MACHINE LEARNING SYSTEM FOR RESOURCE PREDICTION AND TOOL OPTIONS OPTIMIZATION FOR APPLICATION-SPECIFIC INTEGRATED CIRCUIT (ASIC) DESIGN

MARC GARCIA MASSANEDA

Thesis supervisor: JONÀS CASANOVA BACHS (MARVELL TECHNOLOGY UK LTD SUCURSAL EN ESPAÑA)

Tutor: JORDI CORTADELLA FORTUNY (Department of Computer Science)

Degree: Bachelor's Degree in Data Science and Engineering

Thesis report

Facultat d'informàtica de Barcelona (FIB)
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB)
Facultat de Matemàtiques i Estadística (FME)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Acknowledgements

Many people have helped me throughout the thesis. First of all, I would like to thank Jonàs Casanova for guiding the project on a day-to-day basis and providing everything I needed. Also, I would like to thank Prof. Jordi Cortadella for being my tutor; and my family and friends for always supporting me.

Finally, I would like to thank Marvell Technology, Inc. for giving me the opportunity to develop this project in a friendly and helpful environment.

Barcelona, 16th June 2023

Marc Garcia Massaneda

Abstract

This thesis consists in developing machine learning models to predict metrics in the field of microchip design. It has been developed in collaboration with Marvell Technology, Inc., which is a company dedicated to the design and production of microchips.

Microchips are designed and tested through some processes that require spending many resources. These resources need to be requested beforehand, and they can be wasted if the request doesn't match reality. This can cost the company a lot of money and time.

The main goal of this project is to solve this problem using machine learning algorithms and techniques. The creation of different models is carried out to find the most optimal solution for this problem.

These machine learning models are fed with data obtained from both experimental and real projects of the company. Data comes with many problems like inconsistency and outliers. Therefore, proper preprocessing is done, as well as some transformations to get valuable information for the models.

The outcome of this project is the finding of a model that uses machine learning to be able to predict these metrics successfully to strongly reduce the waste of resources.

Keywords

Microchips design, machine learning, linear regression, decision tree, random forest, neural network, bag of words, one-hot encoding, term frequency-inverse document frequency, word2vec, mean absolute percentage error.

Contents

Chapter 1

Introduction

Marvell is a company that manufactures microchips for electronic devices. The process is very long, but the focus is on the step before getting any physical form of the microchip: the hardware design. I work in the CAD department, and our duty is to develop computer-based software to aid in the design process of microchips.

To test the different designs, processes known as jobs are executed and report different quality metrics of the corresponding design. A job is a program that is executed on a computer and completes a task. In our case, jobs execute a process that has a task in the design of microchips. These jobs are executed in a computing grid, which is a group of computers that share resources. The resources of this grid are limited. Taking into account that jobs can be running for hours and even for days, there may be a moment when all the jobs that want to be executed can't fit in the grid. That's why there is a queue system in the grid that allocates a job in a computer if there is any available, or leaves it waiting until there is any free space for it. The management of the grid is very important to execute the maximum number of jobs in the minimum amount of time. This impacts the company in terms of productivity, so also in time and money.

1.1 The problem

An important feature of the computing grid is that the computers that compose it have different capacities, meaning that they are able to fit jobs that consume more or fewer resources. These resources are basically the maximum amount of memory that the job needs throughout the execution. To keep things simple, the term used will be memory. A really important thing to be aware of is that the memory that each job needs has to be requested before running the job.

The problem comes from the fact that the memory that a job consumes is only known when it ends executing, so the request has to be done without knowing exactly how much memory the job needs. The current solution is to use user-defined values: predefined values or guessed by the user, but they are known to be really inaccurate, which translates to misuse of the resources. Two situations happen if the request of memory and the actual memory used differ:

- *More memory requested than memory used*: The job is allocated to a computer with more memory available than what the job uses, so there is memory on the computer that is wasted for a period of time that the job is executing.
- *Less memory requested than memory used*: The job may be allocated in a computer with less memory available than what the job uses, so when the computer runs out of memory, the job is killed without finishing the execution, and it has to be executed again in another computer.

These situations happen often because users that execute jobs don't get any penalty if they are inaccurate: as long as their job does not get killed, they still get the results and they are happy with it. However, behind the scenes, the company is losing when users are not responsible for resources. To demonstrate that the problem actually happens, here is the situation with jobs of the last 2 months of a real project of the company.

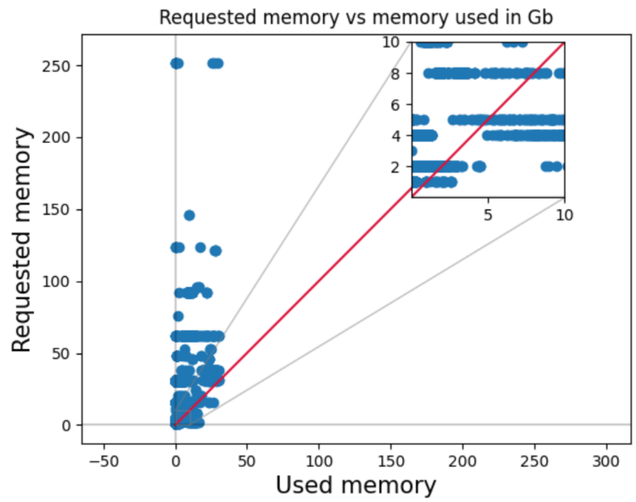


Figure 1.1: Scatter plot of requested memory compared to used memory of jobs of the last two months.

Looking at the scatter plot, it can be clearly seen that some jobs request much more memory than what they use. Furthermore, the patterns formed don't try to follow the red line, which would be the ideal situation: jobs requesting the same memory they use.

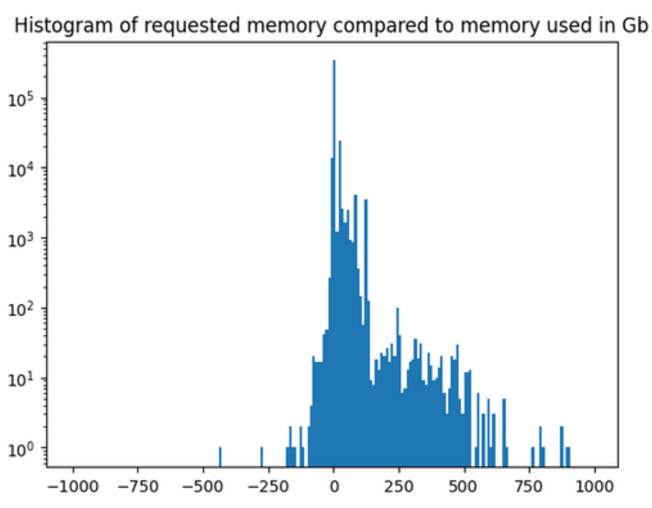


Figure 1.2: Histogram of errors (requested memory − used memory) in Gb of jobs of the last two months. The y axis is in log scale.

Looking at the histogram, the situation does not look pretty good. Most errors are of a few Gb, but there are many cases in which the error is so high. Taking into account that an error of 50 Gb starts to be unacceptable, it is clear that there are too many jobs with an unacceptable error. Furthermore, it is important to see that almost all jobs have positive errors, meaning that jobs tend to request more memory than what they use, not less.

Clearly, a change is needed. This thesis presents a solution that mainly consists of predicting the memory used by jobs, with the aim of reducing the difference between requested memory and actual memory used.

In this thesis, two types of models are developed. The first will make predictions for any job in the company. The second one will make predictions for only a subset of jobs that report many more metrics of the process they are executing.

1.2 Goals and motivations

The main goal of this thesis is the creation of a system that predicts the memory that is going to be used by a job. Moreover, the runtime is also going to be predicted to give more value to the user. Runtime can't be controlled as easy as memory because external factors also have an impact: speed of computer in the grid, connection issues... Therefore, results regarding runtime are expected to be worse than memory. These are the metrics that are aimed to be predicted with their exact definition:

- *Memory*: Maximum amount of memory that the job uses when it is being executed. It is expressed in gigabytes (Gb), which is equivalent to $2^{30} = 1024^3$ bytes.

- *Runtime*: The time that the job is executing on a computer, from being allocated to the computer until the computer is freed. It is expressed in seconds (s).

The motivations behind the predictions of these metrics are multiple:

1. *Grid optimization*: Predicting memory accurately would have a great impact on the grid. Resources would be allocated to the jobs accordingly to their needs, and no more waste of resources or jobs killed.

2. *Outliers detection*: The prediction of the metrics can also have a preventive use. Users who have spent many years executing jobs have a little intuition about the resources that a job may be using. Therefore, if the prediction they get is so different from what they think, that could be used as a warning to double-check the job and maybe find a bad configuration, what is known as an outlier. The idea is to not waste resources on the execution of outliers and be able to detect them from the beginning.

3. *Users helper*: Finally, there are other interesting uses like simple information for the users. If they know that the runtime is going to be some hours long, they are able to organize themselves accordingly, and in the end, they are more productive thanks to it. It may make the organization of your day clearer.

1.3 Document structure

The structure of the rest of the document is as follows. First of all the state of art is presented, which contains deep information on the machine learning techniques and algorithms that have been used to develop the models. The next section contains everything related to the models: the origin of data, how it has been processed, what types of machine learning models have been used, and finally the results obtained with each of them. In the end, a few conclusions are explained, to wrap up the thesis.

Chapter 2

State of the art

In this chapter, there are explained all the machine learning techniques that have been used throughout the thesis. Before, there is an introduction to how machine learning is used today and which problems can be solved with it.

Machine learning is a field that takes part in many different types of sectors: financial markets, healthcare, social media, and the list goes on and on. This tells how easy it is to apply these techniques to different types of data and achieve good results. A task related to predicting future behavior based on past examples can be really useful and that is why machine learning use has been growing exponentially in recent years.

In the following sections, all the techniques and algorithms that have an impact on this thesis are explained in detail to make the thesis easier to understand.

2.1 Machine learning models

First of all, let's take a look at all the different machine learning models that have been tested in the thesis. Choosing the right model is crucial because it has a significant impact on both accuracy and efficiency. It is important to consider the strengths and weaknesses of each model to determine which is the most adequate.

2.1.1 Linear Regression

Linear regression explains the relationship between an output variable and a linear combination of input variables. Input variables are often called independent, and the output variable is called dependent because it depends on input variables. Linear regression is simple if there is only one independent variable, whereas it is multiple if there are two or more independent variables. The form of the model is expressed as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \epsilon_i, \quad i = 1, ..., n$$

Where n is the number of observations, p is the number of independent variables, y_i is the dependent variable of the ith observation, x_{ij} is the jth independent variable of the ith observation, β_j is the coefficient of every jth independent variable and finally ϵ_i is the error of the ith observation. Errors follow an independent identically distributed normal.

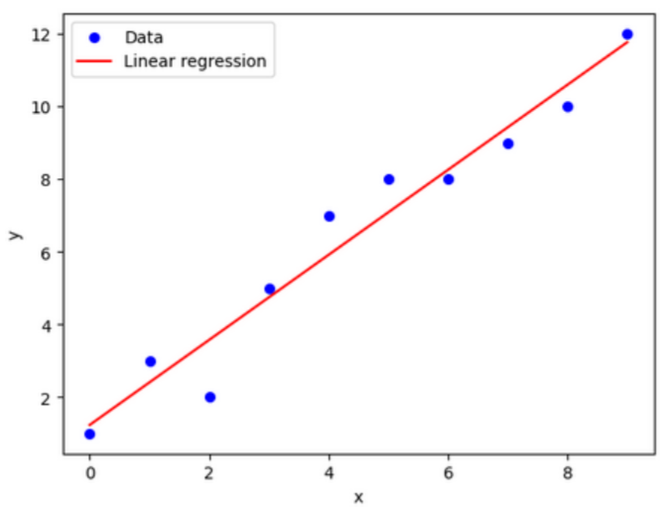


Figure 2.1: Simple linear regression with synthetic data.

The method used in this case to choose the unknown parameters β is called Ordinary least squares. Basically, it consists in minimizing the sum of the squares of the differences between the observed dependent variable, which is the real value, and the result of the linear function of the independent variables. Formally, it would be seen as:

$$min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

A linear model is a way of writing several multiple linear regressions simultaneously. Therefore, we can have more than one output variable. The form of the model is expressed as:

$$y_{ij} = \beta_{0j} + \beta_{1j}x_{i1} + \beta_{2j}x_{i2} + ... + \beta_{pj}x_{ip} + \epsilon_{ij}, \quad i = 1, ..., n, \quad j = 1, ..., m$$

The new variable that appears m is the number of output variables.

This model is used when the relationship between independent and dependent variables is linear. Otherwise, the model is not adequate and might not fit the task well. Moreover, when there is a linear dependence between inputs the model can misbehave, thus inputs should ideally be independent between them.

The main advantage of this model is the simplicity to understand it: just looking at the coefficients you can understand how much effect has each input on the output. On the other hand, the main disadvantage is the need for linearity in data, if dependence is not linear, the model ought not to be used.

2.1.2 Decision Tree

A decision tree is a non-linear model that follows a tree structure with flows. The model was first described by Breiman in 1984 [1]. Each internal node represents a group of features that aims to split up data, and the branches are the outcome of a node that directs data to the following node. Finally, a leaf represents the final result, which can be a class or a numerical value depending on the tree type. The types of decision trees are:

- *Classification tree*: The outcome is a class that represents data. It is used in classification tasks.

- *Regression tree*: The outcome is a numerical value. It is used in regression tasks.

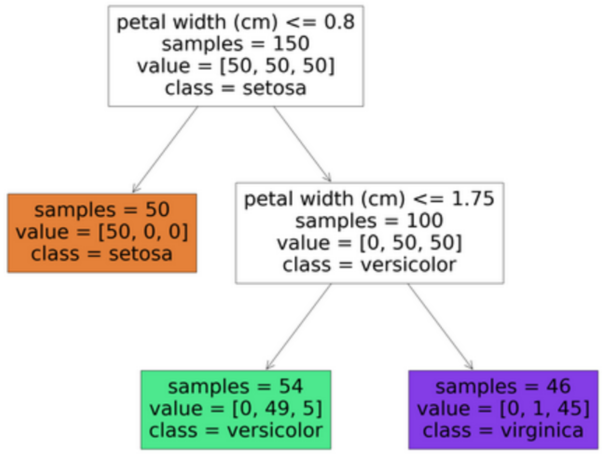


Figure 2.2: Classification tree with synthetic data.

The advantages of using a decision tree are their simplicity to understand how data is split as they can be graphically represented and interpreted by even non-experts. They also use a white box model, which means that one can clearly explain how the model works, how predictions are produced, and what is the influence of each variable. There is also an in-built feature selection, which means that only the most relevant variables are considered.

On the other hand, the model can be very non-robust: a little change in data can result in large changes in predictions. Also, the model can lead to overfitting if the tree is too complex. This complexity can be seen when the tree has too much depth. A balance has to be found between a too simple little tree and a huge over-complex tree to generalize well from training data.

2.1.3 Random Forest

A random forest is a non-linear model that is so alike to a decision tree. It was first described by Tin Kam Ho in 1995 [5], when she was in search of an improvement for decision trees. The likeliness comes from the fact that a random forest is a combination of many decision trees. The goal of this combination is to gain more precision and stability compared to the basic decision tree, solving the habit of overfitting that decision trees have. Like in decision trees, a random forest can be of two types: classification random forest and regression random forest.

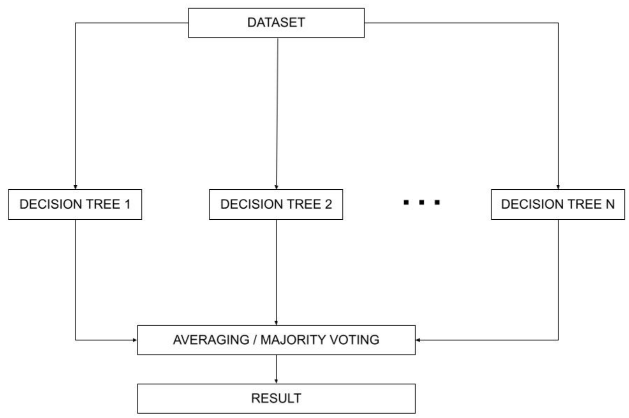


Figure 2.3: Schema of how a random forest works.

Looking at Figure 2.3 we can see how a random forest is built, how the result of each decision tree is combined and how they end in a final outcome. Random forest works on the bagging principle, that is an ensemble technique. This technique divides data and gives a random subset to each tree so each one is trained independently. This gives diversity to the model because each tree uses the variables differently, and ultimately leads to generalizing better from data.

The final result is combined, in a process known as aggregation. The output of the classification random forest is the class selected by most trees, as if there was a voting. The output of the regression random forest is the average of the output values of each decision tree.

2.1.4 Artificial Neural Network

A neural network is a model that tries to recognize relationships in a data set like a human brain does, that is why it is called an artificial network of neurons. The simplest form of the model is called *perceptron* and it consists of a single neuron. It was the first implemented artificial neural network and it was invented by Frank Rosenblatt in 1958 [9]. The model can be used for both classification and regression. The model is composed of several layers that contain nodes. Each node or neuron is the place where computation happens. This computation consists in multiplying each input by a weight linked to the input, then summing all the results and finally applying an activation function. Here's how the computation is done formally:

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

Where n is the number of inputs, w_i is the weight of the *ith* input, x_i is the *ith* input and g is the activation function. The activation function can be chosen among *Sigmoid*, *ReLu* or *tanh* mainly.

A neural network is composed of many layers, each containing many nodes. The first layer is called input layer, the last one is called output layer and the ones in the middle are called hidden layers.

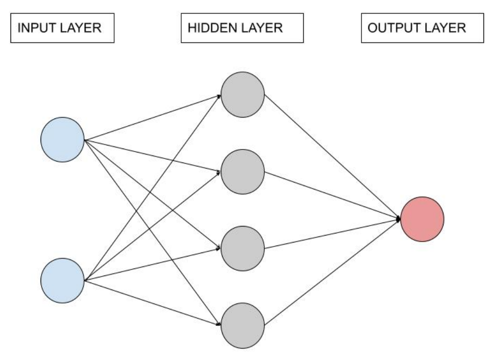


Figure 2.4: Schema of a simple neural network.

Neural networks can have different sizes, from really complex ones with many neurons and many hidden layers to simple neural networks like the one in Figure 2.4, that only has one hidden layer. The deeper the network, the more complex it is, but more complexity doesn't necessarily lead to better results, so an optimal point needs to be found.

The complexity of the model allows modelling more difficult data sets, that is why this type of learning is called deep learning. The main downside is that it is a black-box model, meaning that the learning that the model does can't be followed nor understood completely, then you can't be sure of the inputs that have the most impact for example.

2.2 Natural language processing techniques

The other group of algorithms that takes part in the thesis is natural language processing (NLP). Natural language processing is a field that tries to make machines understand human language. In this thesis, there are some challenges that require interpreting natural language, therefore some techniques of natural language processing have been tested.

2.2.1 Bag-of-Words

The Bag-of-Words (BoW) technique is used in natural language processing to simplify the representation of text. The main idea behind it is to count the occurrences of each word in a document and build a vocabulary with word frequencies. The goal is to transform text into a numerical representation. Here's how the model works:

In the beginning, a document is provided:

d1: 'He likes to play with his sister. My sister likes playing with her friend.'

The first step is to count the words. The model does a preprocessing before counting like lemmatization, that considers the context and converts each word to its meaningful base form. For example 'likes' would count as its base form 'like'. Another typical preprocessing is converting all the words to their form in lowercase. Here's how the example document would be counted after applying lemmatization and lowercasing:

BoW1 = {'he': 1, 'like': 2, 'to': 1, 'play': 2, 'with': 2, 'his': 1, 'sister': 2, 'my': 1, 'play': 1, 'her': 1, 'friend': 1};

The last step is to transform this dictionary to a numerical representation, a vector:

(1) [1, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1]

This numerical representation allows having many useful implementations like text classification. On the other hand, the model has some limitations like not providing semantic information about the words or not giving information of either the location or correlation of words.

2.2.2 Term Frequency–Inverse Document Frequency

The Term Frequency-Inverse Document Frequency (TF-IDF) is a technique that takes another step compared to what BoW accomplishes. Essentially the model gives a numerical value that expresses the importance of a word for a document in a collection. So we don't only focus on one document and its words, but also on all the documents in the collection. The technique has its origin on Karen Spärck Jones introduction of the inverse document frequency (IDF) in 1972 [10].

The numerical value is the product of two factors: the term frequency (TF), which is the times that the term or word appears in the document (value from BoW), and the inverse frequency of the document (IDF), which is a value obtained from dividing the number of total documents in the collection to the number of documents containing the word. Then you take the logarithm of the result. The IDF is a way of giving importance to the term in the document if the term only appears in a few documents in the collection, whereas if it appears in the majority of the documents, the importance of the term in the document decreases. Here is the formula to calculate the TF-IDF of a term t for a document d in a collection of documents D:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, d, D)$$

The term frequency of a term in a document is calculated by simply counting the times that the term t appears in the document d. The inverse document frequency is calculated with the following formula where, as in the previous formula, t is the term, d the document, and D the collection (there are other ways to calculate IDF). It is important to notice that the 1 is added so that the words that appear in all the documents can be considered.

$$idf(t, d, D) = log \frac{\mid D \mid}{\mid \{d : t \in d\} \mid} + 1$$

Let's make a practical example to understand how everything is calculated. Imagine we have the following collection of documents:

D: {d1: 'He likes eating carrots.', d2: 'She makes cakes with carrots.'}

Let's focus on a concrete term and a concrete document to calculate the TF-IDF of the term in the document, for example 'carrot' in $d1$. Notice that like in BoW, in TF-IDF there is also a preprocessing like lemmatization. Applying it, 'carrots' is converted to its base form 'carrot'. First of all, let's calculate its TF:

$$TF('carrot', d1) = 1$$

It's simple, 'carrot' appears once in document $d1$. The following step is to calculate the IDF:

$$IDF('carrot', d1, D) = log\frac{\mid D \mid}{\mid \{d : t \in d\} \mid} + 1 = log\frac{2}{2} + 1 = 1$$

As 'carrot' appears in all the documents, IDF equals 1. Multiplying both factors, we would conclude that the TF-IDF value of the term 'carrot' in $d1$ is 1.

2.2.3 Word2Vec

The Word2Vec model is another technique of NLP that brings information about something that both BoW and TF-IDF can't accomplish: the context. The context is detected because the algorithm uses a neural network model that learns word associations. The technique was developed by Tomas Mikolov in 2013 at Google [8]. Some implementations are the detection of synonymous words or the prediction of the following word.

What the model does is input a big document to the neural network and then, it outputs word embeddings, which are real-valued vector representations of each word. Words are encoded in a way that words that have their embeddings closer in space are expected to have similar meanings.

Similarity between words is calculated with the cosine similarity. It has the following formula:

$$SIM(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|}$$

Where A and B are two words. This vector subspace of words has arithmetic properties. When words are added or subtracted, other words can be obtained that make sense with the operation done. For example:

$$\text{King} - \text{Man} + \text{Woman} = \text{Queen}$$

As seen, this model goes another step further compared to the models previously explained, but the downside is that the numerical representation obtained is harder to understand.

Chapter 3

Models

In this chapter, there is a deep explanation of all the different models that have been developed. Moreover, there is explained the data pipeline: how data has been treated and transformed from its source to being fed to the models.

3.1 Data

Reliable data is an important factor when training a machine learning model: if data is biased and contains outliers, the quality of the model decreases. Therefore, it is so important to clean and process data before feeding it to any model if good results want to be accomplished.

Data provided by the company is heterogeneous and contains multiple metrics. More than one model has been developed to find which input metrics and machine learning techniques are the best to model the problem of this thesis.

3.1.1 Input metrics information

In this section all the input metrics that can be used in a model are defined:

- *Job name*: Identifier of a job. It is a sequence of words divided by a dot.
- *Job path*: Identifier of the location of a job. It is a sequence of words divided by a slash.

- *Requested memory*: Number of Gb that are requested for the execution of a job
- *Requested CPU*: Number of cores that are requested for the execution of a job.
- *Block*: It is the name for a given block, which is a logical unit that has a function inside the microchip.
- *Session*: It is the name of a task during the process of the design of the microchip.
- *Recipe*: Optimization target of the job. It could be area, power, timing, etc. or even a combination of them.
- *Block width*: Width of the area available for the block.
- *Block height*: Height of the area available for the block.

The following metrics are also job inputs, but they are extracted from the previous job, as the behaviors of continuous jobs are so similar:

- *Previous runtime*: The time that the previous job was executing in a computer.
- *Previous memory*: Maximum amount of memory that the previous job used when it was being executed.
- *Previous area*: Multiple metrics that report the area available for the block. Reported by the previous job.
- *Previous instances count*: Number of instances used. Reported by the previous job.
- *Previous design utilization*: Ratio (area of the block / area available). Reported by the previous job.
- *Previous leakage*: Power consumption of the block when it is not performing any computational task. Reported by the previous job.
- *Previous timing*: Multiple metrics that report timing in different parts of the design process. Reported by the previous job.

All the input metrics presented are known to have some correlation with the memory used and runtime of the job.

3.1.2 Data extraction

Data is extracted from an Elastic Search API. An Elastic Search API contains a NoSQL database that allows both extracting and filtering data with more speed than a typical relational database. To pull data, a query is needed.

Through this query, a first filtering of jobs is done. Jobs need to follow the following conditions:

- *Recent jobs*: The considered time to be recent is that jobs were not executed more than 2 months ago. This is done to avoid having outdated data.
- *Runtime at least of 5 minutes*: The runtime of jobs has to be of 5 minutes or more, so that they are worth considering. If runtime is lower, chances are high that jobs had a problem executing.
- *Exit code = 0*: Exit code indicates if a job has finished the execution with success. All codes different from 0 mean that the job has had a problem.
- *Requested CPU and requested memory greater than 0*: If no resources were requested, the job has not been executed well.
- *Memory used greater than 0*: Memory used has to be more than 0 to ensure that the job has been executed well.

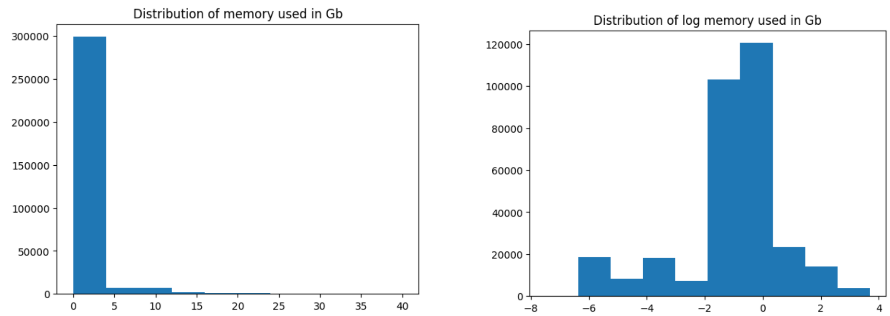
This first filtering consists in functional filtering, basically it checks that jobs have been executed correctly.

3.1.3 Data cleaning

After making the request to the API, a dictionary is received containing all the jobs that passed the first filtering. Next step is to transform the dictionary into a data frame. Afterwards, the useful columns for each model are selected.

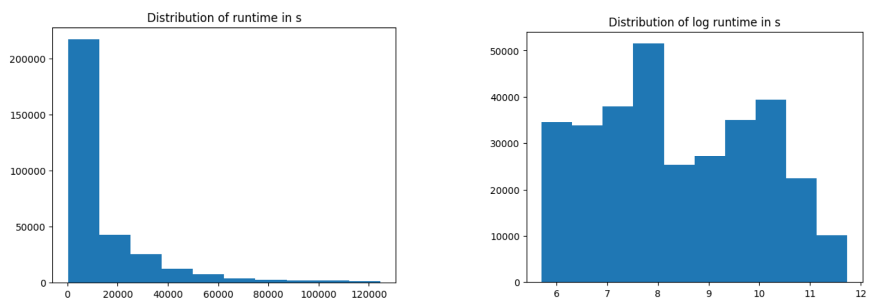
The following step is to check the data type of all the metrics: sometimes a metric has more than one data type, or sometimes what should be numerical is in fact a string. These situations are solved before proceeding.

Finally, statistical filtering takes part. Runtime and memory used of the jobs follow a negative exponential distribution. To be able to do the statistical filtering, the output metrics are applied a *log* transformation to change their distribution to a more normal one.



(a) Memory initial distribution. (b) Memory transformed distribution.

Figure 3.1: *Log* transformation of memory.



(a) Runtime initial distribution. (b) Runtime transformed distribution.

Figure 3.2: *Log* transformation of runtime.

Then, to remove statistical outliers, we filter out all the jobs that follow at least one of these conditions:

$$(1) \quad runtime > mean(runtime) + 2 \times std(runtime)$$

$$(2) \quad runtime < mean(runtime) - 2 \times std(runtime)$$

$$(3) \quad memory > mean(memory) + 2 \times std(memory)$$

$$(4) \quad memory < mean(memory) - 2 \times std(memory)$$

The first two conditions have their focus on the runtime, and the other two on memory. With this strategy, if a job has its memory or runtime outside of the 95% of the cases approximately, it is removed.

Missing values

In this data set, missing values are found in two forms: empty strings or nulls. Whenever a missing value is encountered in any metric, the job is filtered out, because a metric not reported means that the job did not execute well, so in essence, an outlier is removed. Only a tiny percentage of the data set is removed when filtering out missing values, so the filtering doesn't lead to insufficient data.

The data set used of the real company project has initially around 1,500,000 jobs. After the functional cleaning, the data set is reduced to around 330,000 jobs. Finally, after the statistical filtering, the data set is reduced to 320,000 jobs. From the initial state to the clean state, the number of jobs is reduced by an 80% approximately. Examining the numbers, the data set exhibits a low degree of cleanliness from its inception. Nevertheless, the quantity of jobs is more than sufficient to be able to train a model successfully, so the cleaning process turns out to be positive.

3.2 Methodology

In this section, there are described the environment used to develop the models, the machine learning models that have been tested, and the strategy used to train and test the different models.

3.2.1 Environment

The programming language that has been used to develop the models is *Python* [4]. The motivation behind the choice is that *Python* is a flexible programming language that supports many libraries dedicated to machine learning, crucial for this thesis. The libraries that have been mainly used are *Pandas* [2] for everything related to the treatment of data, and *Sci-kit Learn* [7] for all the machine learning techniques. Other important libraries used are *Matplotlib* [3] to make visualizations and *Keras* [11] for everything related to neural networks.

Models are trained and used to predict separately. That is done because it would be a waste of time if the model had to be trained every time a prediction wants to be made. So, the first step is to train the model and then it is saved. Whenever a prediction has to be made with the model, the model is loaded rapidly and the prediction is made. The saving and loading of the model are done thanks to the library *Pickle*.

3.2.2 Model Schema

The process followed when building a model is explained here. Each model is defined by two things:

1. The input it receives.
2. The machine learning model that is used to model data.

Machine learning models

The core of each model is defined by a machine learning model. In other words, each model built to predict a type of input is developed with different core machine learning models, which are the following:

- Linear model.
- Decision tree.
- Random forest.
- Neural network.

Depending on the input, one machine learning model might be more suitable. That's why when building a model, the different combinations of the model with each machine learning model are also built to find which is the best.

Train-Test data split

Before training a model, data is split between training and testing data: 80% of data for training data and the 20% other for testing data. Training data is used, as its name says, to train the model. Then, when the model is trained, testing data is used to test the models: get accuracy metrics and compare their predictions to the real values of the output metrics of testing data jobs.

One important thing to have in mind is that this split is the same for all the models, to ensure that the comparison between models is done fairly and with the same data. Finally, the step of splitting data into training and testing data is not done randomly as usual. The strategy followed is to use the oldest jobs to train the model and the most recent ones to test it. This is because a simulation of the real world wants to be done, where past jobs are used to train the model, and the new ones need to be predicted.

Hyperparameter selection

Another important topic to be treated is hyperparameter selection. Machine learning models like decision trees, random forests and neural networks have hyperparameters, which are important to optimize to get the best results for each model. There are different strategies to tune the hyperparameters of a model:

- *Grid search*: Also called exhaustive search. It consists in declaring a set of possible options for each hyperparameter. Then, every combination is tested to find the most optimal one. The computational cost grows exponentially, so there may be a moment when the search is not feasible.

- *Random search*: It consists in defining a search space for each hyperparameter. Then, it randomly samples points in that space. The number of iterations is set by the user. The advantage is that it doesn't need much computational cost, the user defines how much is needed through the number of iterations. However, since it sets the values of the hyperparameters at random, there is a risk of missing the most optimal set of hyperparameters.

- *Bayesian optimization*: Unlike the previous methods, this method learns from previous iterations. It creates a probabilistic model, in which it maps the different hyperparameters to their corresponding score probability. The number of iterations is also set by the user. The total number of iterations required to find the optimal set of hyperparameters is reduced because there is no need to explore the entire sample space. Nonetheless, the computational cost per operation grows because additional time is required to determine the next values to evaluate.

For decision tree and random forest, the method chosen is grid search, as the number of hyperparameters are not many and the search space known to be optimal is not big, so all the values can be explored.

On the other hand, for the model neural network, bayesian optimization has been used. The explanation is that there are many more hyperparameters to explore and the search space is not really clear, so many more values should be explored. Furthermore, a neural network takes more time to train, so testing all the possibilities would not be feasible.

Cross-validation

After choosing the method to select the hyperparameters, there is still an important topic to take into account: how to evaluate a set of hyperparameters. The first thing that comes to mind is training error, which consists in calculating the error of the entire training data set after using it to train the model. The problem though, is that the samples used to calculate the error have already been used to train the model. Then, the error doesn't tell if the model is able to generalize from data. Problems of overfitting can appear, the model could learn too much from training data.

Cross-validation is the alternative. It is a method that aims to evaluate a model but with data that the model has not been trained with. The method used has been the K-fold algorithm. This method divides training data into K parts. In this case, the choice has been $K = 10$, as it has been demonstrated that it works well with data sets of the real world [6].

The idea is to split the training data set into these K parts and do K iterations. In each iteration, $K - 1$ parts are used to train the model, and the remaining one to test it. In every iteration, the part used to test the model needs to be a different one. In the end, the error of every testing part is averaged. With this method, a fair error of the model is obtained, as data tested doesn't take part in the training process when considered.

The goodness of each model is measured with the metric MAPE (Mean Absolute Percentage Error). The metric will be covered in detail in chapter

3.3 Generic model

The first model developed is the generic model. As said previously, data is heterogeneous depending on the company project data is pulled from. The first challenge is to create a model that works with jobs from any project, so there is a clear intention: only use metrics available on all the projects. With that in mind, input metrics that are common for all the projects are the following:

- *Job name*: Identifier of a job. It is a sequence of words divided by a dot.
- *Job path*: Identifier of the location of a job. It is a sequence of words divided by a slash.
- *Requested memory*: Number of Gb that are requested for the execution of a job.
- *Requested CPU*: Number of cores that are requested for the execution of a job.

All data that is to be fed to a model needs to be numerical. Out of the 4 variables of this model, both requested memory and requested CPU are numerical, so they don't need to be processed. However, job name and job path are a sequence of strings, so they need to be somehow converted.

3.3.1 Vectorization

Vectorization is the process of transforming into a vector. In this case, as what needs to be converted is a sequence of strings, the logical conversion is a sequence of numbers, or which is the same, a vector of numbers.

The first step is to split each word into a vector of words. In the case of job name, words have to be split when a dot is found, whereas in the job path, it has to be done when a slash is encountered. Here is an example:

Job name: w1.w2.w3.w4.w5 $\rightarrow$ [w1, w2, w3, w4, w5]
Job path: w1/w2/w3/w4/w5 $\rightarrow$ [w1, w2, w3, w4, w5]

The next step is to convert this vector of strings to a numerical vector. To do so, there are many options that have been considered and tested. All the options have been described deeply in chapter 2: *State of the art*.

One-hot encoding

The idea is that every different single word that appears in any vector of any job counts as a new metric. Then, if a job has that word in its vector, the value of the metric for that job would be 1, otherwise it would be 0.

The encoding is ideal in terms of information keeping, as all the information is kept. Nonetheless, it is also the one that costs more computational memory. Each different word found would make the data set bigger. If the vocabulary size is on the scale of thousands of different words, which is the case in this thesis, it is not feasible.

Count vectorizer

This vectorizer works similarly to the one-hot encoding. Basically, it counts the times each word appears in the string. Terms don't get repeated in any vector, so it would work exactly as one-hot encoding. However, there is an extra functionality.

This vectorizer can limit the words to be considered so that they can fit in memory. The settings applied are the following:

- Only 2,000 words are considered.
- The words to be considered are the ones with more frequency among all the jobs, without considering the words that appear in more than the 95% of the jobs, because they give less information.

TF-IDF vectorizer

With this vectorizer, we apply an extra variable: document frequency. Each word has the TF-IDF value of that word in the vector of the job.

With this new feature, some words that have great importance with the count vectorizer can have their importance lowered, as they might appear in most of the jobs.

Similarly with the count vectorizer, the vocabulary can be limited. The settings applied are the same applied with the counter vectorizer:

- Only 2,000 words are considered.
- The words to be considered are the ones with more frequency among all the jobs, without considering the words that appear in more than the 95% of the jobs.

Word2Vec vectorizer

Finally, with this vectorizer, there is a big change compared to the previous ones: not only it gives focus to word frequencies but also to their context.

The aim is to convert each word of the job vector to its space embedding. Then, to have a fixed size, all the embeddings in the vector are averaged as it is a simple way to keep the information of all the words in the vector.

The disadvantage of this technique is that word embeddings are calculated in a learning phase with a sample of words. If a new word is encountered afterwards, the vectorizer does not know how to convert it.

3.3.2 Model flow

In this section, the flows of a model can be found, to understand both the process of training a model and the process of obtaining a prediction from a model. The generic model using TF-IDF vectorizer is used as an example, but it can be generalized for all the models.

Training flow

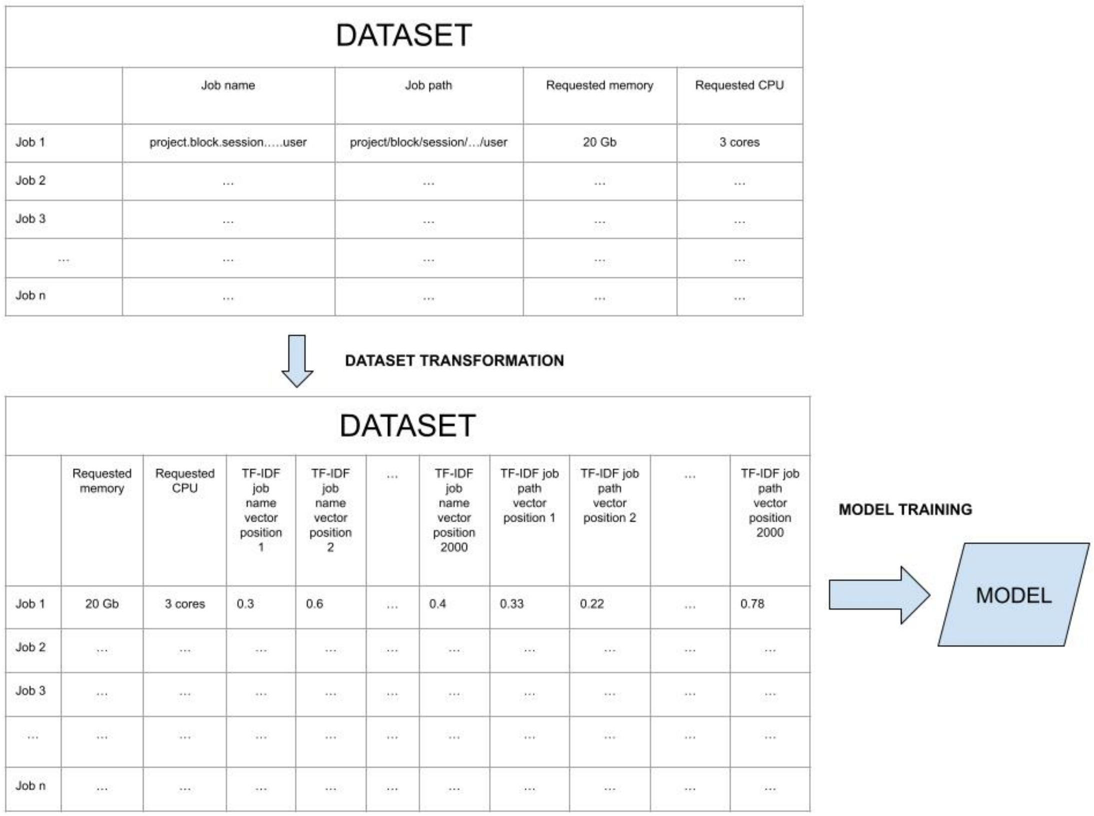


Figure 3.3: Training flow of the generic model using TF-IDF vectorizer.

Prediction flow

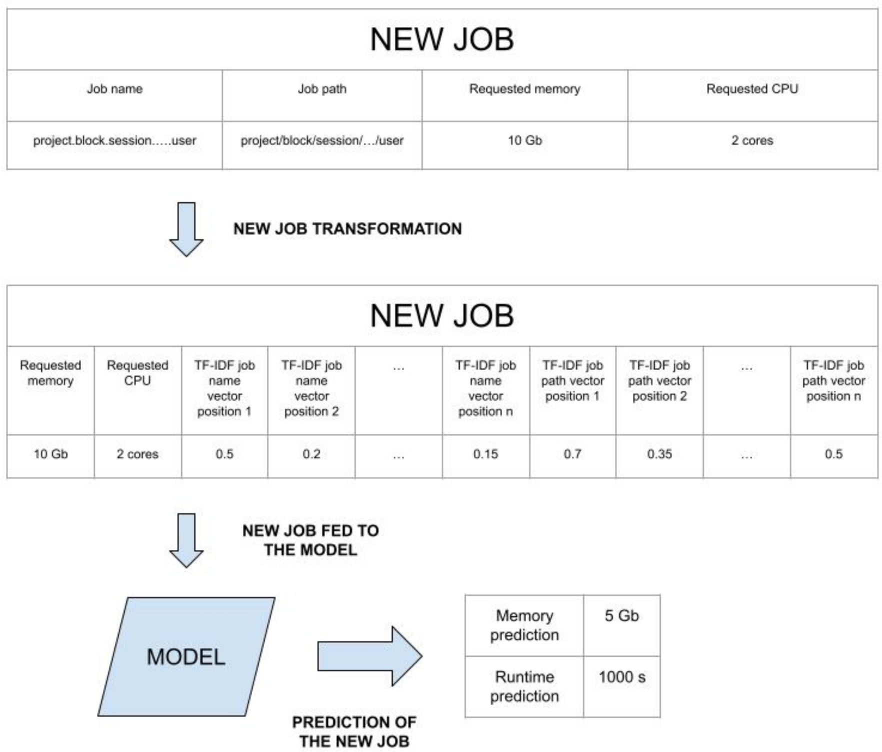


Figure 3.4: Prediction flow of the generic model using TF-IDF vectorizer.

3.4 Baseline metrics model

The second model developed is the baseline form of the metrics model. The metrics model has been exclusively developed for an experimental company project which has all the metrics available, the ones explained in *Section 1.1.2*.

In its baseline form, the aim is to have a similar model to the generic one. The difference is that the most valuable information is already extracted from the job name and the job path: the block and the session of the job. Then, the input of the model is the following:

- *Block*: It is the name for a given block, which is a logical unit that has a function inside the microchip. Each design is composed of blocks, which have a hierarchy: a block can contain other blocks.
- *Session*: It is the name of a task during the process of the design.
- *Requested memory*: Number of Gb that are requested for the execution of a job.
- *Requested CPU*: Number of cores that are requested for the execution of a job.

Again, we have the same problem found in the generic model: both requested memory and requested CPU are numerical but block and session are not. This time though, instead of being a sequence of strings, they are a single string.

The fact of having a single string instead of a sequence of strings reduces the vocabulary size drastically. Therefore, one-hot encoding is possible, as now the vocabulary can fit in computational memory perfectly.

3.5 Advanced metrics model

The third and last model developed is an advanced form of the metrics model. In the baseline model, only four metrics are selected, whereas in this model all the metrics available are fed into the model. The complexity of the model grows, but also accuracy is expected to grow as there is more information. The input of the model is the following:

- *Block*: It is the name for a given block, which is a logical unit that has a function inside the microchip. Each design is composed of blocks, which have a hierarchy: a block can contain other blocks.
- *Session*: It is the name of a task during the process of the design.
- *Requested memory*: Number of Gb that are requested for the execution of a job.
- *Requested CPU*: Number of cores that are requested for the execution of a job.
- *Recipe*: Optimization target of the job. It could be area, power, timing, etc. or even a combination of them.

- *Block width*: Width of the area available for the block.
- *Block height*: Height of the area available for the block.
- *Previous runtime*: The time that the previous job was executing in a computer.
- *Previous memory*: Maximum amount of memory that the previous job used when it was being executed.
- *Previous area*: Multiple metrics that report the area available for the block. Reported by the previous job.
- *Previous instances count*: Number of instances used. Reported by the previous job.
- *Previous design utilization*: Ratio (area of the block / area available). Reported by the previous job.
- *Previous leakage*: Power consumption of the block when it is not performing any computational task. Reported by the previous job.
- *Previous timing*: Multiple metrics that report timing in different parts of the design process. Reported by the previous job.

The majority of the metrics are numerical, so in most of them, no transformation is needed. Nonetheless, there are three that are not: block, session and recipe.

The form of this non-numerical metrics is a single string so, like in the baseline form of the model, one-hot encoding can be applied to each metric because again, the vocabulary is little and can fit into computational memory.

Finally, two metrics are combined: block width and block height. What is interesting about the model is the area of the block. To have this new metric, the following transformation has to be applied:

$$\text{block area} = \text{block width} \times \text{block height}$$

Chapter 4

Results

In this chapter, the models explained in chapter 3: *Models* are tested with the aim to compare them and find the one that outstands from the rest.

To evaluate the models, the metric MAPE (Mean Absolute Percentage Error) is used. This metric gives an intuition on how much error there is in the prediction compared to the true value. The mathematical expression of the metric is the following:

$$\text{MAPE}(y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N-1} \frac{|\hat{y}_i - y_i|}{|y_i|}$$

Where N is the number of predictions done, y_i is the true value for the *ith* example of metric y, and $\hat{y}_i$ is the predicted value for the *ith* example of metric y. Metric y can either be the runtime or the memory.

The choice of MAPE over other evaluation metrics like MAE (Mean Absolute Error) or MSE (Mean Squared Error) is explained by the fact that there is more interest in the error in comparison with the true value than in the error in Gb. This is done because jobs that use a low amount of memory tend to have a low error in Gb, but in reality, they might be having a large error in proportion to what they are using. With the MAPE, all the jobs, no matter their memory usage, are evaluated equally in terms of their error. Furthermore, the metric wants to be used as a margin to add to the prediction, and a percentage is much more useful than an absolute value.

To begin with, let's look at the no model situation to understand where is the initial point. The MAPE value of memory for the no model situation is 4,553.9% in the company project data set seen in section 3.1: *Data*. It is an exorbitant value, totally unacceptable.

In the following sections, all the possible combinations of the models created are tested, using the machine learning models and techniques explained in chapter 2: *State of the art*.

4.1 Generic model

First of all, let's look at the generic model. This model is built with two variants: the machine learning model and the type of vectorizer. Below there are some tables summarizing the results.

The no model situation does not have runtime error because it is not an input, the runtime is not requested. The prediction of the metric has the only goal of being informative.

Count vectorizer

Model	**Memory MAPE (%)**	**Runtime MAPE (%)**
No model	4,553.90	-
Generic model with linear regression	5.63×10^{11}	5.52×10^{10}
Generic model with decision tree	95.40	178.30
Generic model with random forest	90.50	188.50
Generic model with neural network	1,071.70	644.40

Table 4.1: Results of the generic model variations using the count vectorizer.

TF-IDF vectorizer

Model	**Memory MAPE (%)**	**Runtime MAPE (%)**
No model	4,553.90	-
Generic model with linear regression	926.20	862.00
Generic model with decision tree	87.80	221.40
Generic model with random forest	88.70	214.00
Generic model with neural network	1,110.40	780.20

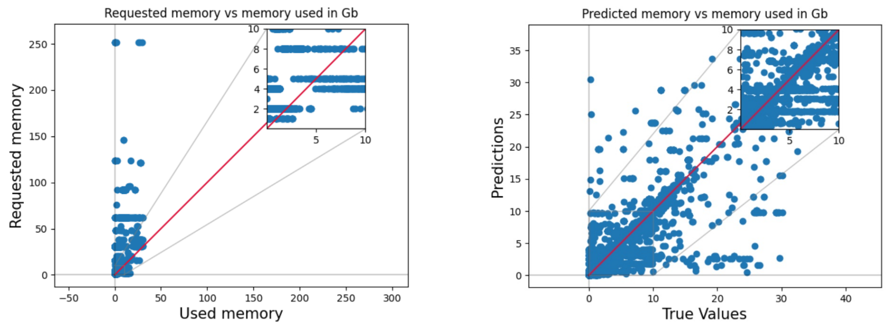
Table 4.2: Results of the generic model variations using the TF-IDF vectorizer.

Word2Vec vectorizer

Model	**Memory MAPE (%)**	**Runtime MAPE (%)**
No model	4,553.90	-
Generic model with linear regression	717.50	890.40
Generic model with decision tree	263.80	492.10
Generic model with random forest	203.10	511.20
Generic model with neural network	417.30	360.70

Table 4.3: Results of the generic model variations using the Word2Vec vectorizer.

Focusing on the memory, it can clearly be seen that any model highly improves the no model situation, with the exception of the linear regression using the count vectorizer, which has an abnormally huge error due to the non-linearity of data. The best model is the decision tree using the TF-IDF vectorizer with 87.8% MAPE.

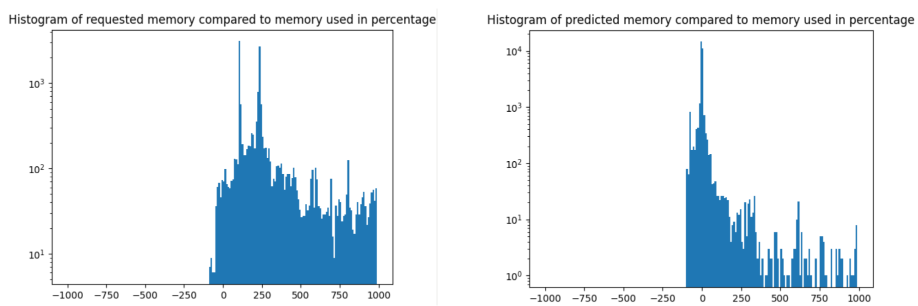


(a) No model.

(b) Best generic model.

Figure 4.1: Comparison of scatter plots without and with the best generic model.

Looking at the scatter plots, the situation seems to improve. In the no model situation, there is no pattern trying to follow the ideal red line ($requested\,memory = used\,memory$), whereas using the best generic model, data follows the red line pattern and in many cases, jobs are on this line.



(a) No model.

(b) Best generic model.

Figure 4.2: Comparison of the histograms of percentage errors without and with the best generic model. The y axis is in log scale.

Looking at the histograms, the error is mitigated a lot. The values represent a percentage error. This percentage is a comparison between the requested or predicted value and the true value. The formula for the error is the following:

$$\text{Percentage error}(y, \hat{y}) = \frac{\hat{y} - y}{y}$$

Where $\hat{y}$ is the requested or predicted value, and y is the true value. In the no model situation, there is not a spike at $error = 0$ and large positive errors count many cases. On the other hand, when using the best generic model, there is a spike at $error = 0$ and huge errors count fewer cases.

Focusing now on the runtime, the best model turns out to be the generic model with a decision tree using the count vectorizer. The MAPE is much higher than in memory due to the fact that the distribution of the runtime values is much bigger than the memory distribution, and it has different external factors difficult to control. Therefore, runtime is more difficult to predict with exactitude.

4.2 Baseline metrics model

The baseline metrics model only has one variant: the machine learning model. Below there is a table with the results of this model, as well as the best variant of the generic model.

It is important to acknowledge that the data set used to do this testing is a different one from the data set used until now. It is a smaller one, with around 17,000 jobs. This is because the metrics for the metrics models are only available for this data set. This data set is not from a real company project, it is a more experimental one. The preprocessing done to this data set is the same applied to the previous data set.

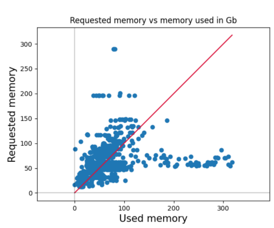
Although the data set has changed, the same data has been applied to all the models to have a fair comparison.

Let's first focus on memory. This data set has a much lower MAPE when no model is applied, just a 54.2%. Nevertheless, it can be improved with the models. All the models improve the no model situation but the linear regression because of non-linearity in data, and the neural networks model, due to the little size of the data set. Neural networks tend to be powerful with huge data sets, but with small ones, they are hardly ever the best choice.

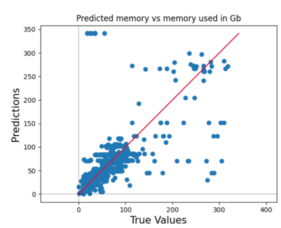
The best model is the one using a random forest with 33.3% MAPE. It both improves the no model situation and the best generic model found: the generic model using a decision tree and a TF-IDF vectorizer.

Model	**Memory MAPE (%)**	**Runtime MAPE (%)**
No model	54.20	-
Generic model with decision tree and TF-IDF vectorizer	38.30	103.80
Baseline metrics model with linear regression	2.73×10^8	3.87×10^9
Baseline metrics model with decision tree	38.00	57.60
Baseline metrics model with random forest	33.30	46.80
Baseline metrics model with neural network	359.30	100.00

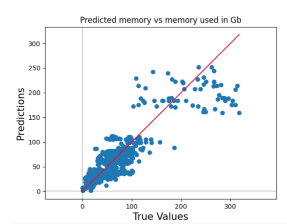
Table 4.4: Results of the baseline metrics model variations.



(a) No model.



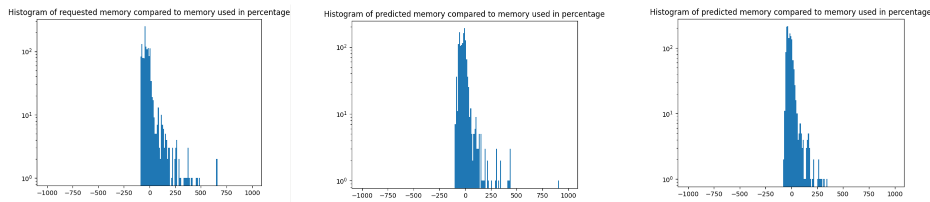
(b) Best generic model.



(c) Best baseline model.

Figure 4.3: Comparison of scatter plots without any model, with the best generic model, and with the best baseline metrics model.

Looking at the scatter plots, the situation improves with each model. In the no model situation, there is again not seen any pattern trying to follow the ideal red line. The best generic model has a better plot, but the best one is accomplished by the best baseline metrics model, the one using a random forest. It is able to predict better the jobs that use more memory.



(a) No model. (b) Best generic model. (c) Best baseline model.

Figure 4.4: Comparison of histograms of percentage errors without any model, with the best generic model, and with the best baseline metrics model. The y axis is in log scale.

Looking at the histograms, the error decreases as a better model is used. With each model improvement, the histogram becomes slimmer and the higher error counts are reduced.

Focusing now on the runtime, the best model is again the baseline metrics model with a random forest with 46.8% MAPE.

4.3 Advanced metrics model

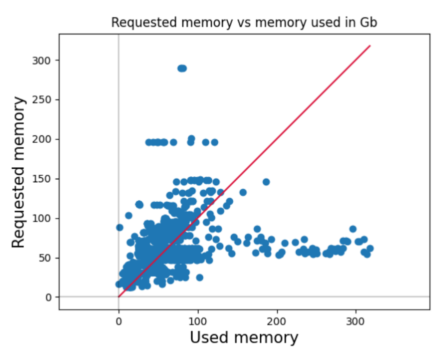
The last model to check is the advanced metrics model. Similarly to the baseline metrics model, it only has one variant: the machine learning model. The difference is that the advanced version uses many more metrics. Below, there is a table with the results of this model, as well as the best variant of the generic model and the best variant of the baseline metrics model.

It is important to remember that the experimental data set used to compare the baseline version is also used here to compare the different models.

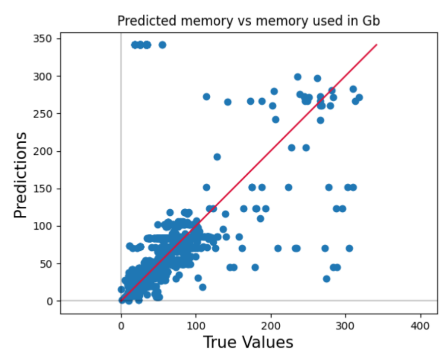
First of all, let's focus on memory. Although the improvement with the baseline model is noticeable, with the advanced model it is even better. The best MAPE is accomplished by the advanced metrics model with random forest, with 21.8%. Again, it is seen that the model using linear regression can't fit this data well.

Model	**Memory MAPE (%)**	**Runtime MAPE (%)**
No model	54.20	-
Generic model with decision tree and TF-IDF vectorizer	38.30	103.80
Baseline metrics model with random forest	33.30	46.80
Advanced metrics model with linear regression	1,732.20	54,836.60
Advanced metrics model with decision tree	26.30	57.10
Advanced metrics model with random forest	21.80	58.90
Advanced metrics model with neural network	99.70	100.00

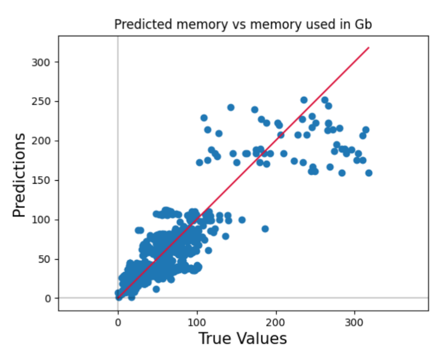
Table 4.5: Results of the advanced metrics model variations.



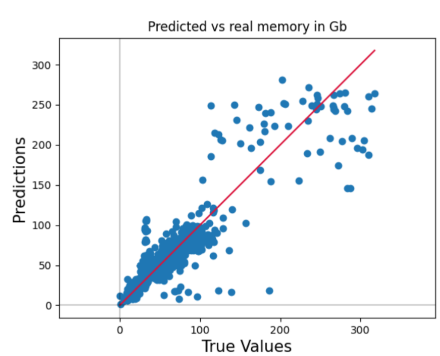
(a) No model.



(b) Best generic model.



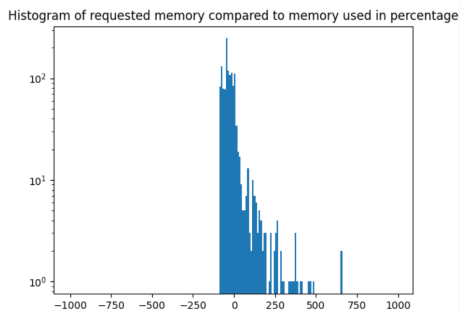
(c) Best baseline model.



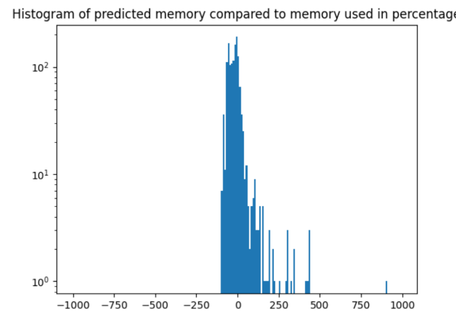
(d) Best advanced model.

Figure 4.5: Comparison of scatter plots without any model, with the best generic model, with the best baseline metrics model, and with the best advanced metrics model.

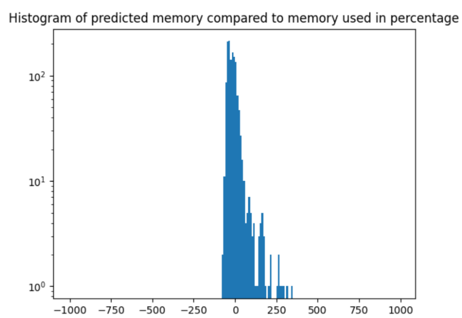
Looking at the scatter plots, the situation again improves with each model. In the no model situation, there is again not seen any pattern trying to follow the ideal red line. The best generic model has a better plot, but the best baseline model and the best advanced model have the best plots. Between these two, a conclusion of which is the better model can't be made.



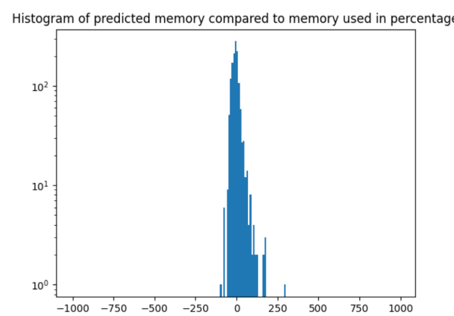
(a) No model.



(b) Best generic model.



(c) Best baseline model.



(d) Best advanced model.

Figure 4.6: Comparison of histograms of percentage errors without any model, with the best generic model, with the best baseline metrics metrics model, and with the best advanced metrics models. The y axis is in log scale.

Looking at the histograms, the error decreases as a better model is used. With each model improvement, the histogram becomes slimmer and the higher error counts are reduced. With the best advanced model, the slimmest histogram is accomplished, which translates to a better result. With the histogram, the advanced model can now be seen as a better model than the baseline model.

Focusing now on the runtime, the situation changes. The advanced models are not able to improve the results accomplished by the best baseline model. Somehow, the new metrics make the model decrease its ability to predict the runtime well.

Chapter 5

Conclusions

It has been demonstrated that the use of machine learning in this thesis has a huge improvement in the company regarding resource usage. We have gone from a chaotic situation where users didn't pay attention to what resources they requested, to how the situation works right now: a model that predicts with accuracy the resources that each job needs.

The best model developed in terms of memory prediction is the advanced metrics model with a random forest. It accomplishes 21.8% MAPE with an experimental data set, which almost divides by three the error that the company has without any model. In a more realistic environment, with a real company project, the reduction would be much higher, as error tends to be higher in real projects than in experimental ones.

In terms of runtime prediction, the accomplishments are also noticeable. The best model is the baseline metrics model with a random forest, that has 46.8% MAPE. Taking into account the big distribution of runtime values, and the external factors that make the values less predictable, this error confirms the theory that runtime is more difficult to predict than memory.

The idea is that these memory predictions will serve as the new memory requested values for the jobs that are about to be executed. That's how the difference between memory requested and memory used will decrease. The actual solution is to simply recommend this predicted value to users, and they can make the choice of whether to use it or not. In the future, the goal is to have a model reliable enough to have everything automated.

This type of idea should be implemented in companies. Not only it is a plus in their economies: money and time are saved; but also contributes to moving to a more ecological world.

5.1 Future Work

There are many ways to keep improving this thesis. Here are some ideas of what's coming next:

- Find more input metrics that can have an impact on the models. The more useful input metrics the models have, the more accuracy they can accomplish.

- Create a model that can predict other output metrics. There are other output metrics that if predicted, they can give a lot of value to the company. The prediction of some timing and power metrics can give more insights about the design before executing any job.

One thing that is left to see is the impact of continuous predictions in the long term. When the model learns from jobs for which memory requested had been predicted previously, the model will be learning from itself, from its previous predictions. So, will it have a good or a bad impact on future predictions? Will this bias affect the model? This is an interesting topic to study in the future.

As machine learning world keeps expanding and growing, surely new algorithms and models will appear that can have an application in this thesis. There will always be some aspects to be improved.

Bibliography

[1] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Cart. *Classification and regression trees*, 1984.

[2] Pandas community. Pandas user guide. https://pandas.pydata.org/docs/user_guide/index.html, May 2023.

[3] Matplotlib development team. Matplotlib user guide. https://matplotlib.org/stable/users/index.html, March 2023.

[4] Python Software Foundation. Python documentation. https://docs.python.org/3/, June 2023.

[5] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

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

[7] Scikit learn developers. Scikit-learn user guide. https://scikit-learn.org/stable/user_guide.html, March 2023.

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

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

[10] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[11] Keras team. Keras api reference. `https://keras.io/api/`, March 2023.