UNIVERSITAT POLITÈCNICA DE CATALUNYA(UPC) - BarcelonaTech
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
MASTER IN INNOVATION AND RESEARCH IN INFORMATICS
ADVANCED COMPUTING

MASTER THESIS

Machine Learning Techniques for resource prediction in nanoelectronic circuit design

Author's name: Narcis Ricart

Supervised by
Jordi Cortadella, Computer Science department and
Jonas Casanova, eSilicon Corporation

3rd July 2017

Abstract

This master's thesis is about the use of machine learning techniques in the field of nanoelectronic circuit design. It has been developed in collaboration with eSilicon Corporation, which is a company specialized in designing and producing Application-specific integrated circuit (ASIC).

While specific integrated circuits require several resources to be designed. Most of them can be classified as expenses from non-recurring engineering. This includes all costs used only once related to the final product. For example, in our project, they mean computational resources, such as CPU usage and memory utilization, but also hours worked by specialized designers. All of them have tremendous costs, which can not be reused for future designs.

The study done in this thesis inverts the current situation, gathering data from the design process to produce valuable information for upcoming designs. This information is related to time, memory and specific attributes of the design.

In order to achieve this goal, some of the most successful and helpful machine learning algorithms are used. Their purpose is to predict key aspects of the design and its computation before any execution is done. For example, if a designer is aware of that the result of an execution in terms of leakage or area utilization, will not satisfy their expectations, then he would be able to decide whether it is worth or not the execution. This implies saving time and computational resources.

These machine learning techniques are fed with data obtained from designs. This data, though, may present us certain drawbacks like missing values, unreliable data due to bugs, errors during the execution and a long list of possible sources of noise and incorrect data. Thus, it is important to preprocess all that information to obtain a trustworthy and stable source of data.

The final outcome of this project can be seen as the combination of data preprocess and machine learning use. This applied to the field of nanoelectronic circuit design, to get predictions of: runtime, CPU time, RAM memory, leakage and area utilization. This can lead to an improvement in productivity by reducing the total cost of computational resources and time spend on them by designers.

Acknowledgements

Several people has helped me making this thesis possible. I would like to thank Prof. Jordi Cortadella to give me the opportunity to work on this project; Jonas Casanova for guiding the project and provide professional advice; Prof. Lluis Belanche for answering any question about machine learning; and my family and friends that have helped me while I made this project.

I would like to thank eSilicon Corporation for granting the whole project and sharing their data and resources; specially Ferran Martorell and Marc Galceran for solving any doubt inside the electronic circuit design world.

Barcelona, 3rd July, 2017

Narcis Ricart Geli

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project is the combination of two complex domains of knowledge, which should be clearly distinguished:

- Application-specific integrated circuit (ASIC) design is the process of designing an integrated circuit for a specific aim. The initial ASICs appear in the eighties in the form of gate arrays (also known as uncommitted logic arrays), which is a logic chip without any specific function but containing logical gates placed at regular positions and a layer that allowed to join the elements as desired. In this kind of technology the customization is done by changing during production the mask of the connection layer. The amount of gates involved increased from a few thousands to hundreds of millions in the modern designs, causing its production and especially its design to be very challenging. Section 1.1 describes the most important steps that are followed to produce ASICs today.

- Machine Learning has been applied in a wide scope of disciplines since its beginning in the fifties, from class classification to image recognition in medical environments. The early years of the discipline were prolific and most of the ideas and concepts that were proposed set the basis of what has became one of the most popular areas of computation. From that point on time the history of machine learning could be summarized as a continuous rise and fall in the interest and impact that it has on the society. In the recent years thanks to the achievements made by the improve in the initialization of methods of deep learning plus the increase of the computational power and the availability of new data, the whole discipline has came back, not only neuronal networks but also a great variety of methods such as support vector machines or ensemble methods. The methods used in this project are described in more detail in Section 2 of this document.

1.1 ASIC design

In this section the key concepts related to ASIC design that are needed to understand the document and the work done during the process are explained. It is also important to highlight that it does not exist the perfect logic component in the sense that depending on the situation a gate or a memory could be good while in another place it could be a bad decision to use it. Thus, the whole process is a trade-off between desirable properties that can not coexist all together at the same time. So while the design has to achieve its logic purpose, the designer has to take care of the following variables:

- Power consumption
- Area
- Performance

This trade-off means that at most two variables can be optimized for a certain task, so if it is needed a smaller area and low power consumption the resulting speed of the component is going to be lower. This starts to give a sense of the hidden difficulty that is behind the design of this kind of circuits.

Another aspect that is needed to be understood is the steps that have to be solved to design an ASIC. As it can be intuitively guessed the whole design can not be done in just one phase. To give an idea of the dimension of what is implied there are millions of components that have to be placed and connected (to power, to each other and to the clock) having constraints on timing, area and power consumption. This is absolutely unfeasible if treated as a unique problem and it is also the reason because the whole ASIC is divided into blocks and steps. All these blocks will have to get through the same steps which are explained below.

Apart from that, even though the blocks are smaller than having the ASIC as a whole the automation of most part of the work is needed. There are a enormous number of components in each block, which implies millions of possibilities to complete a block. Sometimes there are unfeasible constraints, which could add extra difficulty to the whole problem. Even with the amount of components the design is not fully automated and it is still the designer the one that has to change manually some parts when EDA (electronic design automation) tool is not able to find a good solution.

In order to give a general idea of which are the steps that are involved in the design, below a list of the most important stages that are computed along the process is provided. This is not intended to be exhaustive; its intention is to give context to the project and an overview of the difficulty that concerns the design.

- **Import** is the first step, here the net list and the constraints of the block are loaded into the EDA tool and the timing modes are defined.
- **Floorplaning** is the step where the shape of the block is decided. Also meaningful components such as pins and macros are placed, finally the starting/ending cells for the columns and rows are determined.
- **Power grid** is when the power grid is specified, this implies having a high metal layer defined and then some wires that goes through the different metal layers until they reach the components.
- **Placement** takes care of placing all the standard cells and also optimizes the timing constraints.
- **Clock tree building** is the step where all flip-flops are connected to the clock, it has to take into consideration that sometimes can coexist more than one clock in the same block.
- **Routing** up to this moment most of the connections were estimated. At this point the actual local routing is done.
- **Time closure** is the final step, where all constrains related to timing and signal propagation have to be satisfied.

As it can be seen from their description, each of the steps deals with certain problem of the circuit design. This implies that only after the execution of some of them the system has data about some features that before were unknown. For example it is obvious that the system can not have information about the wire length before any connection is done.

Another basic concept is run; a run is a set of steps that are executed one after another. For example a designer could be interested in doing only up to the power grid step, so it would only launch a job to the system with the firsts steps (import,floorplaning and power grid), then all the computation would belong to a run. It is also need to take into account that a run does not need to start from an import, it can also start from an step done in another run.

1.2 Goals and motivations

The main goal of this project consists in creating a system capable of predicting metrics before the execution of a task during the design is executed. The metrics that are predicted in this project are explained in the following list:

- IT metrics
  - *Runtime* is the amount of real time that an execution takes from its beginning to its end.
  - *Memory* is the maximum amount of memory that is used to execute a task.
  - *CPU time* is the sum of total CPU usage that has been used, this takes into account the possibility of parallelism.
- Design metrics
  - The *leakage* is the power consumption even when a block is not being used.
  - *Area utilization* is the percentage of the area occupied by components.

All these metrics have a wide range of possible values and they change from one step to another. For example, runtime and CPU time can last from a couple of hours to a couple of weeks. The same is applied to memory, leakage and area utilization, thus the possible outcome is uncertain before the execution is done.

There are different reasons to predict these concrete variables:

- Reduce the cost of designing:
  - Detect some anomalies during the execution. For example, if a designer launches a run, that its prediction estimates that should take 4 hours of computation and it has been running for more than one day. Then, it can be convenient to warn the designer that probably something is wrong with that execution. This can save time of the designer that is waiting for a result and also it can save computational resources.
  - An accurate prediction of the amount of memory that a certain task needs, will allow an efficient allocation of memory. As a consequence, the system will be able to allocate a much more precise amount of memory for every task. Memory has a huge impact on the overall cost of the design. In fact, the total cost of the machines has an order of magnitude of millions of Euros per year, based on prices from different cloud computation providers.
  - The design metrics predicted can be useful to decide whether an execution is worth being launched or not. For example, let's suppose that a designer wants to launch a task to reduce the leakage, if there is an estimation of the expected leakage that a circuit will have before it is really computed, then it can be decided whether that execution is worth being done. As a consequence a lot of time and resources can be saved.

- Provide valuable information to the designer:
  - Help organizing the work of a designer: Most of the times a designer has a few blocks to implement, but once a process is launched he does not have any accurate value (apart from his intuition) of the amount of time required. So having these predictions would have a good impact on the capacity of the designers to organize their time.

Each of the steps described in Section 1.1 are executed one after another. After the execution of each step, the report is generated. Then, using that information in combination with machine learning algorithms, it is possible to predict the previous described metrics before the execution is done.

To sum up the final aim that is intended in this project is to predict runtime, CPU time, memory, leakage and area utilization to improve the use of the resources during ASICs design and to detect anomalies to warn designers. This is done using machine learning algorithms fed with data from past executions. The predictions obtained are provided between executions of different steps.

1.3 Document structure

The rest of this document is structured as follows. The first section presents the state of the art of the machine learning techniques that have been involved in the project. Then the data available to feed the different procedures is explained, including the preprocess that is applied to it. After having the procedure described, the results are exposed with an interpretation of the meaning. Finally the conclusions are explained, which will be a wrap up of the whole project.

Chapter 2

State of the art

This chapter contains a brief introduction to some problems that are being addressed using machine learning techniques, then a review of the machine learning methods that have been used during the project is also presented.

Machine learning is used to handle a huge variety of problems, such as natural language processing, financial market analysis, medical analysis, image recognition and the list can continue while growing every day. This tells a lot in favour of the adaptability of different algorithms, whose main idea is to find generalized properties of a data set. Then, create a model able to predict some future behaviours based on previous examples (the data available).

There are many different ways to classify methods that are involved in this wide field, the simplest way is to characterize each model by the expected output that is desired, which leaves us with the following list.

Clustering
Clustering algorithms are used when it is wished to group similar data together, while keeping different data in different groups. One example could be a system that assigns tasks to workers based on their skills and the difficulty of the task. This can be done grouping designers by its capacity, so workers with similar skills are gathered inside the same set. Then, tasks can easily be distributed to people depending on their ability to perform a given job.

Regression
A brief definition for *regression* techniques could be all problems that have a real number as a result. This is very interesting for many applications and together with classification methods are the ones that cover a vaster space. In the case that someone wants to predict some numerical value, such as score, price, weight and many others, then a variety of regressors can be used.

Classification
There are *classification* techniques, that are in charge of categorizing an input to one class. It can be the case where the output is not just providing one concrete class but many instead, or maybe it gives a probability of membership to all of them. A classical example would be an image recognizer, which distinguishes images based on the appearance of dogs, cats or elephants

Density estimation
Density estimation is a specific problem which instead of providing a category or a number, what is desired is the underlying distribution function that is followed by the data. In this kind of methods the data is thought to be a random sample belonging to a much bigger population which follows the wished distribution that has to be estimated.

Dimensionality reduction
Dimensionality reduction is the set of algorithms that tries to find a lower dimensional data representation or a smaller number of inputs. Often these techniques are divided into feature extraction and feature selection. The first one is in charge of transforming data from high dimensional space to a lower dimension space; the most well known technique is the linear principal component analysis (PCA). While feature selection is in charge of reducing the number of variables used in the model.

Novelty detection
Novelty detection is the name given to all methods that deal with the detection of some particular data. Particular data means outliers, novel data or anomalous. For example these techniques can be very useful in systems where it is important to keep track of processes and it is desired that no process consumes a huge amount of resources from the system.

As it can be derived from the previous overview of problems that can be solved using machine learning, the main goal of this project fits perfectly in the regression category. Let's remember that as it is explained in Section 1.2 our main aim is to predict the value of some design metrics, in other words the expected value of the output is a real positive value. Therefore the machine learning algorithms that are needed in this project are regressors.

It is necessary to point out that most concepts from machine learning algorithms can be equally used to solve more than one problem. For example depending on the implementation chosen the same concept can be used to classify or regression. For instance, decision trees are used for both purposes. Due the nature of our problem, in this document and also in the project itself, any reference to a machine learning technique usually refers to the regression one.

Once the main ideas of machine learning purpose have been explained, it is important to think about the reasons behind its use. Probably it is clear that our goal fits perfectly as it is a very well known application, which is the prediction of real values. Another reason is that the data available, which is explained in Section 3.1.2, includes several different experiments (of the order of tenth thousands) and a large number of features. The use of any other technique would make unfeasible to manage the huge amount of data that is present in this project. Instead, in machine learning having a large volume of data is a good starting point that helps the models to generalize better. To sum up, the whole project: data and purpose, completely fits to be a machine learning application.

In the following sections, the different machine learning algorithms used to predict CPU time, area utilization, runtime, leakage and memory are presented.

2.1 Linear regression

This technique belongs to a larger family of methods called Generalized Linear Models, which expects the output to be a linear combination of the input variables. Let's say that y is the variable to be estimated, it can be seen as:

$$y = w_0 + w_1 * x_1 \cdots + w_n * x_n$$

Where,

> n is the number of input variables available
>
> x_i is the ith input
>
> w_i is the coefficient of the ith input
>
> w_0 is often called interception and adds an extra degree of freedom to the linear expression

In this case, the method used is also known as *Ordinary Least Squares*, which is the simplest one because it basically minimize the difference between the real value and the predicted by changing the values of the coefficients. It can also be seen as fitting a linear model that minimizes the residual sum of squares error. In a more formal way:

$$min||Xw - y||_2^2$$

The main idea and use of this model can be easily seen in Figure 2.1. Dots represent data points in some dimensional space, the aim of the model is to reduce the total residual square error changing the value of w. As a result and representation of the method there is the straight line which is the linear approximation that fits better our data.

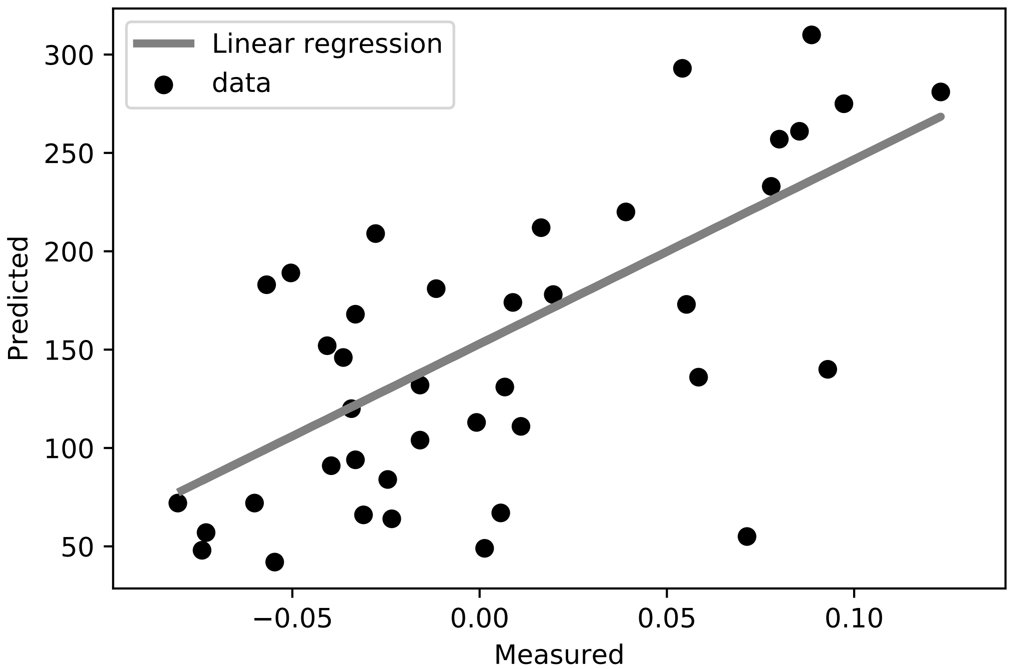

Figure 2.1: Linear regression example with synthetic data.

The main advantages in using this technique are the simplicity to interpret the results and the model itself, because each coefficient has a clear relation of the impact of each variable in the final output. Another excellent point is the lack of parameters to be tuned, the model is based only in changing the w to obtain a result.

This model relies on the independence of the terms. When there is a linear dependence between inputs this model can misbehave. Another drawback is the fact that it can only deal with linear data, so in the case that the sample data is not linear, the model can not do much to predict it.

2.2 Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines is a machine learning method commonly known as MARS. It was first described by Jerome H. Friedman [7] and it is an evolution in the use of linear methods. The main idea is to use linear approximations to describe the data even in cases where data does not follow a linear distribution.

In order to do so, it is needed to have more than one linear model and use each of them to describe only a segment of the whole data set. Another way to see this method is as a set of simple linear models that combined look like a single function. The key concept of this method is the use of hinge functions, which have the form of

$$max(0, x - c)$$

Where c is some constant value and x is some input variable. The form $max(0, c-x)$ is also valid. The final form of these models is the sum of combination of constants, hinge functions and combinations of hinge functions. Here an example of a model is given, y represents the final output:

$$y = \begin{cases} 2.5 \\ +2 * max(0, x_1 - 8.95) \\ +9.2 * max(0, 5.45 - x_2) \\ -4 * max(0, x_3 - 7.67) * max(0, x_1 - 0.61) \end{cases}$$

In this example, there are only three input variables x_1, x_2, x_3. As it can be seen it is possible to combine different functions, use one variable more than once and give weights to each function. This gives a very flexible model that is able to predict non linear models, which represents an improvement of the linear methods. This flexibility usually means a better fit of the model, as it can be seen in the comparison of Figure 2.2 where both models are represented with a discontinuous line. There the MARS model, on the right, is able to fit much better the data than the linear model on the left.

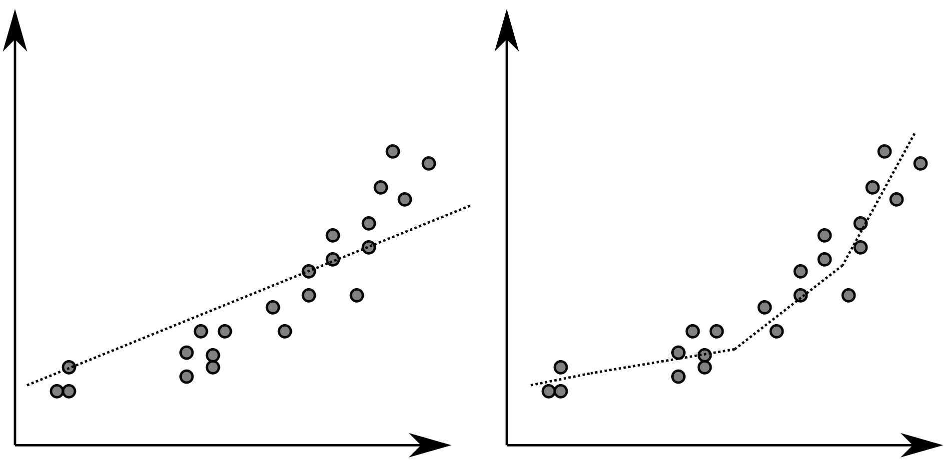

Figure 2.2: Comparison between MARS regression and linear regression with synthetic data, the left plot is for the linear one while the right plot corresponds to MARS algorithm.

Apart from the adaptability of the model, which gives the model the chance of fitting non linear data, another advantage of this model is its interpretation in comparison with other more complex models. The model still gives an understandable intuition of the interaction between variables and the importance they have.

It has some drawbacks, the most important one would be that it is still not as powerful as other methods. Its predictions are usually not as good as the ones obtained with more complex models.

2.3 Classification And Regression Trees

Classification And Regression Trees (CART) is a method introduced in 1984 by Leo Breiman et al.[5] that belongs to the family of decision trees. Decision trees create a set of rules that split data according to one variable in each partition, the rules are applied one after another following a binary tree schema. This means that each node has a rule that divides the sample, while the information about the prediction is provided in the leaf nodes, where the value is given.

The easiest way to understand CART models is to see one of them as the one in Figure 2.3. It is the three first levels of a real CART model for our project, there it can be seen how depending on the rule on the top, it is decided one path or another for a given example. It is possible to see how a variable can be used more than once, *log memory* is used twice; one for the first split and then for the third one is also used for some branches.

It could be a problem for visualization when there is a big tree with a big depth, where the path from the root to the leaves is long. For example, the whole tree shown in Figure 2.3 has a maximum depth of 20, which means a huge number of leaves, even after pruning them. But in general a big advantage offered by CART (and decision trees) is that the models are very understandable and easy to interpret.

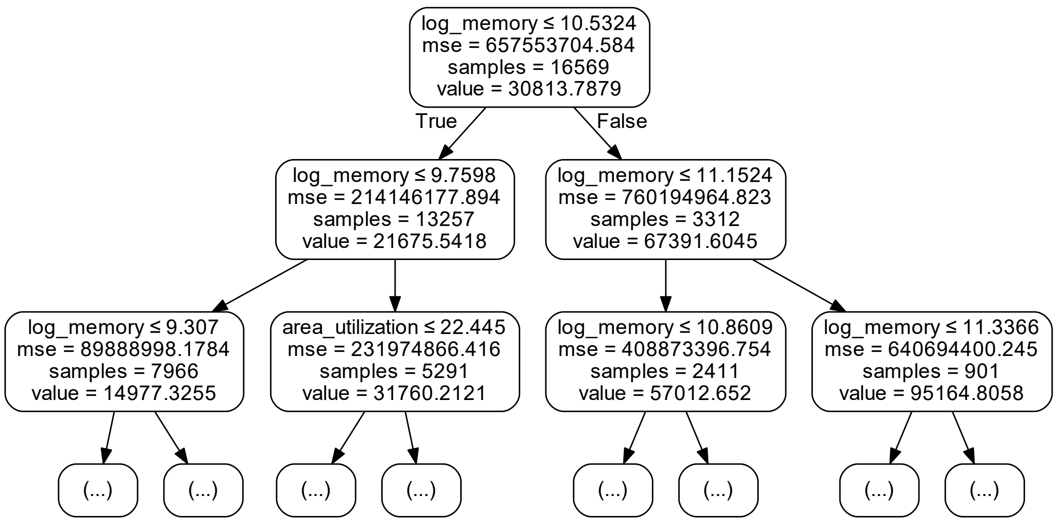

Figure 2.3: Example of the first three levels of a CART tree corresponding to memory.

Another advantage of this technique is that it can handle numerical and categorical data at the same time, because the rules that are applied in each node of the tree are independent from one to another, allowing them to split the branches using whatever variable.

They also have some disadvantages, such as being an unstable estimator. This basically mean that small changes in the data used to create the model may result in very different trees. This disadvantage can be taken as a good opportunity for other methods that are more complex and rely on unstable methods to perform as it will be described in Section 2.6.

The other important drawback of this technique is the overfitting of the data. If there are too many partitions the granularity of the model is very small so any point in the training set is potentially a leaf, which would not characterize the general behaviour of the model. This problem appears in Figure 2.4, which is the same data of the model shown in Figure 2.3 and with another model plotted, which is a CART with a maximum depth of five. The discontinuous line represents the simplest model with depth two, which is the reason why there are only four given values for the whole data set. While the bold one corresponds to a more complex tree with five different levels, as it can be seen this produces a big overfit with values that clearly are noise, which should receive the attention from the model.

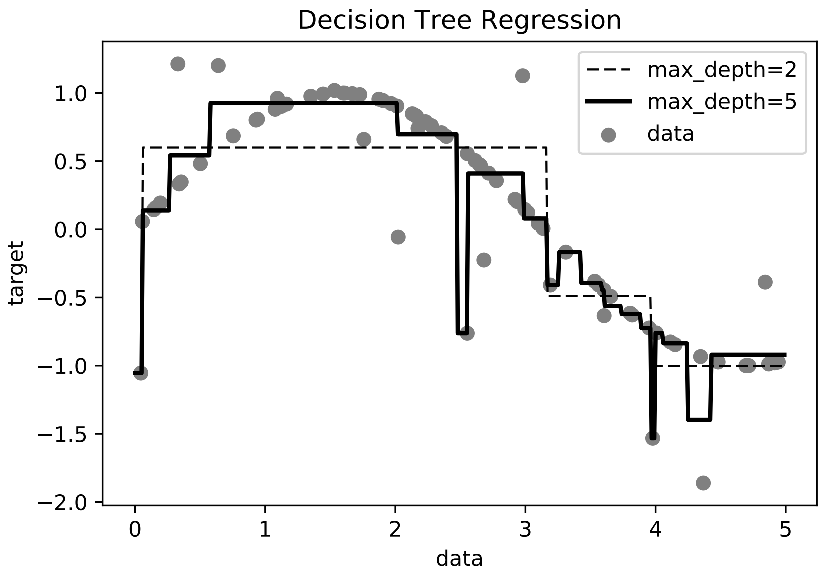

Figure 2.4: CART regression with predicted data from the same example as Figure 2.3.

2.4 Support Vector Regression

Support vector regression (SVR) is a support vector machines (SVM) for regression, instead of classification. SVM was introduced in the sixties but was the contribution of the kernel trick in 1992 [2] that had a great impact of SVM. After just five years in 1997 [6] this technique was adapted also to the regression problem.

The main idea behind this method is to transform data into a high dimension space, where the regression or the classification is much clearer to be done, because a hyper plane can be fitted and the classes are very distant between them. In the classification case is usually clearer to understand this transformation than in the regression problem. In Figure 2.5 the kernel transformation from one dimension to a higher one can be seen. The hyper plane is the red line while the discontinuous lines are called the margins. The kernel function was good chosen because the two classes are better defined in the high dimensional space than in the feature space.

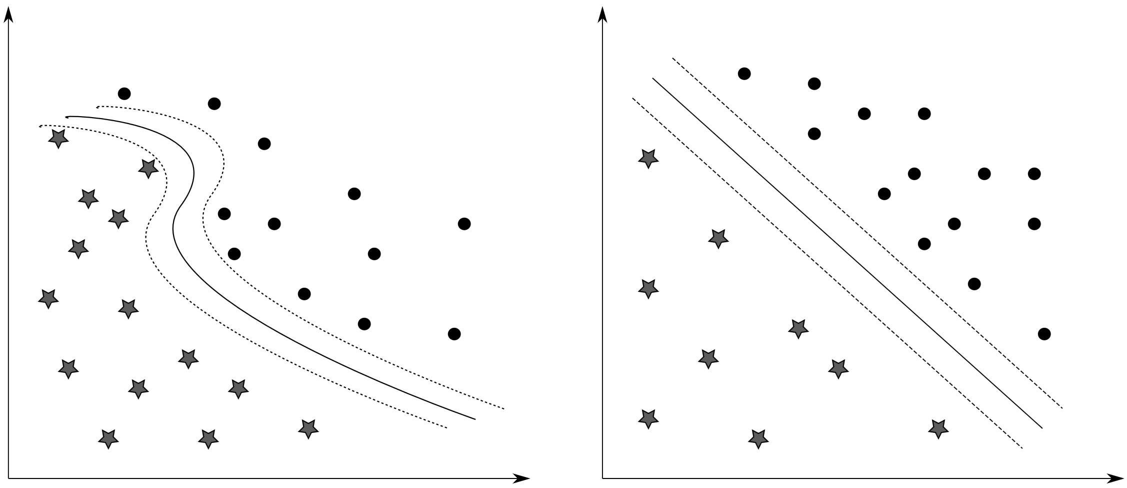

Figure 2.5: SVM kernel transformation example in a two class classification, where the left side represents the data in the feature space while the right side is the representation in a higher dimensional space.

This mapping from low to high dimensions would be very expensive to be computed and it is the reason of using the kernel trick. The trick is essentially computing the inner product of all pair of data in the feature space, instead of computing the coordinates of the data in the high dimension.

Therefore the selection of the kernel function has to be made carefully and it is key in the final outcome of the model. Usually the kernel function is not defined by the user, it is only selected from a set of well known functions, the most used are:

- Polynomial kernels
- Gaussian RBF kernels
- Laplacian RBF kernels
- Sigmoidal kernels

This technique has some parameters that have to be selected, the most obvious one is the kernel function and its parameters. For example to use the polynomial kernel that has the form,

$$k(u, v) = (\langle u, v\rangle + 1)^d$$

the parameter d, which is the degree of the polynomial, has to be selected. There are still two other important parameters to be chosen the C and the ϵ. C is the parameter that penalizes the error term of the slack variables. In Figure 2.6 there are two classes, which are stars and circles and the goal is to classify them using an SVM. Then C would multiply the sum of $c_1 \ldots c_4$, because these points are not well classified and c_i represents how much cost would be needed to get that point to the right position in partition of the space. Obviously this cost is desired to be the minimum. Finally the last parameter of the SVR is the ϵ, which defines a no penalty range with predicts points within a distance ϵ from the real value.

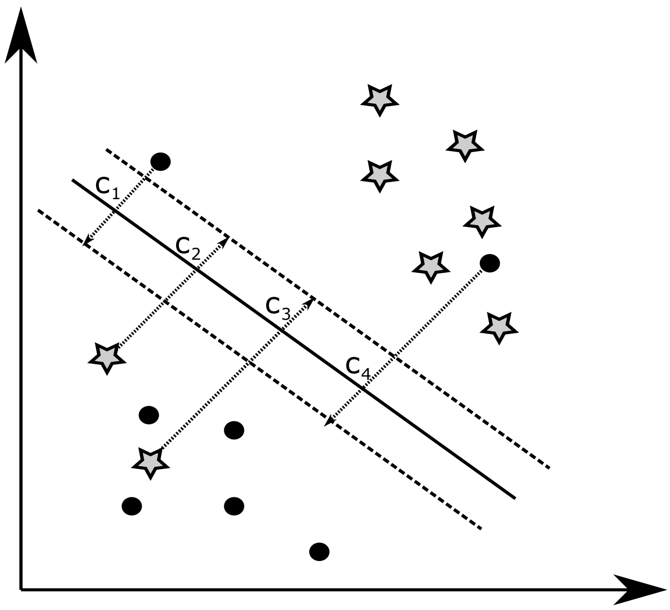

Figure 2.6: SVM error example with four bad classified points

2.5 Artificial Neuronal Network

This technique was one of the first to be developed and the "perceptron" algorithm, which is a name that is usually associated to it, was first introduced in 1958 [11]. But it was not until the introduction of the backpropagation algorithm in the seventies that made the computation efficient enough to be used.

Before explaining the main idea, it is important to define what an artificial neural network (ANN) is. An ANN is a directed graph, composed by nodes and edges that connect them, they are called neurons and connections respectively. A neuron can be seen just as a computational node where a vector of inputs $x_1 \dots x_i$ is combined with a vector of weights $w_1 \dots w_i$ and provides a value as a result. This neurons are grouped forming layers, a layer of neurons share the vector of inputs (note, the weights for each input depends on each neuron it is not shared) and can not be connected with one another. There are two special layers, the initial one, which is also known as the input layer and the final one, which is the output layer, any other layer in between these two is called hidden.

Figure 2.7 represents an ANN with one hidden layer, there the neurons are represented with circles and connection with arrows, each arrow has its own weight in the neuron that is connected. The red neurons represent the input layer with n different features and the output layer represents the result of the regression after all the computation is done.

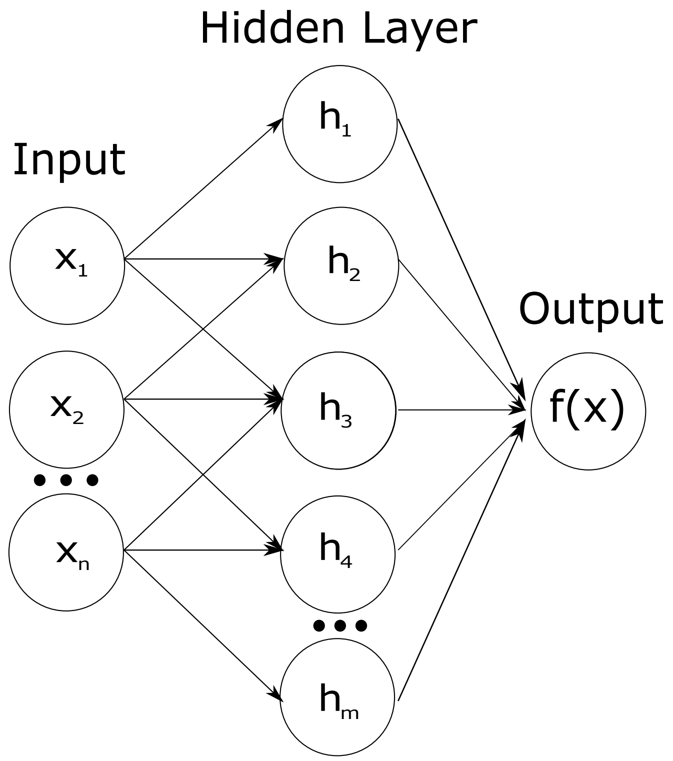

Figure 2.7: Basic structure of an ANN with one hidden layer and n input features

The simplest function to compute the output of a neuron is:

$$y(x) = g(\sum_{i=1}^{n} w_i x_i)$$

Where,

n is the number of inputs

w_i is the weight of the neuron to the *i-th* input

x_i is the *i-th* input of the neuron

g is the activation function, it is the one in charge of providing non linearity to the output, often a *logistic function* or a *tanh function* are used

Once these concepts are explained, it is much easier to formulate the main idea of the ANN model. The idea is to modify the weights in such a way that the error in the output is minimized. This is an iterative problem where two algorithms are combined, the first one is in charge of computing the responsibility of each neuron to the final error, while the other is in charge of updating the value of the weights. The first one is the well known backpropagation algorithm, that takes an advantage of the very well known chain rule of the derivatives. The other important algorithm is a minimization one that updates the weights in a clever way using a variant of a quasi-Newton method which is called L-BFGS (Limited-Broyden–Fletcher–Goldfarb–Shanno)[10].

The method has one clear advantage which is that usually the models that are fitted using ANN perform very good in reality. It is able to perform well even in non linear environments.

This good performance comes with a cost, which is an important number of variables to be tuned and a considerable amount of computation. The list of the most critical parameters is the following one:

- Shape of the hidden layers, this means the amount of neurons to use and how many layers are desired.
- The activation function for the neurons
- α is a parameter that penalizes complex methods. It is also known as regularization term.
- The learning rate, which controls how quickly the update of weights is done.

Another important drawback of this method is its understandability, which is very poor. Just by looking at the weights of an ANN it is extremely hard to conclude anything if possible at all. This is also translated to the parameter selection where it is very difficult to say intuitively which should be the values for the parameters.

2.6 Random forest

Random forest is a technique that, as usual, belongs to a larger family of algorithms which is ensemble methods. The first intuition of the method came with a proposal from Tin Kam Ho in 1995 [8]. But it was in 2001 when Leo Breiman [4] developed the actual technique combining the out-of-bag error with measuring variable importance and bagging (introduced by the same author in 1996 [3]).

Random forest is an averaging method in contrast with the boosting methods, this basically means that the idea behind the algorithm is to generate independent estimators and average their results (for the case of regression, while to classify it would be voting). This can be done because the combination of estimators is usually better than the performance of any individual one by itself. The main reason for this to happen is that the variance of the result is reduced due the use of multiple estimators.

In order to understand how this method internally works, it is important to understand what is bootstrap resampling. It is a technique that consists in sampling N values with replacement for $Z_1^* \dots Z_B^*$, so B potentially different samples are created. Each of the Z_b^* is then used to train a predictor, keep in mind that each predictor have a different set of data. Now let's remember that in Section 2.3, it is explained that CART is an unstable method (small changes in the data set lead to big changes in the model).

Let's try to add all the concepts that can look unrelated at first sight:

- Bootstrap resampling allows having different chunks of the same data
- CART models have a high variation if the data set they use as train change
- Having several independent estimators of the same type, leads to improvement in the prediction.

The result of combining these features is almost a random forest; the only property that is left is the out-of-bag error (OOB). OOB consists in using the not in bootstrap selected data as predictive data, which speeds up the whole computation. The computation of the OOB is defined as:

$$Err^{(0.632)} = 0.368e^{-*} + 0.632Err^{*}$$

Where,

$$Err^* = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{|Z^{-n}|}\sum_{b\in Z^{-n}}\mathbb{I}[t_n \neq f_b^*(x_n)]$$

Having,

$\mathbb{I}(x)$ is 1 if x is true, 0 otherwise

Z^{-n} is the set of resamples that do not contain observation x_n

f_b^* is the model fitted to Z_b^*

$$e^{-*} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{|Z^{n}|}\sum_{b\in Z^{n}}\mathbb{I}[t_n \neq f_b^*(x_n)]$$

where $Z^n = 1, \ldots, B \backslash Z^{-n}$

The previous concepts plus this definition of the OOB is what is integrated in a random forest model. In addition, the model is improved by giving more randomness to estimators, this is called decorrelating trees. It consists in randomly picking a subset of features to be selected during the creation of the CARTs. This means that at random the CART algorithm will be able to select from only a few of the total features to be built. Naturally this decreases the performance of the individual trees, but it increases the overall performance of the forest, due to have a more variance in the independent estimators.

The main advantages of random forest:

- Good performance in the predictions, due the chance of predicting non linear data
- Use of OOB to speed up performance and variable selection
- It is easy to extract the feature importance for the overall forest.

The most remarkable disadvantage is that despite the use of CART models, it is impossible to plot any clear information about the forest. So it is hard to understand why a value is predicted or classified.

Chapter 3

Building Models

The different procedures involved in the project are detailed in this chapter. This includes sections for data, data preprocess and machine learning methodology. The first one explains singularities of the data that has been used, its extraction and its meaning. The second is about transformations that are applied to inputs before an execution of the machine learning method occurs. Finally the methodology section is about the technology used to develop the solution, which machine learning schema has been followed and decisions that have been taken regarding to it.

3.1 Data

One of the main issues when dealing with machine learning is obtaining accurate and reliable data, keeping in mind that it is also important the number of experiments, usually the more available the better it is. In this simple sentence, there are hidden several important problems: obtaining data, accuracy of the data, bad measures or misbehaviours of the algorithm that gathers data. Having a big amount of data of the process is fundamental because it gives the opportunity to methods to capture the general behaviour and not getting focused into concrete examples.

These obstacles have to be overtaken or minimized in order to have a chance to obtain a robust and trustworthy model. The following subsections are devoted to explain the decisions that have been taken to reduce the impact of these problems.

3.1.1 Data extraction

The first step related to data is the preparation of procedures to extract it. In our concrete case, the nature of EDA tools is a clear advantage because they are executed inside a virtual environment, allowing us to capture most of the events and progress done by them.

The main idea of how this is done is having a number of variables computed straight from the design itself, while others are obtained from the report/log generated automatically by the EDA tool after the execution of an step. After all metrics are computed, they are stored in a server. As usual this concept is not as easy as it looks like and there are some problems.

The initial one is that there are different tools used and each of them has its own way of generating the log with its own units, so one tool can provide the runtime in milliseconds but another is using seconds. There is also a problem of names, each provider has its own vocabulary to express the same concept and it can be worse if there is one tool that gives information about some specific concept but the others do not. To solve this issue it is needed an abstract layer that homogenizes all sources of information into one standard.

Another problem appears when a new metric that was not being computed is wanted to be added to the server. Then apart from changing the abstract layer, it is needed that new data starts getting uploaded. This is a process that requires time, because historical samples can not have any information about it, so it is a matter of waiting the creation of new data to slowly fill the metric.

Finally, there are some metrics that can be manually modified by a designer, if a test of some execution is desired. For example the name of the step that has been executed can be sometimes found to have some prefix or suffix added outside the normal procedure. The point here is that it is easy that some names change. This should not be a problem as long as a good standard to describe names is used.

3.1.2 Metrics information

This section is thought to be a guide where all metrics available are explained. This is useful to give an idea of which measures appear during the design and according to this, how accurate the reality is captured. There is a total amount of 52 variables and in order to give a clear idea of them, they are grouped in different clusters depending of their meaning. It can be helpful to understand each described metrics as a column of a matrix, where rows are the different examples or samples.

The number of samples is important in a machine learning project because, usually, a big number of examples allow a better generalization, preventing methods to get stuck with small details. In our case, there are over 55.000 samples of raw data in total, this includes several projects from the last couple of years, which has some drawbacks explained in Section 3.2.

Let's remember that in this project: the runtime, CPU time, memory, leakage and area utilization are the predicted metrics using machine learning algorithms. Thus, the rest of described metrics here are the ones used to create the different machine learning models.

- **Predicted metrics**

  This group is a special set of variables, which are the ones that are predicted using machine learning algorithms. Thus, the rest of groups describe metrics used to create the different models. In other words, this group is the expected output of the models, while the others are their inputs.

  - *Runtime* is the number of seconds from the beginning of the execution until the end of it. In contrast with the CPU time this includes time spend in the scheduler queue or time doing I/O.
  - The *CPU time* is the sum of all seconds that the process has needed all CPUs. So if a run was computed in parallel using 4 CPUs, the final result is the sum of all the time while this CPUs were computing. This is the reason behind that sometimes CPU time is larger than runtime.
  - *Memory* is the peak of memory in mega bytes used during one computation. So it is the real amount of memory used for an execution.
  - *Area utilization* is a ratio between the occupied space inside a block and the empty space still available.
  - *Leakage* is the power consumption in milliwatts even when a block is not being actively used.

- **General information**

  This group contains information about the job that is executed, it follows a hierarchical schema 1 to n where the project level is the root until the most defined level which is the step. So, a step is contained inside only one run, which can have several steps, at the same time a run is done for one particular block, which can have different runs and so on until the project layer. All the variables described in this section are string, with the exception of the time stamp which is an integer value.

  – *Project* represents the name of a project. Each of them can have many different phases.
  – *Phase* is the second layer of the hierarchy, phases belong to one and only one project and can have many blocks.
  – The *Block* metric is the name for a given block, which is a logical unit that has a function inside the overall ASIC. They are developed under one phase only and have at least one run.
  – A *Run* is an arbitrary name given by a designer to identify a run, it usually contains many steps and is always attached to one block.
  – *Step* is a name that describe the work done by the EDA tool, most of the times is one of the names described in Section 1.1.
  – The *Technology* variable describes the technology that is used in that project, technology does not change inside a project.
  – *Tool* is the result of concatenating the EDA tool used to execute an step itself with the version used
  – *Time stamp* is an integer number that tells the number of seconds since the UNIX epoch at submission execution moment.
  – *Previous step* is the name of the previous step with combination of previous run, it is possible to get the predecessor step that generated the actual one.
  – The *Previous run* is the name of the previous run with combination of Previous step, it is possible to get the predecessor run that generated the actual one. Notice that the previous of project, phase and block are not needed because they are always the same for a run and a step.

- **Area**

  The area set of variables contains all metrics related to the area of a block, which are real values except the ones with the count word which are integer numbers.

  – *Total area* is the silicon area in square millimetres occupied by a given block.
  – The *wire length* is the sum of all the wire inside a block in meters.
  – *Combinatorial* is the amount in square millimetres of space occupied with combinatorial standard cells.
  – The *combinatorial count* is the total number of combinatorial standard cells used inside a given block.
  – *Sequential* is the amount in square millimetres of space occupied with flip flops.
  – *Sequential count* is the real number of sequential standard cells in a block
  – The *memory area* are the square millimetres inside the block devoted to memories.
  – *Memory count* is the number of memories used in a block

- **Timing**

  The timing group has several metrics, but they are grouped depending on two problems, that they are representing. There is setup and hold, the first one is a problem generated when a path is too slow and the cycle gets propagated after the next cycle has started. The hold problem is the opposite of the setup, so a path has its signal propagated before it should, causing problems because it is too quickly. Inside each of these two problems there is also another classification depending on whether the signal described is internal or external. Internal means that the affected path is inside the block and it has no interaction with other blocks, and external means that the problem is referred to paths which have interaction with other blocks. So, in total there are two variables with all four combinations resulting in having prefixes: *holdint,holdext,setupint* and *setupext*. For each of them the following information is provided:

  – *Total FEP* is the actual number of failing end points that do not satisfy the constraint that are describing.
  – *TNS* is the total negative slack in nanoseconds, which means how much slack exist inside a block, so it is the addition of all the negative paths.
  – The *WNS* is the worst negative slack, so it is a real negative number that represents the path with more nanoseconds of slack.

- **Congestion**

  The congestion set has five variables, which are computed after dividing the block into segments and then it is computed the amount of chunks with problems. Congestion problems happen when there are too many paths trying to go through the same tracks.

  – *Horizontal congestion* is the absolute value of horizontal problematic segments in a block.
  – The *congestion horizontal ratio* is the ratio between the absolute value and the total number of horizontal chunks.
  – The variable *vertical congestion* represents the absolute value of vertical segments with problems.
  – *Vertical congestion ratio* is the ratio between the absolute value and the total number of vertical chunks.

3.2 Preprocess data

In this section the preprocess applied to the data is detailed, it is a key aspect in any project, where machine learning techniques are used. In our case is the part, where all the knowledge about the process of ASIC design is translated to data. This includes the treatment done to outliers, missing values and some distributions, the manage of categorical variables and how information about a step or a block is used.

3.2.1 Data Treatment

This subsection is devoted to the set of transformations applied to the data. As it is introduced in Section 3.1.2 the total number of samples is over 55.000 and increasing a few hundreds per week, because there are designs that are being done in parallel to this project. But this huge amount of examples has some problems, the main one is that they were stored in different moments in time, using different approaches and some of them do not contain information about all the metrics.

The cause of this heterogeneity of the data is that beforehand it is very difficult to think about the set of variables that can be meaningful to a project like this one. So for example, there is the case of the *congestion* variables that have a 70% of missing values because they were added a couple of months ago and there is no historical data available. This leads us to the first problem, missing values.

Missing values

There are different approaches to solve this problem, one would be compute the value that is missing by doing the mean of the variable and store that value as the real one. This is useful when the metric that is missing is the only problem of the row, meaning that the rest of the data is correct. Another solution is to discard the whole sample, because the information provided can not be trusted.

In our case, both solutions have been applied but it has to be distinguished when one or the other is used. The first approach is suitable for the *congestion* metrics, which have a 70% of missing values, the cause has already been described but it is basically that the variable is being computed for the new runs and it was not calculated before, so the missing values are due to a lack of historical data of this concrete metric. In such case, it is reasonable to fill lacking gaps with information, however the problem is determine which one. If the mean is used, then some steps like import, floorplan and power grid would appear to have congestion problems, because the mean is strictly positive. This information would be non sense because the congestion can be computed once the route has been done and not before. So it is preferable to have a zero instead of the mean, to avoid having strange values.

The other case that appears in our data set is when a huge amount of metrics of a sample is missing, for example the *timing* variables is not provided for all the projects. Then it is unfair to add the average of the column, because it would mean that the *timing* metrics of one concrete project are the mean of the other projects and they are always the same, which would make no sense at all. Unfortunately for this kind of data, the only solution is to be deleted from the data set. This might look a little bit unnecessary but the truth is that having a lot of examples that are not contributing to give a realistic and big picture of the underlying process would be much worse than deleting them.

Apart from the data that contains not computed metrics, there is also the circumstance when something during the execution fails, during the generation of the report or the user simply stops the computation. This usually is reported as having CPU time, runtime and memory zero. This events are also deleted because they are bad results that do not provide information on this metrics apart from the fact that the output for other variables is not defined.

The result of applying these techniques implies getting rid of a big amount of unnecessary examples, reducing the total number of samples to 36.000. It is still a good amount of data to apply machine learning and it is important to keep in mind that the data remaining has no missing values and it could provide a good amount of useful information.

Outliers

The simplest way to understand the issue that is described here is looking at the histogram of the runtime in Figure 3.1. There it can be clearly seen as there is only one column with all values but one which is in the other side of the figure. The existence of this kind of values is clearly an error, because in this case it is basically saying that there is a process that has been running for years.

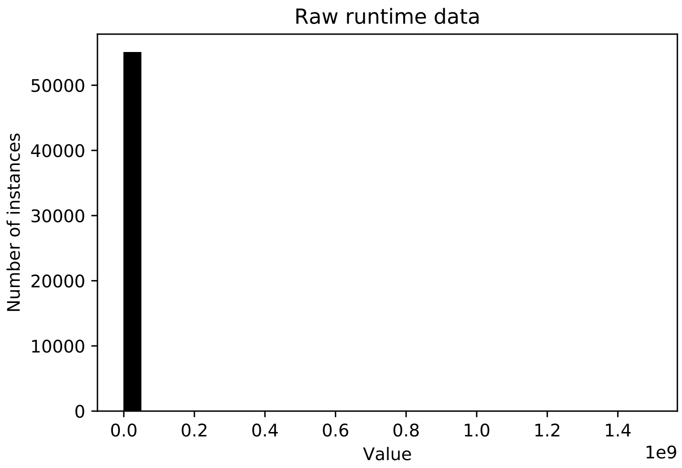

Figure 3.1: Histogram of the runtime time without any missing value.

It is uncommon to have these extreme values, which are clearly outliers or bugs derived from the reporting phase. But there are some of them in runtime, CPU time and memory. In order to clean automatically these extreme values,the mean and standard deviation is computed for each of them and then any value further than 3.5 times the standard deviation plus the mean is deleted.

The 3.5 standard deviation is a value that tries to minimize the number of outliers instances derived from a bad report. It is a number that is derived from looking at the variables, which has a very similar form with huge outliers. This big data points push the mean and the standard deviation to their side, so a distance mean plus 3.5 standard deviation is not a tight bound.

The result of applying this filter to clean the outliers can be seen easily in Figure 3.2. There are less than 800 samples deleted from the combination of the three metrics, which is less than a 5% of the amount left after the missing values are cleaned.

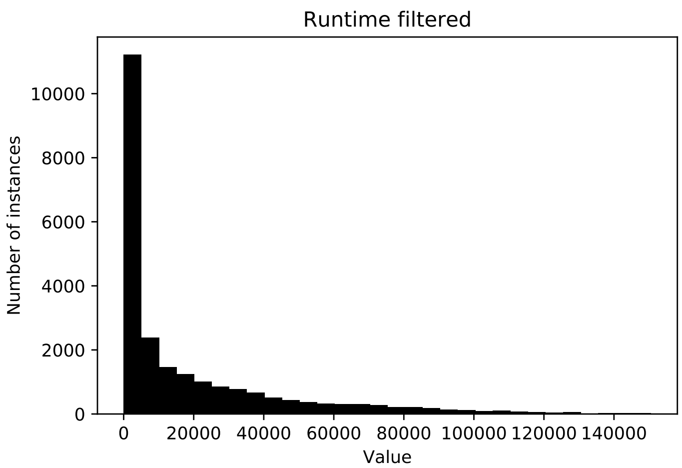

Figure 3.2: Histogram of the runtime once the filter is applied.

Transformations

This part do not deal with a problem itself, but it tries to benefit the machine learning methods applying transformation to the data so it gets easier for it to predict the new distribution. As it can be seen in Figure 3.2, data follows some kind of exponential distribution, which often cause problems to the prediction in any kind of methods.

In order to obtain a more comfortable distribution, like a normal, for the algorithms the natural logarithm is applied to CPU time and runtime, the result seen in Figure 3.3 is for the runtime. The output is stored in another column so both representations of the same variable are available.

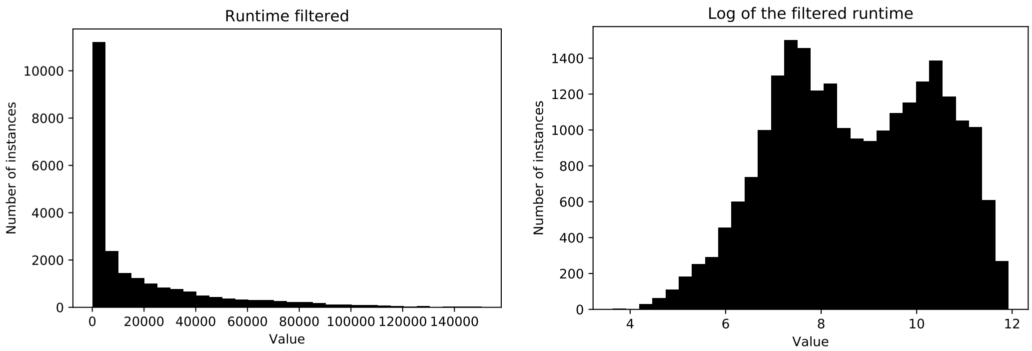

Figure 3.3: On the left, there is the runtime metric filtered and without missing values and on the right there is its transformation once the logarithm is applied.

3.2.2 Categorical data

Usually, it is a challenge to combine numerical and categorical data in a machine learning algorithm, but inside our dataset there is important information from both, in this section it is explain how they are used together.

The problem is clearly the categorical variables, which there are only a few of them that a priori contain important information. Categorical data has two different ways of getting transformed into numerical, depending on whether they have order or not. So with distinction in mind, the following classification is obtained:

- Metrics with order
  - Step
- Metrics without order
  - Tool
  - Technology

Any categorical metric that has a clear order, has a smooth transformation to numerical, which consists in labelling from 0 up to n where n is the number of classes following the order. For example, in *step* the numbers follow the list in Section 1.1.

The other method for converting categorical data to numeric has a higher cost, because it consists in adding a new column, which is a boolean metric, for each different category inside the class. One and only one of the columns of a class have to be true per each row, which indicates the category of the rows. For example, if a closer look is given to the *tool*, then the Table 3.1 appears, the name of the tools has been hidden for confidential issues (the preprocess of Section 3.2.1 were applied to obtain this data). This implies that 7 columns have to be added if the information of *tool* wants to be taken into consideration for the algorithm. In the case of the technology there are 8 different categories, resulting in a total of 15 new boolean columns of zeros and ones.

Tool	**Number of instances**
$Tool_1$	17217
$Tool_2$	11430
$Tool_3$	333
$Tool_4$	2636
$Tool_5$	179
$Tool_6$	1826
$Tool_7$	1721

Table 3.1: The number of instances grouped by tool

3.2.3 Step information

The step is the piece of information that tells us exactly what task is being done by the EDA tool, as it is described in Section 1.1. So the first intuition says that it should be a determinant metric when trying to predict any variable. This feeling can get stronger after looking at the runtime for each individual step, for example if only the power grid is seen like in Figure 3.4, then it looks like the prediction can benefit from the step information.

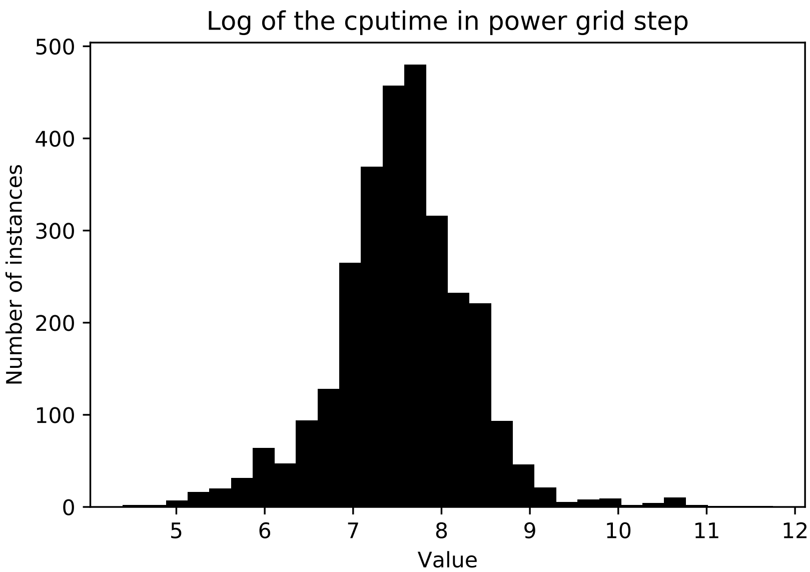

Figure 3.4: Histogram of the logarithm of CPU time data belonging to power grid step.

One approach to provide this information to the system would be using the describe technique in Section 3.2.2 which would be give number to each step in the appropriate order. But as a result of how critical this metric looked like, instead of following that technique, it has been tried to build specific models per each individual step. So the total number of models is multiplied by the number of steps.

3.2.4 Block information

Another factor that would have a beneficial impact on the predicted metrics would be the possibility of having information about the designed block. A senior designer would be able to identify a block by its difficulty looking at its metrics, so having this knowledge somehow translated into the system should make a difference in the performance of the models.

Unfortunately the same technique like in the step can not be applied for one main reason, which is the generalization purpose of the application. The only information that allow a characterization between blocks is their names, so if a model is build per each block (apart from the enormous amount of models that we would have), then it would not be possible to predict anything after many iterations of the design of a block were made. This is because the name of a block is something too specific (also arbitrary) and it does not generalize at all, so the use of that metric is not an option.

So in order to capture the essence of a block, some ratios between variables are computed, this information would not be captured by any of the methods by itself. These divisions have a meaning and a purpose, which basically is to enable the method to compare numbers that just by looking at the absolute value of the involved metrics would be hidden. All results of the following list are new columns to the data set:

$$ratio_1 = \frac{\text{area}}{\text{memory area}}$$

An important information is how much area of a block is occupied by memories.

$$ratio_2 = \frac{\text{area memory}}{\text{memory area count}}$$

This ratio is averaging the size of the memories of a block in order to give an idea of how big (in area, not capacity) the memories of a block are.

$$ratio_3 = \frac{\text{wire length}}{\text{area}}$$

The concept behind this division is to know how a block is occupied by wire length, which could be a property of certain blocks.

$$ratio_4 = \frac{\text{TNS total}}{\text{FEP total}}$$

This ratio is the general one for the *setupint*, *setupext*, *holdint* and *holdext*. It expresses the average negative slack for each of the four variables, which can be very useful because the worse negative slack and the failing points by itself hide this information to the models.

$$ratio_5 = \frac{\text{area}}{\text{CPU time}}$$

The idea is to express the amount of time that is required for square millimetre of area.

$$ratio_6 = \frac{\text{CPU time}}{\text{threads}}$$

The information provided by this ratio should basically be an approximation of the runtime without I/O interruption and user interaction and assuming that the EDA tool implements the algorithm fully in parallel.

3.3 Methodology

In this part of the report are described different decisions involving machine learning, which includes the schema to evaluate and create the models and the environment where the whole set up is build.

3.3.1 Environment

Let's start defining the program language, in our case the chosen one is Python, especially with the use of Sci-kit Learn library. The alternative is usually R and their packages devoted to machine learning, which would also be a good option. There are similarities between both options, they provide a friendly manner to call most of the methods presented in Section 2 without having to implement them, which prevents having a potential source of errors. Apart from the machine learning methods themselves, there is a set of other auxiliary functions that help a lot to perform tasks like visualization, error computation and data transformation. But there is a reason to prefer Python over R for this project, which is its flexibility when it has to be integrated inside a system.

One particularity that is also solved easily thanks to Python is the fact that our program has to be executed first to train the methods but they will need to be executed after they are built to obtain predictions. It would be a waste of resources if any time a prediction is needed, then the whole system has to be computed. So there are two clear stages in the cycle of our program, training the methods and then applying them. Python gives the possibility to serialize any kind of object to a byte string, store it to a file in disk and then the original object can be reconstructed at any desired moment by doing the reverse process. Then this is the way to proceed, any possible object that is needed further in the future is serialized and stored, this includes methods and also objects that deal with the transformation of the data because the exact same transformation has to be applied to all inputs, otherwise it would be unfair.

3.3.2 Model Schema

Here the methodology behind how the machine learning techniques are built is explained. It has been followed the train-test split of data to apply cross validation in the train data set to obtain the parameters of the models. But before explaining with details what briefly is stated previously, it is important to get familiar with the some of the concepts.

Train-Test data split

The data set is split in two different sub sets, in our case the train part has 60% of the total amount of example and test has a 40%, which are the common values in machine learning applications. The split is made at random so each sample inside the set has a probability of being at the train or the test, independently of the metrics that contains or the position in the data set. It has to be random to enforce the two sets to be as uniform as possible, so the conclusions in one set should ideally be applicable to the other. This partition has to be done to avoid the overfitting of our model.

Overfit

Overfitting is the problem that appears when a model is getting the details of the training set, instead of its generalization. This happens when the method is trying to fit into data by reducing the error between the model and the data points. For example in Figure 3.5 it can clearly be seen the problem, on the left of the figure there is a graph where the model is under fit and capture only the main behaviour of the data points, while the other one is an extreme case, from which it is very unlikely to obtain a good prediction for a new point.

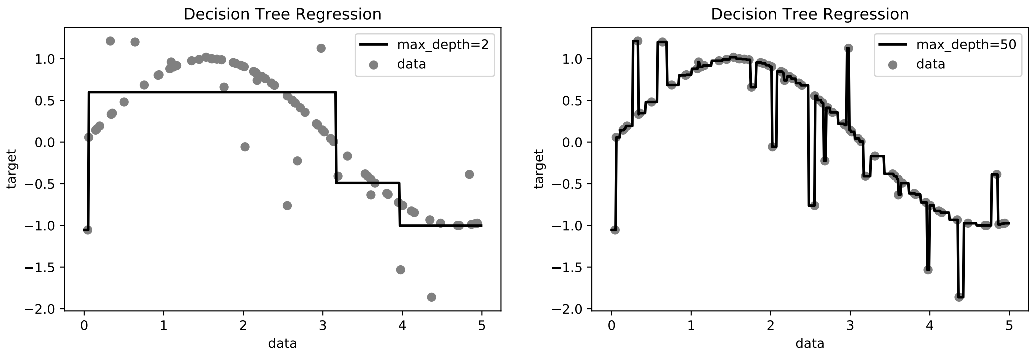

Figure 3.5: On the left, there is an under fitted data, while on the right there is an example of overfitting.

So, overfit is very dangerous because it can lead to have huge error in the prediction, this is the logic behind the split between train and test. A model is trained only with a part of the data and tested the performance with the other set. Proceeding like that the test data has not be seen at any point by the model and it was not considered to build it, then the error of the prediction between them should give an accurate idea of how will perform the model in the future.

This problem is usually avoided or at least minimized having a large data set and penalizing the complex models. Another possible action is to change the parameters that control the model, which are often called hyper parameters.

Parameter selection

As it is explained in Section 2 the parameter of each model has a huge impact in the way it is build, effecting its capacity to give useful predictions. There are two general options to select the parameters for a model:

- Exhaustive search, it is the simplest approach to the problem. Provided a set of possible values for each different parameters for a given model, then all the possible combinations are tried, this implies building a model for each configuration of parameters. Finally the best parameter is obtained. The computational expense grows quickly and it can lead to an unfeasible number of models to be tried.
- Random search was proposed in 2012 [1], a distribution for each parameter is given and a maximum number of iterations. Then using these iterations a good combination of parameter is searched at random, building models and returning the one with the best performance. It has the advantage that it performs quicker than the exhaustive search, if the number of iterations is not very high and it obtains similar results in practice

The problem of overfiting remains because it is clear that any of the proposed solutions will increase it instead of decreasing it. Because the best selected model will be the one with less training error, which would easily be the one that cares more about details and not the generalization. This is because the parameter selection can not be done alone; it has to be done with the use of another technique which is cross-validation

Cross-validation

There are many ways of doing cross-validation, in our case the K-Fold algorithm has been selected with $K = 10$, because it is proven to be the best method for real world data sets [9].

The idea is quite simple, it takes the training subset and it is partitioned in 10 different samples of same size where each point is taken only once. Then for each of the folds a model is build, using the non-selected data as a training set and the selected ones as test data, which is usually called validation data. Finally, the goodness of the model is the average of its performance in each split. The non overfitted models are preferred over the others because they perform better overall folds, while the overfitting ones tend to misbehave.

To sum up, the combination of all the previous concepts is necessary to obtain a reliable model and that is exactly what it is done in this project. Once the data is preprocessed, then it is divided into train and test, the train is used to perform a search of parameters using cross-validation and finally the best model is obtained. The goodness of that model is tested using the r^2 error or coefficient of determination over the predicted values of the model and the real values of the test subset.

Chapter 4

Results

The results obtained after applying the described machine learning techniques in Section 2 and the methodology described in Section 3.2 are detailed in this chapter.

It is important to highlight that any two methods compared in this document have been trained and tested using the same two disjoint sets, so the comparison is based on the same examples because it is intended to be as fair as possible.

The result is usually expressed using the r^2 score or coefficient of determination, which is defined as:

$$r^2 = 1 - \frac{\sum_{i=0}^{n}(y_i - p_i)^2}{\sum_{i=0}^{n}(y_i - \hat{y})^2}$$

Where,

n is the total number of examples

y_i is the true value for the ith example of metric y

p_i is the predicted value for the ith example of metric y

$\hat{y}$ is the mean value of the variable y

As it can be derived straight from the formula, the range of values are inside $(-\infty$,1]. This is due the fact that the prediction can be very far from the real value, making the denominator extremely large. Also, a value of 1 applied to a method means a perfect fit. Often in regression a model gives good predictions when the value is above 0.8 and they are considered excellent above 0.9, obviously this depends on the application but it is a good general way to interpret the value.

4.1 Prediction

In order to understand the decisions taken to obtain the final results, it is necessary to comprehend part of the intermediate outcomes that were obtained. Some of the most transcendental outcomes are shown here, especially those which had an important impact in decisions or conclusions.

Any results shown here are a consequence of applying the preprocess explained in Section 3.2.1 to the target model. This includes, filtering missing values, outlier/bug detection, ratio computation and logarithms transformation. Apart from that, the input variables of any model are all metrics available explained in Section 3.1.2- This is done because a priori there is no knowledge about, which are more determinant to explain the predicted variables.

Let's remember that in this project: runtime, CPU time, memory, leakage and area utilization are the predicted metrics using machine learning algorithms. Thus, all plots, tables or histograms shown in this section, can refer to any of them. For simplicity, it is not shown every possible combination of plots, tables and histograms for the five metrics. Only the ones worth showing are presented in this document.

4.1.1 Initial results and decisions

In this section it is introduced the early decisions that defined the project. One of the main features of this phase is the aim of exploration about how each model performs, if there are differences between them, which parameters of the models would be used and a long list of possibilities that had to be explored because a priori it is impossible to determine much about the problem.

Among the list of design projects with data available, there is one clearly distinguishable from the rest, which is the main project. The essential variation is its size, more or less half of our data belong to it. Also apart from that, it is an ongoing project, which implies two interesting properties, the first one that there is new data being created every week and the other is that any result can easily be integrated to the system and it could help the designers from an early stage of the application.

For these reasons, it has been selected as the starting point for the project and most of the results shown here are built from this data. There is a point where this changes and also data from other projects are mixed together to get a much general model but for this section it can be assumed that the data used is from the main project.

Before showing the results, there is one model called $ZeroModel$, which was not described in Section 2, because it is not a machine learning algorithm. It is the simplest method, which consists in giving the previous value of the metric as a result of the prediction. For example, in the case of the memory, the model would simply give the value of the memory used in the predecessor as the output for the current prediction. Another anomalous value that can be found in the results is the 'All' step, which is not an ASIC step, in fact all steps together form it, so in those models all data available is used.

The first interesting results are shown in Table 4.1, Table 4.2 and Table 4.3, each different table corresponds to one predicted variable. Also in each table appears the different machine learning models as rows and the different steps as columns. The intersection of row and a column gives the r^2 value for the model in the given step, so each value is created by one unique model.

There are a few conclusions that come out of these tables; one is that models seem to predict certain metrics better than others because the models have a larger score in general for the memory than for the CPU time. For example, the random forest has a 0.8573 r^2 value for CPU time in Table 4.1 while it is increased when looking at Table 4.2 where it has a 0.9467. Another is the difference in score between models; there are some of them that tend to perform better than others. Regarding the results tree-based methods, which are CART and Random Forest, have the most precise answers, for instance in the case of CPU time they are the two only models with a higher value than 0.7.

Models	Floorplan	Power grid	Placement	Clock tree	Route	All
Zero Model	0.1013	-0.6121	0.4479	0.0392	-0.4719	-0.3849
cart	-62.9150	0.0296	0.4485	-0.1277	-0.0760	0.7312
Linear Model	-7.9844	-8.9456	-4.5361	-3.4253	-6.1918	-1.6880
SVR	-3.9785	-2.7497	0.4519	-0.8984	-1.6546	0.6440
MARS	-4.8845	0.0084	0.5711	-1.6208	0.1699	0.6573
NeuralNet	-0.5183	-0.0481	0.4780	0.1862	0.0130	0.3978
adaBoost	-0.0008	-0.0139	0.4823	-0.0275	-1.7699	0.0675
Random Forest	0.6369	0.0418	0.5160	0.3122	0.0484	0.8573

Table 4.1: r^2 score of models predicting CPU time for each different step and method

The zero model shows how the variable changes from the previous step to the current one, so if the metric does not change independently of the process, the r^2 would be 1. Having high values, for example in 'All' column from Table 4.2 is almost 0.7, which is quite high. Thus, it is revealing that the memory variable does not vary from one to another. While it explains that area utilization changes a lot, because in Table 4.3 its value is -4.0901

But the most notorious result is the fact that having specific models for steps does not lead to a better predictions, even the work done in each of them is very different from the rest and it should be one of the main factors to determine the resources. In the case of Random Forest, it is extremely surprising that it performs better having all the data together as a set than in any other specific case.

Models	Floorplan	Power grid	Placement	Clock tree	Route	All
Zero Model	0.0607	0.3068	0.4139	0.4255	0.1065	0.6983
cart	0.2466	0.3041	0.4294	0.3180	-0.0871	0.8692
Linear Model	-6.0177	-2.9029	-4.2740	-6.6231	-0.4911	0.6706
SVR	-0.1620	0.1674	0.3727	0.4456	0.1479	0.8542
MARS	0.3203	0.4495	0.5052	0.5311	-0.1060	0.8466
NeuralNet	0.0745	0.1629	0.2979	0.5099	0.0811	0.8132
adaBoost	-3.1133	-0.3734	0.4502	0.3917	0.5342	0.7566
Random Forest	0.2560	0.3693	0.5191	0.5238	0.1768	0.9467

Table 4.2: r^2 score of models predicting memory for each different step and method

Models	Floorplan	Power grid	Placement	Clock tree	Route	All
Zero Model	0.4488	0.6766	0.6195	0.5618	0.4197	-4.0901
cart	0.4660	0.6710	0.6257	0.5633	0.3794	0.8282
Linear Model	-9.0514	-8.0591	-6.7264	-0.4547	-5.3856	0.6603
SVR	0.5164	0.5344	0.5268	0.1315	0.3782	0.4994
MARS	0.4090	0.7273	0.6418	0.4186	0.4004	0.8306
NeuralNet	0.4504	0.5236	0.5935	0.5119	0.2352	0.6956
adaBoost	-0.7533	0.5845	0.6563	0.6208	0.6972	- 0.5617
Random Forest	0.5344	0.6598	0.5413	0.5208	0.5367	0.8720

Table 4.3: r^2 score of models predicting area utilization for each different step and method

This last conclusion is important because it clearly shows that there is no need to specialize models for each step. Therefore, it can be saved a lot of time and resources building just one model, which uses all data available. Then, it is decided that the data is no longer sliced depending on the step.

This result contradicts our a priori believe that the step was one of the major factors in resource consumption and metric prediction. But before being able to express that step is not relevant, an alternative prediction is done, where instead of a regression a classification is done. It is tried to classify the step value, without providing any information of it as input. So this is meant to tell if just with data, the methods are able to determine which step is being executed looking at the information from the previous. The result is a high percentage of success in the classification and less than a 5% is wrongly predicted. To give a visualization of this result, the first three layers of a CART model are shown in Figure 4.1

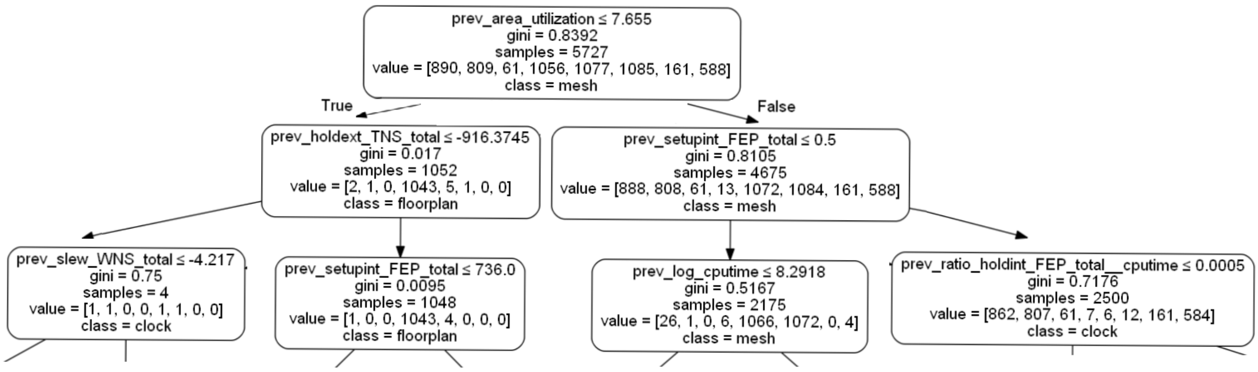

Figure 4.1: CART classification of steps without having information about it as inputs.

The model, where steps can be classified with high accuracy without having any information about it, is a key point to explain the reason behind why is useless to split data by steps. The conclusion is that the information about the steps is already implicit in the data, so there is no need to specify further because each method has that intuition and this specialization comes with the cost of having fewer examples per method, which have an impact to the chances of generalize of the algorithm.

Another result worth showing is a comparison between results with or without preprocess. Without preprocessing is defined as data with filtering missing values, otherwise algorithms would break because they can not deal with nulls directly.

Looking at the results of Table 4.4, it seems that there is a clear relationship between accuracy and preprocess. All scores from machine learning models improve when preprocess is done. Especially in the case of SVR where the value is 0.2263 without preprocess and it rises up to 0.8542 in the other scenario. Overall, the ones that are able to perform better are still the tree-based algorithms, which are also the ones that have the smallest loss when no preprocess is done, from 0.8692 to 0.8307 and from 0.9467 to 0.8937 for CART and random forest, respectively. The results obtained for area utilization and leakage are very similar to Table 4.4, so they are not shown explicitly in this report because the conclusions would be the same.

Models	r^2 without preprocess	r^2 with preprocess
Zero Model	0.7673	0.6983
cart	0.8307	0.8692
Linear Model	0.6489	0.6706
SVR	0.2263	0.8542
MARS	0.7163	0.8466
NeuralNet	0.6064	0.8132
adaBoost	0.5471	0.7566
Random Forest	0.8937	0.9467

Table 4.4: Comparison between performance in models with and without preprocessed data for the memory metric

The impact of applying or not the preprocess to runtime and CPU time is more determinant than in previous cases. The outcome corresponds to Table 4.6 and Table 4.5, there it can be checked that the impact of non preprocessing the data has a huge cost in the overall behaviour of the methods. For instance, the r^2 obtained in MARS model has a value of 0.6573 with preprocess but if nothing is applied to the data the final result is a -9.0136. This drop holds for all the other methods, thus the preprocess is an important and needed step, even for the most robust model which is random forest that goes from an r^2 of 0.8573 to a 0.5827.

Models	r^2 without preprocess	r^2 with preprocess
Zero Model	-0.0542	-0.3849
cart	0.3375	0.7312
Linear Model	-0.6413	-1.6880
SVR	-7.3369	0.6440
MARS	-9.0136	0.6573
NeuralNet	0.0552	0.3978
adaBoost	-0.2174	0.0675
Random Forest	0.5827	0.8573

Table 4.5: Comparison between performance in models with and without preprocessed data for the CPU time metric

Models	r^2 without preprocess	r^2 with preprocess
Zero Model	-0.1906	-0.5738
cart	0.3922	0.7217
Linear Model	0.0599	-0.9198
SVR	-1.5771	0.6781
MARS	-2.3020	0.5995
NeuralNet	0.1788	0.3997
adaBoost	0.3975	-0.0054
Random Forest	0.6146	0.8344

Table 4.6: Comparison between performance in models with and without preprocessed data for the runtime metric

The results on the preprocess of the data are the expected ones; it would not make any sense to obtain similar models if no treatment is done to the data. Especially in the case of runtime and CPU time it is a critical procedure to be done. Without the preprocess there is no model able to predict anything about the desired variable. Apart from that, the results also show that the most precise method is the random forest, which gets a performance much precise than any other method.

Once all the intermediate results are given, it is time to detail and discuss the final results obtained using the main project data. There is a table that gives an overview of the whole performance and also a few figures that are useful to visualize other aspects of the model apart from the r^2.

Table 4.7 provides two important categories of conclusions:

- Model performance, there is a method that gets better results than the rest in all desired metrics : random forest. Apart from that, there is also the CART algorithm that gives good predictions. This outcome is not surprising because as it is explained in Section 2 random forest is an ensemble method that uses the advantages of the CART technique and overcomes its disadvantages to obtain a better model. Another interesting result of the Table 4.7 is the fact that the zero model has a 0.7760 r^2 value predicting leakage is an indicator that it does not vary a lot from one step to another.
- Overall result, all random forest outcomes are very good in the sense of precision and accuracy of prediction. The case of runtime, which it has shown to be the most difficult one to be described, has a value of 0.83 which is a good value for most applications included the one described in this project. Here the main point of predicting runtime is to give information to the designer so he can decide better, for this purpose there is no need to get the exact amount of seconds that a prediction will take. Let's remember that this value can vary from minutes to days. So this method provides that idea with a huge precision. This same argument can be extrapolated to the rest of metrics, which have great models to predict them, especially memory that has an outstanding 0.9467.

Models	Memory	Runtime	Area utilization	CPU time	Leakage
Zero Model	0.6983	-0.5738	-4.0901	-0.3849	0.7760
cart	0.8692	0.7217	0.8282	0.7312	0.8389
Linear Model	0.6706	-0.9198	0.6603	-1.6880	0.6578
SVR	0.8542	0.6781	0.4994	0.6440	0.5623
MARS	0.8466	0.5995	0.8306	0.6573	0.8017
NeuralNet	0.8132	0.3997	0.6956	0.3978	0.8117
adaBoost	0.7566	-0.0054	0.5617	0.0675	0.1272
Random Forest	0.9467	0.8344	0.8720	0.8573	0.9001

Table 4.7: Computed r^2 score for every metric and every model inside the main project.

In order to provide more visual results, a plot representation of the goodness of fit is provided in Figure 4.2 and Figure 4.3 for runtime and memory, respectively. Also a histogram of the distribution in the magnitude of the already mentioned metrics is provided in Figure 4.4 and in Figure 4.5. They are selected because they are the two extreme cases and the rest of models lie in between.

Let's start with the comparison of Figure 4.2 and Figure 4.3, both corresponding to a plot that has as a x-axis the measured values and the predicted values as y-axis. This implies that the discontinuous line is a perfect prediction of a real value. Both are generated using a sample of 2000 points selected at random from their respective test sets. The gray zone is a range of ± 7.5% around the perfect fit, providing a total 15% of width.

As it can be seen, from the figures, the distribution of points is different and in the runtime case they tend to be further from the centre line, than in the case of the memory where it seems that most of the points are between ranges. This effect is a direct cause of the error, which is larger in the first metric, while it is much smaller in the second. In fact, a 70.3 % of the total points of the memory are inside the range, while in the other case this percentage drops to almost half of it to 35%

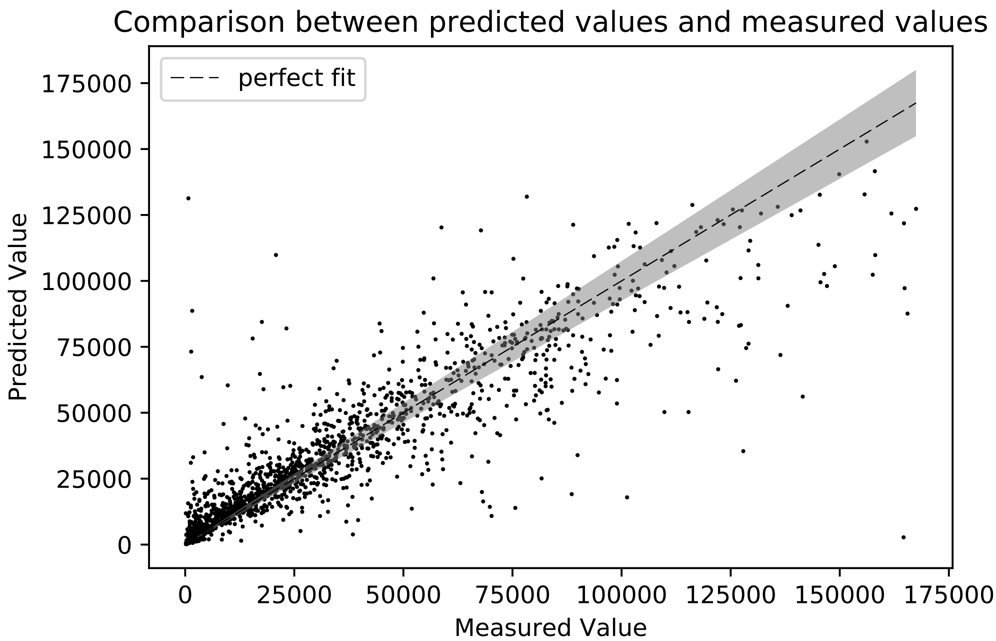

Figure 4.2: Plot with 2000 points of measured value and predicted value for runtime, the diagonal represents a perfect fit between predicted and reality

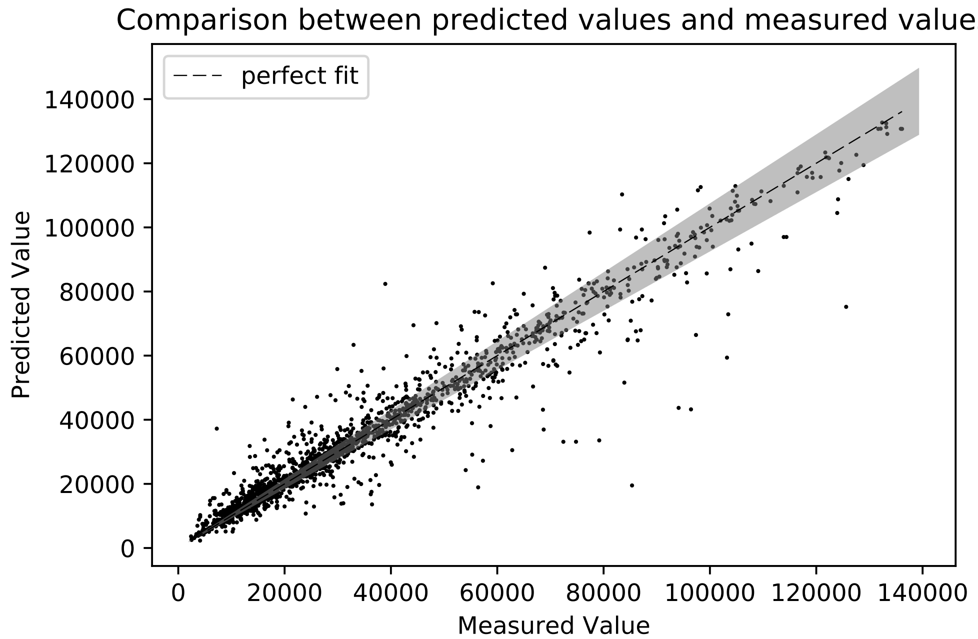

Figure 4.3: Plot of 2000 random points of predicted against real value for memory, where the diagonal represents a perfect fit between predicted and reality

To provide another point of view of the error, there are two histograms of it: one in Figure 4.4 and other in Figure 4.5, for runtime and memory respectively. This representation has been computed as:

$$x_i = \frac{|y_i - p_i|}{y_i}$$

Where,

x_i is the output that is shown in the histogram

y_i is the real value of i-th number

p_i is the predicted value of i-th number

This computation gives a sense of how big is the error related to the actual value. The error decay fast in both cases; especially for memory, where the error is much smaller. In the runtime besides the drop of the error is also notorious the existence of a tail that hardly gets to zero, this indicated that there are some predictions where the error is quite big, because the histogram captures errors three times larger than the actual value. This does not seem to happen for the memory variable where the distribution decays to 0 before reaching the 1.5 error. Actually, the percentage of data with an error smaller than 10% is 35 in the case of runtime and 70 in the case of the memory. This difference gets increased when looking at the data that remains with an error greater than 50% of the true value, for the runtime it is a 19.1% and just a 2.45%. This means that almost 98% of the data has an acceptable error, which is a great result.

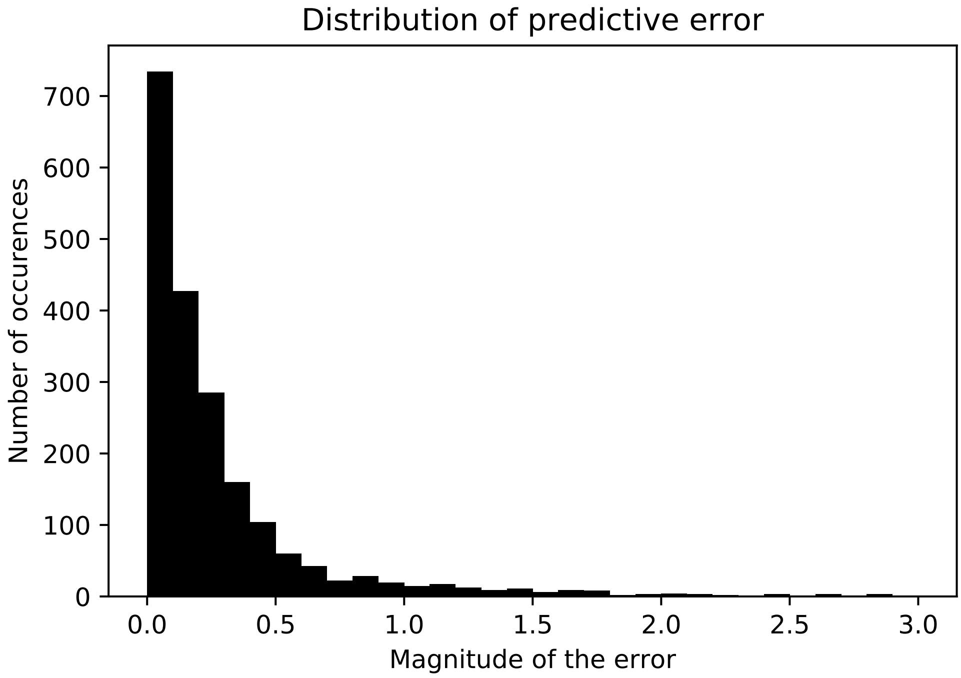

Figure 4.4: Histogram of the error distribution computed as the error divided by the true of runtime value from a total of 2000 points.

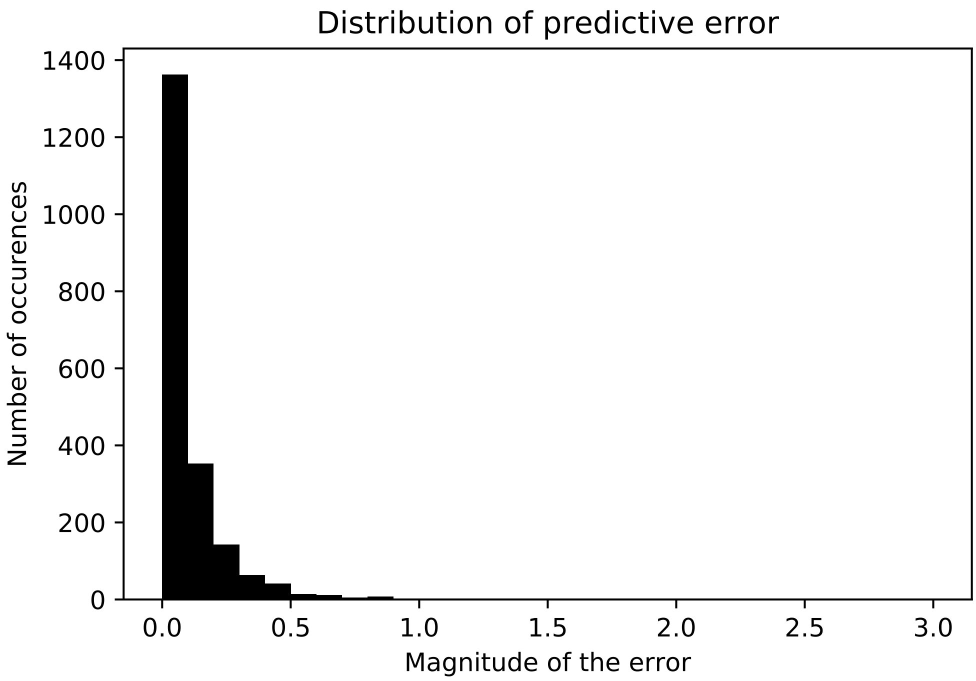

Figure 4.5: Histogram of 2000 points, with the error distribution computed as the error divided by the true of memory value.

Conclusions

In the previous section it has been shown some intermediate results that are important to understand the decisions that have been taken. All information provided is generated using data from only one project, which is the biggest with data available. In order to give an overview of the conclusions obtained after looking at the results, the following list is provided:

- The step metric is not necessary to obtain good models, so there is no need to split data based on step.
- Preprocess has been proved to be a key part of the project. Without the whole preprocess, which consists in outliers filtering, missing values treatment and transformations, there is no chance for any model to predict any metric.
- Regarding to machine learning models, random forest and CART have proved to be the most accurate and robust. Especially random forest, which provides a high precision in the prediction of all metrics.
- Looking at the results by metrics:
  - Memory: There are excellent results for memory models, in fact, it has been shown that it is possible to predict this variable with a low error. Thus, the result can have a big impact in the schedule of memory allocation, which can lead to a reduction of cost.
  - Runtime: The prediction of runtime has a good precision, however it is the variable with the highest error in its prediction. This is due a lack of information of metrics, such as, input/output time or user interaction with the process. But it is still possible to provide an approximate value for this metric, which can allow a better organization for designers and help them to detect anomalies.
  - CPU time variable is predicted with higher accuracy than runtime. This is a consequence of the fact that CPU time is a much more stable metric that does not have many interactions with designers. For example, it can help to detect some misbehaviour where some processes are wasting much more CPU than they should.
  - Area utilization and leakage: The models that predict these two metrics have a very good performance, thus there is a little error for both of them. Using that prediction, it is possible to know before an execution whether it is going to be worth or not, or if at least is going to improve the current solution. Also the detection of anomalies can be a good outcome of using these predictions.

4.1.2 Final results

Having a trustworthy method able to predict any of the desired metrics inside our project is a good starting result. But it would be encouraging if it were possible to costless generalize this behaviour to other design projects, because this would imply having accurate prediction from an early beginning of the design steps for any upcoming project.

The transition from a model that is based only in one project to a model that gathers data from many has not been a problem. Even there are some minor changes to be added they are purely functional, for example it is necessary to add information about project and technology when fetching the previous step, otherwise the algorithm can find a previous from another project. Thus this section focuses in the results of building a wider model using the whole data set. In order to give an interesting visualization of the outcome obtained, the plots, tables and figures used have the same structure and are computed the same way that in Section 4.1.

Table 4.8 contributes giving an overview of the behaviour of the different models for each metric. There it can be seen that the overall performance of the model has a decrease in its accuracy if compared with the case where data from the main project is used for the same purpose in Table 4.7. This drop in the general accuracy of the methods is due the fact that the data used to build them belong to many projects, even from different technologies.

So, a natural though would be that information about technology should be used somehow. In fact it is introduced to methods following the one hot bit encode technique described in Section 3.2.2. The result is that it barely has any impact in the performance in the models. Such a result is very similar to the one obtained with the steps in the previous section. It is unexpected, because it seems that the technology should make a difference between projects but it does not.

There are two metrics that suffer a deeper decrease are runtime and leakage. In both cases the remaining model are still acceptable, especially leakage. Runtime is the only model with an r^2 below 0.80 for all algorithms. The performance of its predictions is not very precise as it will be discussed, but they are still able to give a valuable hint of the order of magnitude of the final result.

On the other side, there is the memory metric, which remains with the highest r^2 value, above 0.90. It is also remarkable the good performance in CPU time and area utilization that is obtain even in a heterogeneous data set.

If we look at the performance of each method, it is more obvious that the best model is the one created by random forest than it was before, which has the highest r^2 for every metric. The second model remains being CART, so the tree-based approach is key in this project. Apart from that, it is necessary to highlight that there are neuronal networks and adaBoost, which have a very irregular performance with values that can go from -1.3123 to 0.6956 and from -1.4172 to 0.7148 for neuronal networks and adaBoost, respectively.

Models	**Memory**	**Runtime**	**Area utilization**	**CPU time**	**Leakage**
Zero Model	0.7059	-0.3950	-1.5968	-0.2820	0.6483
cart	0.8667	0.6393	0.6393	0.7340	0.7772
Linear Model	-1.0475	-6.7189	0.6603	-5.4087	0.3495
SVR	0.8333	0.6781	0.5281	0.5323	-9.7340
MARS	0.8072	0.5995	0.4282	0.5419	0.3704
NeuralNet	-1.3123	-0.5753	0.6956	-0.4013	0.0492
adaBoost	0.7148	-1.4172	0.5617	0.1184	-0.1276
Random Forest	0.9188	0.7643	0.8520	0.8348	0.8154

Table 4.8: r^2 value for metrics and models using all data available.

Now let's get into a visual result of the models, as in the previous section, the same kind of plot and histograms are shown in Figure 4.6, Figure 4.7, Figure 4.8 and Figure 4.9. Plots correspond to predicted values against real values, so the discontinuous line means perfect fit, while the histograms are the distribution of the magnitude on the error computed the same way as in Section 4.1.

To obtain a better visualization both plots in Figure 4.7 and Figure 4.6 are created just using 2000 points selected at random. As it can be seen, the disparity of the points is much higher for the runtime metric, especially at the beginning of the plot where data is distributed with a high relative error with respect to perfect fit. Figure 4.7 has most of the points inside the gray zone, which means that they are not further than ± 7.5% of the actual value that should be obtained. In fact, there is about 75.15% of values within that range, while in the case of runtime the majority is outside it, exactly a 39.25% of the data is inside the gray zone and the rest is outside.

Another outcome that can be seen is that error distribution is not uniform along the x-axis, causing that at the beginning most of the points tend to be over predicted and for bigger measured values, it seems to be the other way around. This behaviour seems to be applicable for both metrics.

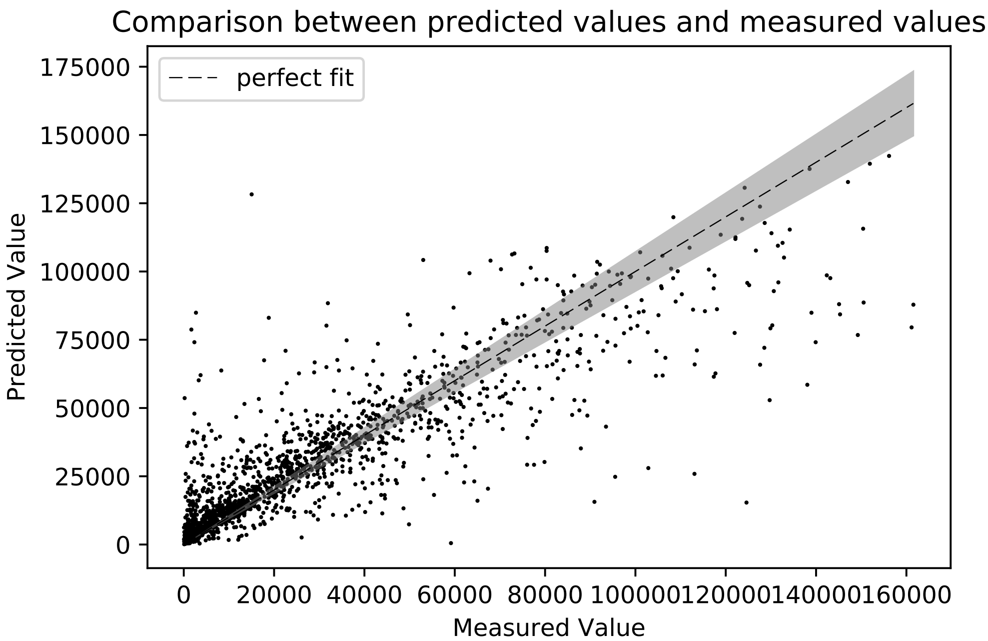

Figure 4.6: Plot with 2000 points of measured value and predicted value for runtime, the diagonal represents a perfect fit between predicted and reality

The histograms represented in Figure 4.8 and Figure 4.9,show how big the error is with respect the real value that is predicted for runtime and memory metrics, respectively. Each column of the histograms represents a 10% of error. This allow us to conclude about how big is the error distributed among points, so when compared, the error of the first one is clearly much larger than in the second. Apart from that, the error decay is much better in the second one, where it looks like the error fades up before getting errors of 150% of the real value, this behaviour does not exist for runtime until the 300% of error.

In fact, in the memory metric only a 36.1% of the total data has an error bigger than 10%, this means that almost two thirds, of the predicted values has an error less than 10 percent. For the runtime metric, the error tends to persist more and 70.2% of samples have an error greater than 10%. The decay is important and there is only a 27.25% of predictions that have an error greater than 50%. But it is still large if compared to the same percentage in the memory case, where only 5.7% of data has an error of 50%.

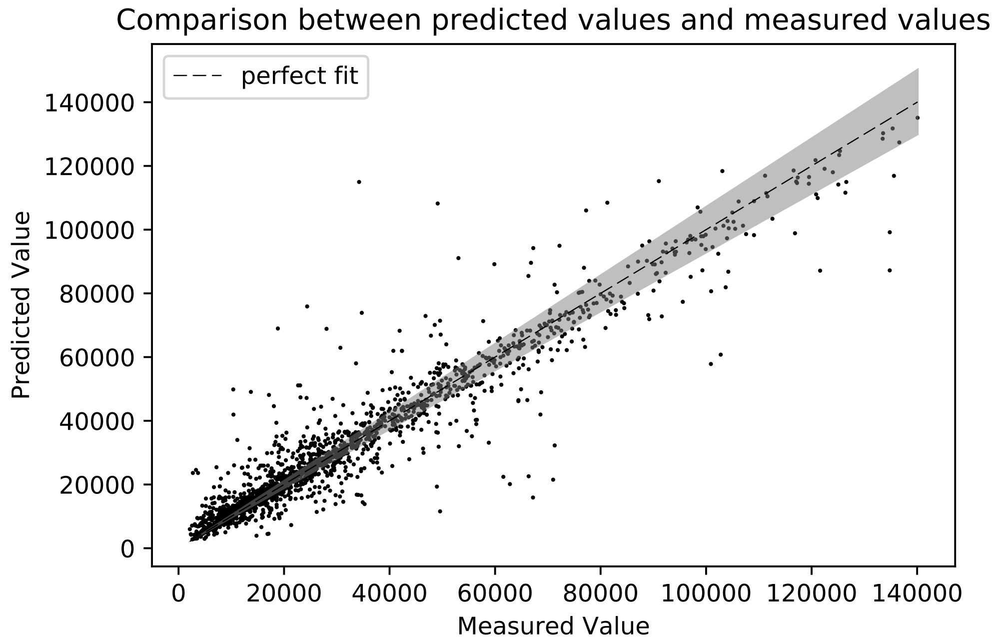

Figure 4.7: Plot of 2000 random points of predicted against real value for memory, where the diagonal represents a perfect fit between predicted and reality

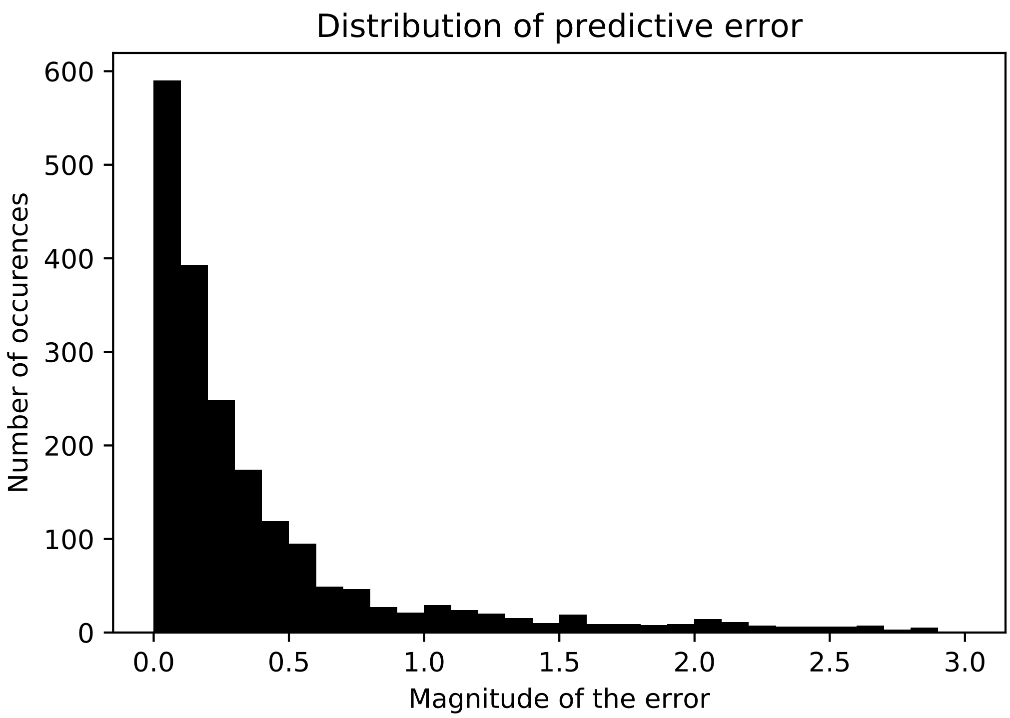

Figure 4.8: Histogram of the error distribution computed as the predicted error divided by the true of runtime value, only 2000 random points are considered.

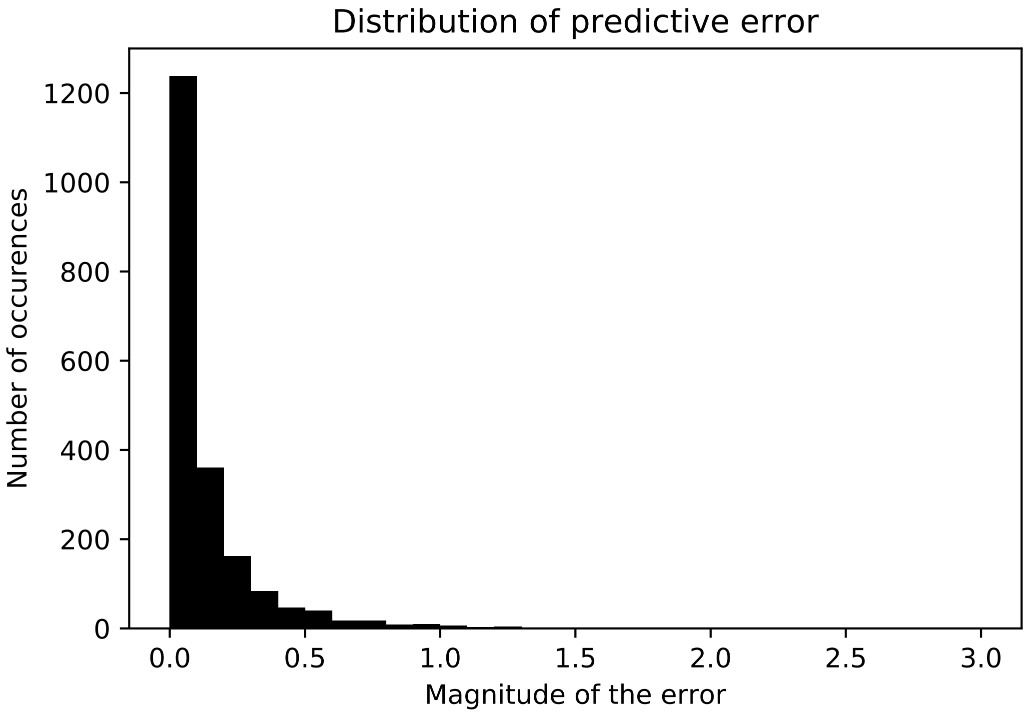

Figure 4.9: Histogram with the distribution of the predictive error divided by the true value of memory value from a total of 2000 points.

Conclusions

In this section it has been shown the results of applying the machine learning techniques described in Section 2 to predict the metrics explained in Section 1.2. This is done using all data available and not only a single project. To give an overview of the conclusions obtained after looking at the results, the following list is provided:

- The use of data from many projects, instead of data from only one project has a negative impact in the accuracy of the models. All algorithms for all variables tend to increase their error, this is a normal consequence of mixing several projects and not having specialized models. But it is still an encouraging outcome, because it generalizes the behaviour of models making them applicable to new projects, due to their project independent property.
- The technology variable has, surprisingly, no impact in the final results.
- Random forest is the best model to predict runtime, CPU time, memory, leakage and area utilization, which are all the desired metrics.
- For CPU time, leakage, memory and area utilization the exact same conclusions and results as the ones described in conclusion subsection from Section 4.1.1 are obtained, but with a lower precision. In the case of runtime, the error tends to be large, despite it still can be used as an indicator of the order of magnitude.

4.1.3 Feature importance

In this section results are shown regarding to which are the variables that have an important impact in the prediction of CPU time, runtime, memory, leakage and area utilization. All data has been used, so it is not only from one project and the results are related to the random forest algorithm.

To compute the feature importance of a random forest, the mean decrease accuracy algorithm is used. The main idea is quite simple; it determines whether a variable is important if the decrease in accuracy is high when the values of that variable are changed at random. This is done for all trees inside a random forest in the OOB space. The sum of the total variable importance is always 1. In this document, just the five with higher value are shown.

Memory

The feature importance for the memory can be seen in Table 4.9, there it can be seen that the logarithm of the memory has a huge impact with a 0.752. It implies that it has much more importance than the rest of variables combined. Then, it can be conclude that the previous memory determines a lot the memory that a process will need in the future. But as it is proved in the previous section, this does not imply that it is sufficient to predict the next memory as the one from the current step. This is what does the Zero model proposed and it has a worse performance than the random forest, even though it has a good result with an 0.70 r^2 value, as it can be seen in Table 4.8.

Variable name	Feature importance
Log memory	0.752
Area utilization	0.026
Ratio Area/CPU	0.016
Log CPU time	0.016
Wire length	0.012

Table 4.9: Top five variables importances for memory metric

Runtime and CPU time

The results of feature importance for runtime and CPU time are very similar, in fact, only the result for runtime are shown in Table 4.10, because the same conclusions and similar values are obtained. From there it can be seen that instead of having one factor that determines a lot the final value, there are four metrics that have a decent importance. The first two are the logarithm of CPU time and the logarithm of runtime with a total combination of 0.24. Then, the third factor is the number of clocks inside a block with a 0.86, followed by a ratio between area and CPU time that has a 0.073. From this it can be conclude that the computation of ratios help models to improve their precision and also that the needed time to compute the previous step is determinant to predict the next time metric. The number of clock sinks gives a sense of how much work will have to be done inside a block, because the more clock sinks, the more difficult is to execute every step.

Variable name	Feature importance
Log CPU time	0.144
Log runtime	0.097
Number of clock sinks	0.086
Ratio Area / CPU time	0.073
Number of std. cells	0.052

Table 4.10: Top five variables importances for runtime metric

Leakage and area utilization

Leakage and area utilization share from the second to the fifth feature importance as the Table 4.11, with different values and order. The first variable for both variables are themselves. They present a mix behaviour between the results obtained for memory and for runtime/CPU time. They have a big first factor with almost 0.45 of importance, followed by two variables that have a value above 0.1. In the case of leakage these variables are area utilization and a ratio. Once again, it can be seen the influence of ratios in the performance of the models.

Variable name	Feature importance
Leakage	0.446
Area utilization	0.123
Ratio Area memory/Number of memories	0.115
Area	0.062
Area memory	0.034

Table 4.11: Top five variables importances for leakage metric

Chapter 5

Conclusion

5.1 Result

The use of machine learning techniques to predict metrics related to nanoelectronic circuit design has been proved successfully. In fact, the good results obtained can have a direct impact on the process of designing ASIC, because there is information regarding metrics of the circuit and its computation that can be predicted before its execution.

An example of this impact is the memory prediction that can be used for automated processes such as memory reservation in the system where many jobs are sent. Currently, the amount of memory needed for an execution is a priori unknown, as a consequence of this fact, the quantity of memory allocated for a process can not be done precisely. This situation implies having memory assigned to tasks that will never use it, thus there is a potential waste of memory.

An accurate prediction of memory gives more precise information about the real needs of the underlying process than the current method. If provided to the system scheduler, then a lot of memory can be saved because only the needed one would be allocated for each task. Such use can have a huge impact in the total amount of required memories for a company, which would be a reduction in the costs of designing circuits.

Predictions regarding to design metrics such as area utilization and leakage allow the designer to decide beforehand if an execution is worth or not. For example, if he wants to satisfy a constrain in leakage and the prediction says that it is not going to achieve the desired value, then the computation may not be worth. This can save an enormous amount of time and computational resources.

At the moment, all prediction's information is embedded in an existence application where the reports of the different executions can be consulted by the designers, this allows them to see how models perform and get familiar with them.

5.2 Future work

There are many possibilities to keep expanding the use of machine learning in the ASIC design field:

- Expand the number of metrics we predict, not just area utilization and leakage, but all other design metrics. to detect anomalies. Each metric has its own ML model.

- Predict total amount of resources for a project, another application can be to determine the number of CPU time, memory and working hours of designers that are needed to produce a whole project. For this purpose, data should be different and more project cases are required, but it would have a major impact in the resource planning for electronic designs. In order to predict future projects, it could be used data at block level because the number of projects is small. The total number of blocks is higher and should work nicely with machine learning techniques.

Apart from other predictions, it is still possible to improve the existing solution by getting more detailed information about the design process that is not being captured at the current moment. For example, by having a sense of how much time a process waits for input, outputs or user interactions can have a positive impact on the prediction of the runtime metric.

Some techniques from data mining can be used to analyze relationships between metrics. For example, how EDA parameters like number of threads affect IT metrics like runtime or memory.

Another possibility can be embedding the whole model into the designers' flow and provide the predictions just before the execution is launched directly in the same application.

Bibliography

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[2] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[3] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. wadsworth & brooks. *Monterey, CA*, 1984.

[6] Harris Drucker, Christopher JC Burges, Linda Kaufman, Alex Smola, Vladimir Vapnik, et al. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[7] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991.

[8] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[9] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[10] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

[11] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.