Fast Frugal Trees vs. CART: Repeatability in defect prediction

Aswin Anil Kumar
North Carolina State University
aanilku@ncsu.edu

Samim Mirhosseini
North Carolina State University
smirhos@ncsu.edu

Sreeram Veluthakkal
North Carolina State University
sveluth@ncsu.edu

Abstract

Introduction: In this essay, we give an overview of Fast Frugal Trees (FFT), state its advantages and explain where we can use FFTs effectively.

Aim: To compare 'repeatability' in terms of computational cost of two popular learning models, namely Fast Frugal Trees (FFT) with Classification and Regression Tree (CART), in defect prediction of software projects.

The main idea of the essay is inspired from the following paper:

Phillips, N. D., Neth, H., Woike, J. K., & Gaissmaier, W. (2017). FFTrees: A toolbox to create, visualize, and evaluate fast-and-frugal decision trees. Judgment and Decision Making, 12(4), 344-368. Retrieved from http://journal.sjdm.org/17/17217/jdm17217.pdf

Keywords Decision Trees, Fast Frugal Trees, Repeatability

1 Criteria

In this section, we explain why each of the criteria below is useful/useless or easy/hard to achieve.

1.1 Model readability

Model readability is useful especially for defending an output against the business user. Using a learner that makes a more readable model is preferred for showing a new idea, although it may not perform the best in some other criteria (No Free Lunch). Model readability is hard to achieve because as only certain learners like decision tree are readable by nature but even those become complex as the size and features of data increases in real world application. Another advantage of model readability is that it opens up our model for critique by the community, thus rapidly improving the model itself [12].

1.2 Actionable conclusions

An actionable conclusion can help Subject Matter Expert (SME) to tune the learner by just looking at the results. It is necessary and useful because the developer who makes the learner might not know the domain or the right parameter tuning for the learner. Also, the user might use the learner in different domains and it is imperative for the user to be able to tune it for a specific domain. The level of complexity in incorporating this criteria to a learner depends on its readability; if it's more readable, it will be more intuitive for the user to make an actionable conclusion.

1.3 Learnability and repeatability of the results

It is important for a learner to use as less amount of RAM, disk and CPU time as possible. In real world applications, we often need to improve and reproduce results to prove accuracy or build improved versions of the learner. Thus, a fast and light learner is useful. This criterion is hard to achieve as faster or lighter learners usually accomplish this by ignoring sets of data or batching. These methodologies are often not considered to create accurate results [5] and it is harder to prove and advocate its usage over a standard, well used, complex learner. This criterion is the main focus of this essay, and we will discuss in more details in key criteria section.

1.4 Multi-goal reasoning

Multi-goal reasoning is when a learner outputs the results based on more than one goal. This is useful and important because most of the real-world applications have multiple goals. This criterion is hard to achieve because making decisions based on multiple goals require scientifically assigning weights to different goals based on domain knowledge, which, is commonly not available with the developer of the learner.

1.5 Anomaly detection

Anomaly detection is to identify an item or event that do not follow expected patterns based on the rest of the dataset [1]. This is useful because the identified items are things like defects or errors in the data. A real-world example of these items is identifying a medical problem or a bank fraud. This criterion is hard to achieve because for an anomaly detector to perform well it needs to be "mature". Maturity here means the learner reaches stability, and it should achieve this fast to be good at anomaly detection.

1.6 Incremental

Incremental updating is essential when using the learner against an infinite stream of data, or if the goal

is to instantly know whether or not the old model still holds with the new condition. If anomaly detection is a goal, it's important to have a learner that is able to incrementally update because an anomaly detector notifies the user something has to change, but an incremental learner helps the user know what to change. The difficulty of achieving this criterion depends on the learner; some learners by nature need the model to be built from scratch when adding new data.

1.7 Sharable

A learner is said to be shareable if it allows to hide or mutate individual's data for privacy and still be able to work well. A shareable learner is useful when the dataset includes sensitive data. For example, medical applications. The difficulty of this criteria depends on the data; if the key fields contain sensitive data it would be hard to hide or mutate the fields without affecting the results.

1.8 Context aware

Context aware means a learner knows that different parts of a dataset will generate different models and behaves differently based on the data settings. Real world applications often require a dynamic model that can adapt to an unpredictable context and this makes this criterion hard to achieve [2]. It is useful to have a data mining framework that is context aware because it could help with the accuracy of output as it is not a one size fit all model.

1.9 Self-tuning

Most of the data mining techniques require manual tuning which makes them usable by only a smaller group of people that are more experienced. A technique with self-tuning capability makes more accurate data mining techniques available to novice data scientists [3]. This criterion is hard to achieve because covering all the patterns of real life data is hard. But in some cases, like defect prediction, it is surprisingly easy as observed by Fu et al [7].

2 Key Criteria

In this paper, we mainly focus on repeatability criteria, why it is important for a learner to satisfy this criterion, and it is hard to achieve.

As stated by Fu et al. [6] "[...] Such long training time limits the ability of (a) a researcher to test the stability of their conclusion via repeated runs with different random seeds; and (b) other researchers to repeat, improve, or even refute that original work.

For example, recently, deep learning was used to find which questions in the Stack Overflow programmer discussion forum can be linked together. Œat deep learning system took 14 hours to execute." Imagine repeating this test for multiple iterations, either to tune the results or verify consistency.

In real world data mining applications, we often need to reproduce results to improve a learner and having a slow learner will make the researcher unproductive. Traditional methods and even methods that process the data in batches can become expensive very fast. It is important to note that we can no longer rely on Moore's law, and the CPUs will not exponentially grow in speed anymore [4], so waiting for more computation power will not help us achieve our goal. The only solution for a doing a complex task is finding a faster way to do it.

3 Critique

When it comes to model readability and actionable conclusions, decision trees have always been in the forefront. But there is a restricted version of decision tree that gives better performance and readability compared to regular decision trees and even some other machine learning algorithms. An FFT is a decision tree where each node has exactly two branches.

Now, keeping up with the No Free Lunch (NFL) theorem, the domain under which FFTs exhibit their brilliance is limited: they are limited to binary classification problems [12].

Decision trees, FFTs included, come under a class of decision algorithms called non-compensatory algorithms. Non- compensatory algorithms use only a subset of all the cues/columns available because the value of these selected cues cannot be outweighed by that of the other cures. i.e. they deliberately ignore information when it knows that the decision it has made with the data available cannot be changed no matter what value the rest of the data has. Compensatory algorithms on the other hand use most or all of the available data because of the way in which they function, the decision made with a certain number of cues could easily be overrun by certain values in other cues.

As noted by Phillips et. al, despite their performance for binary classification task, FFTs are not seen in action as much as some of the other algorithms for the same task. The main reason behind this is that when compared to the more computationally intensive and complex algorithms like SVMs and Random forest, FFTs look inferior and people tend to believe that "more is better" i.e. more the data your learner has access to, the better its prediction will be. This is not always true.

Frugality of FFTs makes people skeptical about using them.

FFTs are extremely resistant to overfitting i.e. they do a better job of predicting unseen data compared to known data. Ironically, in spite of being a huge advantage, this is one of the reasons that FFTs are less popular. While comparing different models, people evaluate them based on how well they fit training data and the FFTs do a relatively poor job at this.

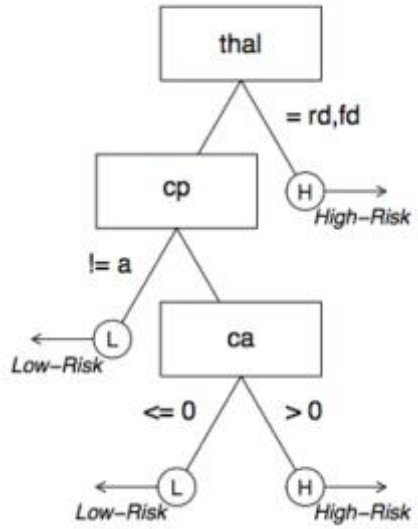

Figure 1. A fast-and-frugal tree (FFT) for classifying patients as either low or high-risk for heart disease [8]

When it comes to model readability, FFT are as simple as it can get. Each node in an FFT answers a single question and based on the answer selects either the left or right branch. Each branch can either be another decision node or an exit branch which results in a final decision. In contrast with regular decision trees, each node in a decision node can have an exit. Because of this FFTs work faster and also makes it easier to understand and use these models. Consider the FFT shown above. This is a tree for deciding whether a patient is at low risk or high risk based on certain tests. We start at the root with the *thal* test. If he/she fails the *thal* test, then we can immediately classify the patient as high risk. Otherwise we go for the *cp* test. If the value is *aa, np* or *ta* we can immediately classify the patient as low risk. Otherwise we go for the *ca* test and then make a final call based on its result. [5]

Contrast this with the approach taken by a regression learner. It will calculate a weighted sum of all the parameters and then check if it's above or below a predetermined threshold to come to the decision. Using an FFT, we can reach at our decision by looking at a single cue in some of the cases. The diagram is easy to understand and implement. [5]

Since the tree uses information in a specific sequential order, it's easy for decision makers in plan for gathering of information[5]. This can be quite useful when information is costly and when it has to be gathered sequentially over time. If we are optimizing for cost, we can have the least expensive tests/decisions at the top levels and the more expensive ones at the bottom. In such a setup, we will end up making a decision with the data that is less expensive to collect alone and go for the more expensive inputs only if we are not able to make a decision with cheaper data alone.

When making a decision on whether to go with FFTs or not, we first need to evaluate the computational and technical resources at our disposal. If the resources are low, then FFTs are the choice.

Consider the cost of obtaining the data to make this decision. If it is very high like in the medical field, we should go with FFTs as we can reach a decision with less data points.

Next consider how important is transparency. In the case of FFTs, it's easy for a decision makers to explain how the model works to people who have very little statistical training. Not only does this make it easier to communicate, it also becomes easy to debug and find out what has gone wrong when your model starts failing in the future due to changes in patterns.

Now consider accuracy. Although FFTs have outperformed several machine learning algorithms in terms of accuracy, keeping with the NFL theorem, FFTs don't guarantee perfect accuracy for all sets of data. Check if FFTs will perform well on the data set that you have.

There are also general classes of data in which FFTs don't perform well. These will be datasets in which each cue/column by itself weekly predicts the outcome but where a function of the cues predicts the outcome with a high level of certainty. E.g.

$$Y = 1.2X_1 + 2.5X_2 + 2.7X_3 + 1.1X_4 + 1.7X_5 + 2.1X_6$$

In such a situation, a regression based model will perform well. Even in such a situation, there is a case in which an FFT will do well. This will be when the weights assigned to the cues are highly skewed. E.g.

$$Y = 100X_1 + 2.5X_2 + 27X_3 + 1.1X_4 + 1.7X_5 + 27X_6$$

Here, an FFT that takes just X_1, X_3, X_6 ignores the rest will perform very well.

Another scenario in which we cannot use FFTs are when it's not possible to justify ignoring information when you make a mistake. For example, a doctor who used a FFT to make a major decision which turned out to be wrong will find it hard to explain why he ignored majority of the symptoms just because his classifier didn't consider them important.

4 Review

In order for an algorithm to be repeatable, it should be able to produce an outcome as fast as possible. FFTs achieve speed by ignoring huge sections of the data and tries to use as less number of decision points as possible. This is what makes them frugal. They are designed in such a way that each node, except for the leaf nodes can result in a final decision, this makes them fast.

This is in stark contrast to a regression learned which needs all the columns in order to come to a decision.

FFTs are a simplified version of decision trees, this makes them highly readable and easy to understand even for someone with no statistical training. A decision tree can get complex when there are too many levels but since FFTs have only two branches at every node, even an FFT with large number of nodes will be easy to understand.

FFTs are so simple that you can describe them in words as seen in Figure 2.

```
FFTrees(diagnosis ~ .,
data = heart.train, my.tree = "If chol > 350,
predict True. If cp != {a} predict False.
If age <= 35, predict False. Otherwise,
predict True").
```

Figure 2. FFT describing in words a prediction chain diagnosing heart diseases [5]

As stated by Phillips et al. "[. . .] Given our results, there is no a priori reason to believe that a more comprehensive or complex algorithm would have been less error-prone than an FFT. [. . .]"

5 Planning

The goal here is to compare 'repeatability' of two popular learning models, namely Fast Frugal Trees (FFT) with Classification and Regression Tree (CART), for defect prediction of software projects. CART is characterized by the fact that it constructs binary trees, namely each internal node has exactly two outgoing edges. The splits are selected using the binary criteria and the obtained tree is pruned by cost–complexity Pruning. CART can handle both numeric and categorical variables and it can easily handle outliers.

Phillips et. al in their paper introduce a package for FFTs and compare its efficiency and prediction accuracy among others against other well-known models. The paper states 'low cost' of FFTs, but that and repeatability is not discussed, which is a key criterion, and is the focus here in our study.

The repeatability of any algorithm is directly tied to the speed in which it can produce an outcome and in the case of FFTs, Phillips et. *al* in their paper use two distinct measures for this: *mcu* and *pci*. MCU is mean number of cues used in arriving at a decision i.e. the number of nodes traversed before we arrive at the final decision. PCI is the percentage of cues or columns ignored.

As seen, fastness and frugality is being measured by the cues required and cues ignored respectively and not by the computational resources taken. Computational limitations are usually in terms of processing power and memory availability and their performance. While cues are a good measure, it may not necessarily translate into faster processing times. Testing stability of a conclusion via 'repeatable' runs is impossible with long running tasks. It is also impossible for other researchers to test the original work. For instance, learning tuning parameters of software cloning detectors as proposed in [11] would take 15 years of CPU time, and it is impossible, economically as well, to repeat it. Thus, here the comparison between the two learning methods will be on 'task run time and other metrics' and baseline the computational requirements in a controlled and similar experimental environment across all runs.

The models will be built using the packages in R for CART and also FFT. The data and the comparison metrics are described below.

The decision problems addressed here is a binary classification tasks for which data is obtained and modified from multiple releases of 15 open source projects will be investigated: Apache Ant, Apache Camel, Ckjm, Apache Forrest, Apache Ivy, JEdit, Apache Log4j, Apache Lucene, PBeans, Apache POI, Apache Synapse, Apache Tomcat, Apache Velocity, Apache Xalan-Java, Apache Xerces. [10] The data is modified to a binary classification problem of presence of bugs.

The metrics suite in the data is as suggested by Chidamber and Kemerer [9] and the data repository is at openscience.us/repo/defect/ck/

References

[1] Hodge, V.J. and Austin, J. (2004) *A survey of outlier detection methodologies. Artificial Intelligence Review*, 22 (2). pp. 85-126.

[2] Singh, Sachin, Pravin Vajirkar, and Yugyung Lee. "*Context-based data mining using ontologies.*" Conceptual Modeling-ER 2003(2003): 405-418.

[3] Daniel Barbara. 2002. *Applications of Data Mining in Computer Security*. Sushil Jajodia (Ed.). Kluwer Academic Publishers, Norwell, MA, USA.

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

[5] Phillips, N. D., Neth, H., Woike, J. K., & Gaissmaier, W. (2017). FFTrees: *A toolbox to create, visualize, and evaluate fast-and-frugal decision trees. Judgment and Decision Making*, 12(4), 344-368. Retrieved from http://journal.sjdm.org/17/17217/jdm17217.pdf

[6] Wei Fu, Tim Menzies. 2017. *Easy over Hard: A Case Study on Deep Learning.* In Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4-8, 2017 (ESEC/FSE'17), 12 pages. DOI: 10.1145/3106237.3106256

[7] Wei Fu, Tim Menzies, Xipeng Shen. *Tuning for Software Analytics: is it Really Necessary?* CoRR, abs/1609.01759.

[8] Nathaniel D. Phillips. 2017. *FFTrees: Fast-and-frugal decision trees.* (August 2017). Retrieved October 23, 2017 from https://cran.r-project.org/web/packages/FFTrees/vignettes/guide.html

[9] Jureczko, M., Spinellis D. 2010. *Using Object-Oriented Design Metrics to Predict Software Defects. In Models and Methods of System Dependability.* Oficyna Wydawnicza Politechniki Wrocławskiej. 69-81. http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html

[10] Marian Jureczko and Lech Madeyski. 2010. *Towards identifying software project clusters with regard to defect prediction.* In Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE '10). ACM, New York, NY, USA, Article 9 , 10 pages.
DOI=http://dx.doi.org.prox.lib.ncsu.edu/10.1145/1868328.1868342

[11] Tiantian Wang, Mark Harman, Yue Jia, and Jens Krinke. 2013. *Searching for better configurations: a rigorous approach to clone evaluation.* In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 455–465.

[12] Menzies, Tim. Foundations of Software Science, txt.github.io/fss17/