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

Md Rayhanur Rahman

Motivation

- Software Source Code evolves - so does the source code metrics
- Software Defect Prediction works on top of those metrics
- Learners need to relearn everything as the source code changes
- Relearning from scratch is not cost effective due to starting over things as soon as source code gets committed
- On top of this, if the dataset gets larger and larger, relearning can become infeasible
- Intuitively, Decision Trees, Random Forests suffer such aforementioned problem
- What if we deploy an incremental/online/stream based decision tree based learner…???

Research Questions

- Three non-streaming learners (CART, RF & FFT) vs one streaming learner (VFDT) will be deployed to predict software defects from datasets that will be fed to the learners with increasing numbers of examples
- We want to observe…
    a. Their prediction performance in defect prediction (RQ1)
    b. Impact of parameter tuning on their performance (RQ2)
    c. The computational resource usage (RQ3)

Baselines Targeted

- Usefulness
- Stable
- Cheap
- Streaming
- Robust

Learners Used for Comparison

- Classification and Regression Tree (CART)
- Random Forest (RF)
- Fast Frugal Tree (FFT)
- Very Fast Decision Tree (VFDT) [1]

How VFDT Works...

- It keeps consuming examples until the hoeffding bound is reaches
- Reaching hoeffding bound means
  - With the probability of 1 - δ, the true mean of a random variable r is at least r' - ε where,
    - $\varepsilon^2 = R^2\ln(1/\delta)/2n$
- To find the best attribute in a node, calculate the information gain of all the attributes when a new examples comes and rank the attributes from best to worst
- If the difference of top two attribute's information gain is bigger than the ε, we can split.
- If not, we have to stream in more examples unless the bound satisfies
- Do it recursively from top to bottom

Datasets

- Software defect prediction datasets
- Class label is Bugs and value is either 0 or 1
- Attributes are software code metrics
- Four datasets:
  - Abinit (around 90,000 examples, 27 attributes)
  - Lammps (around 42,000 examples, 40 attributes)
  - Libmesh (around 25,000 examples, 40 attributes)
  - Mdanalysis (around 10,000 examples, 37 attributes)
- Source: https://github.com/se4sci/defect-prediction/tree/master/src/data/turk_labeled

Preparing the Datasets

- From each dataset, 10 different training and test sets were generated
- Test sets include 10 different set containing 20% of the original dataset
- Training sets include 10 different set containing 80% of the original dataset
- As the dataset were originally obtained sequentially, no stratification is applied
- 10-fold cross validation is not done as learners will be fed increasing number of examples in each pass
- All the learners will be fed these following amount of examples from each of the 10 training set for all of the 4 datasets:
  - .01%, .02%, …, .1%, .2%, …, 1%, 5%, 10%, …., 95%, 100%
- This is how streaming of larger and larger set of training examples is simulated for the learner

Parameter Tuning

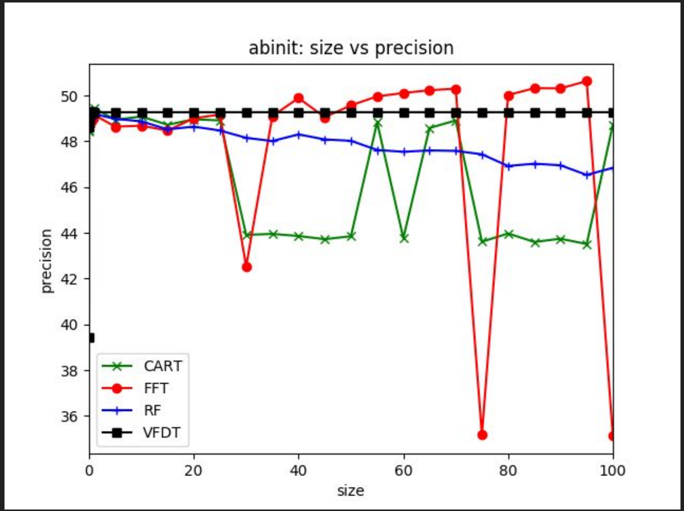
- Differential Evolution [2] Method is applied
- Ranges of the parameters for different learners

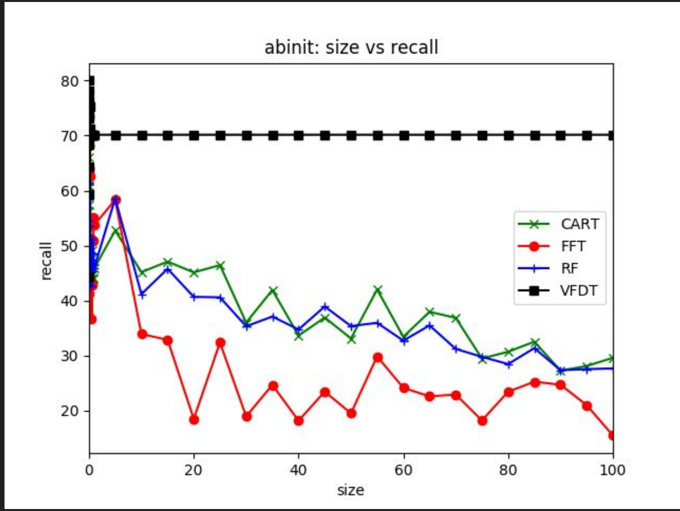
Parameters	Ranges	Learners
Number of Estimator	10 to 100	RF
Maximum Depth	2 to 10	CART, RF, VFDT
Minimum Number of Examples to Split	0 to 1, 5 to 500	CART, RF, VFDT
Minimum Number of Examples at Leaf	0 to 1	CART, RF
τ	.001 to .99	VFDT
n_{min}	5 to 500	VFDT

Evaluation Criteria

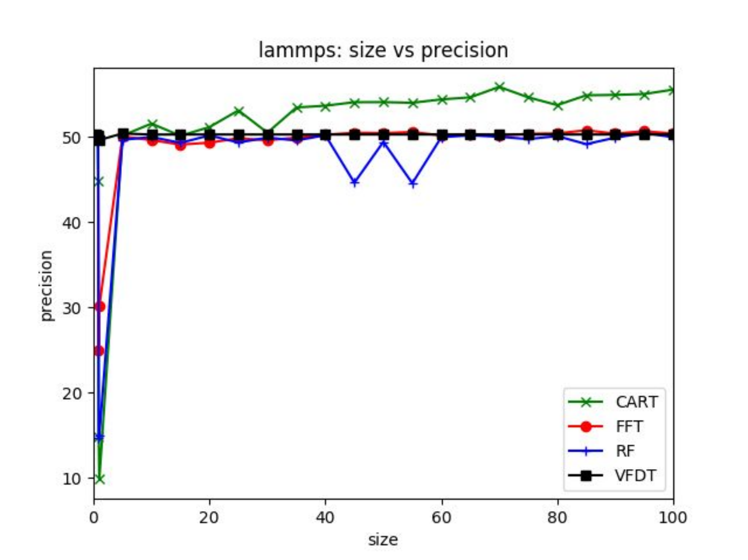
- Prediction Performance
  - Precision
  - Recall
  - False Alarm
- Computer Resource
  - Training Execution Time
  - Primary Memory Consumption

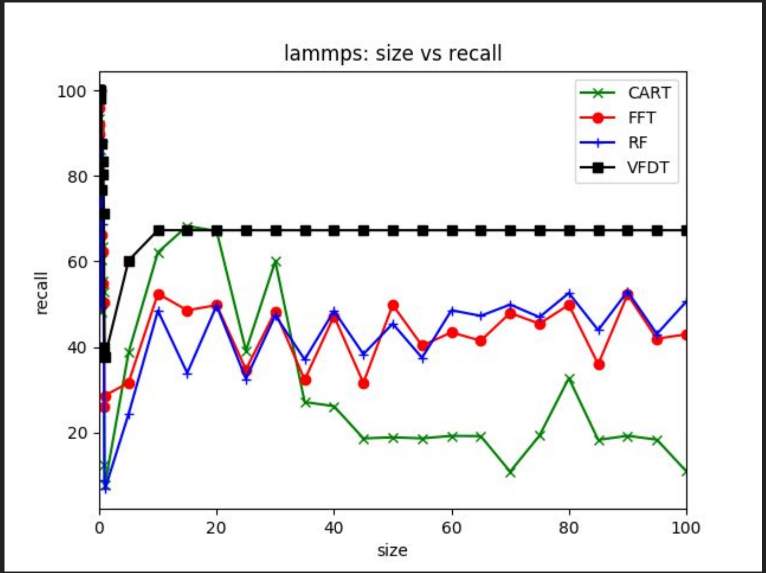
Dataset: Abinit



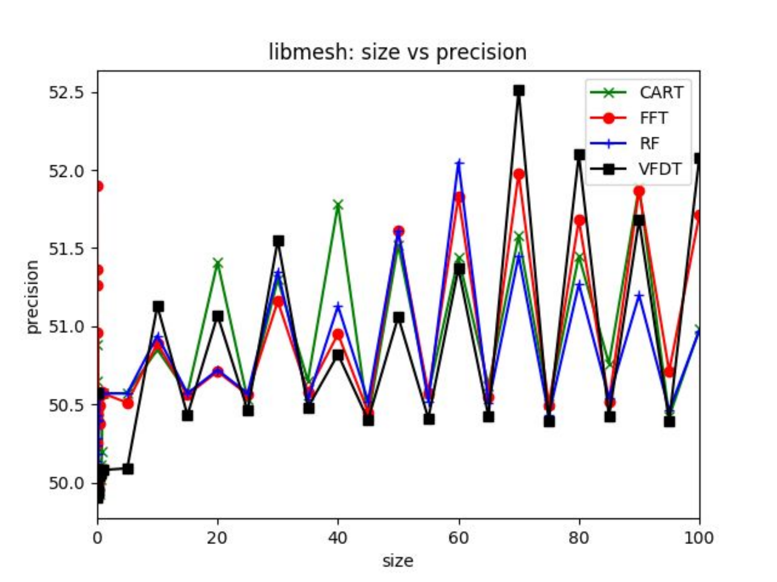


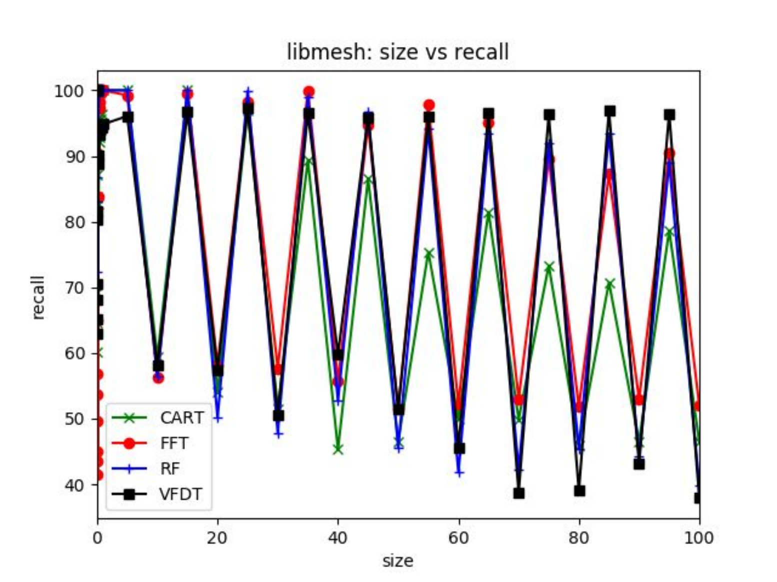
Dataset: Lammps



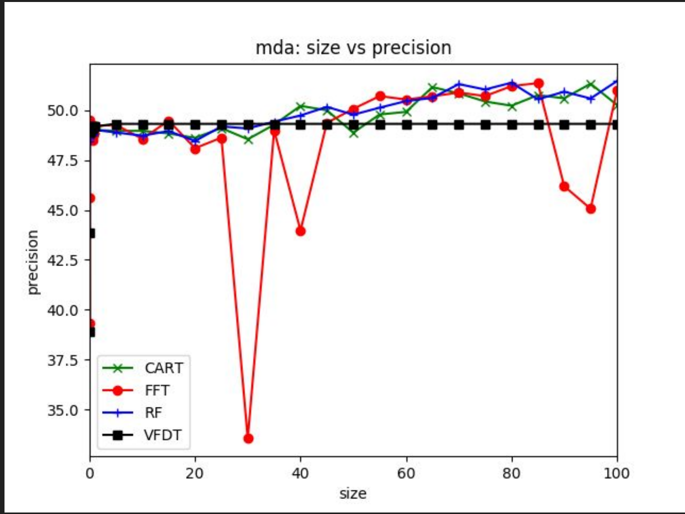


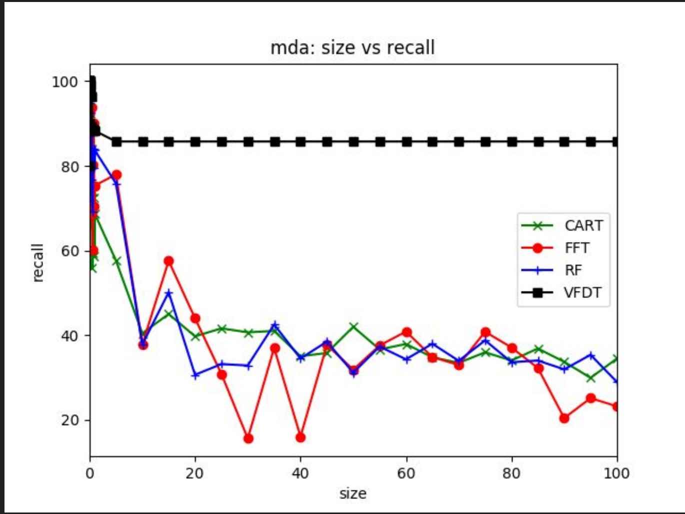
Dataset: Libmesh



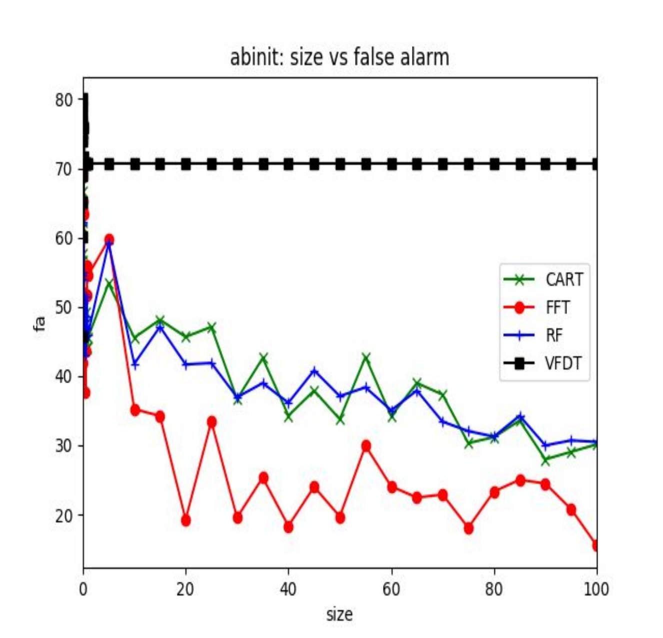


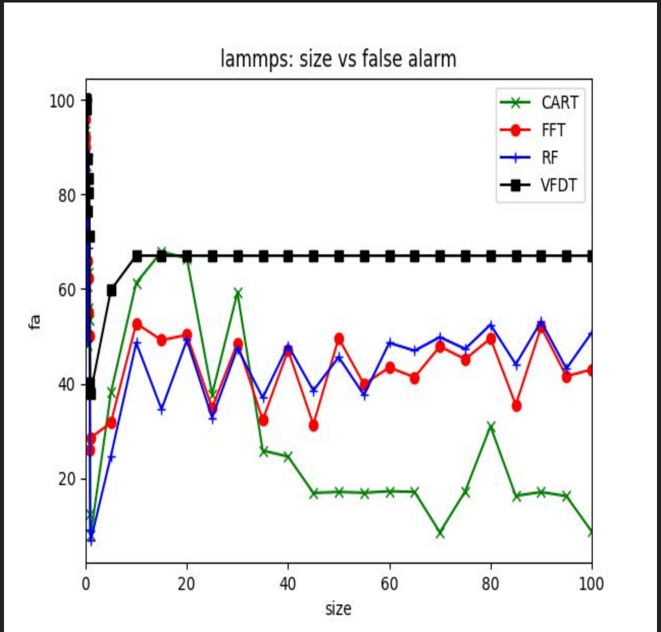
Dataset: Mdanalysis



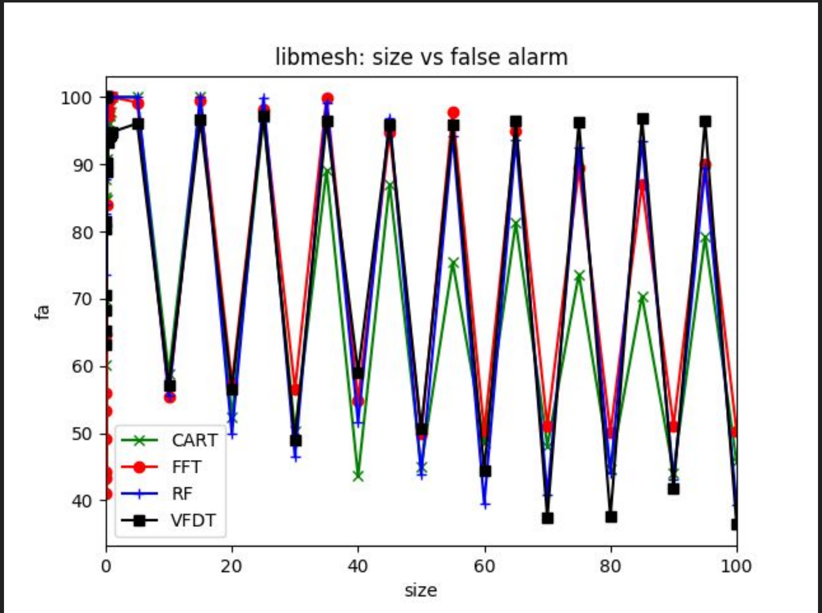


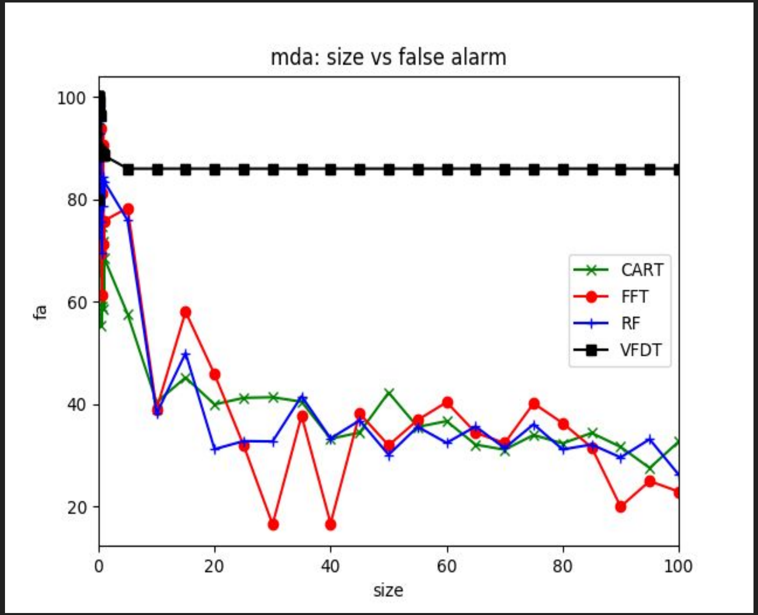
False Alarm for All Datasets





False Alarm for All Datasets





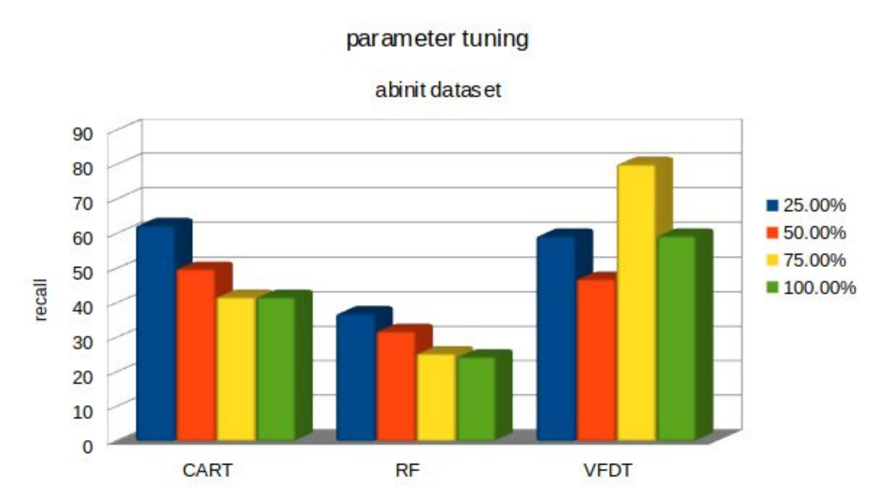
AUC of (size, precision) and (size,recall)

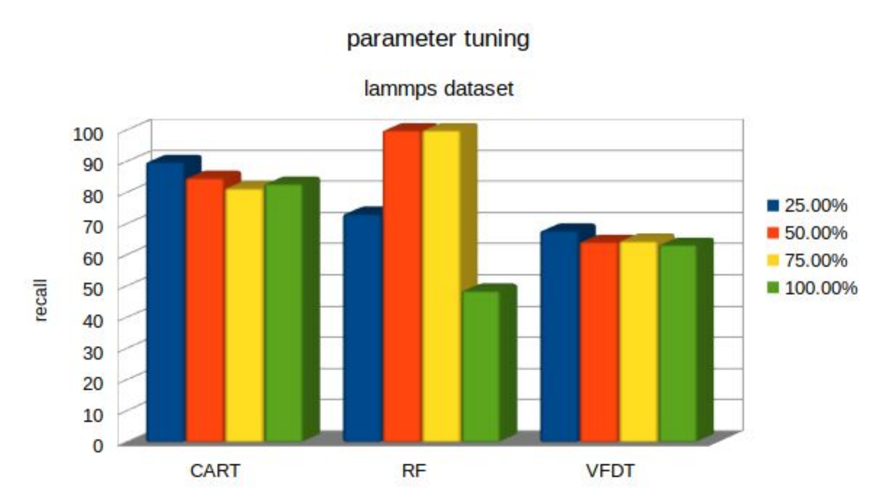
Dataset	Plane	CART	FFT	RF	VFDT
Abinit	size,prec	4606	4816	4785	4926
Abinit	size,rec	3769	2632	3620	7014
Lammps	size,prec	5246	4969	4855	5026
Lammps	size,rec	3091	4338	4336	6653
Libmesh	size,prec	5097	5096	5087	5090
Libmesh	size,rec	6864	7576	7265	7368
Mdanalysis	size,prec	4979	4835	4996	4929
Mdanalysis	size,rec	3928	3691	3908	8585

RQ1: Performance in Defect Prediction

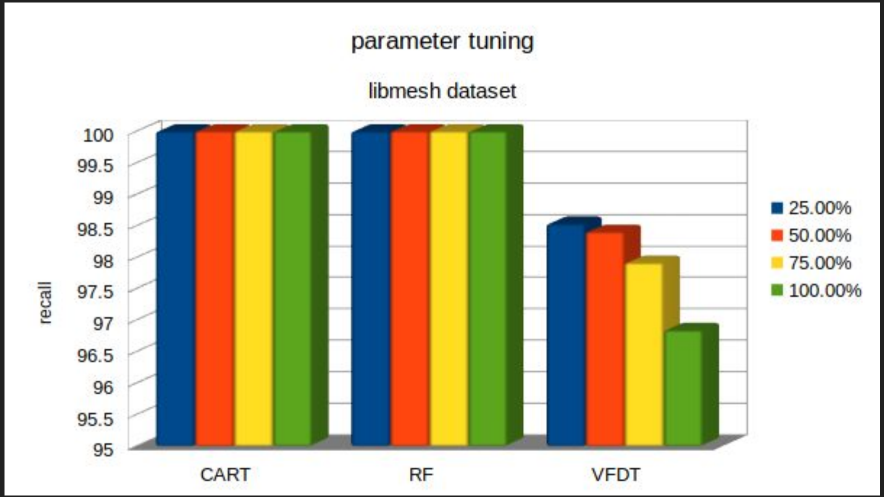
- Precision is overall similar for all learners across all datasets
- But precision is not very far away from the random guess
- Recall and False Alarm varies for datasets and learners
- However, VFDT yields both better recall rate and fluctuate less than the others
- False Alarm rates and recall rates are more or less similar to the recall score of the learners
- VFDT’s stable recall rate is due to the reason of reaching hoeffding bounds down to the maximum depth

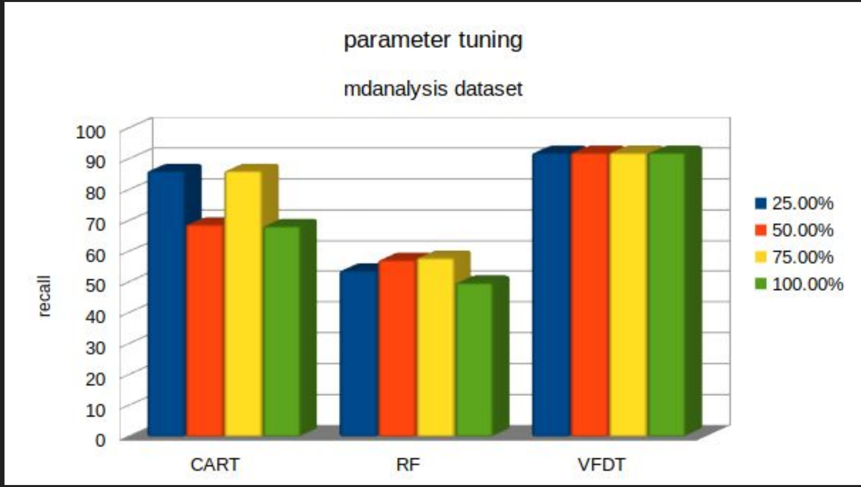
Parameter Tuning at Different Size





Parameter Tuning at Different Size





RQ2: Impact of parameter tuning

- For better result, tuning can be done
- But with streaming data, it’s not very practical
- However, VFDT doesn’t gain much on parameter tuning compared to other two learners

Execution Time

- VFDT is fastest overall (one exception), FFT is the slowest. n_{min} dominates the execution time of VFDT

Memory Consumption

- CART and RF consumed 168MB when learning from the whole abinit dataset
- FFT and VFDT consumed 276 and 288 MB respectively
- However, this comparison is not fair because,
  - The implementation of FFT and BFDT isn’t as efficient as scikit-learn + numpy
  - Python garbage collection mechanism does not release memory as soon as the variables become useless

Review in terms of Baselines

- In terms of precision, not very useful. In terms of recall, VFDT is much more useful than the other three
- VFDT behaves much more stable than the other three
- While learning from newer and newer example consistently, VFDT is more robust
- Theoretically, VFDT is computationally cheaper but parameters have to be set accordingly
- VFDT is a practical option to stream in large number of examples for learning

Discussion and Future Work

- VFDT performed better but not that meaningfully better
  - Poor precision
  - Poor false alarm
- Large datasets can confuse the learner. For example:, observed zigzag in libmesh dataset
  - Need for re-learn
  - Need to change the tree
- VFDT is static, C-VFDT can be a better choice for such scenario
- VFDT is fast but can be made much much faster using multi-core processing
- The effectiveness of VFDT and C-VFDT can be explored in Scalable Planners based on software defect prediction

References

1. Mining high-speed data streams - Domingos. Pedro and Hulten, Geoff. ACM SIGKDD - 2000
2. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Rainer S, Kenneth P. Journal of Global Optimization - 1997