Comparison of Tree based Learners in Incremental Dataset of Software Defect Prediction

Md Rayhanur Rahman*
Ph.D. Student
Dept. of CSC
NC State University
Raleigh, NC, USA
mrahman@ncsu.edu

Abstract

In various research fields, data miners are being applied in an intense manner such as in domains of computing, space, business intelligence etc. In software engineering domain, it is also being used extensively. One of the key area where it is being applied is defect prediction. Defect prediction models help software developers to control the quality issues of the software projects. There are a diversified range of data miners utilized to predict defects from the software metrics such as simple regressors and classifiers as well as complex multi objective models. These days, source codes of the software can be obtained from the github and other repositories easily and code metrics can be computed on the fly. This indicates that dataset for defect prediction can grow to a large volume incrementally over time. In such scenario, scalability challenge will appear as the data become larger and larger. In this work, comparison between the offline tree based classifiers and online VFDT classifier to observe the case of classifying defects from large dataset has been performed. From the observation, it is found out that, online classifier behaves more stable as well as robust and produces better performance without consuming more computational resources than the others.

Keywords Defect Prediction, Decision Tree, Random Forest, VFDT, FFT, Online Analysis

1 Introduction

These days, software automation has engulfed all the spheres of our life. Millions of software companies automate new business logic along with replacing legacy systems with its modern descendant. Consequently, software development has become an ongoing process that never stops and hence, there will always be defects in the software sources that need to be eliminated in an constant manner. Fixing these defects is one of the key concern in software quality assurance activities and dedicated human resources spend a significant amount of time in resolving these issues - however if unaddressed, culminates in loss of money, time, consumer satisfaction and in mission critical cases, casualties [8].

Finding and fixing software defects was a manual task decades ago. But these days, data miners are there to help the developers find these defects. These data miners work on the defect prediction models that mainly contain software code quality metrics such as lines of code, cyclometric complexity etc. Based on these data, data miners predict whether a particular software module would contain defects or not.

In order to conduct defect prediction studies [9, 22], use of various classification techniques have been explored to train defect prediction models. For instance, researchers used simple techniques like logistic regression and linear regression to train defect prediction models in earlier studies [1, 10]. However, researchers have kept using more advanced techniques like adaptive regressions, ensemble learning etc as well [2, 6]. Several context sensitive analysis based defect predictors are also explored [15].

Although there have been numerous advanced miners deployed in defect prediction; classification techniques to build defect prediction models have focused solely on the classification performance. However, as the volume of software source codes increases in daily basis, so does the size of defect prediction examples from which the classifiers will predict the defects. Hence, in case of traditional learners consuming the whole data at once, memory and sample size will be a dominant obstacle if those are fed with incrementally large amount of data from time to time. Meanwhile, currently there are many online data-miners are available which are sample size agnostic [4, 5]. Thus exploring the comparison of traditional learners and online data miners have become a priority.

In this paper, we have put three offline classifiers named CART decision tree based classifier, Fast Frugal Tree (FFT), Random forest against online classifier named VFDT [7]. Although, as mentioned above, there are more sophisticated stream based learners available in the literature; VFDT is chosen for the sake of simplicity. Three research questions will be looked into this work in particular:

- **RQ1:** How these four classifiers performs in large defect prediction datasets that will increment over time

*unity id: 200255928

NCSU, FALL 2018, December 7, 2018, Raleigh, NC, USA
2018.

- **RQ2:** How hyper-parameter tuning changes the performance as the data size changes
- **RQ3:** How much computation resources would be used by these four learners

Four real world defect prediction datasets obtained from [13] are used in this work. These datasets contain around $40,000$ examples on an average. From the observation it was found out that, VFDT performs better and more stable and robust manner than the other three as the data size increases. Moreover, the performance differential becomes more prominent among those learners considering data size increase and finding tuning parameters.

Rest of the paper is organized as follows. In section II, baseline criteria will be discussed - upon which the comparison of the learners will be evaluated, In section III and IV, the experiment setup and result analysis will be discussed. Finally, it will be followed by threats to validity analysis and future work scope in the section V and VI.

2 Background and Motivation

In this section, baseline criteria for judging the learners will be first discussed followed by the motivation of this work along with short summery of the learners applied in the corresponding work.

2.1 Baseline Criteria for Data Miners

These days, numerous data miners are being deployed in SE domain. There are also a diversified variation in how those miners tweaked and tuned to fit a particular model. That leaves us with virtually unlimited choices to pick a certain data miner or optimizer to apply in a new problem. However, according to the *no free lunch* theory, these is no single miner which would suit best in every possible models. So, we need to apply commissioning to filter through the possible choices of data miners or optimizers. In that regard, baseline criteria is truly useful. Baseline criteria refers to the important factors, majority of which should be achieved by a learner. It provides key insight to us regarding how effective and efficient the data miners would perform in the real world. Here follows some of the key baseline criteria along with their short description.

2.1.1 Simple and Reasonable

Learners should be simple in a sense that it is easily explicable to the end users. It should also be easy to understand the underlying models, how it works and how to work on to improve further. There are several learners that are not very easy to describe such as Naive Bayes classifiers and Neural Networks. On the other hand, decision tree based models are simple enough to understand and explain to others. However, learners have to be reasonable as well. It should perform well in terms of accuracy and performance. Otherwise being a simple learner producing non-reasonable results does not help the case of data mining activities.

2.1.2 Stable and Robust

Data miners work on examples to build the model and on the basis of that model, make the predictions. Datasets containing loads of examples pose a significant challenge for the miners. Datasets come with lots of unique cases as well context aware information. Even dataset might also be imbalanced while the others might not be complete either. There is also a potential chance of a lot of anomalies hidden in the dataset. All these factors contribute to the fact of being a data miner unstable. This means the learner would produce different types of decisions in case of diversified datasets which is extremely frustrating for the business users. It also prevents further improvement on the data miners. Those also need to be robust throughout most of the cases of different sample size, splits and validation techniques.

2.1.3 Generic

Data miners should be as generic as possible referring to the fact that they should provide a range of possible solution rather than a simple single point one. The benefit of it is to help the end users with a possible widened outlook of the scenario. But if the miners behave too specific or provides only one solution of a certain scenario, it would confuse the end users more. There is also a chance to miss other corner cases and ignore other possible and equally similar solutions which would turn out truly bad in real world scenario.

2.1.4 Replicable

The learner should build model from the dataset and produce outputs fast enough. This ensures that the learner can be invoked iteratively in case of learning, understanding, rebuilding or tweaking. If it takes several hours to produce outputs for a small data, it would become impossible to work on that learner or tune a little bit. At the same time, the learner must also produce the same output every time if is fed the same input. If the outcome of a certain learner is not replicable, then it is impossible to understand, implement and improve the learner.

2.1.5 Goal Aware

These days, all the real world problems do not focus on a single goal. Earlier, optimizers were used to maximize or minimize a value over a set of constraints but now a days, complex situations are there where there is no optimal solution. Hence, data miners being applied in those fields should be goal aware. This means, rather focusing on a single goal, the miner should produce output in such a manner so that the output should reflect overall realization of multiple goals. There might be conflicting goals and often, the correlation of the goals and datasets are quite complex. Nonetheless, the miner should be able to handle such conflicts, complexity and trade-off cases and produce results that provides a satisfaction of all the goals.

2.1.6 Anomaly Aware

It is almost certain and usual that the datasets would contain a percentage of noisy and anomalous examples. Data miners should be able to detect and cancel out those. However, sometimes data miners confuse with the true example as anomalous example and thus discard those or learn something misleading. A good data miner should be able to handle all these cases.

2.1.7 Context Aware

Despite the fact that a dataset should represent a generic scenario of a particular phenomenon, often this is the case that many context specific information are hidden in the dataset. So in the dataset, there might be a region of locality which is almost similar to other generic examples but might vary in one or two attributes and indicates different labels. Those regions are in fact, responsible for difference in decision making process. Thus, a good data miner should be able to find out those context specific local regions and learn the phenomenon protecting the generic learning model intact as well. This baseline is very critical in fields of health, e-commerce etc.

2.1.8 Incremental

Modern days, datasets tend to be large. They also get incremented periodically. So it is in vain if a data miner builds model from a dataset and then can't reuse the already built model if it wants consider the newer examples. So data miner should be able to extend the existing knowledge of model reading the newer examples. It should also be able to run over infinite stream of datasets as well as relearn in case of anomalies.

2.1.9 Shareable

The dataset and the obtained knowledge by the learner should be shareable so that the obtained learning can be applied in different contexts. However, while sharing the dataset, privacy should be a big concern. So miners should be able to hide the actual data and interpret the outcomes at the same time so that sensitive information of the dataset are protected.

2.1.10 Tunable

There are a variety of parameters on every learners that can affect the accuracy, performance and stability of the learners. However, for a large dataset, if it is necessary to find the best set of parameters for the miner, then it would take a hefty amount of time to do so. Hence, a data miner should be able to tune itself so that it can be fit to a certain dataset or certain domain specific models to obtain better results.

In addition, there are several other baselines that deserves to be mentioned:

- Be applicable to mixed qualitative and quantitative data
- Have no parameters within the modeling process that require tuning.
- Be publicly available via a reference implementation and associated environment for execution
- Be computationally cheap in a reasonable sense

2.1.11 Criteria for this Work

All of the aforementioned criteria are more or less critical for any learner. However, set of baselines critical for a particular domain depends on many contexts as well. In some cases, computational performance might be the most critical. In other case, the accuracy might be the most critical one. In the context of this work, the performance of various tree based learners on incremental data will be observed. Hence, for our work, here are the list of critical baseline we will look more closely:

1. **Usefulness and Reasonable:** The output of the miners should be at least better than some random guess. In this work, the miner would predict whether a given example contains bugs or not. The typical evaluation metrics such as precision and recall of the learners would be put into the focus. The examples will be fed into the miners in an incremental manner and it will be observed how those evaluation measure changes over the new datasets that fed into the learners. Also the outputs need to be better than the established learners.
2. **Stable:** Stability is quite important in case of prediction. If the output of the learners produces varying degree of outputs, then the decision making process would be frustrating. In the context of this work, the stability of the prediction of the learners will be probed in terms of incrementally updated set of examples and observed how these learners perform.
3. **Cheap:** The computational resources being used by the learners is another important aspect. Learners might produce great results but if it comes at a price of heavy computational resources such as cpu time and memory, then it might not become that useful in real world scenario. In this work, learners will learn from a range of small datasets to large one. So, information on how much cpu time and memory resources those learner will use to produce results will be gathered.
4. **Robust:** In case of incremental datasets, the learners need to accommodate all the newer examples. However, this might also confuse learners with different examples or noisy examples. So it needs to be seen how robustly the learner can handle all the examples that will be added in the larger datasets.

5. **Streaming:** In case of very big dataset, the learners should be able to stream in the newer examples and fit those in the existing models. Otherwise, the learners need to completely relearn everything in case of the arrival of the newer data. In the context of this work, this is a very critical requirement.
6. **Tuning:** Tuning of the parameters of the learners is another important baseline. In this work, the learners will be fed with increasingly bigger amount of example set and it will be observed how the initial tuning performs for the rest of the data. It will also be observed how the optimal set of tuned parameters changes the performance when newer examples comes into the context.

2.2 Motivation

In the domain of software defect prediction, several learners have been applied which include logistic regression, neural networks, decision tress, ensemble learning, evolutionary algorithms and context sensitive analysis [1–3, 6, 10, 15, 16, 24]. All of these learners load the whole data into primary memory and perform the classification task. However, things have changed rapidly over years. These days, source codes are being committed to the source code repository periodically. The frequency of committing newer code base is ranged from hours and to days. In order to help developers with new potential defects while the new code base is being committed, necessity to relearn the whole defect prediction model is necessary. As the new code is being pushed, software source code metrics which are being used to build defect prediction models need to be recomputed. Hence, the newer examples of software defects would come into play. Meanwhile, assuming that newer version of source code being committed to the repository at least once in a day results into re-computation and relearning the whole thing from the scratch. Traditional decision tree based learners such as Classification and Regression Tree (CART), Random Forest (RF), Fast Frugal Trees (FFT) would need to relearn everything which is very cpu and memory extensive. If, high speed stream based learners would have been used in this scenario, then there would be no need to relearn everything from scratch as the streaming data miners automatically accommodates the newer examples with the existing models. That signifies the motivation of this work. It needs to be observed how streaming learners perform against the offline learners in terms of feeding newer set of examples periodically. Hence, in this work, CART, RF and FFT will be compared against Very Fast Decision Tree (VFDT) which is an online stream based decision tree learner from the aspect of aforementioned baseline criteria.

2.3 Classifiers

In this work, the performance of four decision tree based classifiers will be observed. Here is a short description of each of these classifiers.

2.3.1 Decision Tree

Decision Tree learners use decision tree based models as a predictive tool to perform classification or regression over a set of examples. In this scope of work, only classification trees will be used where the tree will be used to predict the class labels of a given set of examples. Each interior node refers to independent variables and edges refer to the value of that independent variable. Each leaf corresponds to the value of a target variable (or label or class) on the basis of all independent variables lying on the path from root to leaf. Each of these internal nodes are computed recursively from root. First, from the root, the best attribute is chosen on the basis of information gain from that attribute. After selecting the first best attribute, the root will have children based on the attribute value. Then in each of the child nodes, this process will go on recursively. The stopping criteria can be either a threshold of information gain or the depth of the tree. The leaves of the tree contains the probability of each labels associated with the choice of independent attributes from root to the leave. The advantage of decision tree is that it is simple, understandable and interpretable. It can handle both numerical and categorical data and it can be used in multiple classification and regression problem. However, decision trees would suffer from over-fitting issue. Sometimes, the constructed decision model is very complicated as well. It also become unstable sometimes due to the variation in the training dataset. Moreover, finding the optimal decision tree for a particular model is not straightforward process. In the scope of this work, the standard decision tree library from the python scikit-learn package will be used. It uses an optimized version of CART (classification and regression Trees) which is a variant of C4.5 decision tree [21].

2.3.2 Random Forest

Random Forest is an ensemble learning methods where multiple decision trees will be constructed from the training class and during testing, the class will be predicted that is the mode class predicted by all of the decision trees of that random forest. The intuition of random forest is to tackle the problem of over-fitting suffered by the decision trees. In order to build the decision trees during the training time, example data of the training dataset will be randomly chosen along with replacement. During the computation of best splitting criteria, it would do that so from a randomly selected subset of the training example set. As a result, each tree tends to tackle the bias. Moreover, bagging along with random subset selection also ensures that individual trees have low correlation to each other leading to an ensemble that is both of low bias and low variance. In the scope of this work, we use the Random Forest library from scikit-learn package [20].

2.3.3 Fast and Frugal Tree

Fast and Frugal trees is a type of decision tree where a set of hierarchical rules are obtained from a training set. This tree has a depth of four usually and two branches in the each node. Each node represents a splitting attribute associated with a certain value or range. One of the branch denotes the sub-tree which will be used for further training if the condition is satisfied. The another branch is simply a leaf which denotes the cases if the condition is not satisfied. The leaf also contains the class information with associated probability. This type of tree is useful for quick interpretation of the decision models obtained from a training dataset. It is also useful for the dataset where number of key attributes is more or less, equal to the depth of the decision tree. In this work, Fast and Frugal tree has been implemented from the scratch [18] using unsupervised discretization to select the best splitting attribute.

2.3.4 VFDT - Very Fast Decision Tree

Established decision tree learners like ID3, C4.5 or CART assume that all training examples can be stored at once in primary memory. This is the performance bottleneck those suffer while learning from a big set of training example. Disk-based decision tree learners such as SLIQ and SPRINT meanwhile, stores the examples on secondary memory [14, 17]. But it will also be handicapped by the low disk I/O speed and disk space availability. With an aim to solve this problem, VFDT has been proposed in [7] which can learn from extremely large datasets by reading only a small portion of the whole set of examples. It can be achieved by noticing the fact that, in order to find the best attribute to of a given node, only a small subset of the training examples can be sufficient enough. From a stream of examples, the first attribute will be chosen for the root. After that, the succeeding examples will be routed to the appropriate nodes and used to choose the appropriate attributes recursively. Meanwhile, it needs to known exactly how many example is needed to make such decisions. However, by using hoeffding bound [11], this can be calculated.

Assuming a real-valued random variable r ranging R (e.g., for a probability the range is 1, and for an information gain the range is $logc$, where c is the number of classes), n independent observations of this variable is made, and their mean $\bar{r}$ is computed. The Hoeffding bound states that, with probability $1-\delta$, the true mean of the variable is at least $\bar{r}-\epsilon$, where

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{1}$$

In the figure 1, the generic algorithm of the VFDT is given. In this work, VFDT is implemented by python completely from the scratch [18].

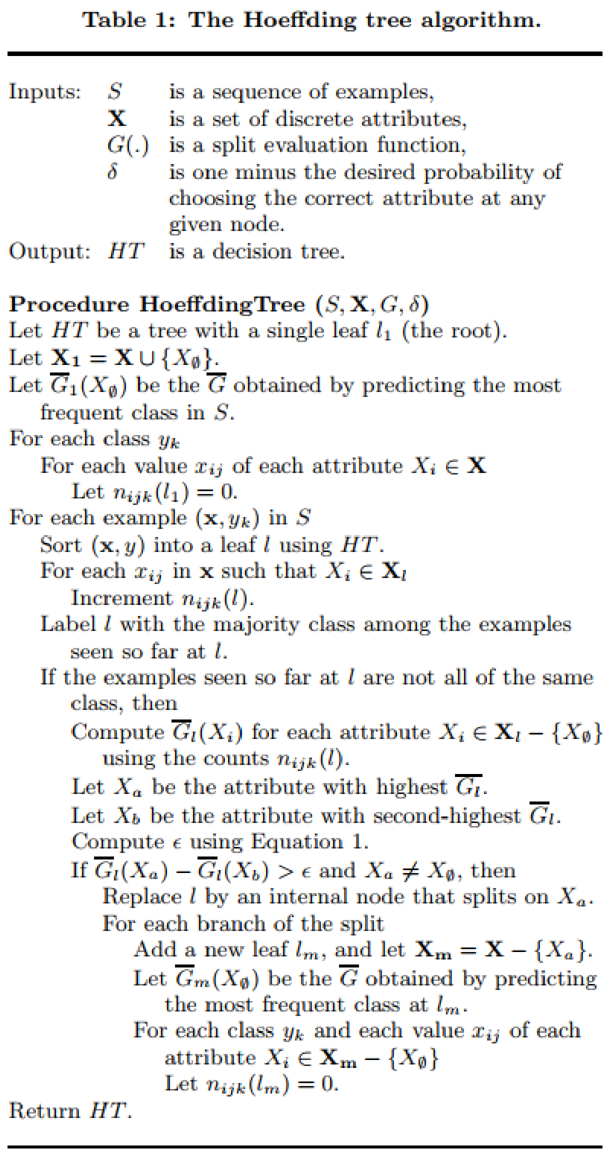


Table 1: The Hoeffding tree algorithm.

```
Inputs:  $S$       is a sequence of examples,
         $\mathbf{X}$       is a set of discrete attributes,
         $G(.)$    is a split evaluation function,
         $\delta$       is one minus the desired probability of
                  choosing the correct attribute at any
                  given node.
Output:  $HT$      is a decision tree.

Procedure HoeffdingTree $(S, \mathbf{X}, G, \delta)$
Let $HT$ be a tree with a single leaf $l_1$ (the root).
Let $\mathbf{X_1} = \mathbf{X} \cup \{X_\emptyset\}$.
Let $\overline{G}_1(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most
    frequent class in $S$.
For each class $y_k$
    For each value $x_{ij}$ of each attribute $X_i \in \mathbf{X}$
        Let $n_{ijk}(l_1) = 0$.
For each example $(\mathbf{x}, y_k)$ in $S$
    Sort $(\mathbf{x}, y)$ into a leaf $l$ using $HT$.
    For each $x_{ij}$ in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
        Increment $n_{ijk}(l)$.
    Label $l$ with the majority class among the examples
        seen so far at $l$.
    If the examples seen so far at $l$ are not all of the same
        class, then
        Compute $\overline{G}_l(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$
            using the counts $n_{ijk}(l)$.
        Let $X_a$ be the attribute with highest $\overline{G}_l$.
        Let $X_b$ be the attribute with second-highest $\overline{G}_l$.
        Compute $\epsilon$ using Equation 1.
        If $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \epsilon$ and $X_a \neq X_\emptyset$, then
            Replace $l$ by an internal node that splits on $X_a$.
            For each branch of the split
                Add a new leaf $l_m$, and let $\mathbf{X_m} = \mathbf{X} - \{X_a\}$.
                Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting
                    the most frequent class at $l_m$.
                For each class $y_k$ and each value $x_{ij}$ of each
                    attribute $X_i \in \mathbf{X_m} - \{X_\emptyset\}$
                    Let $n_{ijk}(l_m) = 0$.
Return $HT$.
```

Figure 1. The VFDT Algorithm [7]

2.4 Tuning Method

As CART, Random Forest and VFDT have several parameters that can pose impact in the classification performance, parameter tuning is necessary before deploying the learners for classifications tasks. In this work, differential evolution method [23] is used for finding the sub-optimal parameters for the classifiers. It finds the sub-optimal solution for a problem by iteratively searching the candidate solution on the basis of a given fitness function. First it generates an initial population of candidate solutions and then generate subsequent set of candidate solutions by applying mutation and crossover methods. After that, all the possible solutions will be tested against a fitness function and best set of solutions will be kept. This process is done iteratively for several pass and final set of best solution will be considered. However, it must be kept in mind that this process does not guarantee that the theoretical optimal solution has been found. In this work, this process has been implemented from the scratch.

3 Experiment Design

In this work, four data miners named CART, FFT, RF and VFDT will be applied on four software defect prediction datasets. These datasets will be fed to the miners incrementally. Then the performance of all four miners based on these datasets will be observed. The experiment setup is described below:

3.1 Dataset

Although, there are so many standard defect prediction datasets, datasets that came from different sequential committed version of the same source code was chosen. It will facilitate experimenting with large set of examples and incremented examples over a period of time. Here is the descriptions of the datasets:

Name	Examples	Attributes
abinit	89303	29
lammps	41130	40
libmesh	24673	40
mdanalysis	11694	37

Table 1. Dataset Description

After choosing the datasets, training and test example set need to be generated. However, stratification and n-fold cross validation are not an option because those examples in the datasets are needed to be fed into the miners in incremental order (i.e. 5%, 10%, 15%... of the data). This is why 80% of the examples for each dataset were randomly and the rest of the data were kept for testing set without breaking the sequence of the data. Thus, 10 different set of training and testing example pairs for each dataset was generated.

3.2 Testing Environment Setup

The work is performed in a traditional laptop PC. Here is the information of the execution environment.

OS Family	Arch Linux
CPU	Intel Core i7 8550U
Memory	16GB 2400MHz
Disk	SSD 240GB
Language & Runtime	Python 3.7
Data Miner Library	Scikit-Learn 0.20

Table 2. Testing Environment Information

3.3 Parameter Tuning

As CART, Random Forest and VFDT have several parameters that can pose impact in the classification performance, parameter tuning is necessary before deploying the learners for classifications tasks. Here is the last of parameters that are needed to be tuned for the classifiers. However, the tuner itself needs some parameters set before using it. In this experiment, iteration, mutation and cross-over rate were kept at 10, 0.8 & 0.7 respectively.

1. **Number of estimators:** This parameter is only applicable to random forest classifier. This is the number of trees that will be used for classification task in the random forest.
2. **Tree depth:** This parameter denotes the maximal depth of the decision tree.
3. **Minimum sample leaves:** This denotes the minimum number of samples required to split an internal node.
4. **Minimum number of examples to split:** This denotes the minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least of this value in each of the left and right branches.
5. **Ties:** When two or more attributes have very similar Gs, potentially many examples will be required to decide between them with high confidence. This will be time consuming. Thus VFDT can optionally decide that there is effectively a tie and split on the current best attribute if $\delta G < \epsilon < \tau$, where τ is a user-specified threshold.
6. **Frequency of Computation of G:** The most critical part of the time cost per example is recomputing G after each example arrives. It is inefficient to calculate G at every occasion. Thus VFDT allows the user to specify a minimum number of new examples n_min that must be accumulated at a leaf before G is recomputed. This effectively reduces the global time spent on G computations by a factor of n_min.

Parameter	Range	Applicable Classifiers
Estimator	(10, 100)	RF
Depth	(2, 10)	CART, RF, VFDT
Splits	(0, 1), (5, 500)	CART, RF, VFDT
Leaves	(0, 1)	CART, RF
τ	(.001, .99)	VFDT
n_min	(5, 500)	VFDT

3.4 Evaluation Criteria

The evaluation criteria are discussed below:

1. **Memory:** It denotes the maximum amount of primary memory used by the python program containing classifiers. In this work, it will be measured in Megabytes.
2. **Time:** It denotes tha amount of time taken by the program to train from the dataset and construct the decision model. In this work, it will be measured in seconds.
3. **Precision:** This is the ratio of all detected bugs that are true positives and detected bugs that are true and false positives.

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{2}$$

4. **Recall:** This is the ratio of all detected bugs that are true positives and detected bugs that are false negatives and true positives.

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{3}$$

5. **False Alarm:** This is the ratio of all detected bugs that are false positives and detected bugs that are true negatives and false positives.

$$false\ alarm = \frac{false\ positive}{false\ positive + true\ negative} \tag{4}$$

It is noteworthy that *accuracy* is not considered as it does not provide a good view of the prediction quality. In case of imbalanced data, it also outputs misleading information.

4 Result Analysis

In this section we will discuss the findings in the light of research questions mentioned in section I.

4.1 RQ1: How these four classifiers performs in large defect prediction datasets that will increment over time

It will be observed how CART, RF, FFT and VFDT performs in terms of precision and recall. Before deploying the learners, parameter tuning was performed and applied the parameters obtained at 25% dataset size for the whole dataset.

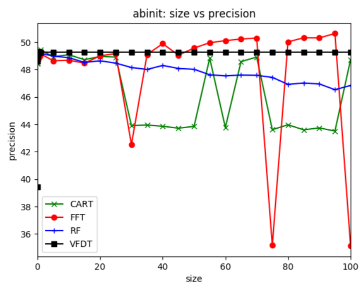


Figure 2. Size vs Precision for abinit dataset

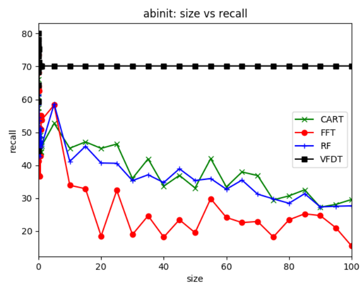


Figure 3. Size vs Recall for abinit dataset

From the figure 2 and 2 it can be seen that, with increase of the examples in abinit dataset, precision score for VFDT does not fluctuate but for the other three learners, it fluctuates, mostly FFT. However, in case of recall score, VFDT performs far better and stable across all of the sizes for the dataset. For the other three learners, the score gets lower and lower when the size increases.

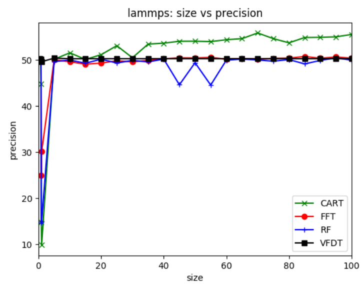


Figure 4. Size vs Precision for lammps dataset

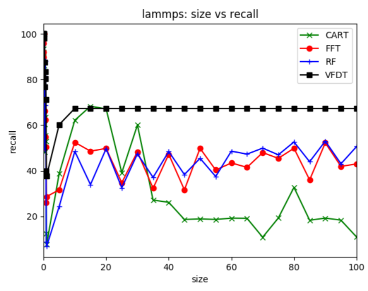


Figure 5. Size vs Recall for lammps dataset

From the figure 4 and 4 it can be seen that, with increase of the examples in lammps dataset, precision score for all the learners are similar and CART is performing slightly better than the others However, in case of recall score, VFDT slightly performs far better and stable across all of the sizes for the dataset. For the other three learners, the score fluctuates back and forth. However, CART performs the worst among the four learners.

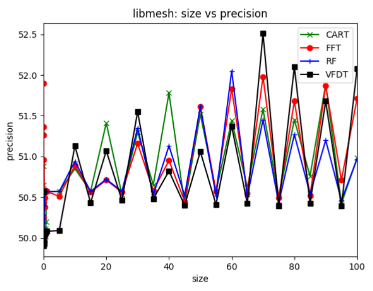


Figure 6. Size vs Precision for libmesh dataset

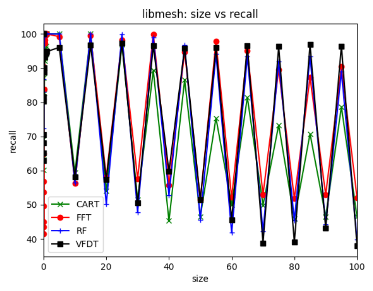


Figure 7. Size vs Recall for libmesh dataset

From the figure 6 and 6 it can be seen that, with increase of the examples in libmesh dataset, precision and recall score for all the learners are similar. However, it also shows that with the increase of dataset size, the recall scores behaves very unstable, for all of the four learners.

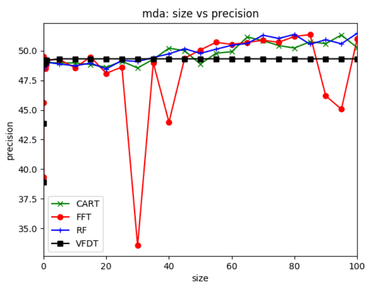


Figure 8. Size vs Precision for mdanalysis dataset

From the figure 8 and 8 it can be seen that, with increase of the examples in mdanalysis dataset, precision score for all the learners are similar and CART as well as FFT are performing slightly better than the others However, in case of recall score, VFDT performs far better and stable across all of the sizes for the dataset. For the other three learners, the score fluctuates back and forth a ittle bit. However, FFT performs the worst among the four learners.

From the figure 11, it can be seen that false alarm count for all the datasets are higher. It can also be seen that false alarm

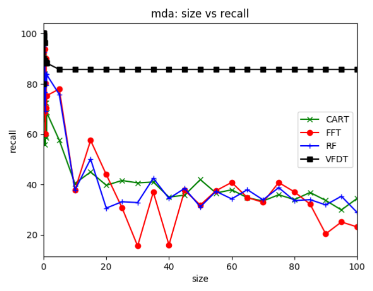
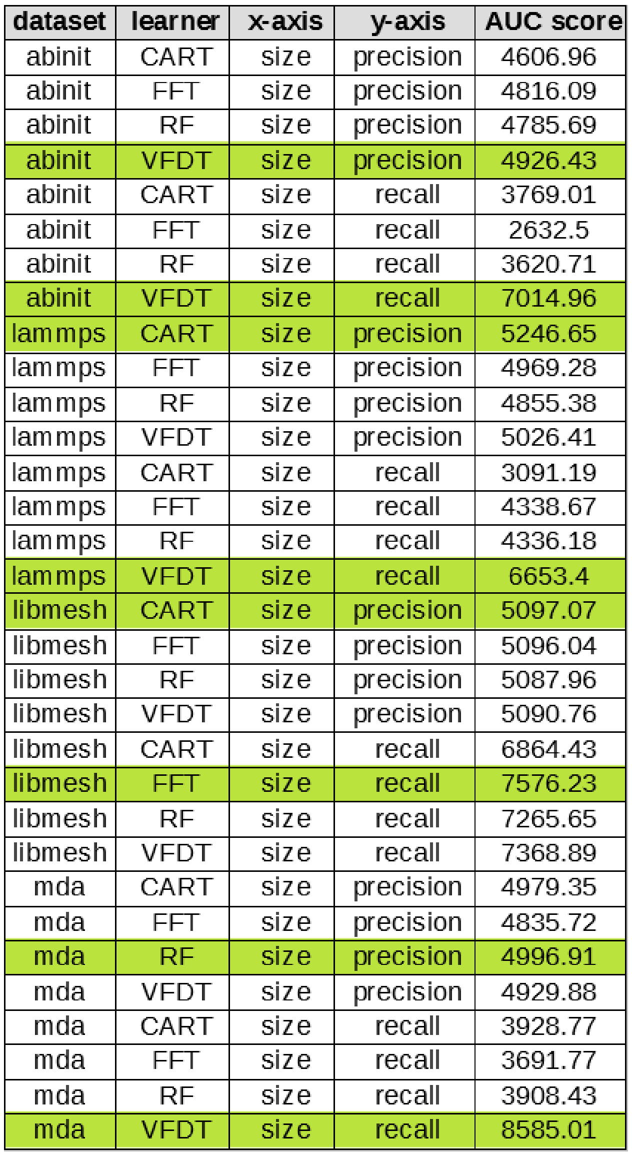


Figure 9. Size vs Recall for mdanalysis dataset

score is similar to recall score across all the datasets. Hence, confusion matrix of the learners were looked into and it was observed that across all size, the number of True Negatives (TN) and False Negatives (FN) are higher than that of True Positives (TP) and False Positives (FP). Moreover, TN and FN are similar. Theoretically, if the precision score is around 50%, then recall score and false alarm score would become similar. This happens because, if precision is around 0.5, then TP will be similar to FP. So, that will mean, the value of $\frac{TP}{TP+FN}$ and $\frac{FP}{FP+FN}$ will be similar. Thus it can be concluded that, for this particular scope recall and false alarm will be similar.

In principle, the key observation from the analysis of four datasets are given below:

- VFDT performs better in recall score across all the dataset size. This can also be seen in the AUC score given in 10
- VFDT achieves slightly a lower precision score than the other three learners. This can also be seen in the AUC score given in 10
- Precision and recall scores of all the three learners except VFDT either goes up or down with the increase of data fed into them. For VFDT, it remains stable. As, VFDT reaches hoeffding bound and grows tree accordingly, with the increase of the data size, its tree structure does not change. Hence, its precision and recall score also stay unchanged.
- As tuning of the learners was performed with 25% of the examples, it is obvious that optimal parameters for VFDT did not change while newer examples came. But with the arrival of newer examples, optimal set of parameters of CART and RF have changes. This is why their precision and recall score dropped.
- the precision score also tells that all the learner performed marginally better than a random guess.
- High rate of false alarm is a genuine concern. It is also associated with the fact of precision score not getting better than the random guess.



dataset	learner	x-axis	y-axis	AUC score
abinit	CART	size	precision	4606.96
abinit	FFT	size	precision	4816.09
abinit	RF	size	precision	4785.69
abinit	VFDT	size	precision	4926.43
abinit	CART	size	recall	3769.01
abinit	FFT	size	recall	2632.5
abinit	RF	size	recall	3620.71
abinit	VFDT	size	recall	7014.96
lammps	CART	size	precision	5246.65
lammps	FFT	size	precision	4969.28
lammps	RF	size	precision	4855.38
lammps	VFDT	size	precision	5026.41
lammps	CART	size	recall	3091.19
lammps	FFT	size	recall	4338.67
lammps	RF	size	recall	4336.18
lammps	VFDT	size	recall	6653.4
libmesh	CART	size	precision	5097.07
libmesh	FFT	size	precision	5096.04
libmesh	RF	size	precision	5087.96
libmesh	VFDT	size	precision	5090.76
libmesh	CART	size	recall	6864.43
libmesh	FFT	size	recall	7576.23
libmesh	RF	size	recall	7265.65
libmesh	VFDT	size	recall	7368.89
mda	CART	size	precision	4979.35
mda	FFT	size	precision	4835.72
mda	RF	size	precision	4996.91
mda	VFDT	size	precision	4929.88
mda	CART	size	recall	3928.77
mda	FFT	size	recall	3691.77
mda	RF	size	recall	3908.43
mda	VFDT	size	recall	8585.01

Figure 10. AUC of size-recall for all datasets

4.2 How hyper-parameter tuning changes the performance as the data size changes

From the analysis in previous section, it is obvious that when the learner faces substantial amount of newer examples, their prediction performance goes down. Consequently, it remains to be seen whether further parameter tuning improves their performance or not. Moreover, it was understood that, if the false alarm rate needs to decreased, increase in the precision score is needed. So at first parameter tuning was applied to increase the f1 score with very high iteration count. As it is time consuming, parameter tuning was only applied for CART on 25% of the datasets. Initially RF, FFT or VFDT were not chosen because applying parameter tuning on them is time consuming due to the facts mentioned below:

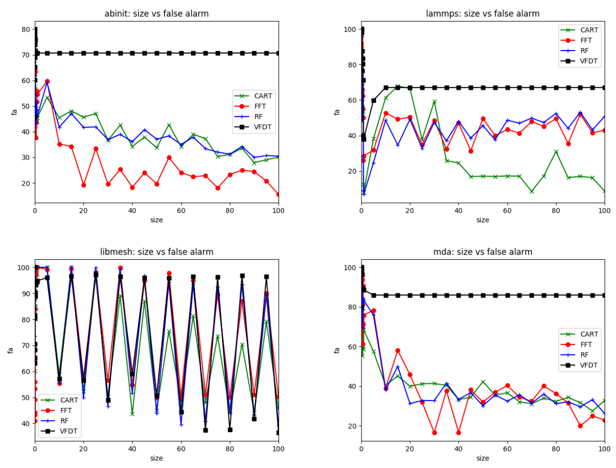


Figure 11. Data Size vs False Alarm

1. Tuning was already applied to all the learners on all the datasets across 40 split size with 10 random sample of it. Still, decent f1 score was not found. So, there might be a possibility that, any better precision and recall at the same time could not be found.
2. FFT is inherently time consuming for discretization on more than 25 attributes. It is also built from scratched and not as optimized as scikit-learn CART
3. RF needs to build at least 10 estimators and thus at least 10 times expensive than CART
4. VFDT can be much faster than CART. However, parameter tuning with n_min and τ can be time consuming as well.

After running the differential evolution algorithm, it was found out that the best precision score obtained is 0.56 from *lammps* dataset and worst obtained is 0.48 from *abinit* dataset. After that, the DE was run once again but with a goal to maximize the difference between recall and false alarm rate. But a maximum difference of 0.07 in *lammps* dataset was found.

Keeping this in mind, another three parameter tuning was performed while the datasets finished processing 50%, 75% and 100% of the total examples. It was observed that precision score does not vary significantly across the dataset size from all the learners, hence in the differential evolution method, it was tried to optimize the parameters on the basis of recall score. Here is the summary of the findings:

It is obvious from Figure 12, 13, 14, 15 that, for most of the cases recall score could have been improved if further tuning was deployed after seeing a substantial amount of newer examples. It is also obvious that recall score of VFDT can be improved farther. However, in case of learning for a large stream of data periodically, it is really inefficient to apply tuning each time the dataset learns.

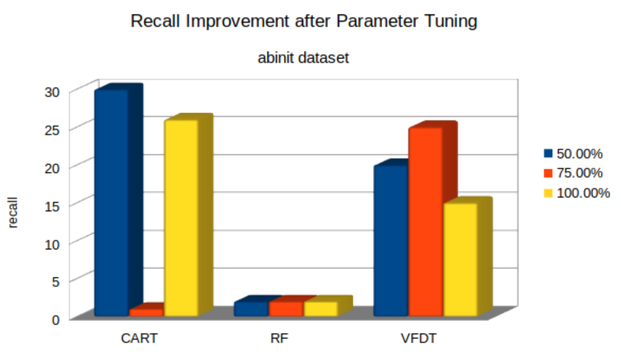


Figure 12. Size vs Recall for mdanalysis dataset

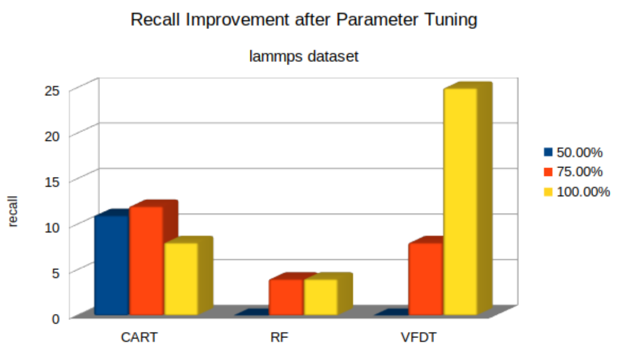


Figure 13. Size vs Recall for mdanalysis dataset

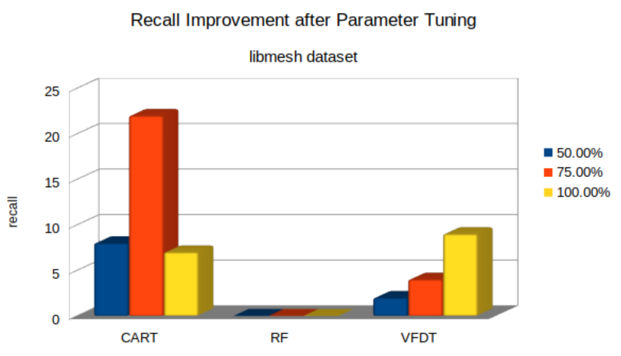


Figure 14. Size vs Recall for mdanalysis dataset

4.3 RQ3: How much computation resources would be used by these four learners

It is also important to observe the computational resources being used by the learners because, with the increase of data size, the need for the learners to become cheap in terms of time and memory is paramount. Here, the findings from the analysis is presented below.

From the above fig 16, it can be seen that FFT is very expensive in terms of computation time. Apart from libmesh dataset, it is also seen that CART, RF and VFDT all takes

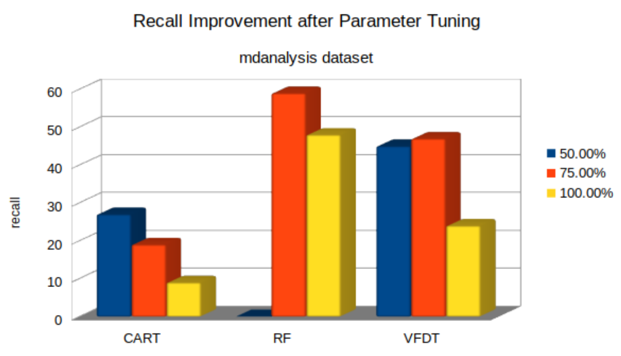


Figure 15. Size vs Recall for mdanalysis dataset

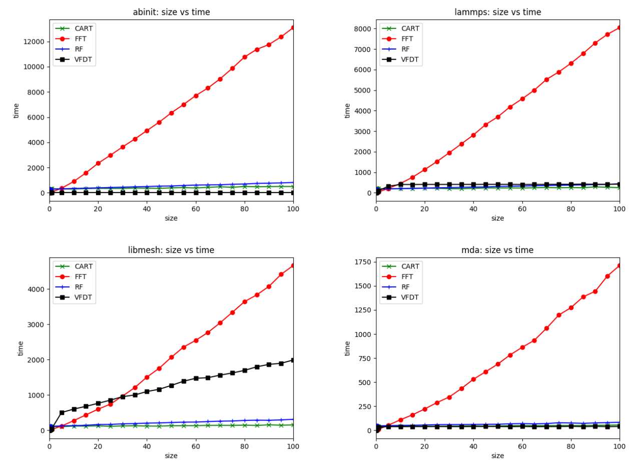


Figure 16. Data Size vs Time

similar time to operate. However, such comparison between VFDT, CART and RF is not conclusive as the dataset is very small compared to the available primary memory in the testbed system. Hence, it was needed to deploy the learner with a very big dataset that is similar to the size of primary memory of the computer. Unfortunately, such big dataset is not available for software defect prediction. Hence, a huge physics dataset for atomic particles was collected from UCI Machine Learning dataset repository [19]. The size of the dataset is around 8Gb of size. It contains 110000000 rows and 29 columns. All the class labels are either boolean and attributes are continuous. From the run, here is the outcome. From the table 3 above, it is obvious that VFDT scales in terms of both computation time and memory usage if the dataset size is substantially large.

4.4 Revisiting Baseline Criteria:

The initital motivation is to observe whether stream based VFDT can do better than the offline classifiers. Hence, based on the observation above, here is the observation from the aspect of the aforementioned baseline.

Learner	Time	Memory
CART	332 sec	6.1 GB
RF	425 sec	6.1 GB
VFDT	34 sec	271 MB

Table 3. Computation Resource Usage for Very Large Dataset

1. **Useful:** In terms of precision, no learner is useful as those are not better than random guess significantly. However, in terms of recall, VFDT have outperformed others.
2. **Reasonable:** VFDT has performed no worse than the other established learners in this case.
3. **Stable:** VFDT has shown much more stable result than the other three
4. **Robust:** Across different datasets and splits, VFDT has shown more robust performance than the other three.
5. **Cheap:** VFDT is computationally cheaper than the others.
6. **Streaming:** VFDT can handle large datasets sequentially and it does not need to load the whole set of examples to primary memory.
7. **Tunable:** Parameter tuning yielded better recall rates for VFDT than the others across all the sizes.

5 Threats to Validity

1. **Sampling Bias:** Like every classification experiment, this work is also subject to have some bias in sampling training and test data. Although 10 disjoint training and testing datasets for each dataset was sampled, it could have been better if the sampling number was higher
2. **Random Bias:** Some of the learners in the experiments utilized random value generation such as RF. Rerunning such classifiers multiple times could given a less biased outputs.
3. **Parameter Bias:** In differential evolution algorithm, some parameters were used to find the best parameters for the learners. So inherently, the tuning method also suffers from some degree of biasness induced by the parameters in the tuning algorithms.
4. **Code Optimization:** Program written from scratch for VFDT and FFT are not as optimized as the CART and RF from scikit-learn package. So the experiment for computational resource could have been bias free if RF and CART were built from the scratch as well.
5. **Dataset:** The experiment performed is only based on the four datasets. It should also be conducted on other large defect datasets to get a more neutral perception.
6. **Cross Validation:** The performance of the learners could have been better observed if cross-validation was performed. However, as the data is ordered, it was not possible.

6 Future Work Direction and Conclusion

Despite the fact that, VFDT has gained in all the baseline criteria that are looked into so far, it has also opened several future work directions. Firstly, the experiment should also be performed in other large software defect datasets to gain even better understanding. Second, the issue of high rate of false alarm should be probed. Moreover, using multi-threading and multi-processing, VFDT can be made much more faster as well. Finally, C-VFDT [12] - which is a variant of VFDT can be much more useful in the defect prediction scenario as it can change its tree structure due to any potential distribution change or concept drift in large dataset [25] - can be deployed in defect prediction and the performance should be observed.

References

[1] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering* 22, 10 (1996), 751–761.

[2] Nicolas Bettenburg, Meiyappan Nagappan, and Ahmed E Hassan. 2012. Think locally, act globally: Improving defect and effort prediction models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 60–69.

[3] Lionel C Briand, Jürgen Wüst, and Hakim Lounis. 2001. Replicated case studies for investigating quality factors in object-oriented designs. *Empirical software engineering* 6, 1 (2001), 11–58.

[4] Luca Canzian and Mihaela Van Der Schaar. 2015. Real-time stream mining: online knowledge extraction using classifier networks. *IEEE Network* 29, 5 (2015), 10–16.

[5] Gianmarco De Francisci Morales. 2013. SAMOA: A platform for mining big data streams. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 777–778.

[6] Thomas G Dietterich. 2000. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning* 40, 2 (2000), 139–157.

[7] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 71–80.

[8] Len Erlikh. 2000. Leveraging legacy system dollars for e-business. *IT professional* 2, 3 (2000), 17–23.

[9] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. 2012. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1276–1304.

[10] Ahmed E Hassan. 2009. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 78–88.

[11] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30.

[12] Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 97–106.

[13] Rahul Krishna. 2018. github/se4sci/defect-prediction. https://git.io/fpKiu. [Online; last accessed 2-Dec-2008].

[14] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. 1996. SLIQ: A fast scalable classifier for data mining. In *International Conference on Extending Database Technology*. Springer, 18–32.

[15] Tim Menzies, Andrew Butcher, Andrian Marcus, Thomas Zimmermann, and David Cok. 2011. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 343–351.

[16] Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2014. Cross-project defect prediction models: L'union fait la force. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 164–173.

[17] J Quinlan and R Cameron-Jones. 1995. Oversearching and layered search in empirical learning. *breast cancer* 286 (1995), 2–7.

[18] Md Rayhanur Rahman. 2018. github/brokenquark/NCSUFSS18. https://git.io/fpKMM. [Online; last accessed 2-Dec-2008].

[19] UCI ML Repository. 2018. Higgs Dataset. https://archive.ics.uci.edu/ml/datasets/HIGGS. [Online; last accessed 2-Dec-2008].

[20] scikit learn. 2018. sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Online; last accessed 2-Dec-2008].

[21] scikit learn. 2018. sklearn.tree.DecisionTreeClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Online; last accessed 2-Dec-2008].

[22] Emad Shihab. 2012. *An exploration of challenges limiting pragmatic software defect prediction.* Ph.D. Dissertation.

[23] Rainer Storn and Kenneth Price. 1997. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization* 11, 4 (1997), 341–359.

[24] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. 2007. Predicting defects for eclipse. In *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*. IEEE, 9–9.

[25] Indrė Žliobaitė. 2010. Learning under concept drift: an overview. *arXiv preprint arXiv:1010.4784* (2010).