Improving Classification Accuracy of a Machine Learning approach for FPGA Timing Closure

Que Yanghua, Nachiket Kapre
School of Computer Engineering
Nanyang Technological University
Singapore, 639798
nachiket@ieee.org

Harnhua Ng, Kirvy Teo
Plunify Inc.
82 Geylang Lor 23
Singapore 388409
harnhua@plunify.com

***Abstract*—**
We can use Cloud Computing and Machine Learning to help deliver timing closure of FPGA designs using InTime [2], [3]. This approach requires no modification to the input RTL and relies exclusively on manipulating the CAD tool parameters that drive the optimization heuristics. By running multiple combinations of the parameters in parallel, we *learn* from results and identify which parameters caused an improvement in the final results. By systematically building a classification model and training it with the results of the parallel CAD runs, we can build an accurate estimation flow for helping identify which parameters are more likely to improve the timing. In this paper, we consider strategies for improving the predictive accuracy of our classifier models to help guide the CAD run towards timing convergence. With ensemble learning we are able to increase average AUC score from 0.74 to 0.79, which could also translate into 2.7$\times$ savings in machine learning effort.

I. Introduction

InTime [3], [2], [5] is a plugin for Xilinx and Altera FPGA CAD tools that allows an FPGA designer to deliver timing closure for their digital design in an automated manner. With dropping computing costs it becomes possible to trivially spawn hundreds of FPGA CAD runs in parallel for potentially faster convergence. Microsoft Bing FPGA acceleration team[1] uses the parallel capacity of the Azure cloud platform to run concurrent seed exploration of the same design to exploit statistical variation in the performance of the FPGA CAD tools. InTime goes beyond this simplistic approach by learning the exact design-specific combination of FPGA CAD tool parameters that deliver timing closure for that particular design. A typical CAD tool may expose hundreds of parameters in the synthesis, technology mapping, and place-and-route steps. For example, Quartus 14.1 exposes hundreds of boolean parameters resulting in an intractably large search space of $\approx 2^{100s}$ possible combinations. Instead of relying on brute force exploration, InTime cleverly searches a subset of this space by choosing which combinations to evaluate with the assistance of machine learning routines.

The key challenge for InTime's machine learning routines is the accuracy of predicting which combinations of CAD parameters to explore. This is particularly important as FPGA CAD runs are slow and can take hours to days of runtime on modern designs. Furthermore, we want to quickly converge to the fastest possible design to satisfy time-to-market targets of our end customer. Another advantage of faster timing closure is the reduction in the number of parallel CAD runs that we must invoke thereby reducing cloud computing costs.

[1]FPGA 2016 Designer's Day invited talk by Eric Chung.

In this paper, we configure and compare various **classification** algorithms that can work well for the structure of the data generated for FPGA timing closure problems. A key challenge that is unique is the presence of very limited data (100s of runs) when compared to the datasets used in the mainstream machine learning community as well as the separation in the final timing scores (very noisy).

The key contributions of this report include:

- Configuration and optimization of the **classifiers** to improve the accuracy of prediction while reducing false positive rates.
- Quantification and characterization of these optimizations across various open-source benchmarks in terms of machine learning metrics (*e.g.* ROC) and FPGA outcomes (*e.g.* iteration counts).

II. Configuring Machine Learning

The initial InTime release relied on simple Naïve Bayesian classifier to organize the learning process. In that case, we generated suitable candidate CAD parameter combinations by testing against trained model. With this approach, InTime was still able to reduce timing scores by 10$\times$ [2] in under 200 runs of the CAD tool across a range of open-source FPGA designs. While these results were promising, there was room for improvements. Our dataset sizes are fairly small (*i.e.* 200-400 samples per design, compared to other millions of records in big-data applications) due to the long runtimes of the CAD tools and finite computing costs at our disposal. What makes this even more challenging is the finicky nature of the resulting timing slacks to even the slightest change in input CAD parameter conditions. In order to overcome these constraints, we explore a broader set of classification algorithms.

- **Logistic regression:** (R function `LogitBoost`) Regression-based techniques (*e.g.* linear regression) is often used to predict quantitative outputs *e.g.* TNS score. However, our initial attempts at using linear regression had limited success due to the unpredictability of the TNS scores. Thus, we switched to logistic regression where each CAD combination is simply evaluated as GOOD or BAD with reference to a threshold TNS score.

- **Bagging:** (R function `treebag`) Decision tree model uses a branch-split structure and segmentation of the predictor space to best fit the observed data. This manner of partitioning naturally fits intuition that certain combinations drive us towards convergence and certain combinations make timing scores worse. However, decision trees alone have low accuracy since it highly depends on the region splitting decisions made in each branch node. We can improve the accuracy of decision trees using Bagging. This is also a tree-based algorithm, but constructs multiple trees $\hat{f}^b(x)$; one in each sampled subset of the input observations. The resulting classifier averages ($\frac{1}{B}$) across these trees, thereby lowering variance and achieving higher prediction accuracy.
- **Random Forest:** (R function `rf`) Random Forest is an adaptation of Bagging where we want to account for correlations between the input CAD parameters and mitigate noises caused by irrelevant features. Random Forest considers random subsets of features x of the input observations X (unlike Bagging) and constructs a decision tree in each subset (like Bagging). Thus, the influence of minority predictors is suppressed and the prediction is more reliable.
- **Support Vector Machine (SVM):** (R function `svmRadial`) Support vector classifiers construct hyperplanes to separate the data into binary classes. SVMs support non-linear boundaries between classes by suitably enlarging this space to fit the non-linearity using the kernel function K. It is computationally intensive as we must construct inner products between all support vectors x_i in S. While this is a computationally-complex classifier, the runtime is still marginal compared to the CAD time.
- **Neural Network:** (R function `nnet`) Neural network is a two-stage classification model, inspired from biological networks of neurons. In the first stage, we construct derived features Z_m through linear combinations of the inputs X using a suitable activation function σ. In the second stage, the output Y is then modeled with linear combinations of these intermediate features Y. The output function g_k then transforms the prediction T into the eventual classifier outputs. Neural networks are harder to train and have many configurations to manage for correct evaluation.
- **Stack:** (code name `stack`) In this strategy, we run all the algorithms described here multiple times using the timing slack database. This allows us to train a sub-model per classification algorithm. We then train a second-level model that combines the predictions of the sub-models using linear regression (Generalized Linear Model).
- **Ensemble:** (code name `ensem`) This is another technique that also requires running all algorithms, but instead of using a second level combiner model, we simply compute a linear weighted average of the sub-model predictions. It is computationally cheaper than **Stack** but can often deliver similar quality of result.

III. Results

We implemented the improved machine learning routines as plugins for InTime using *caret* [1] packages in `R` [4]. We consider OpenCores benchmarks in this evaluation. These benchmarks were compiled using Quartus 14.1 and mapped to Cyclone devices.

To compare the range of different machine learning techniques on our benchmark set, we evaluate the respective trends in the ROC curves. We want the good classifiers to deliver ROC curves that lean towards the top-left quadrant of the plots with high true positive rates and low false positive rates.

In Fig. 1, we show the effect of classification algorithm on final quality. Across our set, *Stack* emerges as the most stable and robust algorithm for the FPGA CAD timing score dataset. There is a high density of high TPR and low FPR when considering *Stack*. The *Logistic Regression* and *Neural Network* have relatively poor performance with the curves trending close to the diagonal. Along the diagonal, the machine learning routine is performing no worse than a random coin toss. Other routines such as *Bagging*, *Random Forest* and *Bucket* methods have a few really good outlier curves but perform mid-range in the exploration with average distribution of curves away from the top-left quadrant. *Decision Tree* implementation tends to simultaneously jump to high true positive and false positive rates resulting in poor performance.

In Fig. 2, we evaluate individual circuit benchmarks to compare the effectiveness of different algorithms for each design. In the plot, we highlight the result of *Stack* algorithm in bold. As we see, it consistently delivers top-of-the-line performance over the complete classification range. In small stretches, it cedes its dominance for *vga*, *viterbi* and *flow* benchmarks. In this set, the *viterbi* and *flow* benchmark (and to some extend *xge*) have tough-to-meet timing constraints to begin with and stubbornly deliver high false positive rates (despite their high accuracy) for most machine learning algorithms. These curves are closer to the diagonal and are not particularly amenable to the machine learning approach. At present, there are realistic limits to the extent of timing margins that InTime can squeeze out of the tools as some constraints may be infeasible. In future work, we hope to be able to predict a realistic timing target window our tools can target.

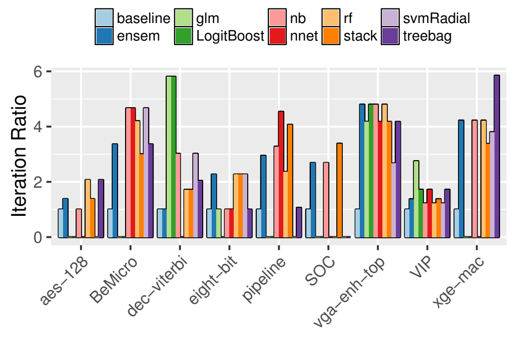

Fig. 3: Iteration Count Reduction when using new algorithms vs. original InTime baseline.

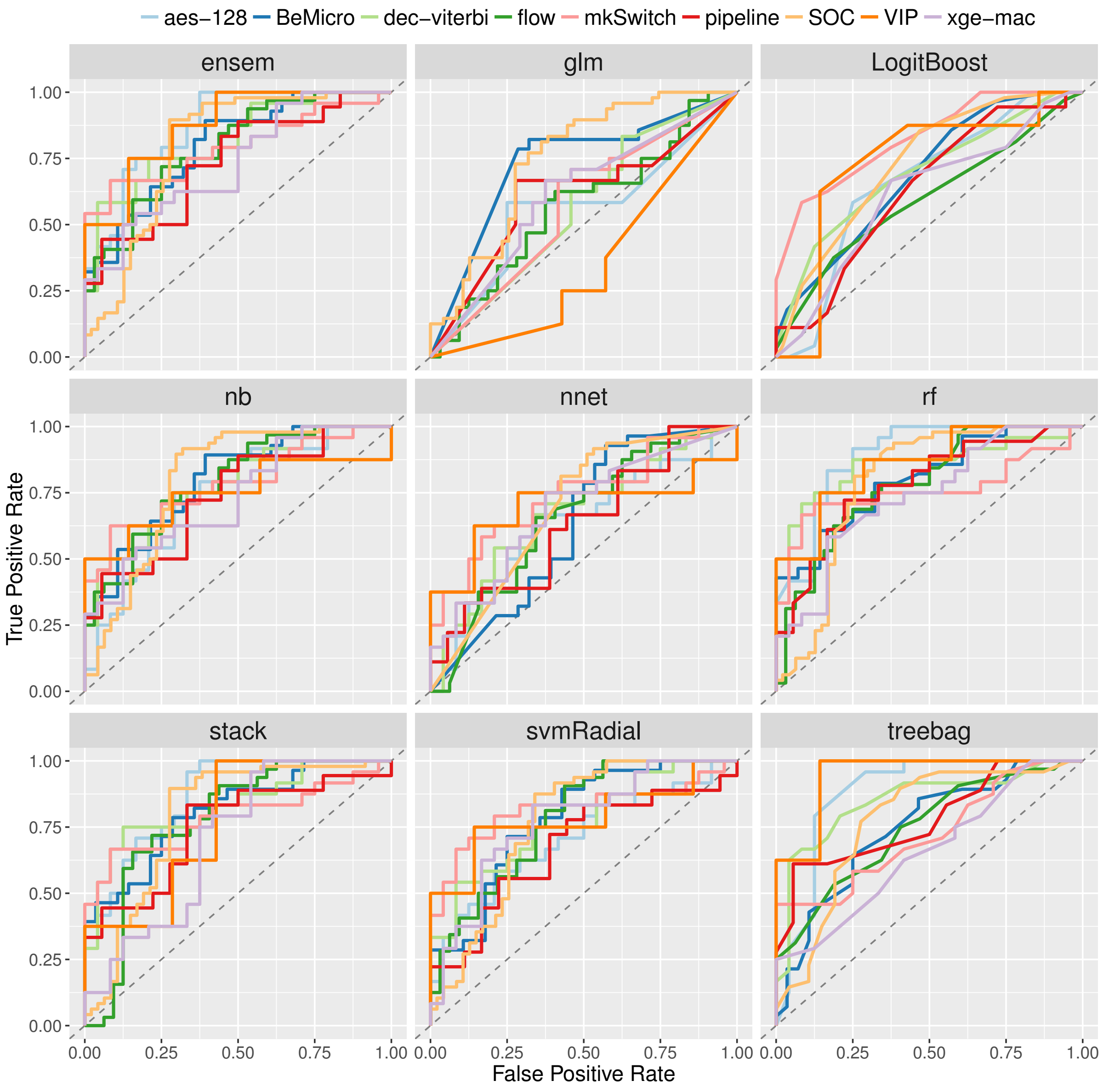

Fig. 1: ROC curves for the various Machine Learning algorithms across various benchmarks. Good ML algorithms should have ROC curves in the top-left quadrant of the plot. Ensemble methods such as *Stacking* and *Bucket* work best and have high density in the top-left quadrant. The tree-based methods such as *Bagging*, *Random Forest* also have a few good solutions in the top-left quadrant of the plot. *Neural Network* and *Logistic Regression* are closer to the diagonal and slightly less effective.

Finally, in Figure 3, we show the reduction factor in number of CAD iterations required when comparing original InTime baseline against our improved approach. As we see, we are able to deliver a 2–6$\times$ (mean 2.7 $\times$ for `Stack` method) reduction in iteration counts. This means that we can build a model that is as accurate with far less training samples, which are essentially CAD running records harvested from the cloud farm; and we can save a lot of time and computing resources as a result. This shows how exploiting the unique characteristics of the FPGA design as part of the CAD flow can not only improve AUC scores but also translate into wins for acceleration of timing closure.

IV. Conclusions

In this paper, we show how to improve the predictive accuracy of InTime's classification routines to help speedup timing closure for FPGA designs by purely manipulating the CAD tool parameter assignments. Specifically, we show how to configure and use the Stack-based approach that combines the predictions of multiple classification algorithms to boost overall robustness of the prediction. With this technique, we gain an AUC score boost of 0.05 on average. This improvement also translates into a 2.7$\times$ reduction in time to timing closure.

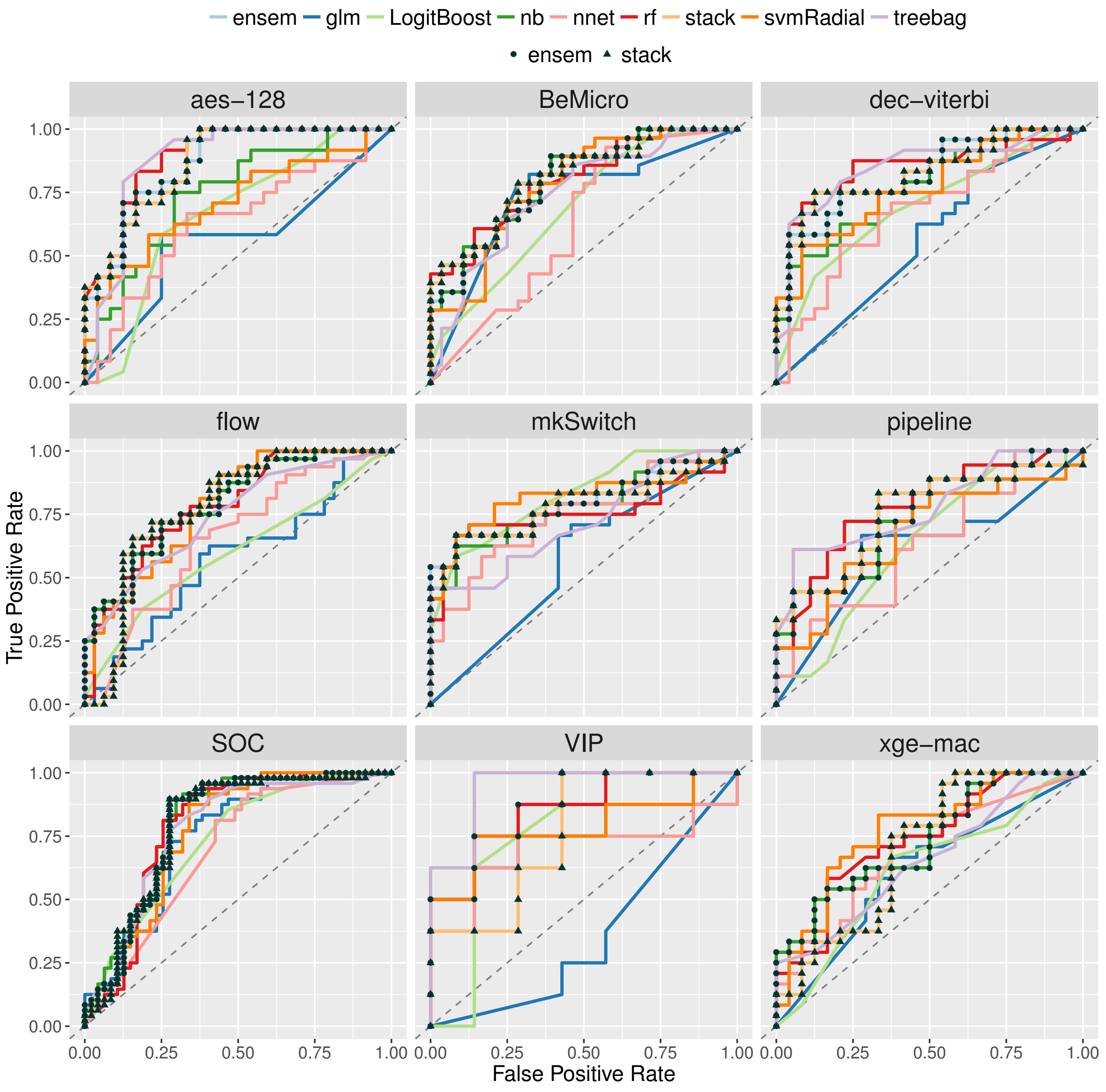

Fig. 2: ROC curves for the various benchmarks across different Machine Learning algorithms. Almost all other benchmarks do particularly well, ROC curves are closer to the top-left corner than the diagonal, especially when using *Stack* (shown in bold black) or *Bucket* algorithms. There are a few cases where they do not pick the winning or best solution but those cases are infrequent and the penalty for the choice is not catastrophic. Benchmarks such as *bitcoin*, *flow* and *viterbi* are hard for timing closure, and show performance closer to the diagonal. These cases are scoring high in certain classification thresholds, but deliver poor convergence due to skew (most classifications are in the negative class).

References

[1] M. K. C. from Jed Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, and L. Scrucca. *caret: Classification and Regression Training*, 2015. R package version 6.0-52.

[2] N. Kapre, B. Chandrashekaran, H. Ng, and K. Teo. Driving timing convergence of fpga designs through machine learning and cloud computing. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 119–126, May 2015.

[3] N. Kapre, H. Ng, K. Teo, and J. Naude. Intime: A machine learning approach for efficient selection of fpga cad tool parameters. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 23–26, New York, NY, USA, 2015. ACM.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[5] Q. Yanghua, C. Adaikkala Raj, H. Ng, K. Teo, and N. Kapre. Case for design-specific machine learning in timing closure of fpga designs. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 169–172, New York, NY, USA, 2016. ACM.