Technische Universität München
Fakultät für Elektro- und Informationstechnik
Lehrstuhl für Entwurfsautomatisierung

Machine Learning Methods for Statistical Circuit Design

Bachelor Thesis

Baris Yigit

Technische Universität München
Fakultät für Elektro- und Informationstechnik
Lehrstuhl für Entwurfsautomatisierung

Machine Learning Methods for Statistical Circuit Design

Bachelor Thesis

Baris Yigit

Supervisor:	Dr. Bing Li
Supervising Professor:	Prof. Dr.-Ing. Ulf Schlichtmann
Topic issued:	15.11.2016
Date of submission:	06.06.2017

Baris Yigit
Christoph-Probst-Str. 16
80805 München

Statement of Authorship

I, Baris Yigit, confirm that the work presented in this thesis has been performed and interpreted solely by myself except where explicitly identified to the contrary. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged. I confirm that this work has not been submitted elsewhere in any other form for the fulfillment of any other degree or qualification.

München, Tuesday 6th June, 2017

Abstract

At nanometer scale manufacturing, process variations have a significant impact on circuit performance. To handle those, post-silicon clock tuning buffers can be included into the circuit balancing timing budgets of neighboring critical paths. The state of the art is a sampling-based approach, in which integer linear programming is required for every sample one at a time. The runtime complexity of this approach is the number of samples multiplied by the required time for one integer linear programming solution. Existing work tries to reduce the number of samples leaving the problem of a long runtime for each iteration unsolved. In this thesis, we propose a machine learning approach to reduce the runtime by learning the positions and sizes of post-silicon tuning buffers. Experimental results demonstrate that we can predict buffer locations and sizes with a very good accuracy (90% and higher) and achieve a significant yield improvement (up to 18.80%) with a high speed-up compared to existing work (up to 19.22 times faster).

Table of Contents

List of Figures

List of Tables

1 Introduction

Manufacturing in nanometer dimensions can produce chips with significant differences in performance due to both environmental and procedural variations. Thus, chips with the same specification coming from the same foundry will have different timing behaviors resulting in a large timing margin which causes unacceptable oversizing to maintain yield. To solve this problem, researchers have developed various circuit components and mechanisms to save failed chips after manufacturing.

One of the post-silicon tuning elements are post-silicon tuning buffers. In [1], the structure of a delay buffer is presented which is illustrated in Fig. 1.1. The delay element, which consists of three parts, creates the delay between the incoming clock signal and the outgoing clock signal. By setting the three configuration bits (0, 1, 2) via the scan chain, the delay value can be altered. Inserted during design phase, post-silicon tuning buffers can be configured after manufacturing allowing for optimal reaction to process variations. Between two adjacent combinational stages, the clock edges are pushed towards the stage with smaller combinational delays and away from the one with the critical path giving the latter more time. Due to process variations, critical paths are unique to every manufactured chip making post-silicon tuning the only effective way to counteract them. This allows for unique circuits to have unique buffer delay settings.

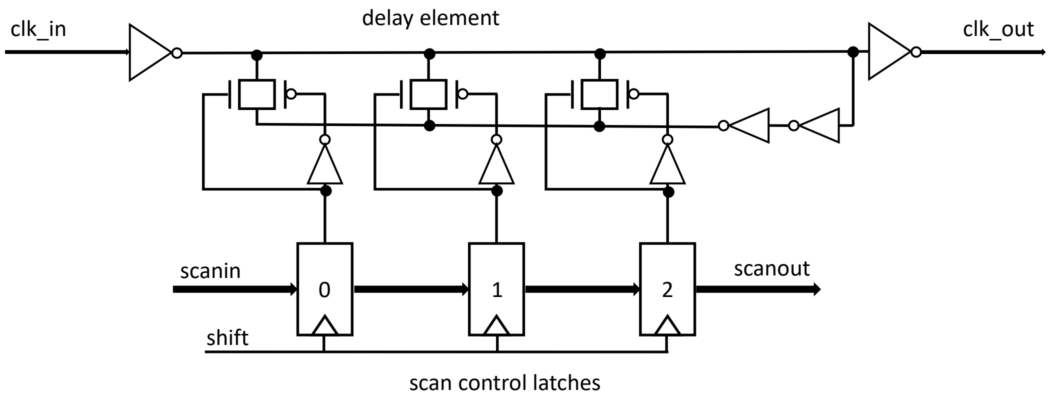

Fig. 1.1 Post-silicon tuning buffer in [1].

In [2], the locations and sizes of post-silicon tuning buffers are determined at design phase by emulating manufactured chips using Monte Carlo simulations. In order to get a good representation, a high number of samples is required and the locations and sizes for buffers must be determined for every sample by integer linear programming. This will inevitably result in long runtimes allowing for runtime improvement using machine learning methods.

Machine learning has a large variety of applications in research and industry. In the financial sector, for example, it is used for fraud detection trying to find anomalies to reduce security risks. In global health, on the other hand, models are developed to analyze the influence of the environment on public health. Basically, any problem with a large amount of data to be analyzed can be tackled using machine learning. The objective of machine learning models is to find underlying patterns in data. After this first step, sophisticated predictions about future outcomes and trends can be made. One concept in machine learning is called supervised machine learning, which can be explained using Fig. 1.2. In supervised learning, the model is given input data with the corresponding output in the training stage. The model parameters are tuned minimizing the prediction error and searching for the best possible prediction based on the present data. After the training is completed, the parameters are fixed and predictions for new input data can be made. The performance of a machine learning model is measured in its capability of correctly predicting the output of unseen input data. Thus, a tradeoff must be made between sufficiently training a model to enable it to find patterns and yet keeping it general enough to perform well on unseen data.

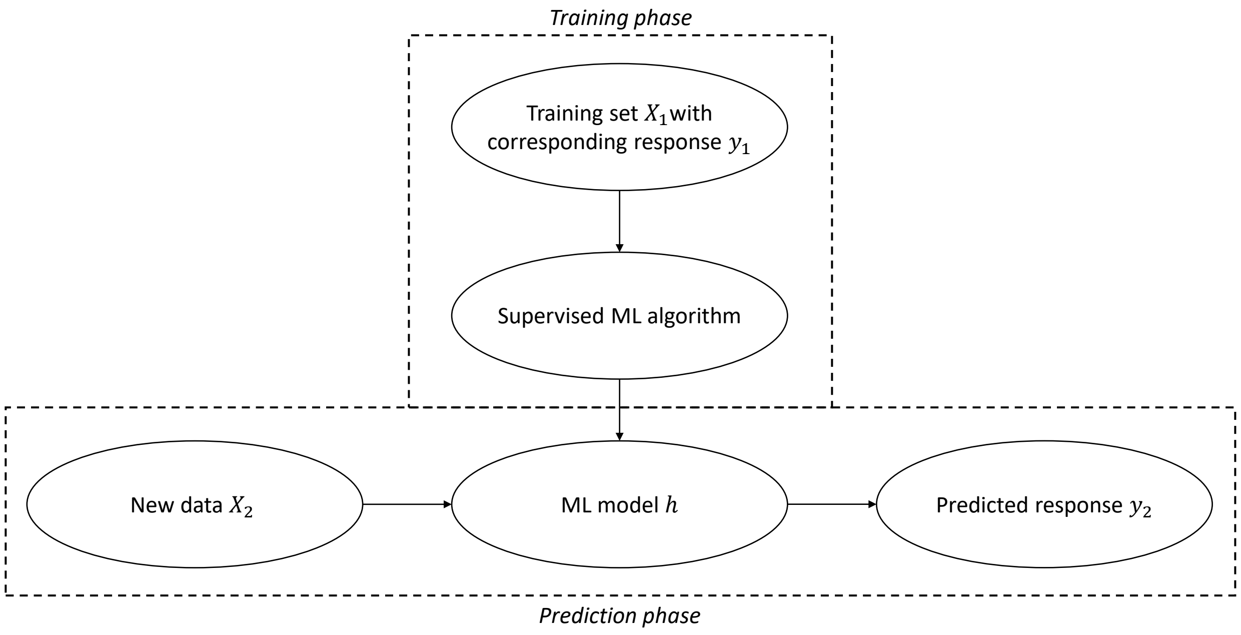

Fig. 1.2 Concept of supervised machine learning (ML) with training and prediction phases. Based on Fig. 2 in [3].

In the design automation field, machine learning has numerous applications. One of the latest is presented in [3]. Here, machine learning is used in the optimization of 3-D integrated circuits. Since electrical and thermal behavior is strongly correlated, they must be modeled together and thus have a high computational cost. Bayesian Optimization is used to find the optimal input parameters iteratively. However, these are limited to five parameters in the beginning of the algorithm. Other applications are, for example, for statistical path selection considering large process variations [4], in which a novel multilayer process space coverage metric is proposed, for parametric yield estimation for analog/mixed signal circuits [5], and for noise sensor placement for voltage emergency detection in dynamic noise management systems [6].

Leaving VLSI applications, further work has been done on comparing different machine learning techniques for learning from a small number of training samples [7]. Data describing real-world problems often results in small training sets. Furthermore, many applications show imbalance in data. In [7], this is tackled by comparing the performance of four different machine learning techniques, which are Support Vector Machines (SVM), Logistic Regression, Naïve Bayes, and Multinomial Naïve Bayes. It is shown that different algorithms excel in different regions of the learning surface whose axes are the number of positives and the number of negatives in the training set which are varied independently from each other. This way, a change of class distributions can be modeled. The results are surprising to the authors: Naïve Bayes show to be insensitive to a change in class distributions while SVM are sensitive to the same changes. This is unexpected because Naïve Bayes classifiers model the class distribution in the priors and are thus expected to be sensitive. Since SVM are non-parametric, they are expected to be less distribution-sensitive. This can help us understand the behavior of different algorithms. However, the application in [7] was text classification and thus, the input is very different from our application. The predictors are made up of binary or nonnegative integer features which, for example, enables the usage of multinomial Naïve Bayes. This does not apply to our inputs which are drawn from Gaussian distributions.

A sampling-based algorithm to insert clock tuning buffers into circuits is proposed in [2]. For every sample, a limited amount of buffer locations is found and the ones that appear the most are selected as the result. The goal of the algorithm is to improve yield after manufacturing of chips and the predominant challenge is the process variability. The inputs to the algorithm are the circuit structure with statistical path delays, buffer specification with the maximum allowed range, and the maximum number of buffers to be inserted into the circuit. The outputs are the buffer locations and sizes considering the setup and hold time constraints of the circuits. The number and sizes of buffers should be kept as small as possible since they require die area and additional testing to be configured. Moreover, the bounds of the ranges, i.e. the maximum and minimum buffer delay values, should be determined. These could be unique to buffers and include negative values. Since a high number of samples is required and one integer linear programming (ILP) problem is solved for every sample, this method has a long runtime. It is extended in [8] to process multiple samples at a time incorporating their relation to each other. A low-discrepancy sample sequence (Sobol sequence) improves the execution efficiency of this method. However, this only reduces the number of samples and yet, one ILP problem still has to be solved for every sample.

In this thesis, we propose a machine learning approach as application to the sampling-based buffer insertion algorithm in [2] to reduce the runtime needed to process each sample. After solving the ILP problem for only a few samples, we use these to train a supervised learning model. In the end, the machine learning model can predict buffer locations and sizes faster than using the ILP approach for every sample. We first analyze and prepare the data from [2]. In a two-stage approach, we first use a classification method to determine whether a buffer is tuned or not. In the second step, a regression model predicts the amount of tuning determining the buffer ranges. Finally, buffers are grouped according to correlation and distance such that multiple flip-flops can be tuned by the same buffer.

The rest of the thesis is organized as follows. In section 2, we describe the buffer insertion problem and present the motivation. In section 3, we propose the solution by inserting buffers using machine learning. Experimental results are presented in section 4 and a conclusion is given in section 5.

2 Problem Formulation and Motivation

In a circuit, post-silicon tuning buffers can change the point in time in which the clock edges reach the flip-flops by modifying the clock path delays to flip-flops after manufacturing. This can be done for each chip individually which is important because process variations result in unique features of chips. The clock tuning by post-silicon buffers can be explained using Fig. 2.1a, where four flip-flops are connected by combinational paths forming a loop. Without the four clock tuning buffers connected to the flip-flops, the minimum clock period of this design is 8 since that is the delay of the longest combinational path depicted between F1 and F2. With the buffers, the minimum clock period can be tuned to a value of 6 by adjusting the clock delays and moving the clock edges. The corresponding tuning values can be seen in Fig. 2.1a next to each post-silicon buffer. Now, the path between F1 and F2 can finish signal propagation with a clock period of 6 because the rising clock edge of F2 is shifted by 2 units by the buffer value x_2 leaving the path 6+2=8 time units. This reduces the timing budget for the path between F2 and F3. However, the buffer value x_3 delays the clock edge of flip-flop F3 by 3 units leaving the path the required 6-2+3=7 time units. Analogously, the other paths fulfill their timing requirements.

Note that if the tuning values are unbound, there is an infinite number of solutions for the tuning values to solve timing requirements with a clock period of 6. For example, we could add 1 to every buffer value x_1 to x_4 and still achieve a minimum clock period of 6. This would, however, be unfavorable since we would then need four tuning buffers. In the configuration in Fig. 2.1a, we only need three buffers because buffer value x_1 is 0 and thus, F1 does not require a tuning buffer.

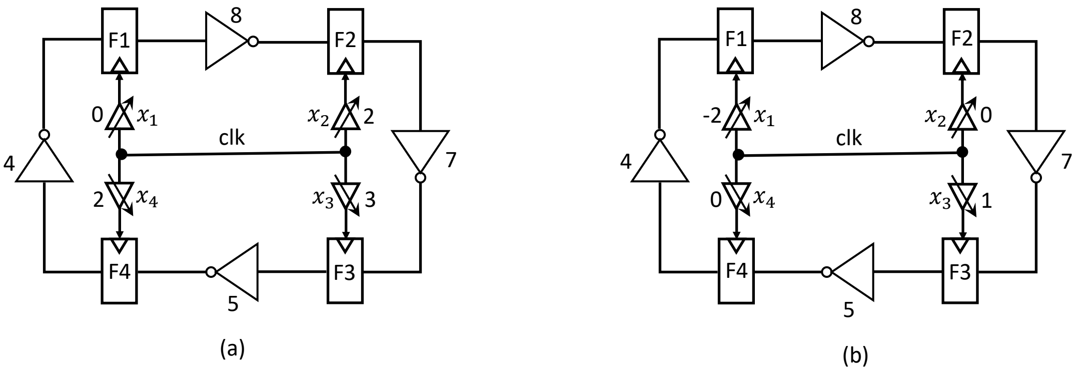

Fig. 2.1 Clock period reduction using post-silicon tuning buffers. Based on Fig. 2 in [8].

By allowing negative buffer values, we can reduce the number of tuning buffers in our example even further to two, depicted in Fig. 2.1b. This is achieved by subtracting every buffer value in Fig. 2.1a by 2 time units. As a result, the buffer value x_1 is -2 which means the clock edge of F1 is not shifted back, but moved two time units earlier than our reference clock edge. In other words, the timing slack of the path between F4 and F1 is shifted clockwise to the critical path between F1 and F2 directly instead of using three counterclockwise shifts via the other flip-flops. This negative delay can be introduced by shortening the original clock path. The original clock path without tuning would lead to a clock signal arrival at the predefined refence time which is called 0. In this manner, negative buffer values can be used to decrease the number of required post-silicon tuning buffers reducing the area and configuration costs. Thus, the goal of our buffer insertion algorithm is to find the smallest number of buffers with which the chips can be tuned to better performance after manufacturing.

In the simple timing example in Fig. 2.1, setup and hold times are omitted, i.e. set to zero. Of course, for the further analysis, they must be included in the timing constraints to have a proper representation of the actual problem. Adding clock tuning buffers changes the timing constraints for the circuit. This can be explained using Fig. 2.2. Since we have the same application, the timing constraints are also the same and thus we use the timing model given in [2]. In Fig. 2.2, two flip-flops, FF$_i$ and FF$_j$, are connected to each other by logic gates. If the clock signal changes from low to high at reference time 0, flip-flops i and j are going to notice this switch after a delay of x_i and x_j, respectively, introduced by the corresponding post-silicon tuning buffers. Thus, the corresponding setup and hold time constraints look like the following.

$$x_i + d_{ij,max} \leq x_j + T - s_j \tag{2.1}$$

$$x_i + d_{ij,min} \geq x_j + h_j \tag{2.2}$$

with the introduced delay values x_i and x_j, the maximum and minimum combinational delays $d_{ij,max}$ and $d_{ij,min}$, the setup and hold times s_j and h_j of flip-flop j and the clock period T. Without the introduced buffer delay values, these inequations reduce to the well-known/ general timing constraints of digital circuits. A manufactured chip has fixed values for all the variables in these two inequations except the buffer delay values which are thus determined by linear programming.

Buffer delay values have a limited range due to area constraints. To fulfill the objective of keeping the buffers as small as possible, their ranges should be kept as small as possible which can be represented in the following way.

$$l_i \leq x_i \leq l_i + \tau_i \tag{2.3}$$

where the lower bound of buffer i is l_i and the range is τ_i. Here, x_i can only take discrete values. Additionally, the tuning values in [2] have a maximum of 20 discrete steps, i.e. the range τ_i can have a maximum value of 20 multiplied by the buffer value resolution of each circuit.

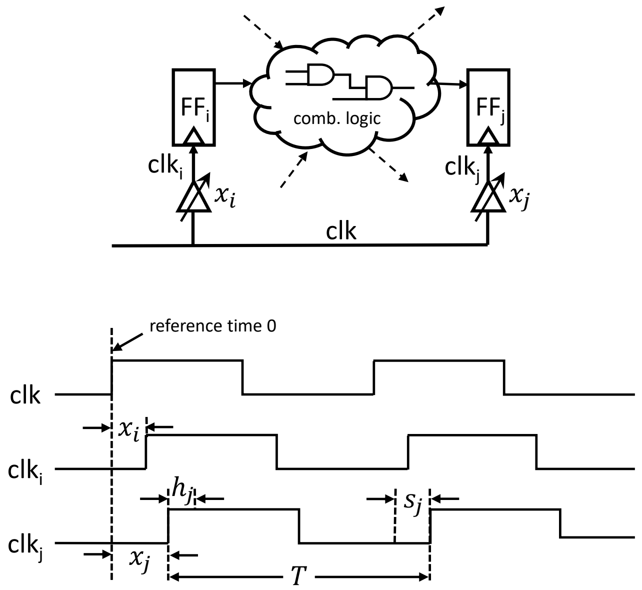

Fig. 2.2 Timing of circuits with tuning buffers in [2].

During design phase, the combinational delays $d_{ij,max}$ and $d_{ij,min}$, as well as the setup time s_j and the hold time h_j are considered to have statistical distributions due to process variations. They are sampled in the sampling-based approach in [2], i.e. manufactured chips are emulated, and the values become fixed. Of course, the delay values are also statistical since the buffers are also subject to process variations. However, these can be combined with the other statistical variables in the two inequations and thus the delay values are treated as fixed variables in [2]. In the proposed machine learning approach, the timing values are the inputs and the buffer delay values are the outputs to be predicted.

Note that Fig. 2.2 also explains the model of negative buffer values discussed above. In those cases, the rising edge of the delayed clock values would be moved into the other direction relative to the reference time 0, i.e. to the left.

Post-silicon tuning buffers are inserted between logic synthesis and physical design of the chip. Logic synthesis must be completed so that timing behavior can be analyzed and combinational delays can be evaluated. Though buffers are tuned after manufacturing, their locations and sizes are required in the physical design phase and therefore need to be calculated before. Thus, due to the statistical nature of circuit design, the circuits must be modeled statistically to get the path delays. One way of doing this is a sampling-based approach in which every simulation emulates one produced chip with its unique properties.

The challenge of the random variables in the setup and hold constraint inequations is solved by using ILP-based Monte Carlo simulation. Monte Carlo is used to sample the delays of the circuits which are the inputs to the ILP model. Then, in a relatively long computation, the tuning values of the buffers are determined. With a large number of samples, a trend can be seen where to insert buffers because the flip-flops which are critical to the yield are tuned more often than others. One ILP solution must be found for every sample which requires a lot of valuable computation time. On average, 75% of the samples do not require tuning or cannot be “saved” even with tuning. This leaves 2,000 to 3,000 samples per circuit in which the ILP model must determine adjustment values for buffers. Thus, every single ILP iteration has a computation time of at least 0.025 s (for circuit s9234) and up to 2.5 s (for circuit pci_bridge32). On the other hand, after training a machine learning model, the prediction time for one sample has a magnitude of 10^{-6} seconds because it only takes one matrix multiplication. This means the machine learning model can achieve a speed-up of four to six orders of magnitude in prediction. Thus, we include a model learning the locations and sizes of the buffers from a smaller number of samples first and then predicting those for all the other samples avoiding the usage of the ILP solver.

3 Buffer Insertion Using Machine Learning

We apply machine learning methods to the algorithm proposed in [2]. Since the choice of the machine learning method depends strongly on the data, we analyze the data used in [2] thoroughly. The data is drawn from Monte Carlo simulations of the delays of different benchmark circuits and then pre-filtered. The biggest challenge is the small number of samples in comparison to their relatively high dimensionality. To combat this, we develop one machine learning model for each flip-flop in a circuit rather than using one model for the entire chip. This reduces the dimension of the samples significantly and allows for the usage of simple machine learning models. The flow of actions can be seen in Fig. 3.1.

We propose a two-step solution. In the first step, a classifier is developed to predict whether a buffer is tuned or not. The predominant challenge here is the class imbalance in the data because the goal is to keep the number of tuned buffers low. Effectively, a classification problem arises with a minority class representing the times that each buffer is tuned and a majority class for the cases where there is no tuning. In a second step, the sizes of the buffers and their ranges are predicted by using a regression model. Since the negative class would distort the buffer values immensely, only the samples that were predicted positively by the classifier before are used. However, this results in a very small number of samples to use for training which is essentially the big challenge for this step. Thus, only simple machine learning models for regression are considered.

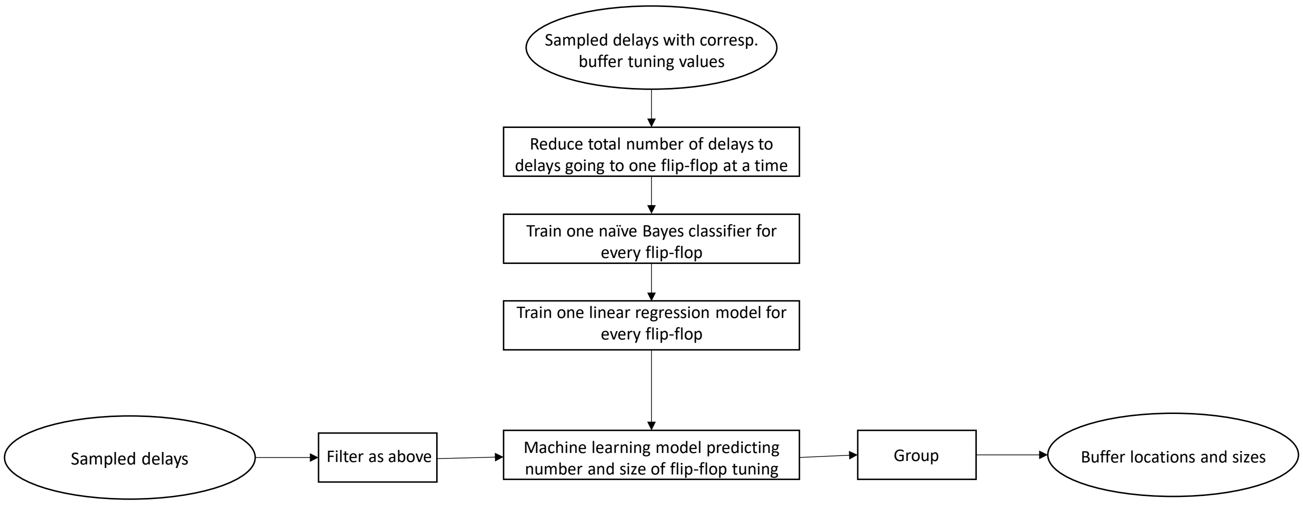
Sampled delays with corresp. buffer tuning values
Reduce total number of delays to delays going to one flip-flop at a time
Train one naïve Bayes classifier for every flip-flop
Train one linear regression model for every flip-flop
Sampled delays
Filter as above
Machine learning model predicting number and size of flip-flop tuning
Group
Buffer locations and sizes

Fig. 3.1 Training of two-stage ML model and prediction of buffer locations and sizes.

3.1 Data Preparation

The primary inputs of the proposed method are the sampled delay values of different circuits. Despite the pre-filtering of the delay values that is performed in [2], their dimensionality is still very high. In all the tested benchmarks, the dimensionality of the circuit delays exceeded the limited number of samples we allow to keep the runtime short. In other words, the high dimensionality of the data would require an unreasonable amount of training samples. Thus, we need to decrease the dimensions drastically. This is done by developing one model per flip-flop instead of one per circuit. In these models, we only consider the delays in the fan-in of one flip-flop one at a time reducing the input dimensions by more than two orders of magnitude, as explained in Fig. 3.2. To determine the delay value x_i, we develop one machine learning model for flip-flop i and only consider the delays represented by the solid arrows neglecting the dashed ones. This is valid because the tuning of a flip-flop almost exclusively depends on the delays of the paths leading to it. However, the other buffers of the circuit are not neglected in this model. Since they are considered for in the ILP solution used in training, they are automatically included in our approach.

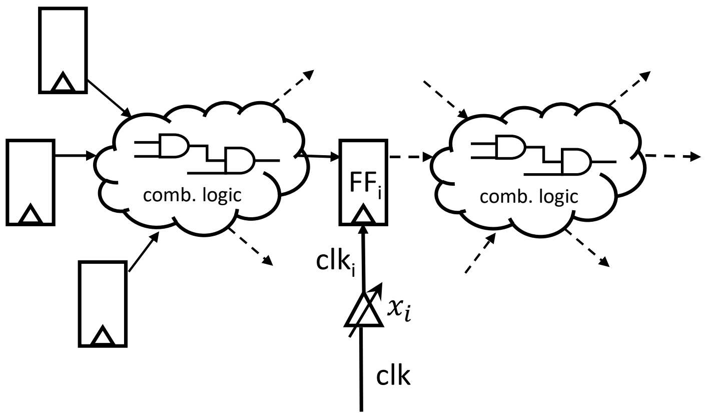

Fig. 3.2 Dimensionality reduction by using one model per flip-flop with only fan-in delays.

3.2 Classification

The first step in the proposed method is a binary classifier. As the name predicts, a binary classifier decides whether some input belongs to one class or the other. This is usually done in a probabilistic approach, i.e. the model determines how likely it is for the input to belong to one class or the other. In addition to the above-mentioned class imbalance, we need to tackle the problem of finding the best suited classifier for the given data. The basic idea is finding the simplest model that can still guarantee a satisfactory accuracy with a reasonable runtime. In finding the best method, we compare the features of different classifiers but most importantly, their accuracy after being applied to the data. Comparable features are for example if the model is discriminative or generative and if the model is parametric or non-parametric [9].

Logistic regression is a linear classification model, in which a linear combination of input and model parameters is plugged into the sigmoid function which returns values from 0 to 1. Neural network classifiers are a series of two or more logistic regression models stacked on top of each other and can thus model more complex and nonlinear behavior. Support vector machines are a kernel-based approach which find a hyperplane separating the data by using some of them as decision boundary.

Discriminative models, like neural networks (NN), usually maximize the conditional log likelihood $\sum_{i=1}^{N} \log p(y_i | \boldsymbol{x}_i, \theta)$ during training to tune the parameters. However, when training a generative model, like a Naïve Bayes classifier (NBC), usually the joint log likelihood $\sum_{i=1}^{N} \log p(y_i, \boldsymbol{x}_i | \theta)$ is maximized. This means that a generative classifier models the distribution of the features, which can be rather complex, whereas a discriminative one only models relatively easy class posteriors, e.g. a simple sigmoidal function. Both the NBC and NN classifier are parametric models, i.e. they have a fixed number of parameters. Non-parametric classifiers, like support vector machines (SVM), do not make these assumptions and are thus more flexible [9].

Training for NBCs is significantly easier and hence faster than for logistic regression. While the latter requires the solution of a convex optimization problem, Naïve Bayes classifiers can be fit very easily by counting and averaging. A big advantage of logistic regression, on the other hand, is that the input data can be preprocessed. This means, instead of using x the model can use some transformation $\varphi(x)$ and the classification could be easier in the new feature space. In NBC, this can be difficult when modelling feature distributions since the new features are correlated complexly. Additionally, Naïve Bayes classification makes strong independence assumptions. The features are treated as completely uncorrelated to each other which is mostly not valid. However, the NBC performs well in many applications because the posterior probability of the right class does not necessarily need to be perfect if it is higher than that of the wrong class in a binary classification [9].

Neural networks require too many training samples for our application. Logistic regression models perform poorly on the input data and are thus excluded from the discussion. Support vector machines have a long runtime and need the SMOTE algorithm, explained in chapter 3.2.3., to produce good results. Thus, because of its simplicity and short runtime, we evaluate the performance of NBC for the buffer insertion problem.

3.2.1 Naïve Bayes Classification

Naïve Bayes Classifiers owe their name to the Bayes' theorem

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}. \tag{3.1}$$

The idea is to determine posterior class probabilities $p(y|x)$ for classification by evaluating prior class probabilities $p(y)$ and the feature distribution $p(x|y)$, called likelihood in Bayes' theorem. The classifier is called "naïve" because it assumes that the features are conditionally independent given the class label [9]. That way, if D is the number of features, the feature distribution (class conditional density) can be written as a product of one dimensional densities

$$p(\boldsymbol{x}|y = c) = \prod_{j=1}^{D} p(x_j|y = c). \tag{3.2}$$

When fitting a naïve Bayes classifier, we determine the model parameters θ by computing the maximum likelihood estimate (MLE). Therefore, we choose the θ that maximizes the joint conditional probability $p(\boldsymbol{x}_i, y_i|\theta)$ or equivalently, its logarithm $\log(p(\boldsymbol{x}_i, y_i|\theta))$ [9].

During prediction, we determine the probability of y being class c given the training data D and a new sample $\boldsymbol{x}$, i.e. the probability $p(y = c|\boldsymbol{x}, D)$ [9]. This is done for all classes and then the sample is assigned to the class with the highest posterior class probability. Thus it appears that the independence assumption may be wrong but still leads to a good classifier. The posterior probability might not be very accurate, but as long as it is higher than the wrong class, the classifier predicts the right class and performs perfectly.

3.2.2 Implementation

We implement the classification in MATLAB using the existing machine learning functions to get a fast evaluation and comparison of the classification results. We do not use the machine learning toolbox in MATLAB since it may be able to compare different classification methods but can only develop one model at a time. As we specified above, we build one model for every flip-flop in a circuit. This results in a range from a couple of models per circuit for the smaller ones to up to 100 models for the larger benchmark circuits. Obviously, this is unhandy to be handled by the machine learning toolbox. In addition, the predominant interest is in finding the best overall accuracy for every circuit and not optimal individual accuracies for single flip-flops. For the classification problem, the MATLAB functions $fitcnb$ and $fitcsvm$ are compared.

The MATLAB function $fitcnb$ trains a multiclass naïve Bayes model which is a binary classifier in this application. Its simplest syntax is $NBCmodel = fitcnb(X, Y)$. In this form, the function takes the inputs X, which is a matrix containing the features as columns and predictors as rows, and Y, which is a vector with the class labels, and returns the trained naïve Bayes classifier $NBCmodel$. Several parameters of this function can be adjusted. One is, for instance, the kernel smoother function which we set to a normal (Gaussian) function. Other tunable parameters are the priors of the classes. They can be set to uniform, i.e. all class prior probabilities are set to the inverse of the number of classes which would be one half each. The default setting here, however, is the empirical setting which means that the class prior probabilities are the relative frequencies of every class in the response variable Y which is used in training. Further interesting settings are the weights that are assigned to the features. With this setting, one could prioritize certain input features given the knowledge of which features are more important than others. By default, this is a vector of ones [10].

The MATLAB function $fitcsvm$, on the other hand, trains a support vector machine (SVM) model for binary classification. The syntax is the same as above $SVMmodel = fitcsvm(X, Y)$. Again, the variable X contains the predictors in its rows and the variable Y contains the class labels and the SVM classifier $SVMmodel$ is returned by the function. Again, there are a lot of parameter settings to be considered and changed. One crucial setting is the kernel function. The default setting for two-class classification is linear, i.e. the software fits a hyperplane in the feature space to try to separate the two classes. If that is not sufficient or the data is not separable in the feature space, Gaussian or polynomial kernels can be used to perform a space transformation into a different linear space and fit a hyperplane separating the transformed predictors. Additionally, this can be altered by using kernel scaling which is a positive scalar all predictor values are divided by before performing the transformation. The second important alternation that can be made is standardizing the predictor data. If this flag is set to true, the software centers and scales each feature in the input data by the feature mean and standard deviation, respectively. Note that MATLAB might train the classifier with the standardized predictors but it stores the unstandardized data. Like above, both the class probability priors and weights can be set by the user. However, a different and more important parameter is the box constraint. It is defined to be controlling the tolerance of margin-violating observations and thus, it helps to prevent overfitting. In other words, increasing the value of the box constraint parameter will result in a smoother decision boundary by assigning less support vectors with the drawback of longer training times. On the contrary, a small box constraint will lead to a better fitted decision boundary with a higher chance of overfitting [11].

3.2.3 SMOTE

As mentioned above, we work with very imbalanced datasets: the minority to majority class ratio is at best 1:5 but on average below 1:10. This means, the machine learning models will be trained to always predict the majority class to get high accuracies. Obviously, this is very unfavorable. There are several possibilities to take on the challenge imposed by imbalanced datasets that have been developed by researchers over the past couple of years. In general, two basic ideas exist to handle this problem: either the majority class is under-sampled or the minority class is oversampled. Due to the small amount of data available, we focus on the latter. One of the oversampling methods is called "Synthetic Minority Over-sampling Technique" (SMOTE) and has been proposed in [12]. The basic idea is – just as the name predicts – to produce synthetic samples of the minority class and add them to the training samples to prevent the system from only predicting one label. The pseudo-code given in [12] is evaluated and the SMOTE technique is implemented in MATLAB to see how it affects the different learning algorithms. SVM classification can be improved with SMOTE while NBC cannot. However, NBC produce better results without SMOTE than SVM with SMOTE and hence, NBC is used.

3.3 Regression

In the second stage, a regression model is built to predict the sizes and ranges of buffers. Here, we only use those samples that were classified as positives before. If one sample is predicted as negative, its tuning is 0 and no tuning value prediction is needed. This flow can be seen in Fig. 3.3.

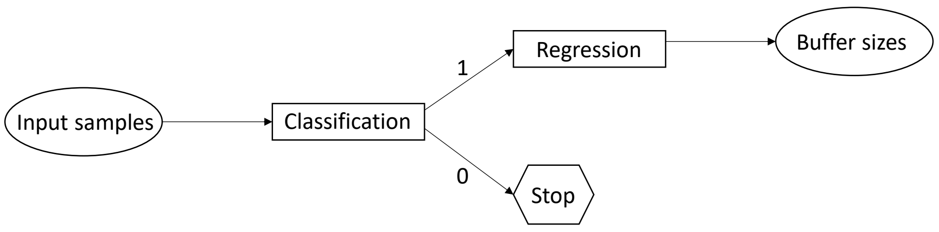

Fig. 3.3 Classification and Regression flow.

Analogously to the classifier, the optimal regression model must be found trading off accuracy against runtime. However, as an additional constraint, the number of training samples is extremely low since only a very small number of samples are predicted positive in the classification step. Hence, only simple models, for which small training sizes are sufficient, are considered. Linear regression is an example for a parametric regression model and is a very commonly used machine learning technique. For this application, we consider linear regression since it is fast and performs better on small dimensional data.

3.3.1 Linear Regression

Linear regression owes its name to the fact that the response is modeled as a linear combination of the input. In the simplest case, this gives the linear combination of the input variables $f(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x}$ with the input vector $\boldsymbol{x}$ and the model weight vector $\boldsymbol{w}$. Since this model is linear in $\boldsymbol{w}$ and $\boldsymbol{x}$, it is very limited. It can be extended, though, by replacing $\boldsymbol{x}$ with some fixed nonlinear functions $\varphi(\boldsymbol{x})$, called basis functions, which yields $f(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{w}^T\varphi(\boldsymbol{x})$. Common choices for the basis functions are polynomial, Gaussian and sigmoidal. Note that this model is still a linear function of the weight parameters and is thus a linear regression model. The target value y, i.e. the response value to every input, is assumed to be the model output $f(\boldsymbol{x},\boldsymbol{w})$ with some Gaussian noise ϵ, so $y = f(\boldsymbol{x},\boldsymbol{w}) + \epsilon$, where ϵ has zero mean and variance β^{-1}. Making this more explicit, the model can be rewritten as the conditional probability density

$$p(y|\boldsymbol{x},\boldsymbol{w},\beta) = \mathcal{N}(y|f(\boldsymbol{x},\boldsymbol{w}),\beta^{-1}) \tag{3.3}$$

where $\mathcal{N}$ is the normal distribution [13].

When fitting the model to the data, we want to find the optimal set of parameters w. The most common way to do this is the maximum likelihood estimation, which means we try to find the parameters that maximize the probability p when we plug in the training data. This is equivalent to maximizing the log likelihood $\ln(p)$ or minimizing the negative log likelihood $-\ln(p)$. This is equivalent to minimizing the residual sum of squares (RSS)

$$RSS(w) = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{w}^T\varphi(\boldsymbol{x}_n)\right)^2 \tag{3.4}$$

which is a measure of the distance of the predicted values to the target values [13].

In words, the goal is that the predicted values are as close to the target values as possible. For a 1D model, this is made visual in Fig. 3.4, in which the RSS is the sum of the squared distances of the target values to the predicted curve.

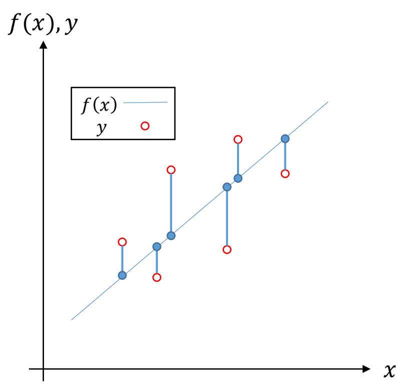

Fig. 3.4 Minimizing the RSS is minimizing the sum of the lengths of the vertical blue lines connecting the training points (red) to their approximations (blue). The blue diagonal line $f(x) = w_0 + w_1x$ is the least squares regression line. Based on Figure 7.2 in [9].

3.3.2 Implementation

For regression, again existing functionality in MATLAB is used to get fast evaluation and comparable results. Again, the machine learning toolbox is unhandy for this application because it can only compare the accuracies of different machine learning methods for one model at a time. Since we develop a lot of small models, i.e. one for every flip-flop, and a good overall accuracy per circuit is the goal, all the learning is performed in one run and compare the different methods afterwards. In this step, the MATLAB functions $glmfit$ and $fitglm$ are compared for linear regression.

To perform linear regression in MATLAB, there are two ways. One can either use the function $glmfit$, which was introduced before 2006, or the function $fitglm$, which was introduced in the 2013 MATLAB release. As one can tell, $fitglm$ is the newer version and is supposed to replace $glmfit$. However, both functions fit a generalized linear regression model to the input data. The differences are in the way they are called by the user. The simplest syntax for $glmfit$ is $w = glmfit(X, Y, distr)$ where the variable X contains the predictors in its rows, the variable Y contains the response and the variable $distr$ specifies the distribution of the response variable. The output w of this function is a vector of dimensionality $\dim(X) + 1$, which contains the coefficient estimates for the fitted model. The additional dimension is the introduced bias. The function $fitglm$, on the other hand, has the same syntax as the classification functions described above. It is called by $mdl = fitglm(X, Y)$ where X is a matrix with the predictors and Y the vector with responses. In contrast to $glmfit$ and analogously to the classification functions above, the output is the entire generalized linear regression model mdl. This means, mdl does not only contain the coefficient estimate, but also the training data, the model description and diagnostic information. A further difference is that $fitglm$ is more flexible with the inputs, which do not have to be matrices, but could also be tables, for example. Both functions can model the distribution of the response variable in five different ways, which are normal, binomial, Poisson, Gamma and inverse Gaussian. The default setting is normal, which is the "classical" linear regression approach, in which a linear curve is fit into the data. In addition to the distribution, there are some other parameters which are not investigated further since they are not useful in our application of these functions [14, 15].

In the proposed approach, finding the right buffers to be tuned is more important for the final result than the actual size of the buffer in every single iteration. Once the buffer locations are present, which is done in the classification step, the ranges of the buffers are determined in the regression step. However, these ranges always include the value 0 since no matter how many times a buffer is tuned, most of the times its tuning value is 0. Thus, we only need to determine one more value, which is the maximal tuning value for positive buffers or the minimal tuning value for negative buffers. Buffers which are tuned positive in some samples and negative in others are so rare that they can be neglected. This makes the value 0 the second range boundary automatically. Furthermore, every value that lies in between the two range boundaries does not contribute to the decision of the size of a buffer. Of course, this does not mean that those values are completely unused because the fact that there is tuning at all is part of the decision of buffer locations in classification. Hence, in regression, further reduction and simplification of the input dimensions and model complexity is considered. Therefore, we primarily focus on the simplest regression model, namely linear regression. In addition, the input dimensionality is reduced to four using only the maximum, minimum, mean and median of every predictor.

3.4 Grouping

In the last step of the buffer insertion algorithm in [2], the buffers are grouped together to reduce the total number of physical buffers on the chip aiming at tuning the highest number of flip-flops possible. This is done in a two-step grouping algorithm evaluating both the tuning value correlation and the distance of the flip-flops to be grouped. If these two decision criteria are met, the flip-flops are connected to the same physical buffer and the tuning values are shared. In the first step, the correlation coefficient of individual buffer pairs is calculated. If multiple buffer show a correlation coefficient above a certain threshold (here 0.8) and their distance is smaller than a pre-defined number (here ten times the minimal distance between buffers), they are grouped together and connected to the same physical buffer. We develop a very similar grouping algorithm with the only information available, i.e. the sample delays. The same two decision criteria are used: the similarity of the tuning values and the distance. First, the correlation of the tuning values is calculated. However, for the distance, we simply check if two flip-flops, modeled by two nodes, share one edge. If two buffers fulfill these criteria, they build a group. If further buffers have a strong correlation to both buffers in a group and a short distance to either flip-flop, they are added to the group. With these algorithm, there might be buffers that cannot be grouped with others. In that case, one buffer would tune a single flip-flop. However, these are only kept if they have a high number of tunings which means they are more likely to improve the yield significantly.

4 Experimental Results

The experimental results are split into two parts. In the first part, the results of the machine learning techniques are presented, i.e. how well do the models fit the data and how accurate are the predictions. This is done for different circuits, different machine learning methods and different amounts of training for both classification and regression. In the second part, we use these results to see how the overall yield of the circuits change after buffer insertion with machine learning in comparison to the results in [2], which are the results of buffer insertion without machine learning.

4.1 Evaluation of the Machine Learning Methods

The proposed machine learning methods were implemented in MATLAB and tested using a 2.20 GHz CPU with four threads. We demonstrate the prediction accuracies for classification and regression for different amount of training and different predictor inputs using the ISCAS89 benchmark circuits.

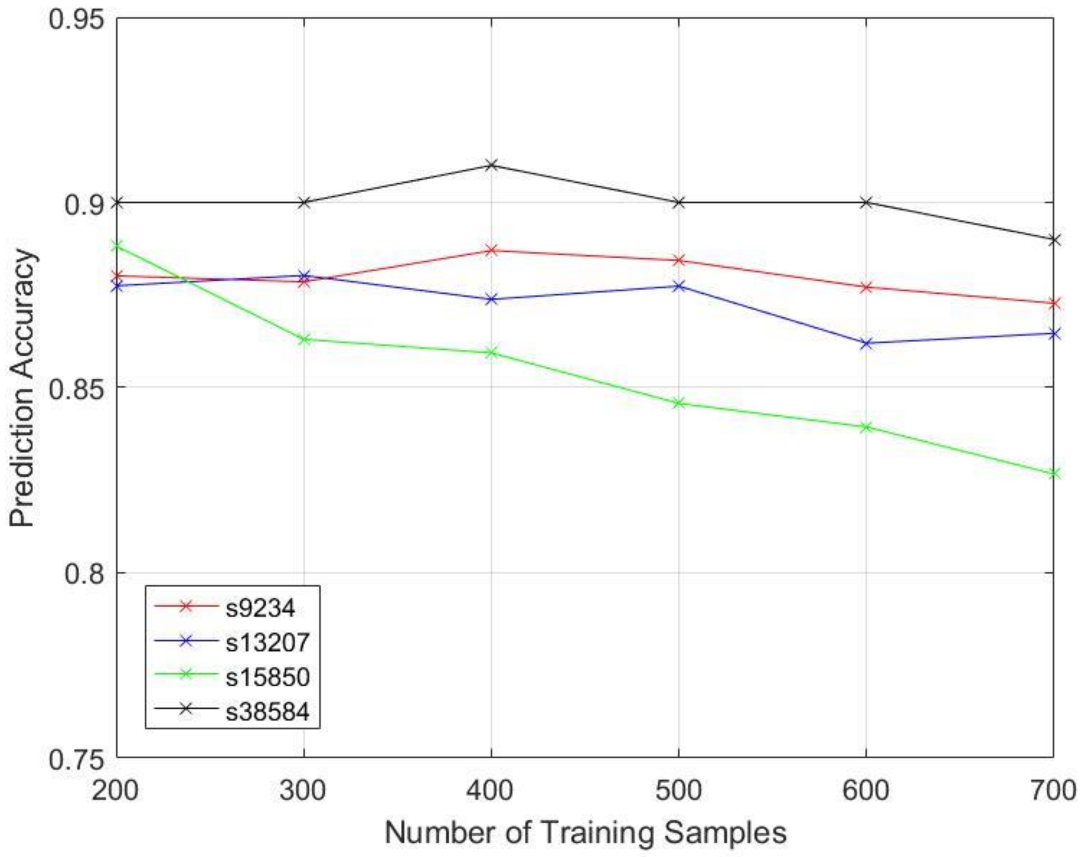

Fig. 4.1 Prediction accuracy of Naive Bayes Classifier with unmodified predictors.

We first show the effect of different amount of training on the prediction accuracy of our classification methods for different circuits. Here, prediction accuracy is the ratio of true positives plus true negatives over the entire test set, i.e. how many examples are predicted correctly. In Fig. 4.1, the prediction accuracy for Naïve Bayes Classification can be seen for the four ISCAS89 benchmark circuits plotted against different amount of training. Despite the significant change of training samples, the prediction accuracies do not vary a lot. The least amount of variation can be seen in circuit s9234 with only 1.5% variation in prediction accuracy, whereas s15850 shows the highest amount of variation with still only 6%. This shows that naïve Bayes classifiers show a good accuracy with even small amount of training. In general, our experiments show even better results for smaller amounts of training than for larger amounts. With the freedom to choose any amount of training, we can find an accuracy of 88% or better for every circuit.

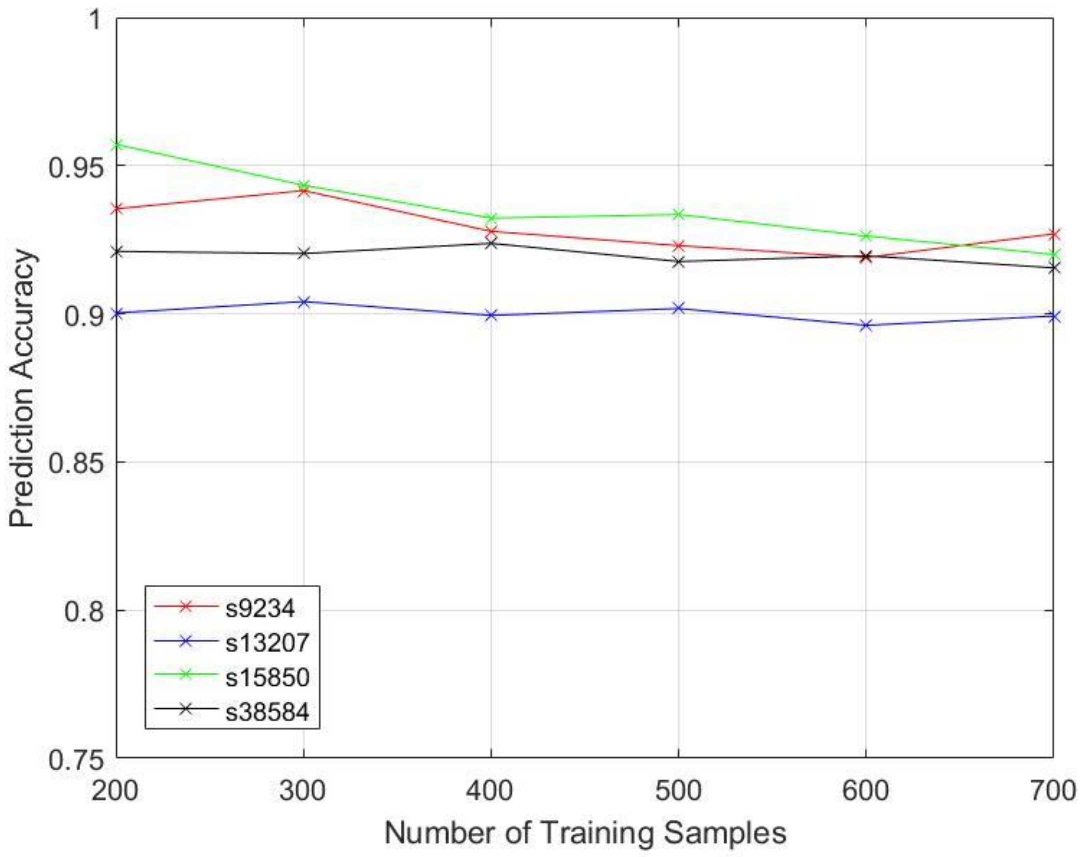

Fig. 4.2 Prediction accuracy of Naive Bayes Classifier with modified predictors.

We take this one step further by modifying the input. Instead of taking the entire predictor with all delay values going to a flip-flop, we only take four values. The new predictors are only the median, mean, minimum and maximum values of the previous predictors. In Fig. 4.2, the prediction accuracy of the Naïve Bayes Classification is shown with the same benchmark circuits and the modified input plotted against different amount of training. For every circuit, this modified classification gives better results (90% and more) than using the entire predictor. Again, the prediction accuracy does not change by a lot when the training set is increased. In fact, smaller amounts of training show better classification accuracies than larger ones. This proves that one more time that Naïve Bayes Classifiers perform well with even a small number of training.

For regression, we also analyze the effect of different amount of training on the prediction accuracy. Here, prediction accuracy shows how close the predicted value is to the actual, ILP-calculated value. In

Fig. 4.3, the prediction accuracy of linear regression for the four ISCAS89 benchmark circuits is plotted against different amount of training. Several results can be drawn. First, the accuracies for different circuits vary by a lot. For example, the circuit s13207 shows prediction accuracies around 80% and higher, whereas circuit s9234 starts from approximately 54%. Second, they show different behavior for increased training. While circuit s9234 generally shows better prediction with rising training, this is not true for the other circuits. All in all, the prediction accuracies seem not high enough.

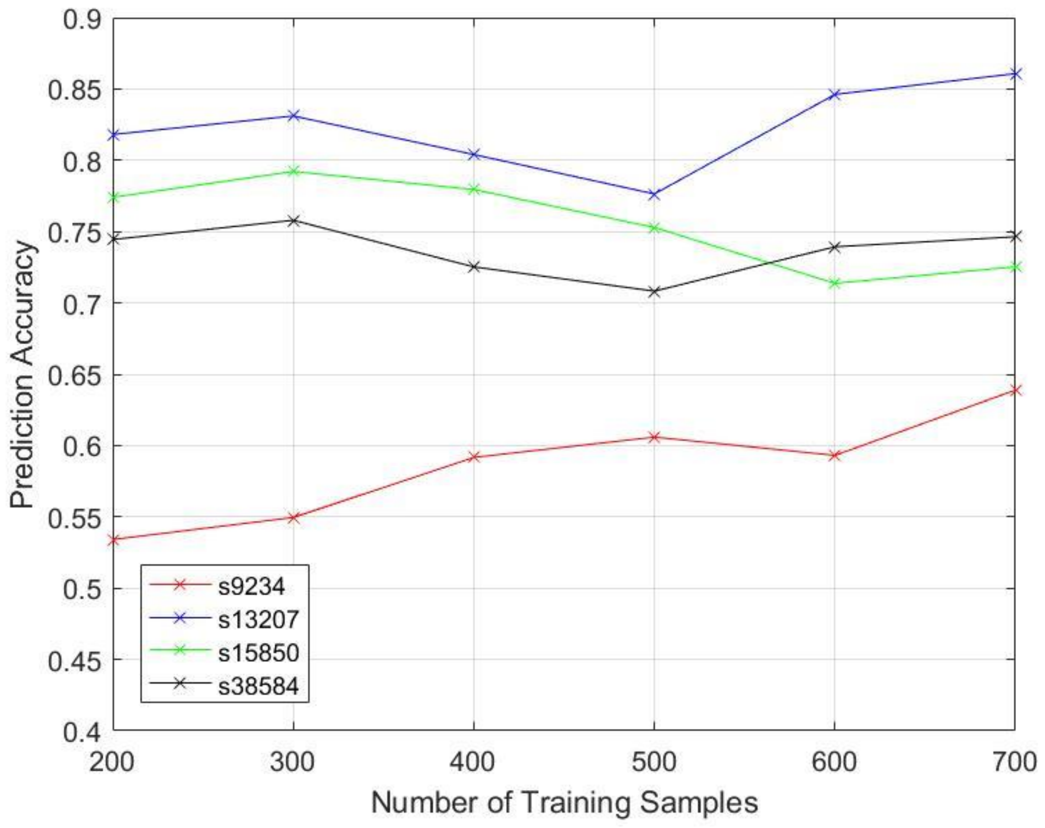

Fig. 4.3 Overall accuracy of linear regression for unmodified predictors.

However, the final ranges of the buffers are more important than predicting every single value correctly. Since 0 is always included in the ranges, we need to only predict the minimal value for negative buffers and the maximum value for positive ones. If we compare the ranges obtained by the predicted buffer values to the ranges obtained by the actual, ILP-calculated buffer values, we get the results in Fig. 4.4. For every circuit and every training size, the prediction accuracies are very high with 90% and above.

Analogously to classification, we take this one step further by modifying the input. Instead of taking the entire predictor with all delay values going to a flip-flop, we only take four values. The new predictors are only the median, mean, minimum and maximum values of the previous predictors. In Fig. 4.5, the prediction accuracy of linear regression with the same benchmark circuits and the modified input is plotted against different amount of training. The prediction accuracy is improved for all circuits.

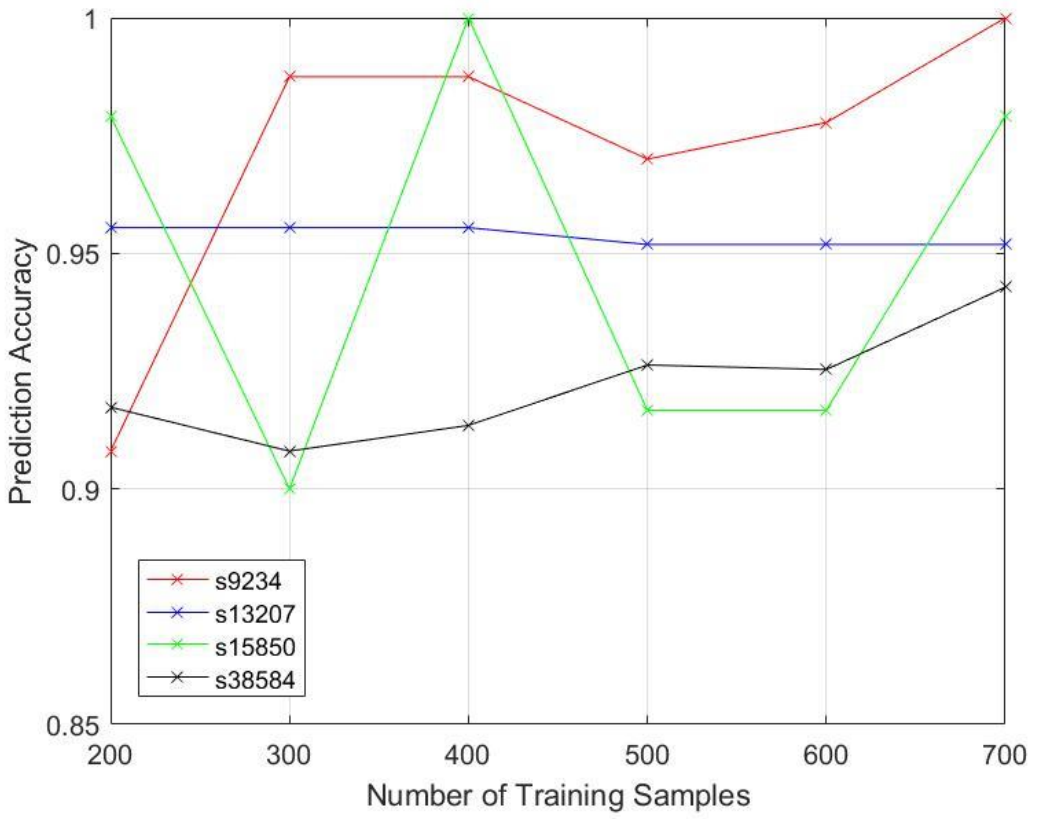

Fig. 4.4 Range accuracy of linear regression for modified predictors.

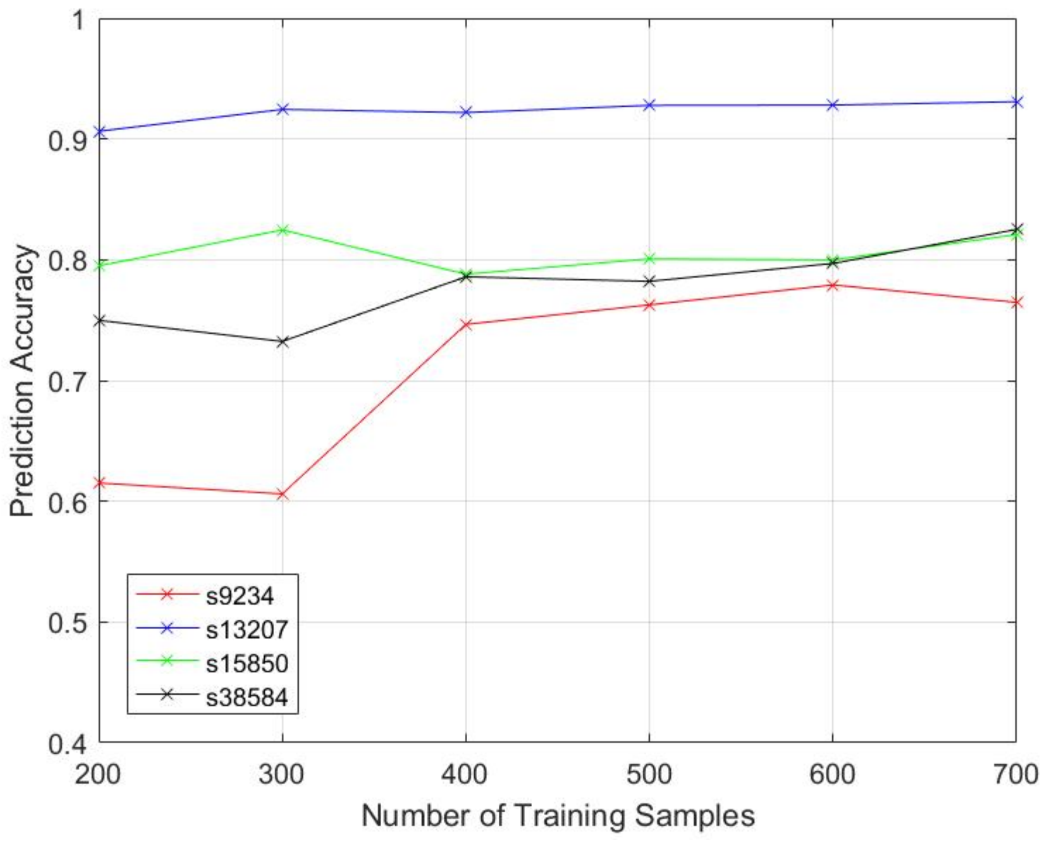

Fig. 4.5 Overall accuracy of linear regression for modified predictors.

Again, the final ranges of the buffers are more important than predicting every single value correctly. In Fig. 4.6, the prediction accuracy of the ranges obtained by the predicted buffer values compared to the ranges obtained by the actual, ILP-calculated buffer values is presented. For every circuit and every training size, the prediction accuracies are high with at least 88% or more. Furthermore, except for circuit s38584, all circuits have a range prediction of over 95% for most of the training sizes.

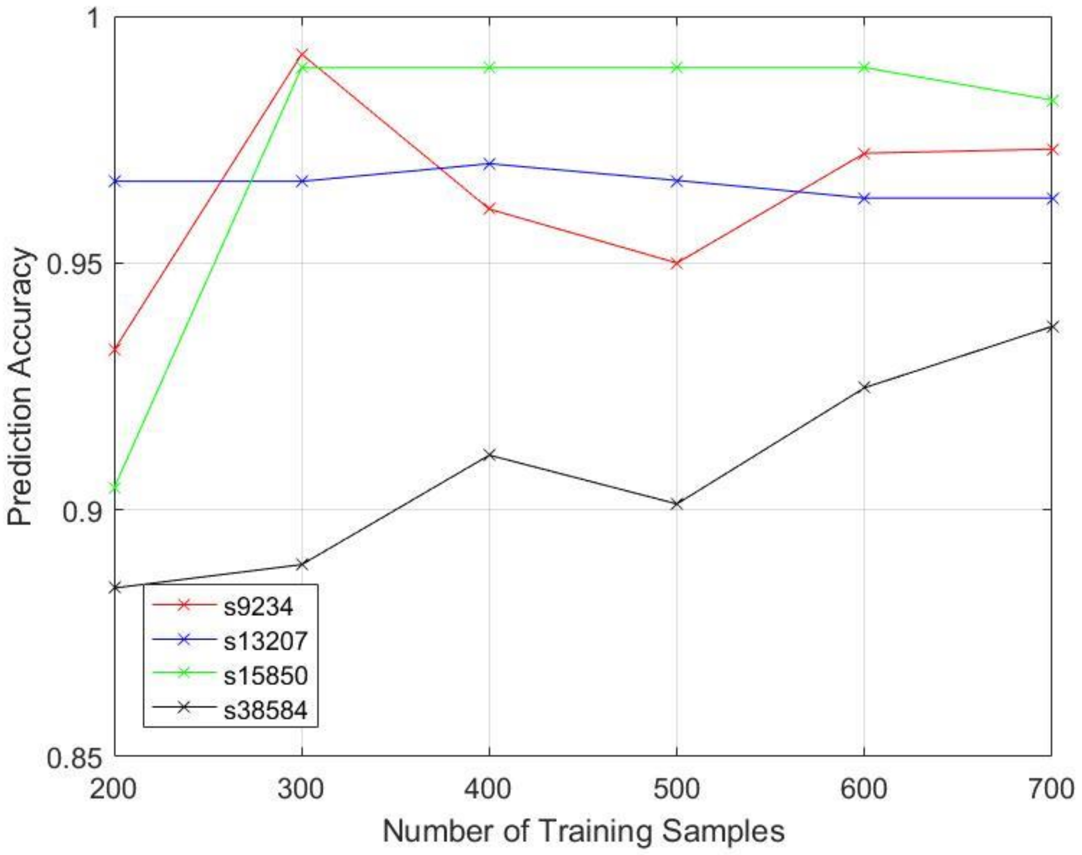

Fig. 4.6 Range accuracy of linear regression for modified predictors.

4.2 Yield and Timing Results

The proposed method was implemented in MATLAB and tested using a 2.20 GHz CPU with four threads. We demonstrate the results using seven circuits from the ISCAS89 benchmark set and from the TAU 2013 variation-aware timing analysis contest. Since we take over the data from [2], the maximum allowed buffer values are set to 1/8 of the original clock period and all tuning delays are discrete with 20 steps. We use 500 samples for training and 500 for testing.

The experimental results on the overall yield of the circuits after buffer insertion with machine learning can be seen in Table 1. First, the mean μ_T and the standard deviation σ_T of the clock period are calculated without post-silicon tuning buffers [2]. The original yields Y_o of the circuits with a clock period of μ_T, $\mu_T + 0.5\sigma_T$, and $\mu_T + \sigma_T$ are given as 50%, 69.15% and 84.13%, respectively. In contrast to these, the yield with post-silicon tuning buffers can be seen in the columns Y $(\%)$ and the yield improvement, i.e. $Y - Y_o$, is shown in Y_i $(\%)$. These show that post-silicon tuning buffers can improve the yield significantly (18.80%). The column N_b shows the number of buffers that are inserted into each circuit. This number is limited to 1% of the number of flip-flops in the corresponding circuit. We only determine the buffer locations and sizes for μ_T, but use them to additionally evaluate the yield for $\mu_T + 0.5\sigma_T$ and $\mu_T + \sigma_T$.

Table 1 Results of buffer number and yield improvement

Circuit			μ_T				$\mu_T + 0.5\sigma_T$		$\mu_T + \sigma_T$	
	n_s	n_g	N_b	Y (%)	Y_i (%)	$T(s)$	Y (%)	Y_i (%)	Y (%)	Y_i (%)
s9234	211	5597	2	52.37%	2.37%	1.3172	71.16%	2.01%	85.00%	0.87%
s13207	638	7951	6	60.26%	10.26%	1.4968	78.45%	9.30%	90.40%	6.27%
s15850	534	9772	5	68.80%	18.80%	2.2766	84.53%	15.38%	93.31%	9.18%
s38584	1426	19253	14	64.62%	14.62%	3.6592	80.90%	11.75%	91.44%	7.31%
mem_ctrl	1065	10327	10	56.51%	6.51%	17.1010	74.26%	5.11%	87.38%	3.25%
usb_funct	1746	14381	17	58.99%	8.99%	17.9384	76.28%	7.13%	88.82%	4.69%
ac97_ctrl	2199	12494	21	52.69%	2.69%	9.3191	71.85%	2.70%	85.83%	1.70%

In Fig. 4.7, we compare the yield of this method to the yield in the proposed method in [2] with respect to μ_T. We train our model with the input of [2] and use their output as response variable. However, we use a different grouping algorithm. As expected, the yield improvement of this method is similar to the one in [2] and the difference mostly stems from the different grouping.

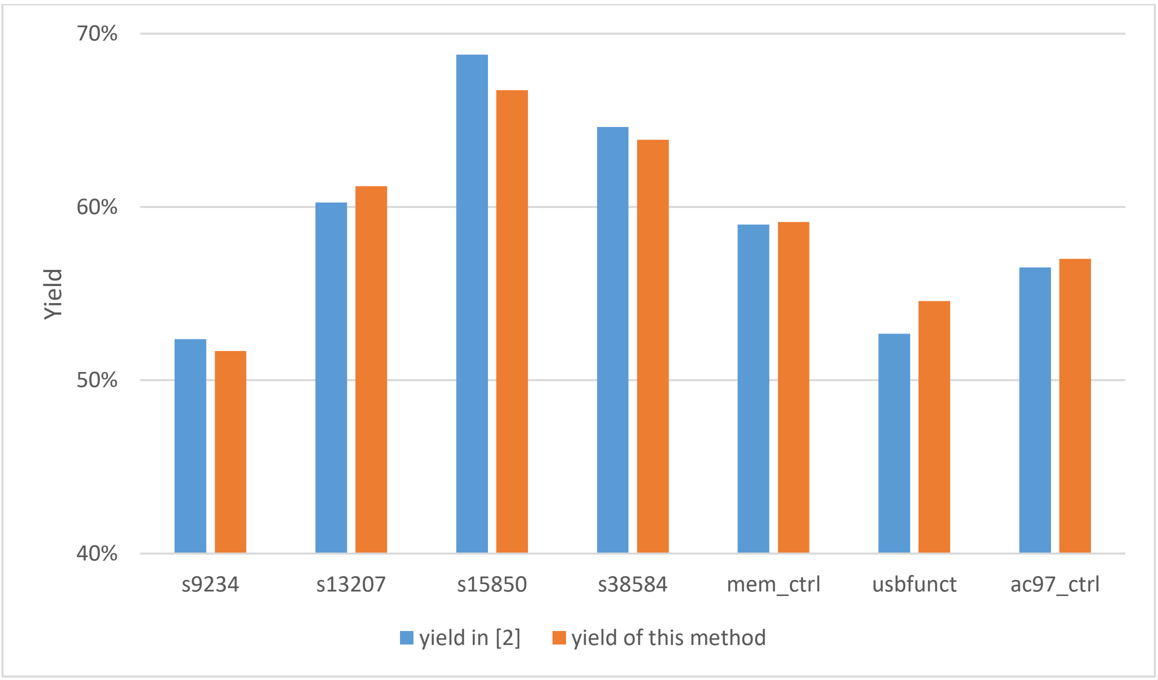

Fig. 4.7 Yield comparison.

The runtime of our method is shown in Table 1 in the column $T(s)$. It consists almost exclusively of the training time for the machine learning methods. Since the buffer locations and sizes of the first 500 samples calculated by [2] are used for training, the runtime for the ILP solution of 500 samples needs to be added to the runtime of our method for the comparison with [2] which can be seen in Fig. 4.8.

The runtime in [2] almost exclusively consists of the runtime for the ILP solutions. Thus, the speed-up equation looks like the following:

$$speed - up = \frac{runtime\ in\ [2]}{runtime\ of\ this\ method} \approx \frac{runtime\ for\ 10{,}000\ ILP\ solutions}{runtime\ for\ 500\ ILP\ solutions\ +\ training\ time}.$$

If the training time is negligible compared to the runtime for 500 ILP solutions, we achieve the maximum speed-up of 20. The speed-up can be seen in Table 2.

Table 2 Speed-up

Circuit	s9234	s13207	s15850	s38584	mem_ctrl	usb_funct	ac97_ctrl
Speed-up	13.46	16.78	16.61	19.22	15.58	17.20	18.45

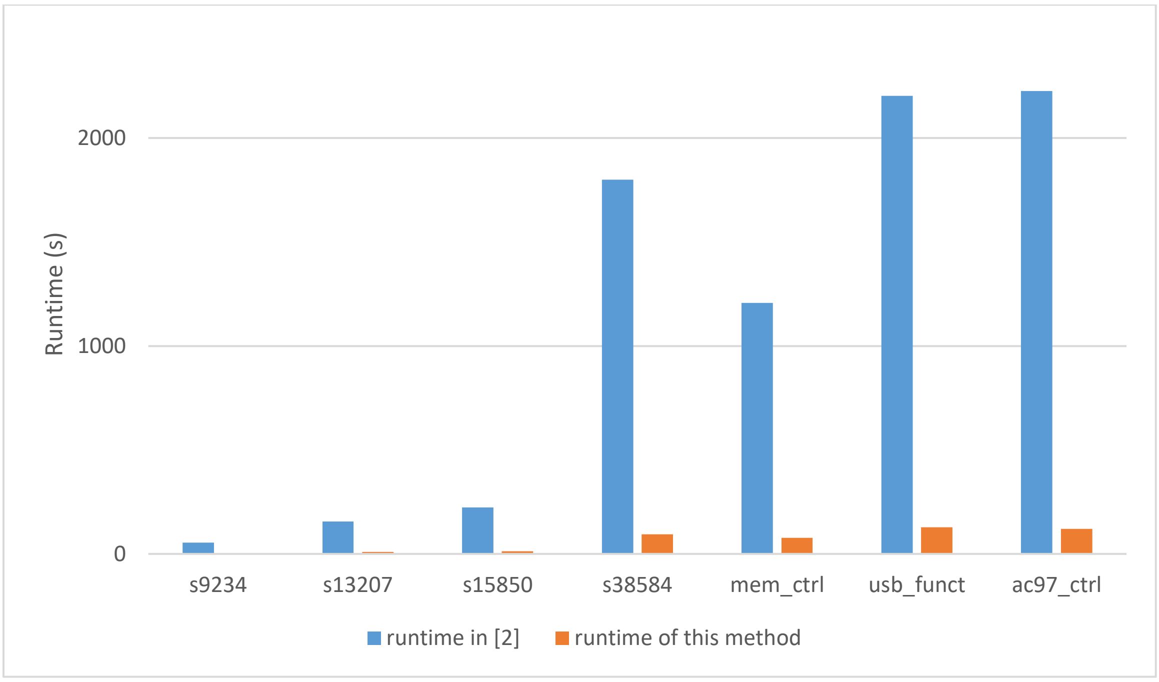

Fig. 4.8 Runtime comparison.

5 Conclusion

In this thesis, we propose a machine learning approach to speed-up the sampling-based method to determine post-silicon tuning buffer locations and ranges proposed in [2]. Using only a few samples for training, the model can predict buffer locations and sizes for yield improvement effectively and fast. We use a Naïve Bayes classifier to determine the buffer locations and a linear regression model to predict their sizes. Experimental results confirm that a large speed-up can be achieved with machine learning models while keeping the yield improvement like the one in [2]. Future tasks of this work include learning of learning, in which we try to find a machine learning model which can predict the parameters of the previous machine learning models.

References

[1] S. Naffziger, B. Stackhouse, T. Grutkowski, D. Josephson, J. Desai, E. Alon, M. Horowitz, “The implementation of a 2-core multi-threaded Itanium family processor,” *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 197-209, Jan. 2006.

[2] G. L. Zhang, B. Li, U. Schlichtmann, “Sampling-based buffer insertion for post-silicon yield improvement under process variability,” in *Proc., Design, Autom., and Test Europe Conf.*, 2016, pp. 1457-1460.

[3] S. J. Park, B. Bae, J. Kim, M. Swaminathan, “Application of Machine Learning for Optimization of 3-D Integrated Circuits and Systems,” *IEEE Trans. VLSI Syst.*, vol. pp, no. 99, pp. 1-10, 2017.

[4] J. Xiong, Y. Shi, V. Zolotov, and C. Visweswariah, “Statistical multilayer process space coverage for at-speed test," in *Proc. Design Autom. Conf.,* 2009, pp. 340-345.

[5] F. Gong, H. Yu, Y. Shi, and L. He, “Variability-aware parametric yield estimation for analog/mixed-signal circuits: Concepts, algorithms, and challenges,” *IEEE Design & Test*, vol. 31, no. 4, pp. 6-15, 2014.

[6] T. Wang, C. Zhang, J. Xiong, and Y. Shi, “Eagle-eye: a near-optimal statistical framework for noise sensor placement,” in *Proc. Int. Conf. Comput.-Aided Des.*, 2013, pp. 437-443.

[7] Forman G., Cohen I. (2004) Learning from Little: Comparison of Classifiers Given Little Training. In: Boulicaut JF., Esposito F., Giannotti F., Pedreschi D. (eds) Knowledge Discovery in Databases: PKDD 2004. PKDD 2004. Lecture Notes in Computer Science, vol 3202. Springer, Berlin, Heidelberg.

[8] G. L. Zhang, B. Li, J. Liu, Y. Shi, and U. Schlichtmann, “Design-phase buffer allocation for post-silicon clock binning by iterative learning,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, 2017, doi: 10.1109/TCAD.2017.2702632.

[9] K. P. Murphy, “Machine Learning: A Probabilistic Perspective,” The MIT Press, 2012, pp. 82-85, pp. 217-219 and pp. 267-270.

[10]The MathWorks, Inc., “Train multiclass naive Bayes model - MATLAB fitcnb,” [Online]. Available: https://www.mathworks.com/help/stats/fitcnb.html [Accessed 04.06.2017].

[11]The MathWorks, Inc., “Train binary support vector machine classifier - MATLAB fitcsvm,” [Online]. Available: https://www.mathworks.com/help/stats/fitcsvm.html [Accessed 04.06.2017].

[12]N. V. Chawla, K. W. Bowyer, L. O. Hall et al., “SMOTE: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, pp. 321-357, 2002.

[13]C. M. Bishop, “Pattern Recognition and Machine Learning (Information Science and Statistics)”, Springer-Verlag New York, Inc., Secaucus, NJ, 2006, pp. 138-141.

[14]The MathWorks, Inc., “Generalized linear model regression - MATLAB glmfit,” [Online]. Available: https://www.mathworks.com/help/stats/glmfit.html [Accessed 04.06.2017].

[15]The MathWorks, Inc., “Create generalized linear regression model - MATLAB fitglm,” [Online]. Available: https://www.mathworks.com/help/stats/fitglm.html [Accessed 04.06.2017].