, Kappa statistics and AUC, but lacked the comparison of statistical

significance. Nevertheless, their results also indicated the RttFor as the classifier of utmost performance.

## 3. CLASSIFICATION MODELS

All the algorithms involve building models iteratively upon a training set. The training set contains multiple independent variables and 1 dependent variable that we want to predict. The model is trained on this dataset and then it is evaluated on previously unseen data, i.e. the testing set. In classification domain, the dependent variable is discrete, unlike in regression domain, where it is continuous. In our case, the dependent variable is a binary class, where 1 indicates the presence of a bug and 0 indicates the absence of bugs and the independent variables are source code features. We present the algorithms of all the three classification models in the remainder of this section.

### 3.1 Logistic Regression

LogReg is a statistical classifier. It is used in various classification problem domains and it is renowned as a robust method. That quality makes this classification algorithm appealing to software engineering domain where data are rarely normally distributed, usually are skewed and contain outliers and missing values. The multivariate LogReg is used for classification problem with multiple independent variables [Tabachnick and Fidell, 2001]. For a training set that consists of a set of features X of size ($N$ x $n$), with $N$ being the number of instances and $n$ being the number of features $m_k$ (with $1 \leq k \leq n$), and of dependent variable Y of size $n$ as the binary class vector, the LogReg classification algorithm can be explained in these steps:

(1) Initiate the search of regression coefficients $C_k$, where $C_0$ is the intercept and $C_k$ are the weights for each feature $m_k$ and build the classification model $\hat{Y}$ as:

$$\hat{Y}(m_1, m_2, \dots m_n) = \frac{e^{C_0+C_1m_1+\cdots+C_nm_n}}{1+e^{C_0+C_1m_1+\cdots+C_nm_n}} \quad (1)$$

(2) Evaluate the model by assessing the natural log likelihood ($NLL$) between the actual ($Y_j$) and the predicted ($\hat{Y}_j$) outcomes for each $j$-th instance (with $1 \leq j \leq N$) as:

$$NLL = \sum_{j=1}^{N}\left[Y_j \ln\left(\hat{Y}_j\right) + (1 - Y_j)ln(1 - Y_j)\right] \quad (2)$$

(3) Optimize the coefficients using maximum likelihood procedure iteratively, until convergence of coefficients is achieved

(4) The output of classification is the probability that a given instance belongs to the class $\hat{Y}_j = 1$

### 3.2 Random Forest

RndFor is an ensemble classifier, proposed by [Breiman, 2001], that takes advantage of randomness induced by splitting the instances and features for multiple classifiers. Decision trees are the classifiers and each of them receives a different training subset. The final classification output is the majority's decision of all the trees. That way, generalization error is reduced, impact of outliers and noise is minimized and model's performance is improved. For a training set that is defined according to previous Subsection 3.1 the RndFor classification algorithm is presented in Figure 1a. With $T_i$ indicating an arbitrary tree and $K$ as the number of trees, it contains following steps:

(1) Assign each tree $T_i$ with a subset of features of size between 1 and $\sqrt{n}$ from the training set X

(2) Take a bootstrap sample of instances from the training set (2/3 for **training** and 1/3 for **error estimation**)

(3) Iteratively grow the trees using CART methodology without pruning, selecting the best feature and splitting the node into two daughters

(4) Average the tree's error testing the subset of trees on **mutual error estimation subset** and testing each tree individually on its own **error estimation subset**

(5) The output of classification is the majority vote of all trees in the forest

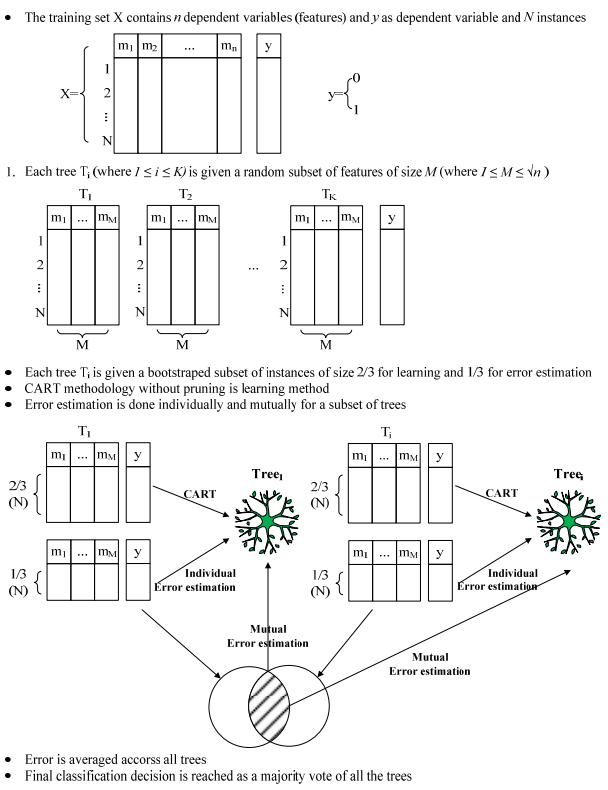


Fig. 1a Random Forest Algorithm

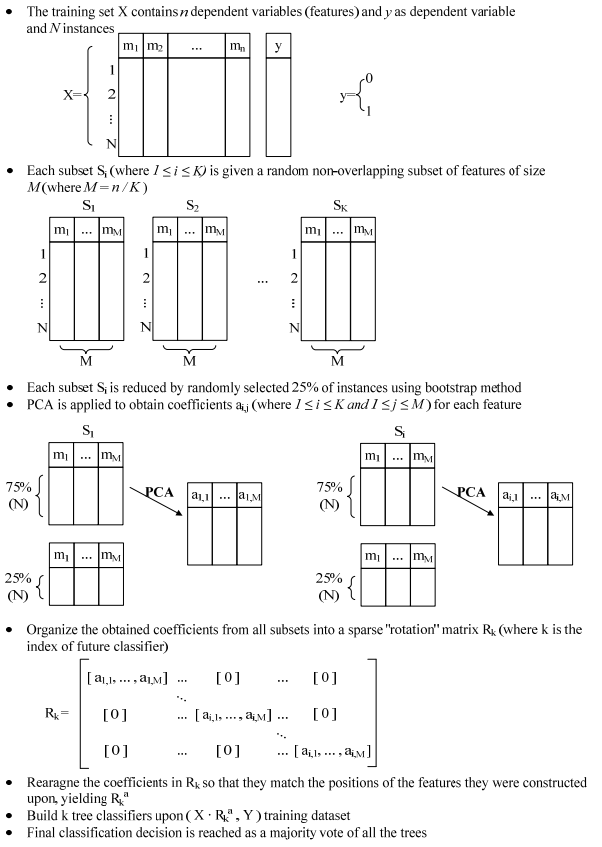


Fig. 1b Rotation Forest Algorithm

## 3.3 Rotation Forest

The RttFor is a novel ensemble classifier algorithm, proposed by [Rodríguez et al., 2006]. It is an algorithm that involves randomized PCA performed upon a selection of features and instances of the training set before building the decision trees. For a training set that is defined according to previous Subsection 3.1, the RttFor classification algorithm is presented in Figure 1b. With $S_i$ indicating an arbitrary subset of training set and $K$ as the total number of subsets, it contains following steps:

(1) Split the features of training set X into $K$ non-overlapping subsets of equal size $M=n/K$

(2) From each subset $S_i$, randomly remove 25% of instances using bootstrap method to ensure the coefficients obtained by PCA are different for each tree

(3) Run PCA on the remaining 75% of the subset and obtain $a_{i,j}$ coefficients for each $i$-th subset and $j$-th feature inside the subset

(4) Organize the $a_{i,j}$ coefficients in a sparse rotation matrix and rearrange the coefficients so that they match the positions of the $n$ features they were build upon to obtain $R_k{}^a$

(5) Repeat the procedure in steps 1 – 4 for each tree in the RttFor and build the tree upon training set $X \cdot R_k{}^a$, Y

(6) The output of classification in the majority vote of all the trees in the forest

## 4. CASE STUDY

The goal of this case study is to evaluate the performance of RttFor in SDP. We used LogReg, a robust and trustworthy statistical classifier used in many other domains and the RndFor, a newer ensemble algorithm that is achieving excellent results in SDP, as the basis of our comparison. Our research question (**RQ**) is how well does RttFor perform in SDP when compared to LogReg and RndFor?

### 4.1 Datasets

We collected datasets from five consecutive releases of Eclipse JDT project: 2.0, 2.1, 3.0, 3.1 and 3.2. Eclipse is an integrated development environment (IDE), mostly written in Java, but offering a wide range of programming languages for development. The Java development tools (JDT) is one of the largest development environments and, due to publicly available source code management repository in GIT and bug tracking repository in Bugzilla, it is often analyzed in SDP research. The number of the source code files for each release of the JDT project and the ratio of NFPr and FPr files is given in Table I.

Table I. Eclipse Datasets

| **Dataset** | **Appearance** | | |
|---|---|---|---|
| | ***Files*** | ***NFpr*** | ***Fpr*** |
| JDT 2.0 | 2021 | 50.1% | 49.9% |
| JDT 2.1 | 2343 | 64.1% | 35.9% |
| JDT 3.0 | 2953 | 58.1% | 41.9% |
| JDT 3.1 | 3394 | 64.5% | 35.5% |
| JDT 3.2 | 1833 | 59.0% | 41.0% |

We collected the data using Bug Code (BuCo) Analyzer tool, which we developed for this purposes [Mauša et al., 2014]. The data collection approach starts with the collection of fixed bugs of severity greater than trivial, for each of the analyzed releases of the JDT project. Then it performs the bug-code linking on the file level of granularity using a regular expression that defines the surrounding characters of a Bug ID in the commit messages and outperforms even some of the complex prediction techniques [Mauša et al., 2014a]. Finally, it computes the 50 software product metrics using LOC Metrics and JHawk for each source code file. All the numerical metrics are used as independent variables for our experiment, so we excluded only the name of superclass. In order to make the dependent variable suitable for classification, the *number of bugs per file* was transformed into binary attribute named *Fault proneness*. Its value is set to 1 for all the files that contain at least 1 bug and set to 0 otherwise. The final structure of dataset is a matrix of size $N$ *x* $n$, where $N$ represents the number of files and $n$ represents the number of features.

### 4.2 Experiment Workflow

We used the Weka Experimenter Environment (version 3.6.9) to perform the experiment. Weka is a popular open source software for machine learning written in java, developed at the University of Waikato, New Zealand. The experiment workflow includes the following steps:

(1) import the data in *arff* format and assign *fault proneness* to be the dependent variable
(2) split the training and testing sets using 10-times 10-fold cross validation
(3) build RttFor, LogReg and RndFor models
(4) evaluate the models' performance in terms of Acc, TPR, FPR, F-measure, Kappa and AUC
(5) perform the paired T-test of significance

The 10-times 10-fold cross validation is used to prepare the training and testing sets. The 10-fold cross validation divides the dataset into 10 parts of equal size, randomly picking instances (rows) and maintaining the number of features (columns). In each of 10 steps, another 1/10 portion of dataset is used as testing set and the remaining 9/10 are used for training. The 10-times 10-fold cross validation iterates the whole procedure 10 times, evaluating the classification model for 100 times in total.
The classification models are all built and evaluated upon the same training and testing datasets, i.e. JDT release. It is important to be aware that the parameters tuning can improve the performance of various classifiers [Chen et al., 2009]. Thus, we performed an examination of their performance with various configurations. We used Weka's CVParameter Selection meta classifier that performs parameter selection by cross-validation for any classifier. For RndFor, we were configuring the number of features per tree from 1 to 8 (8 is the default value, calculated as $\log_2(number\ of\ attributes) + 1$), the maximum depth of tree from 1 to 5 (including the default unlimited depth) and the number of trees from 10 to 50 with step of 5 (default value is 10). For RttFor, we were configuring only the number of groups from 3 to 10 (default value is 3) and left the number of iterations to default value of 10 and the percentage of removed instances to default value of 50%. Since there were no significantly different results obtained by either of these configurations, we left all the parameters to their default values for the main experiment.

### 4.3 Performance Evaluation

The evaluation of binary classification problems is usually done using the confusion matrix. Confusion matrix consists of correctly classified instances, true positive (TP) and true negative (TN), and incorrectly classified instances, false positive (FP) and false negative (FN). In our case, the Positive instances are the ones that are Fault Prone (FPr), i.e. files that have at least 1 bug, and Non-Fault-Prone files (NFPr) are the Negative ones. The evaluation metrics that we used in our experiment are the following ones:

– *Accuracy (Acc) – number of correctly classified files:*

$$Acc = \frac{TP+TN}{TP+FN+FP+TN} \tag{3}$$

– *True Positive Rate (TP_Rate, Recall) – number of correctly classified FPr files in total FPr files:*

$$TP\ Rate = \frac{TP}{TP+FN} \tag{4}$$

– *False Positive Rate (FP Rate) – number of incorrectly classified NFPr files in total false predictions:*

$$FP\ Rate = \frac{FP}{FP+FN} \tag{5}$$

– *F-measure (F) – harmonic mean of TPR and Precision:*

$$F = 2 \cdot \frac{\mathrm{TPR} \cdot \mathrm{Precision}}{TPR+Precision} \tag{6}$$

– *Kappa statistics – accuracy that takes into account the chance of random guessing:*

$$\kappa = \frac{\mathrm{Acc} - \mathrm{Pr}\_\mathrm{rand}}{1 - \mathrm{Pr}\ \_rand} \tag{7}$$

*where Pr_rand is equal to:*

$$Pr\_rand = \frac{\text{TP+FN}}{Total} \cdot \frac{\text{TP+FP}}{Total} + \frac{\text{TN+FP}}{Total} \cdot \frac{\text{TN+FN}}{Total} \quad (8)$$

The usual output of a binary classifier is the probability that a certain instance belongs to the Positive class. Before making the final prediction, a probability threshold value, above which instances are going to be classified as Positive, needs to be determined. That is why all the above mentioned evaluation metrics are calculated with predetermined threshold value. A metric that does not depend on the threshold value is:

– *Area under receiver operating curve (AUC)*:

$$AUC = \int_0^1 \text{ROC\_curve} \quad (9)$$

*where ROC_curve is a graphical plot that illustrates the relation between TP_Rate and FP_Rate for all possible probability threshold values.*

All the used metrics have their values in range [0-1]. The performance of a classifier is better for higher values of Accuracy, TP Rate, F-measure, Kappa statistics and AUC. Only in the case of FP Rate, also known as the false alarm rate, the performance is better for lower values. It is important to use several evaluation metrics because they examine predictive performance from various angles. In the presence of severe data imbalance between the two output classes, it is even more important. For example, Acc can then easily become very high, misleading us to believe in excellent performance. On the other hand, the FP Rate would be also very high, indicating that the prediction is of questionable value [Mauša et al., 2012]. After obtaining the results, we perform the paired T-test in order to discover whether there are significant differences between classifiers. The whole process is repeated for each of the 5 Eclipse JDT datasets and for each of the 6 evaluation metrics.

## 5. RESULTS

The results of the paired T-tests for Acc, TPR, FPR, F-measure, Kappa statistics and AUC are given in Table II. Paired T-test compares only two classifiers at the time in one evaluation metric. First five rows represent the paired T-test comparison of results when the RttFor is the basis of comparison and the second five rows provide the results when the LogReg is compared to other two classifiers. The only remaining combination would be to use the RndFor as a comparison basis, but that one can be deduced from the previous two. The basis of comparison and its results are given in bold. The results are presented with their average value and standard deviation of 100 iterations that are obtained with the 10-times 10-fold cross validation process. The results that exhibit significant difference are marked with * and *v*. Sign * is given for cases in which the compared classifier is performing significantly worse than the basis of comparison and *v* is given for significantly better performance. The results that do not exhibit statistically significant difference have no sign adjacent to them. From results presented in tables II, we draw following observations:

– *Overall summary of results shows that RttFor achieved 29 wins and 14 loses, RndFor achieved 19 wins and 14 loses, LogReg achieved 15 wins and 27 loses*
– *RttFor outperformed RndFor and LogReg in terms of AUC and Kappa statistics in all but 1 case.*
– *LogReg outperformed RndFor and RttFor in terms of FP rate in all but 1 case.*
– *RttFor is outperformed only in terms of TP Rate and FP Rate, 6 times by RndFor and 8 times by LogReg*

Table II. Paried T-tests between Rotation Forest, Random Forest and Logistic Regression

| Dataset | Accuracy | | | TP Rate | | | FP Rate | | |
|---|---|---|---|---|---|---|---|---|---|
| | **RttFor** | RndFor | LogReg | **RttFor** | RndFor | LogReg | **RttFor** | RndFor | LogReg |
| JDT 2.0 | **0.75(0.03)** | 0.73(0.03) * | 0.71(0.03) * | **0.76(0.04)** | 0.77(0.04) | 0.76(0.04) | **0.26(0.04)** | 0.35(0.05) v | 0.37(0.05) v |
| JDT 2.1 | **0.83(0.02)** | 0.82(0.02) | 0.81(0.02) * | **0.86(0.03)** | 0.88(0.03) | 0.88(0.03) | **0.38(0.06)** | 0.46(0.06) v | 0.52(0.05) v |
| JDT 3.0 | **0.79(0.02)** | 0.79(0.02) | 0.78(0.02) * | **0.82(0.03)** | 0.84(0.03) | 0.86(0.03) v | **0.35(0.04)** | 0.40(0.04) v | 0.48(0.04) v |
| JDT 3.1 | **0.82(0.02)** | 0.82(0.02) | 0.81(0.01) * | **0.87(0.03)** | 0.87(0.02) | 0.91(0.02) v | **0.43(0.05)** | 0.49(0.05) v | 0.61(0.05) v |
| JDT 3.2 | **0.81(0.03)** | 0.80(0.02) | 0.80(0.02) | **0.82(0.04)** | 0.85(0.04) v | 0.88(0.03) v | **0.32(0.06)** | 0.38(0.06) v | 0.47(0.06) v |
| | LogReg | RndFor | RttFor | LogReg | RndFor | RttFor | LogReg | RndFor | RttFor |
| JDT 2.0 | **0.71(0.03)** | 0.73(0.03) | 0.75(0.03) v | **0.76(0.04)** | 0.77(0.04) | 0.76(0.04) | **0.37(0.05)** | 0.35(0.05) | 0.26(0.04) * |
| JDT 2.1 | **0.81(0.02)** | 0.82(0.02) | 0.83(0.02) v | **0.88(0.03)** | 0.88(0.03) | 0.86(0.03) | **0.52(0.05)** | 0.46(0.06) * | 0.38(0.06) * |
| JDT 3.0 | **0.78(0.02)** | 0.79(0.02) | 0.79(0.02) v | **0.86(0.03)** | 0.84(0.03) * | 0.82(0.03) * | **0.48(0.04)** | 0.40(0.04) * | 0.35(0.04) * |
| JDT 3.1 | **0.81(0.01)** | 0.82(0.02) | 0.82(0.02) v | **0.91(0.02)** | 0.87(0.02) * | 0.87(0.03) * | **0.61(0.05)** | 0.49(0.05) * | 0.43(0.05) * |
| JDT 3.2 | **0.80(0.02)** | 0.80(0.02) | 0.81(0.03) | **0.88(0.03)** | 0.85(0.04) * | 0.82(0.04) * | **0.47(0.06)** | 0.38(0.06) * | 0.32(0.06) * |
| | **F-measure** | | | **Kappa** | | | **AUC** | | |
| | **RttFor** | RndFor | LogReg | **RttFor** | RndFor | LogReg | **RttFor** | RndFor | LogReg |
| JDT 2.0 | **0.75(0.03)** | 0.73(0.03) * | 0.71(0.03) * | **0.49(0.05)** | 0.42(0.06) * | 0.38(0.06) * | **0.83(0.03)** | 0.79(0.03) * | 0.75(0.03) * |
| JDT 2.1 | **0.83(0.02)** | 0.82(0.02) | 0.81(0.02) * | **0.50(0.06)** | 0.44(0.06) * | 0.39(0.06) * | **0.84(0.02)** | 0.81(0.02) * | 0.79(0.03) * |
| JDT 3.0 | **0.79(0.02)** | 0.79(0.02) | 0.78(0.02) * | **0.48(0.05)** | 0.45(0.04) * | 0.40(0.05) * | **0.82(0.02)** | 0.80(0.02) * | 0.77(0.03) * |
| JDT 3.1 | **0.82(0.02)** | 0.82(0.02) | 0.81(0.01) * | **0.45(0.05)** | 0.41(0.05) * | 0.34(0.05) * | **0.81(0.02)** | 0.79(0.03) * | 0.75(0.03) * |
| JDT 3.2 | **0.81(0.03)** | 0.80(0.02) | 0.80(0.02) | **0.51(0.07)** | 0.48(0.07) | 0.43(0.06) * | **0.84(0.03)** | 0.82(0.03) * | 0.79(0.03) * |
| | LogReg | RndFor | RttFor | LogReg | RndFor | RttFor | LogReg | RndFor | RttFor |
| JDT 2.0 | **0.71(0.03)** | 0.73(0.03) | 0.75(0.03) v | **0.38(0.06)** | 0.42(0.06) | 0.49(0.05) v | **0.75(0.03)** | 0.79(0.03) v | 0.83(0.03) v |
| JDT 2.1 | **0.81(0.02)** | 0.82(0.02) | 0.83(0.02) v | **0.39(0.06)** | 0.44(0.06) v | 0.50(0.06) v | **0.79(0.03)** | 0.81(0.02) v | 0.84(0.02) v |
| JDT 3.0 | **0.78(0.02)** | 0.79(0.02) | 0.79(0.02) v | **0.40(0.05)** | 0.45(0.04) v | 0.48(0.05) v | **0.77(0.03)** | 0.80(0.02) v | 0.82(0.02) v |
| JDT 3.1 | **0.81(0.01)** | 0.82(0.02) | 0.82(0.02) v | **0.34(0.05)** | 0.41(0.05) v | 0.45(0.05) v | **0.75(0.03)** | 0.79(0.03) v | 0.81(0.02) v |
| JDT 3.2 | **0.80(0.02)** | 0.80(0.02) | 0.81(0.03) | **0.43(0.06)** | 0.48(0.07) v | 0.51(0.07) v | **0.79(0.03)** | 0.82(0.03) v | 0.84(0.03) v |

* – significantly worse, v – significantly better

## 5.1 Threats to Validity

It is very important to be aware of the threats to validity of our case study [Runeson and Höst, 2009] and we address them in this subsection. The generalization of our results is limited with the choice of the datasets we used. We covered only the evolution through 5 subsequent releases of only 1 project that comes from only 1 open source community. A greater number of case studies like this one, with as many datasets from various domains, is required to achieve a conclusion of greater confidence level. The choice of comparing classification algorithms is another threat to validity. This case study included only 2 classifiers other than RttFor. However, we chose the LogReg and the RndFor due to their very good performance in other case studies. The choice of classification model's parameters is a potential source of bias to our results. That is why we performed an analysis of performance when tuning the parameters. Since we noticed no statistically significant difference between various configurations, we left them to their default values.

## 6. RESULTS

This paper continues the search for a classification algorithm of utmost performance for the SDP research area. We analyzed a promising novel classifier called RttFor that received very limited attention in this field so far. We compared the performance of RttFor with the LogReg and the RndFor in terms of 6 diverse evaluation metrics in order to get as detailed comparison as possible. The classifiers were used upon 5 subsequent releases of Eclipse JDT open source project. These datasets contain between 1800 and 3400 instances, i.e. java files, 48 features describing their complexity and size and 1 binary class variable that indicates weather the file contains defects or not. The comparison was done using paired T-test of significance between results obtained by 10-times 10-fold cross-validation.

The conclusion of our case study and the answer to our **RQ** is that RttFor is indeed a state of the art algorithm for classification purposes. The overall ranking of the three classifiers we analyzed shows that the RttFor is the most successful one, the RndFor is the second best and the LogReg is the least successful classifier. This finding is consistent with other case studies that used RttFor and proves that this classifier needs to be taken into account when performing research in SDP more often. Our future work intentions include a more complex comparison that would include a greater number of classification algorithms. But more importantly, it would include a greater number of datasets, covering longer periods of projects' evolution and a greater number of projects from various contexts.

### REFERENCES


T. Galinac Grbac, P. Runeson and D. Huljenic, 2013. A second replicated quantitative analysis of fault distributions in complex software systems. IEEE Trans. Softw. Eng. 39(4), pp. 462-476

T. Galinac Grbac and D. Huljenić, 2014. On the probability distributions of faults in complex software systems, Information and Software Technology (0950-5849), pp. 1-26.

J. J. Rodríguez, L.I. Kuncheva and C.J. Alonso, 2006. Rotation Forest: A New Classifier Ensemble Method, IEEE Transactions on Pattern Analysis and Machine Learning, vol. 28, no. 10, pp. 1619-1629.

S. Lessmann, B. Basesens and S. Pietsch, 2008. Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, IEEE Transactions On Software Engineering, vol. 34, no. 4, pp. 485-496.

M. F. Amasyali and K. O. Ersoy, 2014. Classifier Ensembles with the Extended Space Forest, IEEE Transactions on Knowledge and Data Engineering, Vol. 26, no. 3, pp. 549-562

C. Pardo, J. F. Diez-Pasor, C. Garcia-Osorio and J. J. Rodriguez, 2013. Rotation Forest for regression, Applied Mathematics and Computing, vol. 219 (19), pp. 9914-9924.

L. I. Kuncheva, J. J. Rodriguez, C. O. Plumpton, D. E. J. Linden and S. J. Johnston, 2010. Random Subspace Ensembles for fMRI Classification, IEEE Transaction on Medical Imaging, Vol. 29., No. 2, pp. 531-542

J. Xia, P. Du, X. He and J. Chanussot, 2014. Hyperspectral Remote Sensing Image Classification Based on Rotation Forest, IEEE Geoscience and Remote Sensing Letters, Vol. 11., no. 1, pp. 239-243

Bailing Zhang, 2013. Reliable Classification of Vehicle Types Based on Cascade Classifier Ensembles, IEEE Transactions on Inteligent Transportation Systems, Vol. 14, no. 1, pp. 322-332

H. Palivela, H. K. Yogish, S. Vijaykumar and K. Patil, 2013. A stury od mining algorithms for finding accurate results and marking irregularities in software fault prediction, International Conferenc on Information Communication and Embedded Systems (ICICES), pp. 524–530.

B. G. Tabachnick and L. S. Fidell, 2001. Using Multivariate Statistics, 5th edition, Pearson, 2007, ISBN 0-205-45938-2

L. Breiman, 2001. Random forests, Machine Learning, Vol. 45, No. 1, pp. 5–32

C. Chen, M.-L. Shyu and S.-C. Chen, 2009. Supervised Multi-Class Classification with Adaptive and Automatic Parameter Tuning, IEEE International Conference on Information Reuse & Integration, IRI '09, Las Vegas, USA, pp. 433-434

G. Mauša, T. Galinac Grbac and B. Dalbelo Bašić, 2014. Software Defect Prediction with Bug-Code Analyzer – a Data Collection Tool Demo, In Proceedings of SoftCOM '14. Split, Croatia

G. Mauša, P. Perković, T. Galinac Grbac and B. Dalbelo Bašič, 2014. Techniques for Bug-Code Linking, In Proceedings of SQAMIA '14, Lovran , Croatia, pp. 47-55

G. Mauša, T. Galinac Grbac, and B. Dalbelo Bašić, 2012. Multivariate logistic regression prediction of fault-proneness in software modules, In Proceedings of MIPRO '12, Opatija, Croatia, pp. 698–703

P. Runeson and M. Höst, 2009. Guidelines for conducting and reporting case study research in software engineering, Empirical Softw. Engg., vol. 14, pp.131–164