Software Bug Prediction using Decision Tree to Achieve Better Readability and Evaluating its Performance.

Raghavendra Nayak Muddur
North Carolina State University
Raleigh, NC 27695
United States
rmuddur@ncsu.edu

Sunil Narasimhamurthy
North Carolina State University
Raleigh, NC 27695
United States
snarasi5@ncsu.edu

Abstract

There is no software that has zero bugs. Given the enormous 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 and fixing of all these bugs. So, there are data mining based solutions which can be helpful in identification, classification, and fault prediction of these defects. For these solutions readability is an important criterion. On running static analysis tools, we get to know the reason on why it is defective. While using data mining techniques to predict faults a developer would like to know why it is classified the way it is, hence a readable model is more preferred. In our research, we are building a readable decision tree. We are using automatic configuration and automatic feature extraction to reduce overhead. Then we are comparing the experimental results with another learner while applying same feature extraction and parameter tuning techniques.

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

1. Research questions

RQ1 Readability: *How readable are decision trees generated with this experiment designed in this paper?*

To answer this question, we used ten dataset Lucene, velocity, poi, xalan, log4j, prop, ivy, camel, xerces, jedit. Maximum depth and number of leaf nodes where restricted, result was we got much smaller tree than non-tuned decision tree. In most case, number of nodes in tree was less than 10.

RQ2 Effectiveness: *What is the performance of readable decision tree in comparison with non-tuned decision tree?*

Readable decision tree must be effective, by which their performance (F1 score or precision or recall) has to be similar or better than non-tuned model. We have used Recursive feature extraction for extracting key parameters, Grid search to tune parameters. Readable decision trees were indeed as effective or better than original decision tree.

RQ3 Contrasting result with Random forest: *How does our readable decision tree compare with same experiment with Random forest?*

Random forest has been consistently good at defect prediction. We wanted to replicate the same steps of feature extraction and grid search for random forest and compare its performance with decision tree. Key idea here was to measure tradeoff. Same experiment setup while improved performance of non-tuned random forest did not give significantly better result than decision tree learners.

In summary contributions of this paper are

- Readable decision trees can be built with parameter tuning, feature extraction which out performs non-tuned decision tree in many cases. Effect of feature extraction and grid search in Random forest performance.

2. Motivation

Readability 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 chosen by trial and error method. Other examples of algorithms with good readability are Decision tree, Fast frugal tree etc. Not all learners with good readability are useful.

Readability is very important in many application domains. Consider a one financial transaction was marked fraudulent, If the transaction was genuine and the customer calls up the customer care. Now customer demands explanation from the financial analyst asking why the particular transaction was marked as fraudulent. If non-readable algorithm was used then there is no way for financial analyst to explain to customer why the transaction was marked fraudulent. Another case where readability could be very important is in medical equipment. If a medical equipment with learner either shows positive of negative, should be able to explain why the particular class was chosen. This is very important since it would allow doctor or medical staff to treat patient appropriately.

The paper we are critiqued for essay 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. The paper claims that Random Forest is a good classifier for software quality prediction. 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. This paper motivated us to build readable decision tree and compare its performance. We did not use NASA dataset instead we used promise dataset. We used ten datasets in our experiment.

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

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

3. Experimental Design

Our experimental design we follow two procedures.

- Analysis of untuned models where we use default model parameter values and original dataset.
- Analysis of tuned models where we use optimized model parameter values and the dataset modified to retain significant attributes and handle class imbalance.

3.1 Analysis of untuned models

In this procedure, the dataset is not modified and we retain the entire set of attributes. Models are trained on (k-1) versions of the dataset and tested on the kth version of the dataset.

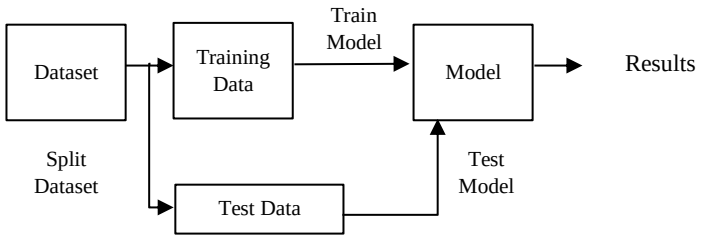

Figure 1: Experiment methodology

Dataset	Training Version	Training Observations	Defective	Testing Version	Testing Observations	Defective
POI	1.5, 2.0, 2.5	936	426 (45%)	3.0	442	281 (63%)
Camel	1.2, 1.4	1480	361 (24%)	1.6	965	188 (19%)
Log4J	1.0,1.1	244	71(29%)	1.2	205	189 (92%)
Lucene	2.0, 2.2	442	235 (53%)	2.4	340	203 (59%)
Synapse	1.0, 1.1	379	76 (20%)	1.2	256	86 (33%)
Velocity	1.4,1.5	410	289 (70%)	1.6	229	78 (34%)
Xalan	2.4, 2.5, 2.6	2411	908 (37%)	2.7	909	898 (98%)
Jedit	4.1, 4.2	679	127 (18%)	4.3	492	11 (2%)
Ivy	1.1, 1.4	352	79 (31%)	2.0	352	40 (11%)
Prop	2,3,4	42006	4451 (10%)	5	8516	1299(15%)

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

What is readable decision tree?

Using a decision tree automatically will not result in readable models. Things to be considered are maximum depth, maximum leaf nodes. There is no one magic answer for selecting optimal max depth and max leaf nodes. For our experiment, we have considered a tree to be readable if its maximum depth is not more than five and number of leaf nodes are no more than eight.

We see that due to the vast number of attributes, we obtain a highly extensive and detailed decision tree. However, this means

that we obtain branches on features that are not important and do not provide a meaningful contribution while making a decision. Also, this affects readability of the decision tree. Hence, it makes sense to reduce our feature set while maintaining performance and improving readability.

3.2 Analysis of tuned models

As we mentioned in the previous section, a large feature set can lead to a complex decision tree which may not always improve performance and affects readability.

In order to achieve this, we can do the following.

3.2.1 SMOTE Analysis

SMOTE is performed on the dataset to handle high class imbalance. It uses an over-sampling approach where the minority class is over-sampled by creating artificial value vs. replacement. A simple implementation of SMOTE works as follows.

1. Select a sample from the feature set and identify k-nearest neighbors.
2. To create an artificial data point, find the distance between the current sample and one of the k-nearest neighbors. Multiply this with a value between 0 and 1 to obtain the new data point.

This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general. In this experiment, we perform SMOTE only on the training dataset.

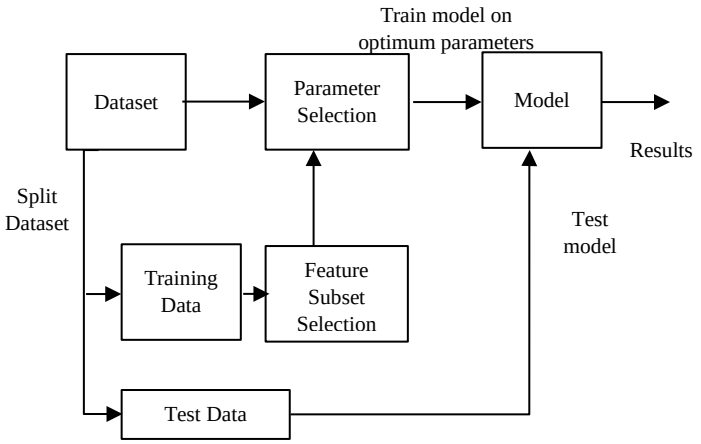

Figure 2: Preprocessing technique

3.2.2 Feature Subset Selection

We can select a subset of features using an iterative feature selection procedure. In this procedure, we iteratively select a subset of features and perform stratified k-fold cross validation to check performance. Stratified k-fold cross validation ensures that each fold contains roughly the same proportions of the two types of class labels. This process of stratification ensures each fold is a good representative of the whole dataset.

3.2.3 Parameter Tuning

We perform parameter tuning to obtain the most optimum model performance after obtaining our modified dataset. This is achieved by choosing different parameter values using grid-search, and performing stratified k-fold cross validation. After covering all combinations of parameter values from performing grid search, we select the model with best performance.

Parameters	Value
max_depth	[3, 4]
max_leaf_nodes	range (4,8,1)
min_samples_split	range (3,25,1)
criterion	["gini", "entropy"]

Table 3: Decision Tree Parameter Training

Parameters	Value
n_estimators	range (50, 150, 10)
max_features	["auto", "sqrt", "log2"]
min_samples_split	range (2,20,1)

Table 4: Random Forest Parameter Training

We can evaluate our model performance using multiple metrics. This is best explained using a Confusion Matrix.

	PREDICTED CLASS		
ACTUAL CLASS		Class = Yes	Class = No
	Class = Yes	TP	FN
	Class = No	FP	TN

Table 5: Confusion Matrix

A confusion matrix (or error matrix) is a special kind of contingency table with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions. The matrix represents the instances in a predicted class while each column represents the instances in an actual class. This helps us visualize the performance of the model. Some performance metrics that can be obtained from a confusion matrix are listed below.

- **True Positives (TP)**: These are cases in which we predicted yes, and the actual class is yes.
- **True Negatives (TN)**: These are cases in which we predicted no, and the actual class is no.
- **False Positives (FP)**: We predicted yes, but the actual class is no. (Also known as a "Type I error.")
- **False Negatives (FN)**: We predicted no, but the actual class is yes. (Also known as a "Type II error.")

- **Accuracy**: How often is the classifier accurate in its classification?
  Accuracy = (TP + TN)/(TP+TN+FP+FN)

- **True Positive Rate (TPR)**: Also known as sensitivity or recall. Tells us how often does is predict yes when the class is actually yes.
  TPR = TP/(TP+FN)

- **True Negative Rate (TNR)**: Also known as specificity. Tells us how often does is predict no when the class is actually no.
  TNR = TN/(TP+FN)

- **Precision**: When it predicts yes, how often is it correct?
  Precision = TP/(TP+FP)

- **F-Measure**: Accesses the trade-off between precision and recall.
  F-Measure = (2*P*R)/(P+R)

3.2 Experiment Infrastructure

We ran our experiments on compute optimized Amazon EC2 C5.9xlarge instance which has 36 cores. We chose this instance type so that we can run our experiments in parallel and get results relatively faster. The cost of running this instance in US-East region is $1.53 per Hour and total cost came out to be $4.59 for 3 Hours of usage.

Our purpose for using a cloud infrastructure service is to take advantage of faster computation speed and a reasonable cost. This in contrast to using a local machine which puts a strain on local resources and affect speed and efficiency.

4. Results

RQ1 Readability: *How readable are decision trees generated with this experiment designed in this paper?*
After running our experiment, we obtained the decision trees with max depth of no more than five and number of leaves no more than eight.

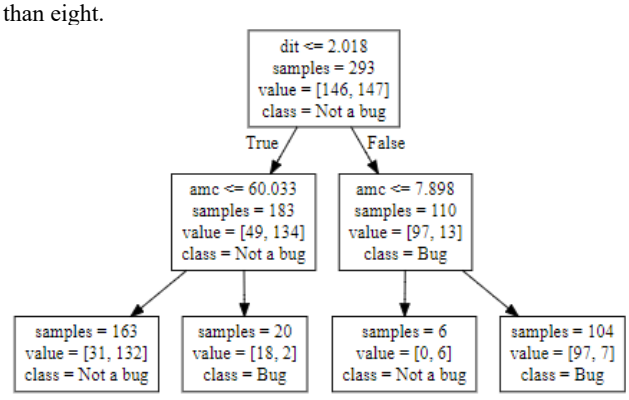

Figure 3: Decision tree for tuned Velocity dataset

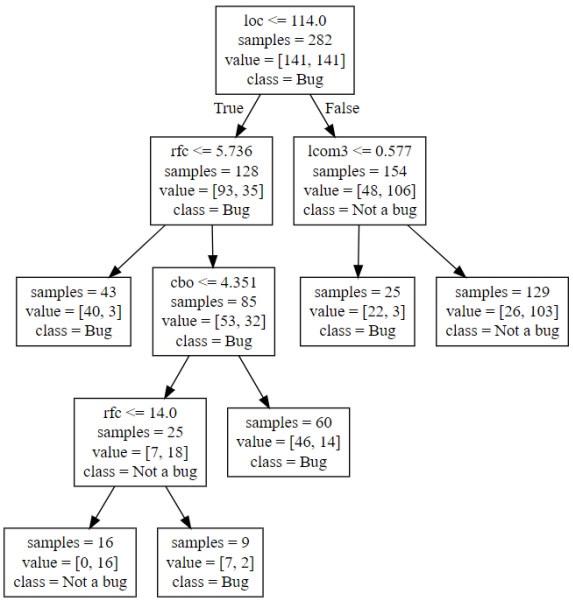

Figure 4: Decision tree for tuned POI dataset

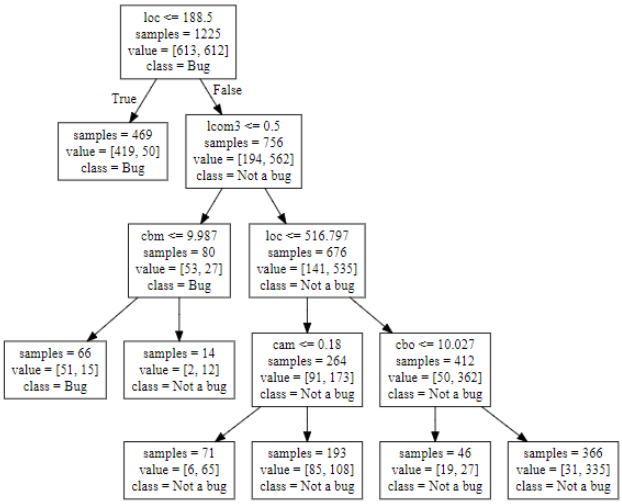

Figure 5: Decision tree for tuned Xalan dataset

The above figures are obtained for the data sets Xalan, POI and Velocity. Similar results were obtained for other datasets as well. We can see that these trees are much smaller and readable.

To achieve these readable decision trees, we had made use of recursive feature selection to select the optimal features and gird search to get optimum parameters. In our AWS C5.x9large EC2 instance feature selection for running decision tree always completed in 0 seconds except for Prop data set which took 7 seconds since it has 42,006 observations.

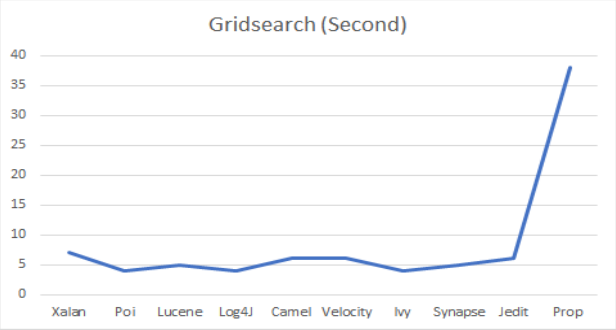

Figure 6: Time taken for running Grid Search after feature selection for each data set

From Figure 6 we can see that time it took to run Grid search for each of the data sets on an average is around 5 seconds except for Prop which took around 38 seconds for the reason explained in the previous paragraph.

RQ2 Effectiveness: *What is the performance of readable decision tree in comparison with non-tuned decision tree?*

The previous research question was concerned only about the readability and did not consider anything about the performance. We have seen that performance is the main criteria to consider the validity of any model. Our key measure in this RQ is to compare how our readable tuned Decision tree fair against the untuned Decision tree.

Precision:

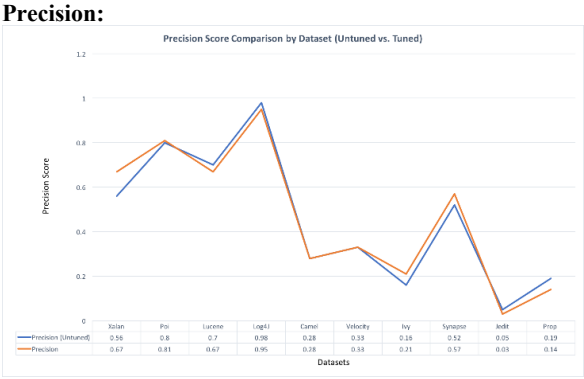

Figure 7: Comparison of precision obtained between tuned and untuned decision tree

From the above Figure 7 we can see that the tuned Decision tree's precision is very much comparable with the precision of untuned Decision tree. There is not much variance.

Recall:

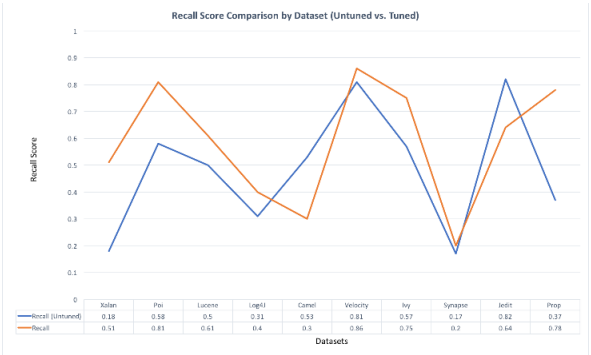

Figure 8: Comparison of recall between tuned and untuned decision tree

The recall for tuned Decision tree performed significantly better compared to untuned one in Xalan recall increased from 0.18 to 0.51 and in Ivy the recall increased from 0.57 to 0.75. Except for jedit and camel where untuned performed slightly better. All the other datasets were very close with tuned performing marginally better.

F1 score:

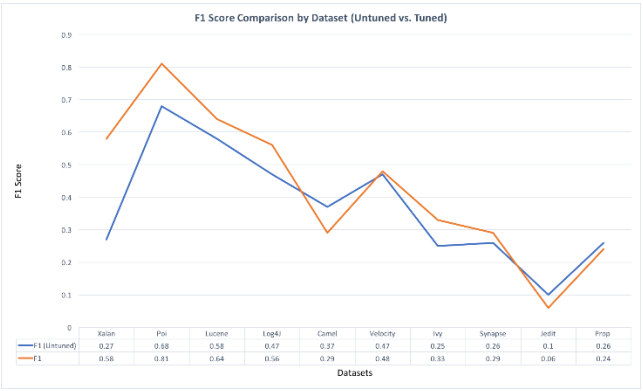

Figure 9: Comparison of F1 score between tuned & untuned decision tree

As we can from the above Figure 9 the tuned Decision tree perform better than untuned except for camel and jedit.

Accuracy:

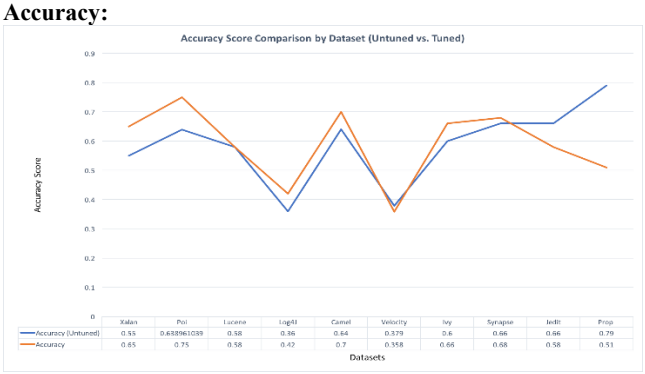

Figure 10: Comparison of accuracy between tuned & untuned decision tree

Accuracy for tuned Decision tree performed significantly better except for camel and jedit.

We can say from precision, recall, f1 score and accuracy that the tuned readable Decision tree has performed better in many cases and not too far behind untuned Decision tree. We can conclude that a readable Decision tree can be reliable without much hit in the performance.

RQ3 Contrasting result with Random forest: *How does our readable decision tree compare with same experiment with Random forest?*

Decision tree not always provides the best performance, and learners such as Random forest perform far more better in terms of accuracy and F1 score. So, in experiment we ran these evaluations and compared the results obtained from Decision tree and Random forest.

We performed the similar experiments where we ran untuned Random forest and recorded precision, recall, f1 score and accuracy. Then we used recursive feature extraction and grid search to tune the Random forest.

Precision:

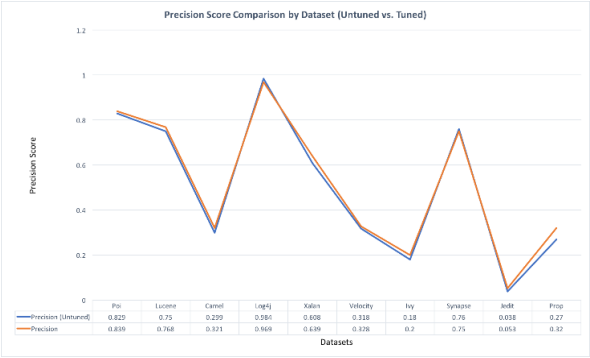

Figure 11: Comparison of precision between tuned & untuned Random forest

We can see from Figure 11 that precision similar for tuned and untuned experiment for Random forest.

Recall:

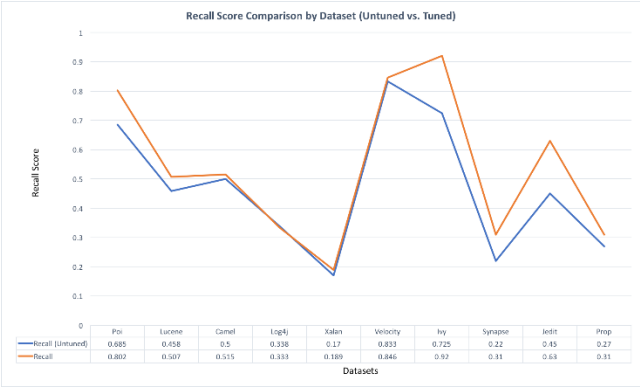

Figure 12: Comparison of recall between tuned & untuned Random forest

We can see from Figure 12 that recall for tuned Random forest was always better. We could see that recall has improved greatly in poi, ivy and jedit data sets. This is consistent with the Random forest result in the paper *Less in More: Minimizing Code Reorganization using XTREE, Menzies et al*. [12].

F1 score:

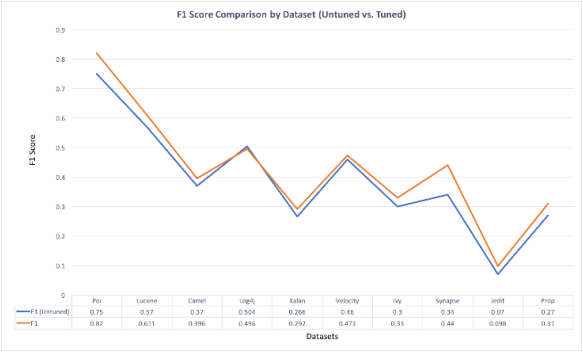

Figure 13: Comparison of f1 score between tuned & untuned Random forest

We can see from Figure 13 that F1 score for tuned Random forest was always better.

Accuracy:

Figure 14: Comparison of accuracy between tuned & untuned Random forest

Similarly, accuracy for tuned Random forest seems to be better.

Grid Search and Feature selection timings:

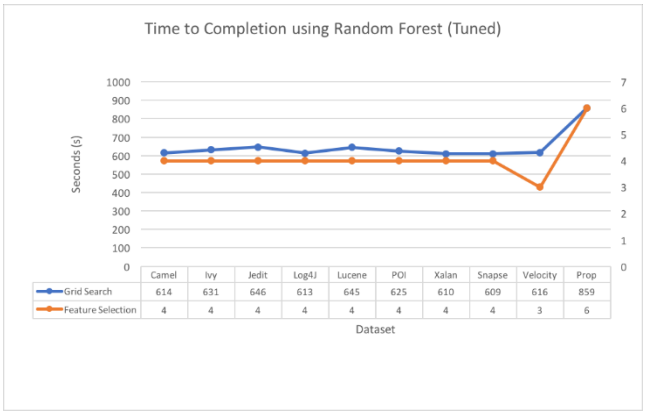

Figure 15: Comparison of run time for Grid search and Feature selection for Random Forest.

From the Figure 15 we can see that time it takes to run Grid search is around 600 seconds except for poi data set which took 859 seconds. It would have been difficult to run these experiments on our local system if not for modern public could infrastructure. Here as explained before we used 36 core CPU and ran took advantage of parallel processing.

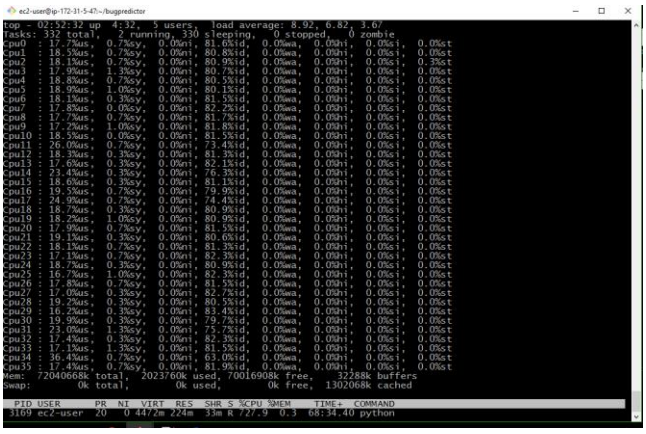

Figure 16: CPU usage for running Grid search for Random forest.

From Figure 16 we can see that running Grid Search utilized all 36 cores.

Decision Tree vs Random Forest:
From same experiment performed for Decision Tree and Random forest we see that there is similarities in their performance.

Precision:

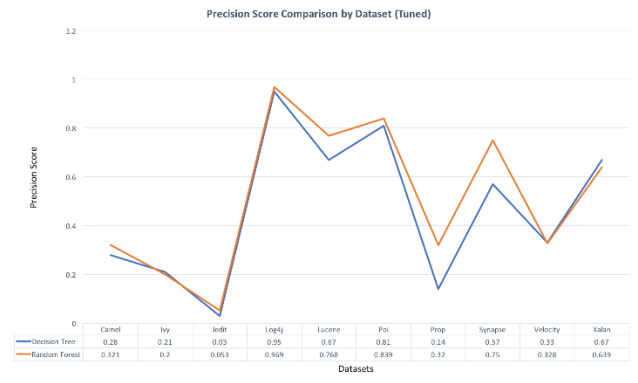

Figure 16: DT vs RF precision.

Recall:

Figure 17: DT vs RF recall.

F1 score:

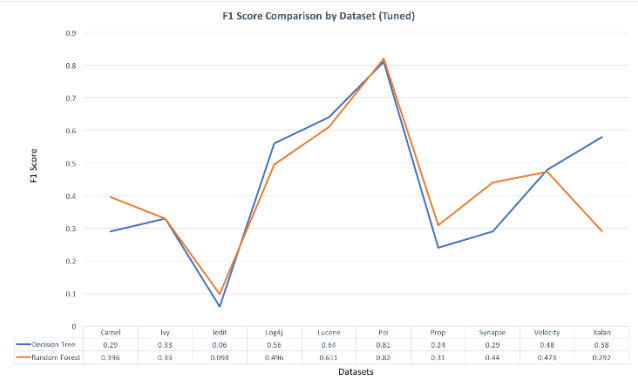

Figure 18: DT vs RF F1 score.

Accuracy:

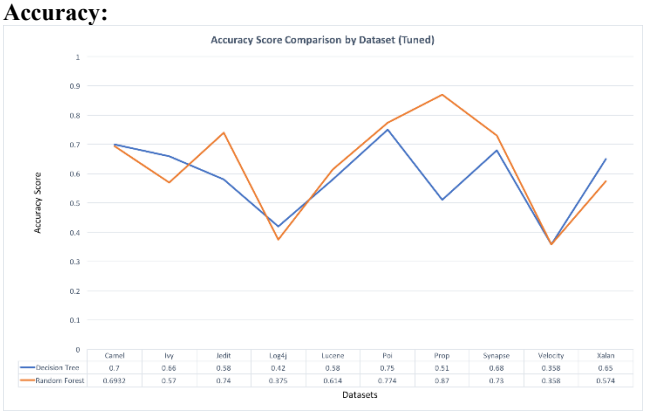

Figure 19: DT vs RF F1 accuracy.

From Figure 16, 17, 18 and 19 we see that except for accuracy the performance is close match between Decision Tree and Random Forest. We can attribute this to the feature selection. Fine tuning to get better result requires further investigation.

5. Threats to Validity

The work done in this project aims to present an optimum data mining process by 'automatically' selecting the best feature set and model parameters for near ideal performance. However, this methodology comes with challenges. While obtaining the dataset is the first step towards data modeling, it is sometimes necessary to run the data through a few pre-processing steps depending on the composition of the dataset. Here is a summary of some of these steps.

- **Inaccurate Data (Missing Data) -** There are many reasons for missing data such as data is not continuously collected, a mistake in data entry, technical problems, etc. For a categorical column with meaningful missing data, we can simply create a new category called "missing", "None" or similar and then handle the categorical feature in the usual way. For missing data where the lack of information carries no information, we need to pre-process the data and replace missing values by guessing the true value by a concept called 0imputation.
- **Noisy Data (Erroneous Data and Outliers) -** The reasons for the existence of noisy data could be a technological problem of gadget that gathers data, a human mistake during data entry and much more.
- **Inconsistent Data -** The presence of inconsistencies is due to the reasons such that existence of duplication within data, human data entry, containing mistakes in codes or names, i.e., violation of data constraints and much more.
- **Feature Engineering** – There might be other feature engineering techniques that might work better. Feature extraction methods such as Principle Component Analysis (PCA) or Linear Discriminatory Analysis (LDA) help combine features together instead of discarding them with the chance of losing important information. Feature elimination in Random Forest might not have had any impact in improving its performance.
- **Domain Knowledge** – While model performance and analysis of metrics provide an idea of how well we understand the underlying structure of the data, not every scenario can be dependent on this. Some scenarios require

expert knowledge of the dataset and source of the dataset in order to assess if a high-performance model is actually adding value to the user.

6. Conclusion and Future work

We could build the readable decision tree which has performance comparable to the untuned Decision tree. We saw that when we conducted the same experiment on Random forest had similar results. While the tuned Random forest performance might not have been the better due to recursive feature elimination which might have impacted its performance. As future work this need a further investigation. Impact of recursive feature elimination on decision tree also needs to be investigated.

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.

[10] Predictions from a Fitted Rpart Object
https://stat.ethz.ch/R-manual/R-devel/library/rpart/html/predict.rpart.html

[11] Breiman and Cutler's Random Forests for Classification and Regression. Package 'randomForest' https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[12] Rahul Krishna, Tim Menzies, and Lucas Layman. 2017. Less is more: Minimizing code reorganization using XTREE. Information and Software Technology 88 (2017), 53–66.