Readable Decision Tree to Achieve Performance of Random Forest for Software Bug Prediction

Raghavendra Nayak
North Carolina State University
rmuddur@ncsu.edu

Sunil Narasimhamurthy
North Carolina State University
snarasi5@ncsu.edu

Abstract

There is no software that has zero bugs. Given the enormous of scale at which software industry is creating new software's, there are hundreds and thousands of bugs getting introduced. It may not be possible for a manual evaluation & fixing of all these bugs. So, there are data mining based solutions which can helpful in identifying and classifying these defects. However, in the paper we are more interested in review of machine learning techniques for software fault prediction with focus on random forest. We would be commenting on the paper by Lan Guo et al.

Keywords Random Forest, Neural Networks, Decision Tree, Fast Frugal Tree, Defect Prediction, Automated Defect Finder.

1 Introduction

Now a day's machine learning is finding its application everywhere. Not forgetting its immense application is traditional empirical software engineering practices. With advancement in cloud and emergence of Micro Service oriented architecture there are lot of moving components in the infrastructure. Micro Services architecture is a part of service oriented architecture where each service has complete flexibility in terms of deployment, development, migration, etc. This coupled with Agile software building practice companies are adapting DevOps culture which lets them deploy software very fast. This capability of CI/CD pipelining application is achievable only if there is heavy reliance on automation practices. Running tests continuously will help in finding obvious bugs but given the urgency placed on delivering a working software a developer may not always find time to develop tests to do total coverage of entire code base. There are chances that a test case might be faulty. Further exploiting such build process there are other techniques such as running static analysis tool to predict the code quality and look for bad smells in code. Given the volume of code there are instances where there can be hell lot of warning and issues. This is ignored in most cases and team would happily set an under estimated lower threshold to get their green check marks on their builds. This is where machine learning can be helpful in finding bugs and prediction faults in the software [7].

2 Criteria

2.1 Model Readability

Reliability refers to the consistency of the results obtained from the research and how well such results can be understood. Some learners are transparent in terms of how it solves the problem. Some are pretty much black box for example Neural Networks. It becomes difficult to train and outcome becomes non-deterministic and which depend of internal parameters which are often chose by trial and error method. Other examples of algorithms with good readability are Decision tree, Random forest etc. Not all learners with good readability are useful. Let this criterion alone not be an advantage of an algorithm [4].

2.2 Actionable Conclusions

Most of the time a learner is relied upon to give Boolean decisions. But not always the case where many criteria decides the outcome. A reliable learner to give actionable conclusion outcome is one such predicts whether certain outcome based on model which it has generated.

2.3 Learnability and Repeatability of Results

Computation has become cheap, this allows many iterations of running learner. However fast learner is preferable over slow ones. Variant exists for complex algorithms which can work over only small sets of data e.g mini batch k-means. Naïve Bayes is fast and low memory. "Most of the learning algorithms operate as "black-boxes", which give predictions based on some input, internal working still being mystery. Moreover, their output is often difficult to interpret or understand by business users".

2.4 Multi-Goal Reasoning

Goal-driven autonomy (GDA) is a reflective model of goal reasoning that controls the focus of an agent's planning activities by dynamically resolving unexpected discrepancies in the world state, which frequently arise when solving tasks in complex environments. Many times, learner must apply learner based on multiple goals. One criterion is to use domination score to calculate weightage for each variable. Regression tree learner normalizes each variable and based on goal the data is sorted and classified.

This helps in reducing all goals into single domination score and apply supervised and unsupervised classification.

2.5 Anomaly Detection

In machine learning and data mining tasks with large data sets it is common to identify data which are outliers/anomalies. Outlier is an observation which is distant from other observations. The choice of whether to ignore such outliers depends on the application. In case such as fraud detection for credit cards or network intrusion detection systems these patterns of anomalies are crucial. In ideal case the learner should be able to identify this anomaly and ignore such observations if it's not crucial. Having an anomaly detection system is relatively harder.

2.6 Incremental Learning

Incremental learning algorithm in ideal sense can adapt to new data sets without forgetting the old knowledge and it doesn't have to compute the whole thing all over again. This type of learner is useful when the new data is continuously consumed to expand the existing model knowledge.
Many machine learning modes support incremental learning such as decision trees, artificial neural networks, incremental SVM etc. Outlier detection is one such good use case of incremental learner.

2.7 Context Aware

Context awareness of a learner is a feature which can adapt to different regions of data based on the current context and can generate appropriate output. It can be classified into two types
a. Active context-awareness: the system adapts itself to the changing environment modifying its behaviour,
b. Passive context-awareness: the system presents the new or updated context to an interested user without modifying its behaviour.

2.8 Self Tuning

Tuning provides several advantages such as it improves performance score of a predictor, changes conclusion on what learners are better than others and decides on what factor are most important. It has proven to improve performance although it is not easy and relatively slow. One such example is Genetic algorithms which involve procedures such as mutation, crossover and selection of best results sets.

3 Key Criteria

3.1 Model Readability

Although random forest is relatively easier to understand most literature around random forest treat it to be a black box. Since it is forest consisting of large number of deep trees it becomes daunting to analyze. Gaining full understanding of full process is infeasible. Even if we are to examine the single tree it is only feasible in the case where there is small depth and less number of trees. A tree depth as small as 10 can have over a thousand nodes. So, in simple words we can say that explanatory model is almost impossible.
Consider a situation where a financial company which has to develop a fraud detection system if some transaction is marked as fraudulent based on numerous factors a financial analyst would not be able to understand why a particular transaction was rejected. Suppose random forest was trained in older data sets and is producing irrelevant outcomes it becomes a mystery.

3.2 Learnability and Repeatability of Results

Theoretically if the number of trees in random forest is not limited then a data set for multiple iteration will produce the same result. However, it not practically possible. The random forest is an ensemble approach. Ensembles are divide and conquer approach. The main intuition behind random forest is that a group of weak learners can come together as strong learners.

4 Critique

The paper [2] we are critiquing is Robust Prediction of Fault-Process by Random Forest. The paper claims to provide a novel methodology at that time to the framework of software quality production. The methodology which is based on Random Forest could be applied to real-world applications in software quality prediction. The data set used were obtained from C, C++ projects all of which were a NASA project. There were totally 15,119 modules. Each data sets contained 21 software metrics, which is shown in Table 1.

Table 1. Five Most Important Attributes in Each Data Set

Data Set	Most Important Attributes
CM1	UniqOp, V, LOBlank, N, E
JM1	V, LOC, E, T, I
KC1	LOC, I, T, E, V
KC2	UniqOpnd, TotalOpnd, iv(G), LOBlank, LOCode
PC1	I, UniqOpnd, LOComment, T, N

A random forest classifier is a collection of tree structures which in turn is a classifier [3]. Classification on a new input vector object is done by examining this object on each tree in forest. Each tree votes and forest select the tree with maximum vote for the given input vector. Each tree is grown as follows

- If number of class in a training data is N, take sample of N classes at random with replacement from the original input. This sample will help in training the tree.

- At every node, k predictors are selected at random from K input variables. Such that k is very small compared to K.
- The best split on k predictors is used to predict the node.
- Each tree is grown to the largest extent possible without pruning.

When tree is trained with picking data with replacement about one third of the data on an average will be left out. Such data is called out of bag data [1].

The paper claims that Random Forest is a good classifier for software quality prediction [2], especially for large scale system for the following reasons:

- It is reported to be consistently accurate relative to the other classification algorithms.
- It runs efficiently on big data sets.
- Its methods are efficient in estimating missing data and maintain accuracy when a chunk of data is missing.
- It tells which attributes are important compared to others.
- It is less prone to noise compared to other methods.

This paper is bit dated and it was published in 2004, while Random Forest was published in 2001. The paper also compares the performance of Random Forest with other statistical methods such as Logistic Regression, Discriminant analysis and claims that random forest is better.

In one project JM1 there are 10,883 modules [2]. Due to noise and outliers unavoidable in such a large data set, other algorithms applied on JM1 have lower prediction accuracy and defect detection rate. Random forest outperforms all the compared methods on JM1 which can be seen in the Figure 1. Random forest has both higher overall accuracy and higher defect detection rate than other methods. Dominance of Random forest is statistically significant which is also shown in the figure.

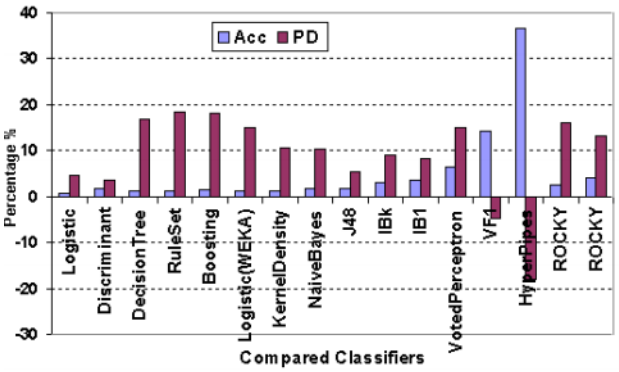

Figure 1. Improvement of random forest predictions over other methods on JM1

Similarly results can be seen in another project PC1 which can be seen in the Figure 2. Also, Random forest outperformed other models.

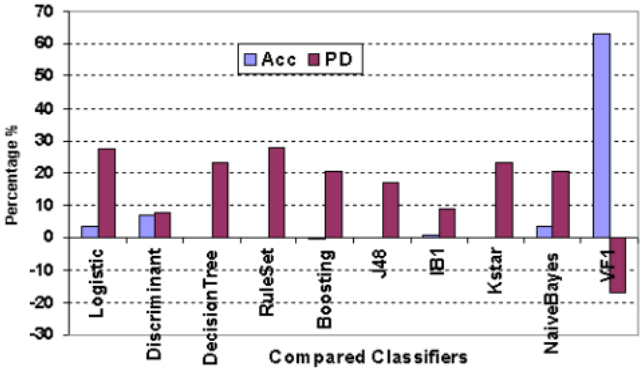

Figure 2. Improvement of random forest predictions over other methods on PC1

In a study in the paper *A systematic review of machine learning techniques for software fault prediction* one of the research question the authors were trying to find is which ML technique was used predominantly is software fault prediction. Top three of them were Decision trees (DT), Bayesian learners (BL) and Ensemble learners (EL) aka Random Forest [1]. The study performed in this paper from January 1991 to October 2013 in the literature that used machine learning techniques Random Forest(EL) was not in the top half as shown in the Table 2. Given this outcome it is not valid to make strong claims made in the earlier paper about Random Forest.

Table 2. Distribution of studies across ML techniques based on classification.

Method	# of Studies	Percent
DT	31	47.7
NN	17	26.16
SVM	18	27.7
BL	31	47.7
EL	12	18.47
EA	8	12.31
RBL	5	7.7
Misc.	16	24.62

So based on these things we are planning to implement a decision tree which would have better readability with lesser depth so that readability of the model is high and there is no randomness.

5 Review

In summary, we reviewed the random forest approach to for bug prediction. We have seen many anomalies in the claims which doesn't comply with the results seen in the usage of ML algorithms over the years. We want to pick up the best learner to be shown in this systematic review paper i.e decision tree.

6 Planning

We evaluated random forest and decision tree from the performance and method readability point of view. As per the paper [9] by Ghotra et al. various algorithms are ranked according to double Scott-Knott test on promise corpus which can be seen in the table 3. We seen that performance of Random Forest is placed in the top row for its high performance and median rank of 1.7. A much readable learner Decision Tree (J48) is ranked 3rd. This shows that there is trade off with respect to performance and readability. Although we feel that decision tree would be much better than random forest in terms of readability in terms of performance there needs to be some improvement to choose it relative to the other leaners. From literature survey [1] we saw however that decision tree is still the most favorite learning algorithm as most used over last 15 years. As told before we want to start with evaluating this implementation of decision tree for predicting software defects keep in mind to improve readability and learnability of the model. Keeping in mind that this has potential to be complicated, so another constraint is to make depth relatively small to make sure that tree doesn't become too complex to be analyzed.
Our plan is to extrapolate on this information and modify decision tree by using discretizer or other techniques to improve its performance.

Table 3. The studied techniques ranked according to the double Scott-Knott test on the promise corpus.

Overall Rank	Classification Technique	Median Rank	Average Rank	Standard Deviation
1	Rsub+J48, SL, Rsub+SL, Bag+SL, LMT, RF+SL, RF+J48, Bag+LMT, Rsub+LMT, and RF+LMT	1.7	1.63	0.33
2	RBFs, Bag+J48, Ad+SL, KNN, RF+NB, Ad+LMT, NB, Rsub+NB, and Bag+NB	2.8	2.84	0.41
3	Ripper, EM, J48, Ad+NB, Bag+SMO, Ad+J48, Ad+SMO, and K-means	5.1	5.13	0.46
4	RF+SMO, Ridor, SMO,and Rsub+SMO	6.5	6.45	0.25

.

For evaluating the above assumption, we are going to use XERCES data set. As our preliminary analysis we ran this data set in R for random forest and decision tree. Dataset was sampled evenly with 75/25 % split. Package "randomForest" from Random Forest library was used and for decision tree "rpart" was used to run our tests.
We can see from the visualization of data set that the class zero is the dominating class and there are totally 62 bug class which can be seen in the Figure 3.

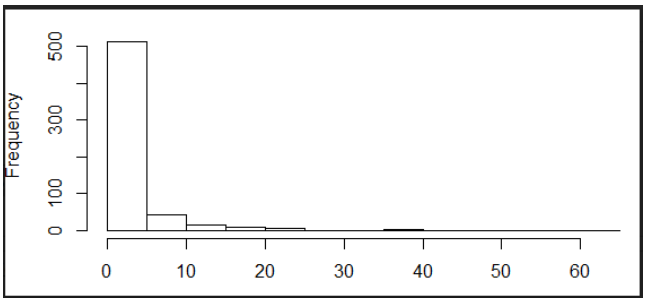

Figure 3. Histogram of bug class in the XERCES data set.

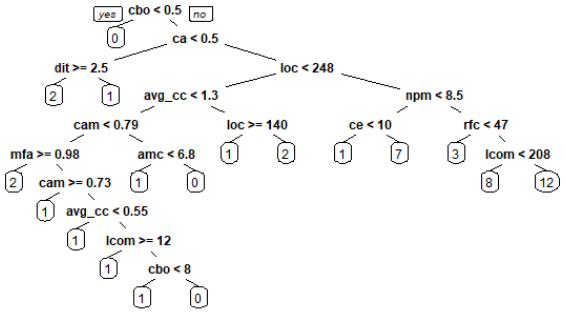

Figure 4. Split from the decision tree that was obtained after running decision tree from R-part.

The accuracy we obtained for decision tree for XERCES data without using any parameter tuning and using all other features was 58 percent whereas for Random Forest it was 64 percent. We can see that these two are comparable thought we have to accept that we have not used any kind of preprocessing or validated data set for any kind of noise.
In the next phase of the project we would be exploring many treatment, discretization techniques which could possibly help us improve and tune the performance of decision tree.

References

[1] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction",Applied Soft Computing, vol. 27, (2015), pp. 504-518.
[2] Guo, L., Ma, Y., Cukic, B., Singh, H., 2004. Robust prediction of fault-proneness by random forests. In: 15th International Symposium on Software Reliability Engineering, pp. 417–428
[3] Breiman. Random forests. Machine Learning, 45(1): 5–32, 2001.
[4] Tim Menzies txt.github.io/fss17/.
[5] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," Software Engineering, IEEE Transactions on, vol. 31,

[6] P. Knab, M. Pinzger, and A. Bernstein, “Predicting defect densities in source code files with decision tree learners,” in Proceedings of the 2006 international workshop on Mining software repositories. ACM, 2006, pp. 119–125
[7] P. S. Sandhu, M. Prashar, P. Bassi, and A. Bisht, “A model for estimation of efforts in development of software systems,” in World Academy of Science, Engineering and Technology, vol. 56, pp. 148– 152, 2009
[8] M. Daud and D. Corne. Human Readable Rule Induction In Medical Data Mining: A Survey Of Existing Algorithms. In Proceedings of the WSEAS European Computing Conference, 2007
[9] B. Ghotra, S. McIntosh, and A. E. Hassan, “Revisiting the impact of classification techniques on the performance of defect prediction models,” in Proc. of the 37th Int’l Conf. on Software Engineering (ICSE), ser. ICSE ’15, 2015, pp. 789–800.