RF and FFT on Studying re-opened bugs in open source software

Satish Gurav
North Carolina State University
United States
sjgurav@ncsu.edu

Abhinandan Deshpande
North Carolina State University
United States
aadeshp2@ncsu.edu

Neel Kapadia
North Carolina State University
United States
ntkapadi@ncsu.edu

Abstract

Bug fixing accounts for a large amount of the software maintenance resources. Generally, bugs are reported, fixed, verified and closed. However, in some cases bugs have to be re-opened. Re-opened bugs increase maintenance costs, degrade the overall user-perceived quality of the software and lead to unnecessary rework by busy practitioners. The existing paper is a study on the prediction of re-opened bugs through a case study on three large open source projects—namely Eclipse, Apache and OpenOffice. We structure our study along four dimensions: (1) the work habits dimension (e.g., the weekday on which the bug was initially closed), (2) the bug report dimension (e.g., the component in which the bug was found) (3) the bug fix dimension (e.g., the amount of time it took to perform the initial fix) and (4) the team dimension (e.g., the experience of the bug fixer). In this paper, however, we will discuss about the effects of parameter tuning on above studied systems. We will discuss and critique about the effects of parameter tuning on the performance of given decision tree, i.e. in this case C 4.5.

Concepts •Data Mining Algorithms •Decision trees •Self-tuning •C 4.5

Keywords Parameter tuning · Bug reports· Re-opened bugs· Open source software

1 Criteria

Evaluation is one of the key points in any data mining process. There are many key factors on which we can evaluate a data earner,

1.1 Model Readability

This is one of the key points for any data mining algorithm. It is very important that the users understand the output and it can only happen when it is readable. If the business users cannot understand the output then it is of very minimal use as it then defeats the whole purpose of data miners. In each of Rexer Analytics' previous Data Miner Surveys, respondents were asked to share their greatest challenges as data miners and explaining data mining to others persisted as a top challenge year after year.

Following are few data mining algorithms that do well in above criterion:

- C 4.5
- Fast Frugal Trees
- Bayes Nets

1.2 Actionable Conclusions

All data mining algorithms predict an output by building succinct model. It is important that whether we can predict the changes that could be made to get different outcomes and this should be possible just by reading the output. Planning is therefore a critical criterion when it comes to user as they are the one seeing the data.

Following are the data mining algorithms with planning,

1. Planners

1.3 Learnability and Repeatability of The Results

It is very important for data mining algorithms to not be CPU or Memory hogging. Operations done by many of the such learners are CPU/GPU hogging and we can no longer rely on Moore's Law [1] to double our computational power every 18 months. [2].
For instance:

- Learning control settings for learners can take days to weeks to years of CPU time [3, 4, 5].
- Lam et al. needed weeks of CPU time to combine deep learning and text mining to localize buggy files from bug reports [6].
- Gu et al. spent 240 hours of GPU time to train a deep learning based method to generate API usage sequences for given natural language query [7].

Following are the algorithms that perform well in this criterion,

- Mini-batch K-means
- Naïve Bayes
- Fast Frugal Trees

1.4 Multi-goal Reasoning

Goal models have been widely used in computer science to represent software requirements, business objectives, and

design qualities. Existing goal modelling techniques, however, have shown limitations of expressiveness and/or tractability in coping with complex real-world problems [8]. In many real-life problems, objectives under consideration conflict with each other, and optimizing a particular solution with respect to a single objective can result in unacceptable results with respect to the other objectives. Therefore it is very critical for algorithms to be multi-objective.
Following are examples of such algorithms,

- Differential Evolution
- Genetic Algorithms

1.5 Anamoly Detection

In data mining, anomaly detection (also outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset[9]. Typically the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are also referred to as outliers, novelties, noise, deviations and exceptions [10]. Three broad categories of anomaly detection techniques exist [10].

- Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set.
- Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherent unbalanced nature of outlier detection).
- Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given normal training data set, and then testing the likelihood of a test instance to be generated by the learnt model.

Following are the few techniques to do just that,

- K-nearest neighbor
- SVM

1.6 Incremental

Incremental learning is a method of machine learning, in which input data is continuously used to extend the existing model's knowledge i.e. to further train the model. It represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time or its size is out of system memory limits. Algorithms that can facilitate incremental learning are known as incremental machine learning algorithms. The aim of incremental learning is for the learning model to adapt to new data without forgetting its existing knowledge, it does not retrain the model. Some incremental learners have built-in some parameter or assumption that controls the relevancy of old data, while others, called stable incremental machine learning algorithms, learn representations of the training data that are not even partially forgotten over time.
Many traditional machine learning algorithms inherently support incremental learning, other algorithms can be adapted to facilitate this. Examples of incremental algorithms include,

- Decision Trees
- Neural Network
- Naïve Bayes

1.7 Sharable

Data sharing is the fundamental step of cross project defect prediction, i.e. the process of using data from one project to predict for defects in another. Prior work on secure data sharing allowed data owners to share their data on a single-party basis for defect prediction via data minimization and obfuscation. However the studied method did not consider that bigger data required the data owner to share more of their data. **D**ata can be mutated to lower the odds of detecting i**n**dividuals within the data and that summarized/mutated data still works well for learning a model as shown with LACE2 by Prof. Menzies [11].

1.8 Context-aware

The knowledge discovery via data mining process (KDDM) is a multiple phase that aims to at a minimum semi-automatically extract new knowledge from existing datasets. For many data mining tasks, the evaluation phase is a challenging one for various reasons. Given this challenge several studies have presented techniques that could be used for the semi-automated evaluation of data mining results. When taken together, these studies suggest the possibility of a common multi-criteria evaluation framework. The use of such a multi-criteria evaluation framework, however, requires that relevant objectives, measures and preference function be identified. This implies that the context of the DM problem is particularly important for the evaluation phase of the KDDM process [12].
Following are the few techniques to do just that,

- Mini-batch k-means
- Nested Mini-batch k-means

1.9 Self Tuning

A self-tuning system is capable of optimizing its own internal running parameters in order to maximize or minimize the fulfilment of an objective function; typically the maximization of efficiency or error minimization. In the field of data mining, choosing control parameters is very important aspect as attributes of a model are highly reliant on these control parameters chosen at the start of data modelling. Evolutionary algorithms like differential evolution and genetic algorithms expand on this area. For example, parameter selection is one of the key criteria in DE.

The choice DE parameters F, CR and NP can have a large impact on optimization performance. Selecting the DE parameters that yield good performance has therefore been the subject of much research. Differential evolution (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found. DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc. DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

2 Key Criteria

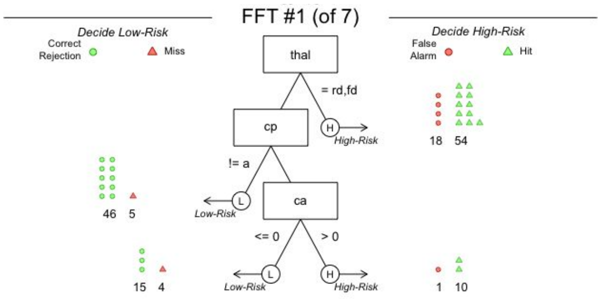
Learnability and Repeatability of The Results

As discussed earlier, methods to increase the efficiency and accuracy of a decision tree or similar model, are using Fast Frugal Trees (FFT) and/or Random Forests.

Fast Frugal Trees

A fast-and-frugal tree (FFT) is a set of hierarchical rules for making decisions based on very little information (usually 4 or fewer). Specifically, it is a decision tree where each node has exactly two branches, where one (or in the cast of the final node, both) branches is an exit branch.

This is an example of FFT:



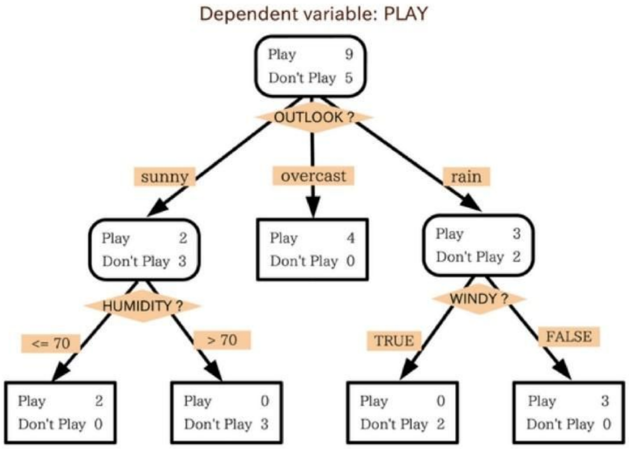
FFT is generally used when the number of features is around 4. Hence, it is an ideal choice for this research topic. Fast Frugal Trees have the following properties:

- Ecologically rational (that is, they exploit structures of information in the environment).
- Founded in evolved psychological capacities such as memory and the perceptual system.
- Fast, frugal, and simple enough to operate effectively when time, knowledge, and computational might are limited.
- Precise enough to be modeled computationally.
- Powerful enough to model both good and poor reasoning.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

An example of Random Forest for the golf dataset is given by:



We will implement FFT and Random Forest for the case study and analyze the performance of these algorithms against that of Decision Tree implementation.

3 Critique

In this paper we are critiquing the decision trees that are being used, which have a very high probability of over-fitting of data and is slower as compared to the FFTs. Hence, in this paper we trying to implement FFT and Random Forest to overcome these drawbacks. As we are only looking for four features in our data, designing FFTs is easy and fast. On the other hand, random forests avoid over-fitting and makes outlier detection easy.

4 Review

In summary, we reviewed the random forest and FFTs for the purpose of examining the reopened bugs and we feel that they will perform much better than decision trees since they are faster and more sensitive to anomalies. We wish to update the previous methods and achieve the same goal but with greater precision.

5 Planning

We plan to implement the system in the following manner:

- Critique the decision tree model, the parameters used to train the model and the dataset used for training.
- Use knowledge of Fast Frugal Trees and Random Forest to design/implement a more efficient way to solve the problem.
- Compare the performance of the models.

Algorithm for FFT:

```r
# Create a fast-and-frugal tree (FFT) predicting heart disease

heart.fft <- FFTrees(formula = diagnosis ~.,

                     data = heart.train,

                     data.test = heart.test,

                     main = "Heart Disease",

                     decision.labels = c("Low-Risk", "High-Risk"))


# Visualize the best training tree applied to the test data

plot(heart.fft, data = "test")
```

Algorithm for Random Forests:

```
1.  N <- root
2.  With node N do
3.  Find the feature F among a random subset of features + threshold value T that split the samples assigned to N into 2 subsets Sleft and Sright so as to maximize the label purity within these subsets
4.  Assign (F,T) to N
5.  If Sleft and Sright too small to be splitted
6.      Attach child leaf nodes Lleft and Lright to N
7.        Tag the leaves with the most present label in Sleft and Sright, resp.
8.  else
9.      Attach child nodes Nleft and Nright to N
10.     Assign Sleft and Sright to them, resp.
11.     Repeat procedure for N = Nleft and N = Nright
```

References

[1] Gordon E Moore and others. 1998. Cramming more components onto integrated circuits. Proc. IEEE 86, 1 (1998), 82–85.

[2] Wei Fu, and Tim Menzies. Easy over Hard: A Case Study on Deep Learning, Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.

[3] Wei Fu, Vivek Nair, and Tim Menzies. 2016. Why is di,erential evolution beŠer than grid search for tuning defect predictors? arXiv preprint arXiv:1609.02613 (2016).

[4] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classication techniques for defect prediction models. In Proceedings of the 38th International Conference on So‡ware Engineering. ACM, 321–332.

[5] Tiantian Wang, Mark Harman, Yue Jia, and Jens Krinke. 2013. Searching for beŠer congurations: a rigorous approach to clone evaluation. In Proceedings of the 2013 9th Joint Meeting on Foundations of So‡ware Engineering. ACM, 455–465.

[6] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2015. Combining deep learning with information retrieval to localize buggy les for bug reports (n). In Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering. IEEE, 476–481.

[7] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of So‡ware Engineering. ACM, 631–642.

[8] Nguyen, C.M., Sebastiani, R., Giorgini, P. et al. Requirements Eng (2016). https://doi.org/10.1007/s00766-016-0263-5

[9] Chandola, V.; Banerjee, A.; Kumar, V. (2009). "Anomaly detection: A survey". ACM Computing Surveys. 41 (3): 1–58. doi:10.1145/1541880.1541882]

[10] Hodge, V. J.; Austin, J. (2004). "A Survey of Outlier Detection Methodologies" (PDF). Artificial Intelligence Review. 22 (2): 85–126. doi:10.1007/s10462-004-4304-y]

[11] Fayola Peters, Tim Menzies, Lucas Layman. LACE2: Better Privacy-Preserving Data Sharing for Cross Project Defect Prediction

[12] A Novel Incremental Data Mining Algorithm Based on FP-growth for Big Data
Hong-Yi Chang; Jia-Chi Lin; Mei-Li
Cheng; Shih-Chang Huang
2016 International Conference on Networking and Network Applications