FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Delivery Forecasting in Project Management

Henrique Miguel Bastos Gonçalves

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Pedro Mendes Moreira

July 27, 2021

Delivery Forecasting in Project Management

Henrique Miguel Bastos Gonçalves

Mestrado Integrado em Engenharia Informática e Computação

July 27, 2021

Abstract

Software estimation is one of the most challenging areas of project management and, in recent decades, has been a recurring topic of investigation among the scientific community.

Accurate estimates of effort and task duration play an important role in the successful management of a project. There are two traditional methods for predicting the effort and task duration of a project, namely, empirical and theory-based methods. Even so, the use of these traditional methods so far still makes it difficult to accurately estimate the effort and duration of tasks, and, since these methods require experts input, they are both susceptible to human bias, which compromises the success of a project.

In view of this difficulty, the present dissertation focuses on the hypothesis that Machine Learning methods can be applied in software estimation and bring benefits for the good forecasting of tasks in project management. Machine Learning methods have emerged with the aim of solving the flaws of traditional methods and increasing the project's success rate by providing good accuracy and greater efficiency in estimates. However, the biggest obstacle for this type of automated approach is the difficulty of selecting and applying appropriate Machine Learning methods and also performing correct sanitation of the data.

Also, according to *state of art*, not many ML implementations have been applied in an industry environment which shows a still existent lack of proof that this can be achieved in such conditions. This has been mainly due to a broad diversification of results and practices, which made it difficult for organizations to follow a way to successfully implement these ML algorithms. Also, problems such as bad pre-processing and overfitting contributed for not so good results as well. With that being said, a solid and stable heterogeneous ensemble approach with a wide variety of regression models and that is robust to the aforementioned problems was explored.

The main objectives of this dissertation are to apply this ensemble approach to task duration and effort forecast successfully, in PROMESSA platform on the Project Control management tool provided by Fraunhofer AICOS, compare it with the *state of art* and conclude if such approach is possible to implement in such industrial environment. The methodology and materials available are further discussed, and every good practice extracted from researches is carefully used to attempt the best possible results.

In the end, this heterogeneous ensemble is built with a stack of 7 regression models such as boosting ensembles, random forest and variances, neural nets, SVM, and KNN. The results show that this ensemble was successfully applied to the PROMESSA platform with promising and competitive results being compared with the *state of art*. The hypothesis is proved to be possible, and future work is discussed to improve the current results.

Keywords: Machine Learning, Supervised learning by regression, Predictive Modeling, Heterogeneous Ensemble

Resumo

A estimativa de software é uma das áreas mais desafiantes da gestão de projetos e, nas últimas décadas, tem sido um tópico recorrente de investigação entre a comunidade científica.

Estimativas precisas do esforço e da duração de tarefas desempenham um papel importante na gestão bem-sucedida de um projeto. Existem dois métodos tradicionais para prever o esforço e a duração de tarefas de um projeto, nomeadamente, os métodos empíricos e os baseados em teoria. Ainda assim, a utilização destes métodos tradicionais até agora ainda acarreta dificuldade em estimar com precisão o esforço e a duração de tarefas, aliado com o facto de que dependem de input de um especialista, ficando suscetível a erros humanos, o que compromete o sucesso do projeto.

Atendendo a esta dificuldade, a presente tese concentra-se na hipótese de os métodos de Machine Learning, serem aplicados na estimativa de software e trazerem benefícios para a boa previsão de tarefas na gestão de projetos. Os métodos de Machine Learning têm surgido com o objetivo de solucionar as falhas dos métodos tradicionais e aumentar a taxa de sucesso do projeto ao fornecer boa precisão e maior eficácia nas estimativas. Porém, o maior obstáculo para este tipo de abordagem automatizada é a dificuldade de selecionar e aplicar o método adequado de Machine Learning e também de realizar um correto saneamento dos dados, uma vez que as pesquisas científicas até este momento ainda não alcançaram resultados conclusivos.

Além disso, de acordo com o *estado da arte*, poucas implementações de ML foram aplicadas em ambiente industrial, o que mostra um buraco ainda existente e a falta de provas de que isso pode ser alcançado em tais condições. Isto deve-se principalmente a uma ampla diversificação de resultados e práticas que dificultam as organizações de seguir uma maneira de implementar com sucesso estes algoritmos de ML. Além disso, problemas como mau pré-processamento e overfitting também contribuíram para resultados menos positivos. Assim sendo, uma abordagem forte e estável através de ensembles heterogéneos com uma ampla variedade de modelos de regressão, que seja robusta para os problemas acima mencionados, foi explorada.

Os principais objetivos desta dissertação passam por aplicar com sucesso esta abordagem de ensemble para a previsão da duração e esforço de tarefas na plataforma PROMESSA, mais especificamente na ferramenta de gestão *Project Control* fornecida pela Fraunhofer AICOS, e compará-la com o *estado da arte* para concluir se tal abordagem é possível implementar em ambiente industrial. A metodologia e os materiais disponíveis são discutidos mais detalhadamente e todas as boas práticas extraídas das pesquisas feitas são cuidadosamente utilizadas para tentar alcançar os melhores resultados possíveis.

No final, este ensemble heterogéneo é construído com uma pilha de 7 modelos de regressão, como boosting ensembles, random forest e variações, neural nets, SVM e KNN. Os resultados mostram que este ensemble foi aplicado com sucesso à plataforma PROMESSA e com resultados promissores e competitivos quando comparados com o *estado da arte*. A hypothesis é comprovada

e trabalhos futuros são discutidos para melhorar os resultados atuais.

Keywords: Machine Learning, Supervised learning by regression, Predictive Modeling, Heterogeneous Ensemble

Acknowledgements

This work was developed within the scope of the project “PROMESSA - PROject ManagEment intellingent ASSistAnt”, with the reference NORTE-01-0247-FEDER-039887, co-financed by the European Regional Development Fund (ERDF), through the North Regional Operational Program (NORTH 2020).

I thank everyone who supported me through out this process like my coordinators Prof. João Moreira and Prof. Filipe Correia from FEUP and everyone who contacted with me from Fraunhofer, with a special mention to Ricardo Graça, who were always there to help me out. I also thank my family who were always there to support my decisions and to help me out in though times.

Henrique Miguel Bastos Gonçalves

"In the middle of every difficulty
lies opportunity."

Albert Einstein

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
COCOMO	COnstructive COst MOdel
DT	Decision Tree
ExtraTrees	Extremely randomized Trees
GBM	Grandient Boosting Machine
GLM	Generalized Linear Model
KNN	K-Nearest Neighbor
LASSO	Least Absolute Shrinkage and Selection Operator
ML	Machine Learning
MLP	MultiLayer Perceptron (same as MLP-ANN)
MLP-ANN	MultiLayer Perceptron Artificial Neural Network
PM	Project Manager
PROMESSA	PROject Management Intelligent aSSistAnt
REST	REpresentational State Transfer
RF	Random Forest
SVM	Support Vector Machine
XGBoost	eXtremme Gradient Boosting
WWW	*World Wide Web*

Chapter 1

Introduction

Software estimation is one of the most challenging areas of project management, being a recurrent topic of scientific research in the last decades. The difficulty in software estimation resides mainly in the initial planning phase when uncertainty regarding functionalities of the final product is substantial, representing an important moment in any project life cycle and where many tend to fail, putting at stake the quality of the whole process and eventually leading to failure [8]. It is known that accurate estimations of development effort or duration play an important role in the successful management of the project. Even though this process is so important, experts can not usually estimate accurately the effort, time, and cost of a project to be developed, due to the uncertainty that underlies their activity [39].

Inaccurate estimates can occur when there is limited knowledge about influencing factors and risks, pressure from the client, and management. Consequently, inaccurate estimates can compromise the delivery time, budget, and quality of the final product, which can ultimately result in project termination if customer satisfaction is not met. Pospieszny [39] states that even with the emergence and wide propagation of agile approaches in software, which slightly decreased project failure, the rate at which projects fail is still substantial. The major reason for this is the deficiency in the estimation and planning of the project [52]. Therefore, accurate estimations of effort and duration of tasks significantly increase the success rate of a project.

Regardless of how good the estimate can be made, the methods used so far rely only on human estimation due to their simplicity and ease of implementation. This, however, is often time-consuming, representing a problem since precious time wasted in estimations could be used to actually implement features in the project [39]. Machine learning methods have been arising to solve the failures traditional methods make and increase the success rate.

1.1 Motivation

Fraunhofer is a partner in the PROMESSA project (PROject Management Intelligent aSSistAnt), a joint effort from Fraunhofer AICOS, StrongStep, FEUP, and INESCTEC, which aims to investigate and develop a system capable of assisting other project management tools, like Project Control from Fraunhofer AICOS, in project management, risk management, and project staffing, using mainly Machine Learning and data analysis. A module with a REST API is intended to be developed for PROMESSA to assist other project management tools like Project Control (Fraunhofer) and SCRAIM (StrongStep). This module will hold all the AI developed for the aforementioned purposes, like the models that will be developed in this dissertation, to efficiently forecast variables like effort and task duration.

The effort or task duration estimations done by project managers are usually time-consuming and subject to human bias. This can be turned around by applying other automated methodologies with predictive models in machine learning. This type of data-driven methods seem to be the future of software engineering, representing a very optimistic alternative to human methods with the capacity of providing good accuracy and increased efficacy of effort and duration estimations, decreasing the boilerplate needed by the experts when planning new projects/tasks.

The greatest obstacle to this type of automated approach is the difficulty of selecting and apply the appropriate methods since research has not yet reached conclusive results. Therefore, this dissertation aims to do a benchmarking of the various machine learning alternatives and select the most effective ones for the problem at hand.

1.2 Objectives

The objectives of this dissertation are lined up with the PROMESSA project, such as:

- Analyze a variety of machine learning methods for delivery forecasting.
- Evaluate which methods fit the most with the Project Control data.
- Validate the models using data from different Project Control projects and prove their efficacy and accuracy.
- Extend the PROMESSA platform to integrate the implementation of these models and test them to prove their contribution.

1.3 Contributions

This dissertation aims at contributing in the following way:

- Explore and apply machine learning methods for software project forecasting in an industry environment where many few implementations have been found and done

- Apply successfully, when compared with state of the art, a more robust heterogeneous ensemble of ML algorithms, following other researchers work and proving this implementation can be done in an industry environment
- Extend PROMESSA module with both effort and duration machine learning methods developed which will allow every project manager to save time, manage resources better and decrease forecasting error

1.4 Hypothesis and Research Questions

This dissertation will focus on the following hypothesis:

"Machine learning methods for software project forecasting can accurately be applied, bringing benefits in terms of reduced forecasting errors and resources management"

This hypothesis is decomposed in the following research questions:

- **RQ1:** What predictors can be used for accurate forecasts and why?
- **RQ2:** Which methods can be used for accurate forecasts, and how are the results comparable with state of the art?

1.5 Structure

This document is composed by 7 chapters:

- **Chapter 1**, serves as contextualization and description of the problem with the proposed objectives and research questions to be developed in the dissertation.
- **Chapter 2**, where some background analysis is made of notions to use among the document.
- **Chapter 3**, an analysis is made on the *state of art* of software development forecasting, providing a complete analysis on the most relevant methods found, structuring a more in-depth research and review about these methods, what benefits and limitation they present and a brief comparison between some of them.
- **Chapter 4**, the problem definition, proposed solutions and the evaluation metrics to validate the solution.
- **Chapter 5**, the methodology is defined, discussing the materials used and how the results were achieved.
- **Chapter 6**, shows the proposed solution and discussion of the final results.
- **Chapter 7**, the conclusions of this dissertation are drawn as well as future work.

Chapter 2

Background

In this chapter, it will be presented an overview of concepts that work as a basis for the rest of the dissertation. Important notions will be explored for better understanding the predictive model building process and how to achieve the best effort, duration, and costs forecasting/estimation. Important conclusions will be taken for the next steps to take on this dissertation and look into important metrics when processing input/output data, validating and comparing models.

2.1 Mythical Man-Month

Typically in software development management, when Project Managers(PMs) have to forecast effort, they do it through the Mythical Man-month phenomenon, elicited by Brooks [10]. So whenever an effort is to be calculated, it is represented by man-month, man-day, or even man-hour. The book talks mainly about the fallacy that usually happens in the management of projects, that after bad predictions are made, and deadlines start to show signs of being delayed usual response is to add more manpower to fulfill the deadline. This is often rather a bad practice, as new team members need to be integrated into the team and the amount of time needed to train newcomers just causes an even bigger delay in project deadlines.

2.2 Prediction Versus Interpretation

In any predictive model, there are two key factors for comparing and choosing between certain models in detriment to others. Most times, developers mainly target models to achieve high levels of accuracy, and it does not matter whether it is a *black box* or a simple, interpretable model. While primary interest is usually to generate accurate predictions, a secondary interest may be to interpret the model and understand how it works. The unfortunate reality is that as pushing towards higher accuracy, models become more complex, and their interpretability becomes more difficult. This is almost always the trade-off done when prediction accuracy is the primary goal [26].

2.3 Model Building Process

Building predictive models to estimate variables might seem like a straightforward process, based on picking the most appropriate modeling technique, train and validate it, and generate a final prediction. This simple approach will most likely not generate a reliable, trustworthy model for predicting new samples. Common reasons why predictive models fail are: inadequate pre-processing of data, inadequate model validation, unjustified extrapolation (like applying a model to data that resides in a space which the model has never seen), or, most importantly, over-fitting the model to existing data [26, p. 3–4]. For this process to actually work, first, data, the objective of modeling, and problem context must be well understood, and only then is it possible to finally proceed to building, evaluating, and selecting models.

Generally, data pre-processing techniques refer to the addition, deletion, or transformation of training set data. These transformations concern mainly predictors (independent variables) in a way that they can be filtered to avoid outliers, samples that are exceptionally far from mainstream of data, they can be scaled due to specific characteristics of the problem and might even experience feature-selection, as there are some predictors that have more influence in the outcome than others [26, p. 27–50]. There are also popular techniques like K-nearest neighbor to avoid and deal with missing values in the dataset.

2.4 Overfitting and Model Tuning

With technology breakthroughs over the years in machine learning and predictive modeling, now many techniques can learn the structure of a set of data so well that when the model is applied to data on which it was built, it just predicts every sample correctly. This type of model is so connected with training data that is said to be over-fit and will usually have poor accuracy when predicting new samples. Figure 2.1 is a good example that illustrates over-fitting as it can be observed two models with two predictors, being the left model an example of an over-fit model, hence various different boundaries overextended to correctly classify every data point, whereas the second one represents a model that is fairly smooth with only one line (not overfitted).

Model tuning, on the other hand, is the process to achieve the best model settings which allow it to achieve its optimal performance. In machine learning, this is accomplished by selecting appropriate “hyper-parameters”. This can be any parameters that individually or combined get the best predictions out of the model used. This is normally achieved by splitting the existing data into training sets used to build and tune the model, and the test sets used to validate it [26, p. 61–70]. Resampling techniques like k-Fold Cross-Validation and its variants can be used for estimating model performance and to achieve model tuning.

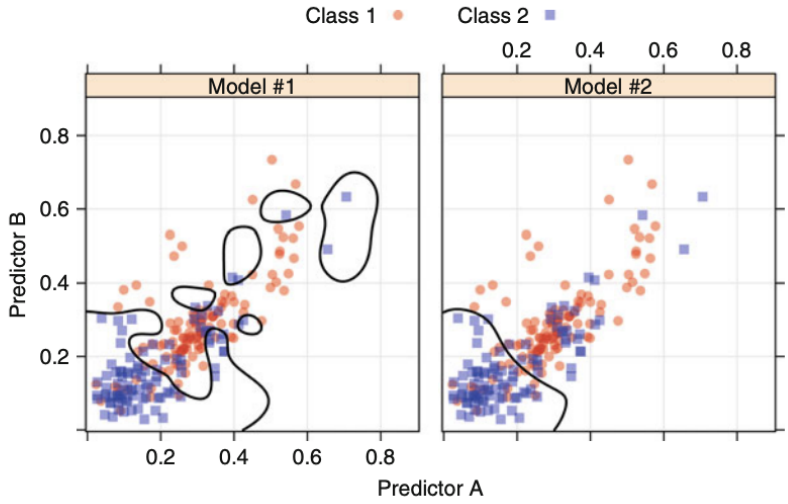

Figure 2.1: An example of a training set with two classes and two predictors. The panels show two different classification models and their associated class boundaries [26]

2.5 Evaluation Metrics For Result Comparison

According to Wen's research [54], evaluation metrics such as the Mean Magnitude of Relative Error (MMRE) (2.5) and Percentage Relative Error Deviation (PRED) (2.6) are widely used to make comparisons between methodologies and their performance. They both use the Magnitude of Relative Error (MRE) (2.4) and have been subject to various critical studies since these are considered to be asymmetric measures and biased accuracy statistics [45]. With that in mind, other standard and non-biased measures will be used, such as Mean Absolute Error (MAE) (2.1) and Root Mean Squared Error (RMSE) (2.2) since they both penalize errors in both directions in the same way. These measures are negatively oriented - the lower the value the better - and since RMSE gives more weight to large errors, then if it differs largely from MAE, then the incidence of larger errors is greater. At last, besides MAE and RMSE, the Squared Correlation (R^2) (2.3), also called *Coefficient of Determination*, will be used to measure the algorithms performance [15].

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.1}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.2}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2} \tag{2.3}$$

$$MRE_i = \frac{|y_i - \hat{y}_i|}{y_i} \tag{2.4}$$

$$\text{MMRE}_\text{i} = \frac{1}{n}\sum_{i=1}^{n} MRE_i \tag{2.5}$$

$$PRED(x) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}1, & \text{if } MRE \leq x\\ 0, & \text{otherwise}\end{cases} \tag{2.6}$$

Where

y – actual value

$\hat{y}$ – estimated value

$\bar{y}$ – mean of actual values

x – percentage, typically 25% or 30%

n – number of samples

2.6 Summary

Software development project managers have an important task at hand in the planning of every new iteration. It is known that failure in this process usually "snowballs" forward and eventually leads to failure in the rest of the development process. A key measure that PMs have to deal with at every planning stage is forecasting. Later, the task duration and total effort forecast will be calculated in months and man-months, respectively. Every time PMs and other managers fail to predict these parameters and delays in schedules start to appear, the normal answer is to add man-power which can, lot of times, put even more pressure to the team and eventually lead to failure, as already mentioned in section 2.1

When building a model, it is important to think about the different stages it faces as this is not a simple process. As observed in Figure 2.2 to build and compare methodologies, first it is needed to attend and comprehend the data that is being worked with. This pre-processing of the data as aforementioned in 2.3 might mean dealing with transformations in data in a way to face outliers, samples that are exceptionally far from the mainstream of the data, missing values, or even to

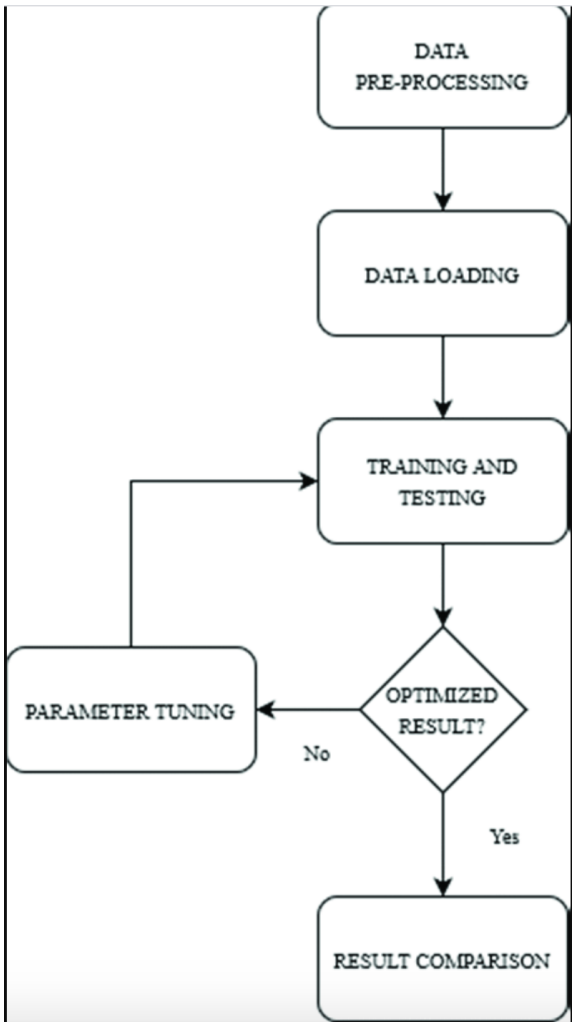

Figure 2.2: Modeling methodology [49]

feature select and disregard unimportant predictors that add noise to the data and decrease model efficiency. Then it comes the training and testing phases which is when the data is split to build the model with specific algorithms and fulfill it with the expected prediction powers. Finally, the results must be analysed and validated to choose the right methodology for the specific problem at hand. This can be done with the key error metrics 2.5, comparing the estimation accuracy between models and other important factors like model resistance to overfitting and how can the results be improved, if not already optimal, with parameter tuning processes as referenced in 2.4. It is also important to note that as pushing towards higher accuracy, models become more complex, and their interpretability becomes more difficult as well as their implementation.

Chapter 3

Forecasting Methods

In this chapter will be presented an overview of the *state of art* in project effort and duration forecasting. Different methods will be explored as their advantages and limitations. After the analysis of these methods and a review of existing work on regression problems, with particular attention to effort and duration estimation, important conclusions will be taken for possible next steps to take on this dissertation.

3.1 Types of Methods

Analysing the existing methods, these can be divided into 3 main types [3, 36, 39, 54]:

- **Empirical Methods** — methods that gather information by observation or analogy of similar projects, and expert experience;
- **Theory-Based Methods** — methods based on explicit theory or logic models;
- **Machine Learning Methods** — methods that work with machine learning and focus on the learning ability inherent.

3.2 Empirical Methods

In this section, some of the existing empirical methods, like expert opinion and analogy methods, will be explored.

3.2.1 Expert-Estimation

Expert-estimation or expert-opinion methods are base on professional's experience and opinions used to forecast metrics like duration and effort in every project planning phase. This type of technique, as shown by Muhammad Usman et al. [53], is the most commonly used for effort and

duration estimation in agile software development projects. Studies reported from Magne Jørgensen [23] also show that in many cases these methods led to more accurate effort estimates than using sophisticated forecasting models. These results were mainly due to the difficulty and complexity of building proper models and understanding the input data, which many times led to overfitting and lower accuracy problems, and bad calibration of these models within the organization they are to be implemented.

Examples of expert-estimation methods commonly used in industry are:

- **Wideband Delphi** — developed by Barry Boehm and John Farquhar, is an expert-opinion method where the team is divided in the estimation team and a moderator [47]. Then, individually, each member of the estimation team should write down his estimates and assumptions that lead him to that prediction. After some brainstorming the team should arrive to an agreement. In the end, the PM and the team review the final results to check if everything is in order. The moderator should not be the PM to avoid human bias and should be someone that knows the Delphi process well to moderate every conversation. Although it is a simple and easy to implement method, it is also really time-consuming (with long meetings) and exposed to human bias either by the moderator or by the estimation team;

- **Planning Poker** — developed by J. Greening [18] and also known as Scrum poker, it is a gamified technique based on consensus for estimation of the effort of development tasks. The team usually gathers together in every planning release meeting to discuss the description of the tasks to be done and then members of the team make estimates by selecting numbered cards (from a Fibonacci sequence) which are then simultaneously revealed (to avoid human bias between members) and afterward discussed. After analysing the highest and lowest estimations the team should try to achieve common ground. If that does not happen then the team must either defer the task, split it, or take the lowest estimation. Planning poker is noted to be one of the most known and used methods in the industry due to its simplicity, satisfactory accuracy, easy implementation and understanding by all stakeholders. Although commonly used in the industry it also has its drawbacks like the fact that it is a manual process so more time consuming than computational methods, it is subject to human bias that may influence member's estimation in order to achieve personal goals and many times the discussion phase is skipped to speed meetings, jeopardizing the method success;

- **PERT** — the Project Evaluation Review Technique (PERT) is a method used to predict the duration of a particular task or activity. For these estimations a PERT chart is used based on three different time estimates for the project: the shortest possible amount of time each task will take, the most probable amount of time, and the longest amount of time tasks can take if things do not go as planned [1]. Although it is a simple and easy to implement method, even with little data, it might be hard to interpret and manage in complex projects and may be susceptible to human bias.

To conclude, these methods prove themselves to be very simple to implement and understand with satisfactory accuracy. However, they are time-consuming and rely on expert knowledge, experience, and perception which may be biased.

3.2.2 Analogy

The use of analogy for software estimation is the process of comparing the proposed project to previously completed similar projects where the information of the project development is known and from where the estimation will be deducted for the new cases.

According to Ali Idri and Amit Sinhal et al. [4, 46], analogy processes are generally composed of three steps:

- **Characterizing** the proposed project;
- **Similarity evaluation**, which means selecting the most similar completed projects whose characteristics have been stored in the historical database;
- **Adaptation**, which means deriving the estimate for the proposed project from the most similar completed projects by analogy.

An example of a known analogy method is Chistopher Schofield's ANaloGy Estimation tooL (ANGEL) [42], which is presented to be one of the most used analogy estimation tools.

The main advantages of analogy based methods are:

- The ability to achieve reasonable levels of accuracy;
- Simple if the organization repeats similar work;
- Estimates are immediately available;
- Encourages detailed documentation.

Although presenting some advantages, these methods can be unreliable and extremely tough to evaluate the dissimilarity among the environments and therefore assess the correctness of the data [46].

3.3 Theory-Based Methods

In this section, existing theory-based methods will be explored and analysed, resulting in a conclusion to which might be applicable.

3.3.1 COCOMO

COnstructive COst MOdel (COCOMO), a theory-based model created by Barry Boehm in 1981 is one of the most used methods for effort estimation. Consists of a hierarchy of three increasingly

detailed and precise forms: basic, intermediate and embedded which use the following equations 3.1 (for the basic form) and 3.2 (for the remaining two forms) to calculate the effort of each delivery [32, 46]:

$$E = a(KLOC)^b \tag{3.1}$$

$$E = a(KLOC)^b * EAF \tag{3.2}$$

Where E represents the software effort computed in man-months. The values of the parameters a and b depend mainly on the class of software project. *KLOC* (kilo line of code - all program instructions and formal statements) is considered a primary element that affects the effort estimation and *EAF* is the Effort Adjust Factors which in the new COCOMO II is the product of all 17 available cost drivers [32]. Software projects are classified based on their complexity and size into three distinct categories:

- **Organic**, a mode used for small projects developed within stable environments with a relaxed requirement specification;
- **Embedded**, a mode used in large projects that are operated under tight constraints.
- **Semi-detached**, a mode used in intermediate projects that lie between the two previous extreme models.

3.3.2 Monte Carlo

Monte Carlo was first applied in the decade of 1930 to estimate the value of π, being only later, in 1946, formalized by Stanislaw Ulam and Enrico Fermi in the Los Alamos Laboratory, USA [29]. This technique uses random sampling techniques, where the inputs are chosen randomly from a data subset and analyses the results over many computational cycles. It returns a range of values instead of single values, retrieving the probability for each range of possible values. This is a simple, easy to interpret method that was made to forecast results of problems that are too difficult to solve in order to arrive to a single exact value. Nowadays, it is mainly being used to manage risks in fields like oil, finance, gas exploration and project management, with many implementations within project management. It is known to be an easy and fast model to learn from the dataset and make decent predictions although it might be exposed to problems of overfitting [34, 35].

3.3.3 Summary

COCOMO is a very used method that normally takes too many inputs that are calculated by experts, meaning that an expert must estimate some inputs in order to make the forecast which opens the method to subjectivity to human bias. Even though it is a simple to implement and relatively easy to understand method, if it is done solely with inputs estimated by humans it generally does not show much accuracy. It may also not be feasible for modern approaches in software

development due to new trends in code reuse, codeless programming and agile development. A possible implementation studied by Suherman et al. [49] of COCOMO that increases its accuracy is the combining of it with other tuning parameter methods and a comparison was made against machine learning methods alone to find out which approach gives better performance. In the findings, it is reported that Random Forest Regression has the best result (54%) to estimate software effort against a MMRE of 73% in SVR (Support Vector Regression) and 115% in COCOMO II with machine learning parameter tuning, meaning that even with an improved parameter tuning process ML methods still present a higher hope for better results.

Monte-Carlo, on the other hand, proves to be a decent method that presents satisfactory accuracy and is easy to implement and understand. Another benefit is the easy interpretation of the degree each input affects the final forecast. Even though being a proven and good alternative for the effort, duration and cost forecast, it might still be exposed to overfitting problems. It does not also seem to present the same level of accuracy when compared with other machine learning models [34].

Other methods like the Software LIfecycle Management(SLIM) method, created by L. Putnam, could be presented, but they generally do not show much accuracy, might not be feasible for modern approaches in software development and do not often take into consideration the skill set of the project team [14, 39].

3.4 Machine Learning Methods

In this section, machine learning methods will be explored for calculating the relationships between a dependent variable and one or many independent variables. In the end there will be comparisons between the various methods.

3.4.1 Linear Regression and variations

Linear regression methods are appropriate for cases where the relationship between the predictors and the target variable follows something close to a flat hyperplane for multiple predictors or a simple straight line in cases where the data only has a single predictor. As such these type of methods can directly or indirectly be written in the following form [26, p. 101–111]:

$$y_i = b_0 + b_1 x_{i1} + \ldots + b_P x_{iP} + e_i \tag{3.3}$$

Where y_i is the response for sample i, b_0 is the estimated intercept, b_P represents the estimated coefficient of the Pth predictor with value x_{iP} and e_i represents the random error that cannot be explained by the model.

If the relationship looks rather curvilinear then predictors can be augmented to capture these relationships having, for instance, a quadratic or cubic form instead. With that being said, these types of models are fast, easy to implement and really simple to interpret, although limited to linear

relationships, sensitive to outliers and need scaling and centering techniques to improve numeric stability.

Variations of these linear regression methods, such as penalized models ridge regression, lasso, and elastic net, that apply penalty functions have been created to reduce overfitting, improve performance and even apply feature selection (e.g. lasso), still present the main limitations stated above for these type of methods [26, p. 101–127] [13].

3.4.2 Neural Networks

Neural networks are powerful non-linear regression techniques inspired by the brain anatomy as observed in Figure 3.1. The outcome is modeled by an intermediary set of layers with unobserved variables (called hidden variables).

These hidden variables are a linear combination of some or all of the predictor variables which typically are transformed by a non-linear function (i.e. sigmoidal). The outcome is then calculated by a linear combination of the hidden variables. Most methods based on ANN use either feed-forward networks where no loops in the network path occur or feedback networks that have recursive loops. It is known that neural networks have a tendency to over-fit the relationship between the predictors and the response due to a large number of regression coefficients, although there exist several different approaches to fight that and increase the model efficiency like Multi-Layer Perceptron Artificial Neural Networks (MLP-ANN) [26, 141–145]. Other limitations of ANN as exposed by Miranda [34] are the difficulty of understanding and interpreting the method being many times called *"black boxes"* and the amount of data required to usefully train the method.

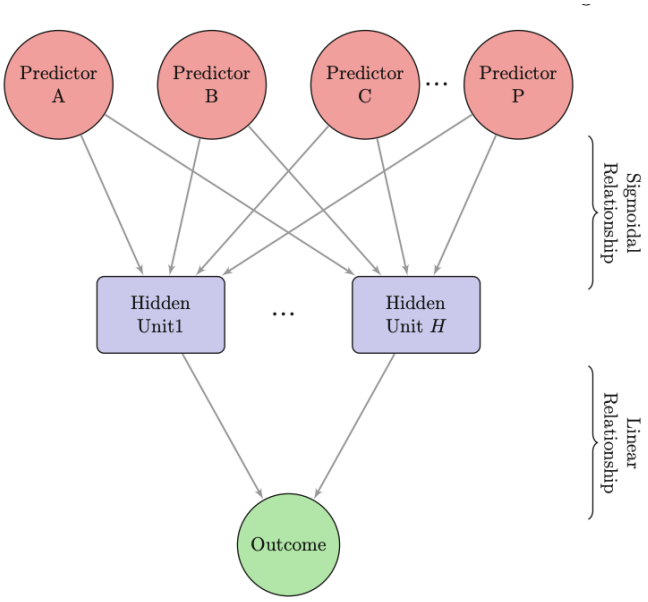

Figure 3.1: Representation of the Neural Network model

3.4.3 Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines (MARS) is a non-linear regression method that builds multiple linear regression models across the range of predictor values. It does this by partitioning the data and run a linear regression model on each different partition [2].

After creating the basis functions (BF) and partitioning the predictor values in several groups, a separate linear regression is modeled on each group separately, creating various regressions lines with a unique slope each. Then the MARS algorithm automatically searches for the best connection spots between separate regression lines, called knots, each with a pair of basis functions that describe the relationship between the environmental and target variables as shown in Figure 3.2 [2].

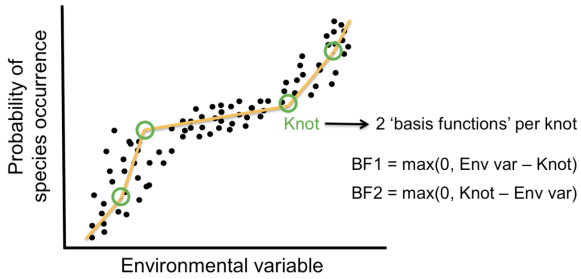

Figure 3.2: Representation of a MARS model [2]

Although it is a model sensitive to overfitting and not very robust to noisy data, it works well with a large number of independent variables, requires very little pre-processing, it automatically does feature selection and is an efficient and fast algorithm, despite its higher complexity when compared with linear methods, but not as complex as neural networks.

3.4.4 Support Vector Machines

Support Vector Machine (SVM) is a machine learning technique used in various areas both for regression and classification. The SVM classifier separates instances from two different classes by using a hyper-plane that tries to maximize the margin, increasing the capacity of generalization of the classifier. For regression other techniques are used such as ε-insensitive regression but most of the concepts stay the same. The instances closer to the borders are the support vectors and they help define the separating line between classes by calculating margins. The number of support vectors represents the complexity of the model [34]. In Figure 3.3 is a representation of the SVM algorithm.

Whenever the data is not linearly separable, the SVM uses concepts such as soft margin, which tries to find a line to separate the data, tolerating some misclassified dots with the penalty, and kernel tricks. The SVM algorithm is implemented in practice using a kernel function to transforms the inputs into the required form [55]. These kernels are:

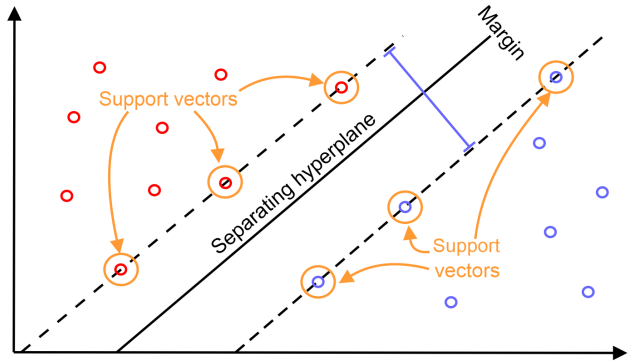

Figure 3.3: Representation of the SVM model [55]

- **Linear Kernel**, used when the data is linearly separable, and mostly when the number of features is large;
- **Polynomial Kernel**, a more generalized form of the linear kernel which distinguishes curved or non-linear input spaces;
- **RBF Kernel**, general-purpose kernel, used when there is no prior knowledge about the data and when the number of features is not so large.

Since the predictors enter the model as the sum of cross products, techniques such as centering and scaling may be needed as relevant differences in the predictor scales can affect the model.

Some advantages of SVM are good performance with unstructured and semistructured data and less risk for overfitting. Disadvantages are the long training time for large datasets, difficulty to interpret and calibrate the model, and the sensitivity to noisy data like missing values and outliers [13].

3.4.5 K-Nearest Neighbors

The KNN approach simply predicts a new sample using the K-closest samples from the training set. To predict a new sample for regression, KNN identifies that sample's KNNs in the predictor space. The predicted response for the new sample is then the mean (or the median) of the K neighbors' responses. This method is dependent on how the user defines distance between samples being the Euclidean distance the most commonly used metric and is defined as follows [26, 159–161]:

$$\left(\sum_{j=1}^{P} \left(x_{aj} - x_{bj} \right)^2 \right)^{1/2} \tag{3.4}$$

Where x_a and x_b are two individual samples.

The KNN method can have poor predictive performance when local predictor structure grows or is not relevant to response. Irrelevant or noisy predictors are one culprit, since these can cause similar samples to be driven away from each other in the predictor space. Hence, removing irrelevant, noisy predictors is a key pre-processing step for KNN. Scaling and centering techniques are also needed since this method is based on distances and different scale predictors will generate distances that are weighted towards larger scale predictors, increasing the bias and lowering performance. Nevertheless, it is a very simple and easy to implement method.

3.4.6 Regression Trees and variations

Tree-based models are based on one or more nested if-then statements for the predictors that partition the data. The data is split into smaller groups that are more homogenous with respect to the result. So at each step a predictor is chosen as its value to be split increasing the number of nodes in the tree until a leaf is reached where a specific outcome is given [26, 173–190].

These type of models are generally fast and easy to implement, being also easy to interpret, except in cases where the predictors and the data are too large and the tree gets very big. In addition, since these models do not require much pre-processing, due to the fact that they can effectively handle any type of predictor (e.g. continuous, categorical, etc.), handle noisy data, and conduct an intrinsic feature selection, they are often chosen for many real-life problems. These, however, are sensitive to overfitting, are unstable (as a minor change in the data can alter the whole tree structure) and, although normally getting decent results, are not always optimal in performance.

To overcome these issues, ensemble methods have been developed that combine many trees (or rule-based models) into one model [26, 13].

The *Bagging ensemble* is a technique that uses *bootstrapping* in conjunction with regression models, such as trees, that in the end combine all individual predictions (e.g. by average) to make the final prediction, reducing the model variance, hence enhancing its capacity to generalize and being able to achieve better performances [26].

The *Random Forest* is a very similar ensemble model to bagging trees, but instead of splitting the data considering the whole set of predictors, it randomly selects from a subset of predictors the best one to do the split. This adds randomness to the model, decreasing variance (making it more robust to overfitting) and improving the results [26, 13].

The *Boosting ensemble* also considers homogeneous weak learners, such as trees, learns them sequentially in a very adaptive way (a base model depends on the previous ones) and in the end combines them following a deterministic strategy [26, 13]. This type of ensemble is very flexible and has been proving to be one of the best in performance, having variations as the XGBoost that have been winning many machine learning competitions lately [11].

3.4.7 Summary

After presenting various machine learning predictive models, a comparison of the advantages and disadvantages, as already described in some aforementioned methods, can be summarized and further detailed in the table 3.1.

Characteristic	EN/Lasso	ANN	SVM	MARS	KNN	DT	Bagg	RF	Boost
Pre-Processing	CS, NZ	CS, NZ	CS	-	CS, NZ	-	-	-	-
Tuning Parameters	1-2	2-3	2-3	1-2	1	0-1	0-1	1-2	3
Automatic Feature Selection	√	×	×	√	×	√	√	√	√
Handling of noisy data	×	×	×	o	×	√	√	√	√
Robustness to overfitting	o	×	√	×	√	×	×	×	×
Interpretability	√	×	×	o	×	o	×	×	×
Predictive performance	×	√	√	o	o	o	o	√	√
Computation Time	√	×	×	√	√	√	o	×	×

Table 3.1: Some characteristics of different machine learning methods [13, 26]. Symbols represent Good (√), Decent (o), Poor (×), Centering and Scaling (CS), Normalization (NZ)
EN - Elastic Net; DT - Decision Trees; Bagg - Bagged Trees; RF - Random Forest; Boost - Boosted Tree

3.5 Related Work

In the last two decades, extensive research has been made in terms of applying data mining, statistical and machine learning algorithms for software estimation. According to Pospieszny [39], most researches focused on estimating effort and duration for the initial phases of a development project since forecasts in these stages are generally more challenging due to uncertainty and limited knowledge, and the failure in such initial phases could prove to be deadly for the success of the project.

A extensive and comprehensive research was made by Wen et al. [54], where a review of 84 studies concerning machine learning methods for effort estimation was made and according to his results, in the last two decades, researchers focused their attention mainly on tailoring individual algorithms for best performance, such as artificial neural networks (ANN), case-based reasoning (CBR) models and decision trees (DT). From this research it was also clear that ML models performed better than other traditional and statistical ones, with mean magnitude relative error (MMRE) ranging from 35-55% and PRED(25) from 45-75%. Researchers also gave a big relevance to the fact that depending on the dataset and the pre-processing approach ML algorithms may perform very differently due to outliers, missing values and overfitting problems.

The discrepancy in results and building ML models is even more perceptible when analysing individual publications like de Barcelos Tronto et al. [6], where it was compared the accuracy of ANN approaches with multiple regression models for effort estimation using COCOMO dataset and for evaluation purposes MMRE and PRED, or Berlin et al. [7] that examined and compared

the accuracy of ANN and linear regression (LR) not only for effort but also for duration prediction in an ISBSG dataset, having both concluded that ANN outperformed the other used models even though the results differ between them. It is also important to note that on Berlin research a log transformation of output variables was made to improve the accuracy. Another example is López-Martín and Abran [30] publication that focused his research on comparing prediction precision between different neural network types on effort estimation with normalisation of dependent variables, cross-validation approaches and MAR with MdAR were used as an accuracy criterion. In this research the Radial Basis Function Neural Network was set to perform better with a high confidence level, although, for instance, in Pospieszny research the MLP-ANN was preferred [39]. Also, Rhaman et al. [40], that compared different types of ANN and DTs, using RMSE for comparison, concluded that DTs were more effective in small-median datasets where in large the ANNs showed more accuracy.

Other interesting researches such as Minku and Yao [33], that worried about the sensitivity of ML methods to noises within the datasets, supported that models should not rely on individual algorithms but instead a conjunction which should increase prediction accuracy and strengthen the models to noisy data and overfitting problems. More researches support this idea with Kocaguneli et al. [24] proposing various ensemble methods, such as boosting, bagging and complex random sampling techniques. Even though the ensemble techniques may post many advantages, Azhar et al. [5] stated that these methods might introduce substantial performance overhead if applied too excessively. With that in mind, Tin Ho [50] concluded that a set of different but limited number of algorithms and simple ensemble approaches such as averaging the obtained estimates should be used for building ML models for effort and duration forecasting. Another successful example of this approach is Pospieszny [39] proposal with an ensemble model of three different ML algorithms: SVM, MLP-ANN and GLM and an average approach to obtain the forecasts, that showed very good results.

Another, yet more generic and recent, extensive research was made by Fernández et al. [15], where a review of 77 popular regression models was made which belong to 19 families: linear and generalized linear models, generalized additive models, least squares, projection methods, LASSO and ridge regression, Bayesian models, Gaussian processes, quantile regression, nearest neighbors, regression trees and rules, random forests, bagging and boosting, neural networks, deep learning and support vector regression. The methods were all implemented in 83 different datasets splitting the data in 50% for training, 25% for tuning parameters and the last 25% for testing. A smart pre-processing and cross-validation strategy was used and for evaluating and comparing the various families and algorithms measures like MAE, RMSE and R^2 were used. This review ranked each algorithm between different sized datasets and concluded that the most effective algorithms for the various different regression problems were in the following order: cubist and M5 regression rules, the gradient boosting machines (gbm), extremely randomized regression trees (extraTrees), SVMs, neural networks, the projection pursuit (ppr) from the projection methods family, nearest neighbors and also bagging ensemble of MARS models had a decent performance. A time and error (in terms of memory and time errors) comparison was also made with the regression rules

and boosting ensembles being faster and less error-prone, with SVMs and neural networks, on the other hand, being more error-prone and slower, although still presenting satisfactory results and even achieving the best results in some databases. A worth mentioning algorithm with high performance that has been winning regression competitions lately is the XGBoost created by Chen et al. [11] which is a variation of the gradient boosting algorithm.

Despite the large number of different approaches taken to build ML models and forecast effort and duration, important recommendations can be extracted for implementing them in practice. Some of these recommendations emphasize pre-processing, like the use of different techniques, such as deletion or imputation of values, which according to Huang et al. [20] depend on the dataset and help dealing with outliers and missing values. As stated by Strike et al. [48] missing values should be discarded, when possible, to reduce bias. As a way to deal and remove outliers Ruan et al. [41] proposes the use of the common rule of three standard deviations from a mean. However, since this method may be biased due to the fact that the mean is also affected by outliers, Christophe Leys et al. [28] proposed instead a median absolute deviation approach or the typical interquartile method to detect and remove outliers. Berlin and López-Martín et al. [7, 30] also indicate that log transformation of effort and duration tend to generate more accurate estimates although not necessary in ML algorithms.

Finally to conclude, besides the aforementioned limitations and, as stated by Pospieszny [39], the application of ML methods to small, outdated datasets (COCOMO, NASA), and the fact that these rarely follow current software methodologies and modern development approaches, led to inconclusive results and few implementations of ML predictive algorithms for effort and duration estimation within organisations. Pospieszny article tries to address these limitations and implement a practical effective heterogeneous ensemble model for effort and duration forecasting which this dissertation aims to explore and extend with a wider variety of regression models to find the best ensemble combination for this delivery forecasting problem.

3.6 Conclusion of state of the art

Due to the fact that estimations, like effort and duration forecasting, play an important role in every planning phase of a development project, being key factors for many projects to either be successful or terminated [52], in the last decades many researches have been made in order to solve these issues.

For that purpose, software estimation techniques based on expert knowledge or by analogy, like PERT, ANGEL, Wideband Delphi, and Planning Poker, as explored in section 3.2, are most widely used due to their simplicity and almost effortless application [56]. Nevertheless, these techniques are usually error-prone and time-consuming, which led to the creation of alternative approaches as seen in section 3.3. These alternatives are based on theory methods like COCOMO, Monte Carlo, SLIM, and even variations like COCOMO II with machine learning tuning methods, in order to improve its prediction ability [36, 49]. These methods also ended up not being ideal as they present identical problems with empirical approaches due to the fact that most parameters are

calculated by experts which makes them subjective to human bias. Besides that, constant updates to new trends in programming, software architecture, and software development methodologies also led the methods to become outdated [17] and not so efficient when compared with other data-driven methods like machine learning [13]. This was also proved in Suherman research [49], where a Random Forest method outperformed a COCOMO II method with parameter tuning.

Therefore, in order to tackle the mentioned limitations, in the last decades research has been made for software estimation machine learning techniques [13, 15, 44, 54] that are considered highly effective for tackling uncertainty and obtained powerful prediction capabilities for effort and duration estimation. Related works, as seen in section 3.5, and several ML algorithms, seen in section 3.4, were presented and explored to compare different approaches for effort and duration forecasting. It was concluded that few implementations could be found in practice (as most organizations still rely on theory-based and expert methods [3]) due to inconclusive results that derived from factors like the focus on specific algorithms, tailoring them for best performance, bad data pre-processing approaches, and models trained in small legacy and outdated datasets which finally led to a disparity of results and not much conclusion. In the end, it was established that no method is perfect for any specific kind of problem and there can be several solutions [46]. Nevertheless, a set of a different but limited number of algorithms ensembled with a simple average approach with good practices in data preparation, to avoid problems such as overfitting, outliers, and useless/noisy data, should be a promising way of tackling the problem. This type of heterogeneous ensembles, besides the clear aforementioned advantages presented, also enable the final model to be more flexible and robust in any type of dataset when compared with the typical homogenous ensemble approaches like the bagging and boosting ensembles. In this way, efficient ML models can be built for effort and duration forecasting [39, 50].

Chapter 4

Problem Description and Proposed Approach

In this section, it is described the problem tackled in this dissertation, presenting a detailed solution to be built and further analysed.

4.1 Problem

Nowadays, every project planning and management is dependent on estimations, like duration, effort and cost of tasks/projects. Due to the difficulty of such estimations many projects fail to accurately estimate certain parameters that may lead to bad schedule prevision and consequently to delays in the project that might have catastrophic consequences for organizations like project termination.

It is known that accurate estimations of development effort or duration play an important role in the successful management of a project. Even though this process is so important, experts can not usually estimate accurately the effort, time and cost of a project or its tasks to be developed, due to uncertainties and other constraints like human bias [39]. Usual estimation methods used in many companies follow strategies like *Poker Planning* that even though presenting satisfactory results, normally take precious time that could be used to actually add value to a project.

With the help of Fraunhofer, PROMESSA (PROject ManagEment Intelligent aSSistAnt) was launched with the aim to develop a system capable of assisting other project management tools. One example of these tools is the Project Control. This is a management tool developed by Fraunhofer Portugal, that aims to assist in high-level management, in terms of measuring the general effort, either financially or in human resources, of any type of project. Project Managers using the Project Control normally face the unavoidable process of planning releases that require effort and duration forecasting for every new task. This process can be both time-costly and error-prone due to subjectivity in human opinion. Besides that, the arising of data-driven methods has proven

to be a way of fighting these limitations and save costs. This dissertation aims to address these limitations and build a proper method that can then be integrated into PROMESSA to aid other tools in the difficult effort and duration forecast inherent in every project.

4.2 Proposed solution

As concluded from the *state of art* section 3.6, the proposed solution will use an averaging ensemble of several different machine learning algorithms (gradient boosting machine - gbm, extremely randomized trees - extraTrees, random forest - RF, SVMs, ANN, lasso, extreme gradient boosting - XGBoost, and KNN) to estimate effort (in man-months) and duration (in months), due to their proved efficacy, adaptability to changing environments, and enhanced robustness to both over and underfitting when compared to individual ML approaches [15, 39].

Before building the model, an effective approach of smart data preparation and feature-selection will be used for every modeling process to use a "cleaner" (in terms of noisy data) input dataset and meaningful input parameters. The process of data preparation, feature-selection, modeling each ML algorithm with k-fold cross-validation, ensembling by average and evaluation of the models is more detailed in Figure 4.1.

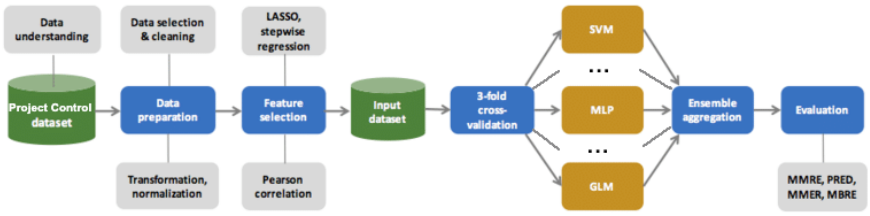

Figure 4.1: Process of building effort and duration ensemble model [39]

The data will be split into 75% for training and testing and the remaining 25% for model tuning. Each ML method will be individually modeled and evaluated. Only then the ensemble model can be built. This approach will allow accurate comparisons between each model and also the proposed ensemble solution, allowing for the second research question proposed in section 1.4 to be answered. In this process, whenever a method is not having a similar or better accuracy when compared to others it could be either excluded or replaced by other ML algorithms, so it does not jeopardize the final ensemble solution [39].

Experiences with different subsets of the data will also be done so that *RQ1* from the research questions section 1.4 may be answered and more comparisons done to make conclusions about what is the most effective approach for the problem at hand.

Finally, the solution proposed will be developed in Python with the aid of the scikit-learn [1] library, because it is a well known and proven solution for building and exploring machine learning techniques as well as all the data pre-processing and validation inherent to every machine learning building process [37, 38].

4.3 Evaluation

Apart from using the standard unbiased error measures seen in section 2.5 such as MAE, RMSE and R^2 to compare the accuracy between methods, two criteria which are widely used in software estimation will also be applied: Mean Magnitude Relative Error (MMRE) and Percentage Relative Error Deviation (PRED). According to Conte et al. [12] a software estimation model is considered accurate when MMRE $\leq$ .25 and PRED (.25) $\geq$ .75. Generally both levels are rarely achieved and for PRED a 30% MRE threshold is often applied instead [22]. Although these measures are said to be deprecated [45], their use will help when comparing this dissertation results with the researched papers and to define a threshold of what can be called a "satisfactory/unsatisfactory" solution. With that in mind, a comparison between models will be made instead through MAE, RMSE and R^2 with particular attention for RMSE since this metric gives more weight to large errors which in effort and duration estimation of deliveries is not very affordable [15].

[1] More information about the scikit-learn library here: https://scikit-learn.org/

Chapter 5

Methodology

In this chapter, the data will be discussed in more detail and how it was prepared to feed the models. The pre-processing and parameter tuning process will also be explained and how to achieve the final heterogeneous ensemble model for the Project Control tool in PROMESSA platform.

5.1 Data Understanding and Preparation

In this section the structure of the dataset available during this project will be explained to aid in understanding the algorithms results. The dataset was extracted from the Project Control environment from Fraunhofer and was stored on a MySQL database.

This database is composed by 1127 different projects from the beginnings of 2013 til the endings of 2019 counting 22463 different work packages (sub-projects within the projects) and with a total of 54153 tasks with people allocated. The most important aspects of the database structure schema and data can be further specified in more detail:

- **Project** - represents the different projects and their details;
- **Baseline** - represents an instance of a project, like a specific moment within the project. Whenever a project is redefined or changes in the deadlines occur, a new baseline is created representing a new instance of that project with the new edited details;
- **WorkPackage** - represents the sub-projects or phases in which a project is subdivided;
- **Task** - represents a single task to be done in a work package of a project;
- **Allocation** - represents the allocation of each user to a task with details such as effort, dates, description of work, etc.;

After some analysis and discussions with Project Control data specialists, only the few tables described above were actually considered to aid in the task effort and duration forecast. Other

tables describing users, organizations, contacts, activities, investments, etc., were considered not important and thus excluded from the input dataset for simplification purposes.

Most of these other tables were excluded on the basis that whenever a new task is being created there is no way to know that information beforehand. Take the users for example, each user will be allocated to a task only after it is completely defined and specified, so this data becomes irrelevant since it can not be inserted as input upon creation of new tasks. Other information was just adding noise or not being relevant.

It was left for tables such as *Project*, *Baseline*, *WorkPackage*, *Task* and *Allocation* to work as input and output for this problem, with the last table, *Allocation*, being used mostly as an auxiliary to calculate the effort of the tasks. In the end, the data used and its structure can be seen in more detail in the figure 5.1.

Even with the selection of certain tables there were still many parameters to deal with. Being most of them useless or not having any weight to predict the target variables, a filter on these parameters was applied with the help of data specialists from Fraunhofer, in order to reduce the number of predictors and noise to feed the models. To further understand how this filter was done and what data can actually affect the output, task total effort in man-months and task duration in months, it will be detailed below the meaning of each parameter and which ones retain any meaning for the target calculation.

Table **Project**:

- **type_mask (p_type)** - type of the project. It can be either an European, National, Industry, Internal, Reserved, a Thesis or an Other type of project;
- **aicos_funding_rate (afr)** - the rate at which the company funds are applied;
- **equipment_funding_type_mask (eftm)** - the types of funding to be applied in terms of equipment for the project;
- **backlog_type_mask (btm)** - type of backlog being used in the project. If it is an Industry, EU, National or Other type of backlog.
- **revenue_distribution_mask (rdm)** - tells if the revenue is distributed and applied in a Linear, Planned, Real, or Other type of way;
- **department_mask** - represents the departments in which the project is inserted. Examples are: research and development, databases, financial, governmental, human resources, IT, administration and legal;
- **business_field_mask** - field of business in which the project is inserted in. Examples are: Ambient Assisted Living, Information and Communication Technologies for Development, Non-Research and Development and Others.
- **business_subfield_mask** - business subfield in wich the project is inserted into;

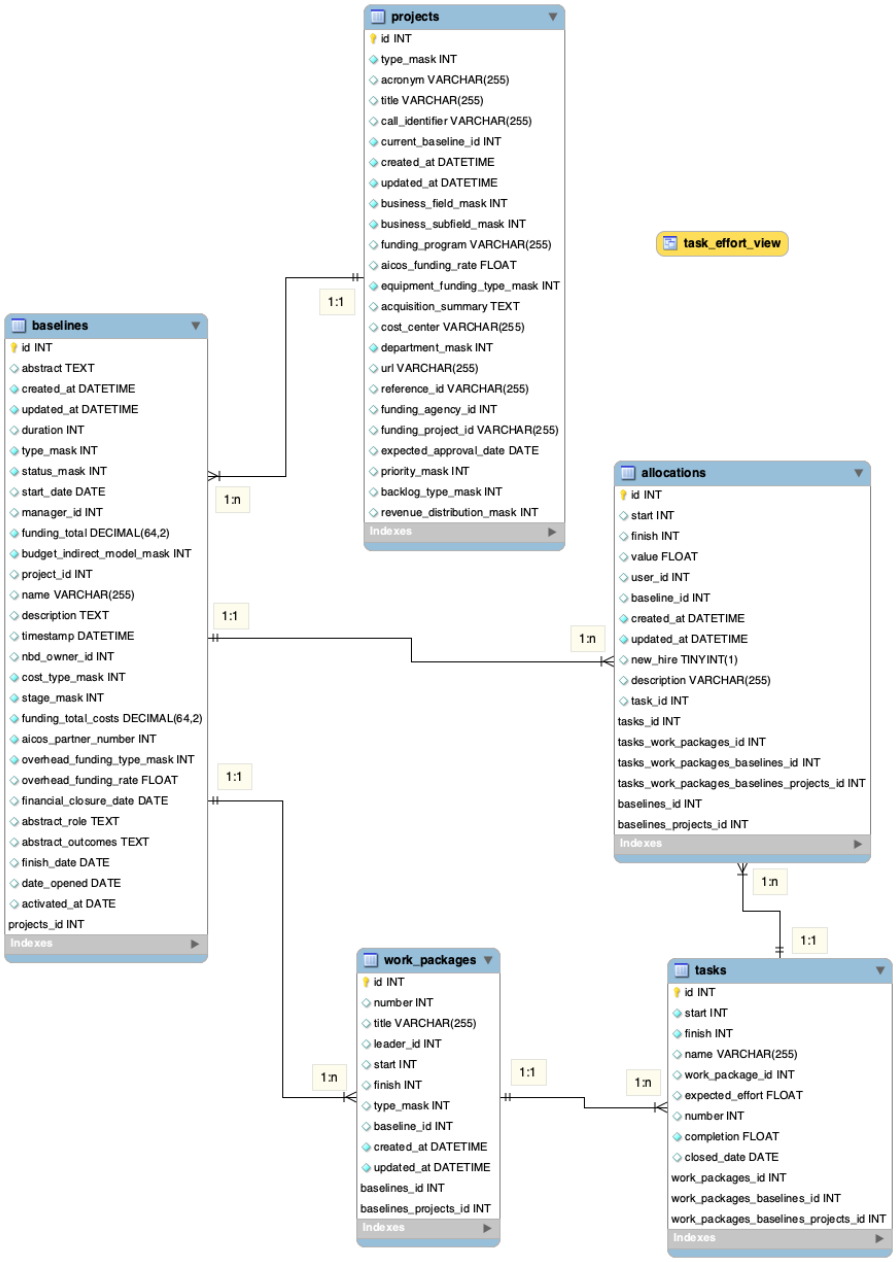

Figure 5.1: Subset of the Project Control Schema that is actually used to achieve the effort/duration forecasts

- **priority_mask** - the priority of the project. It can be either low, medium or high;

Note that the "mask" parameters described above are categorical variables that are already encoded into numeric in the database. This will avoid the boilerplate work of encoding these parameters as most ML algorithms require them to be numeric, especially in a python sklearn implementation.

Acronym, *title*, *call_identifier*, *funding_program*, *acquisition_summary*, *cost_center*, *url*, and

all the ids and dates parameters are all extra information with types such as strings and dates that were considered irrelevant for calculating the target variables. They are mostly names of project, funding programs, funding centers, etc., and other types of descriptions like abstracts of the project and so not much information can be taken to help with the forecast.

Table **Baseline**:

- **type_mask (b_type)** - type of baseline. Examples are lead, proposal, contract, forecast, budget, backup, billable hours and current. These represent the different phases a project, and its baselines, go through from the moment of creation/definition until the moment it gets approved for budget and is developed/completed;
- **duration** - the total duration of the baseline;
- **cost_type_mask** - type of costs applied in the project. These can be either base, EU, QREN or Industry costs.
- **stage_mask** - the different stages a baseline goes through. It can be lead, proposal and project.
- **status_mask** - the current status of the baseline. It can be something like if it has been approved for proposal, submitted, under analysis, in initial lead description, active, suspended, budgeting, etc.
- **funding_total** - the total funding available for this baseline/project;
- **budget_indirect_model_mask (bimm)** - model in which the budget is applied;
- **funding_total_costs** - the total funding applied to this baseline/project;
- **aicos_partner_number (apn)** - the number that identifies the Fraunhofer partner in the baseline;
- **overhead_funding_type_mask (ofm)** - type of funding overhead that the baseline experiences at the moment;
- **overhead_funding_rate (ofr)** - rate at which the overhead funding is applied.

Abstract, *name*, *description*, *abstract_role*, *abstract_outcomes*, and most of the ids and dates parameters are all extra information with types such as strings and dates that were considered irrelevant or not much information could be extracted to help calculating the target variables.

Table **WorkPackage**:

- **type_mask (w_type)** - type of work package. Examples are research and development, demonstration, management, and other;
- **number** - a number that represents the order in which the work package was created in the project to which it belongs. It starts from 0, which is the first work package created, 1 is the second and so on;

- **start** - the month in which the work package starts. Many times a work package is not created right away from the start and even if it is, does not mean it will start in the beginning of the project but can, for instance, start at month 4 which signifies that it will only be worked on the fourth month after the project begins;

- **finish** - the month in which the work package ends.

Parameters like *Title*, the ids and dates were not considered to be relevant for the forecasting and thus excluded from the set of inputs. On the other hand, parameters *start* and *finish* themselves do not seem to add much information to the target forecasting, but a surrogate variable calculated as the difference between them results in the work package duration which is much more informative and one can see its big relevance especially to forecast the duration of the tasks contained in that work package. This surrogate variable is calculated as in equation 5.1.

$$wp\_duration = finish - start + 1 \tag{5.1}$$

Since the *start* variable always represents the work package starting month from the very first day of that month and the *finish* the finishing month by its last day, then to calculate the work package duration we have to add 1 after subtracting the *finish* and *start*.

Table **Tasks**:

- **number** - a number that represents the order in which the task was created in its work package. This starts with the 0, so if the number is 0 then it is the first task created, 1 is the second and so on;

- **start** - the month in which the task starts;

- **finish** - the month in which the task ends.

Parameters like *completion*, the ids and dates were not considered to be relevant and thus excluded. The *name* parameter is a string that describes the task. Normally these types of variables do not have useful information for the forecast or any knowledge that could be extracted and is not already spread through other categorical variables. For instance, any useful information that could be taken from the *baseline abstract* variable is already represented in variables such as *type_mask*, *business_field_mask*, *department_mask*, etc. However, since there are no variables with such information for each task and the target variables are directly related to the task, maybe there is an opportunity to extract knowledge through data mining from the *name* parameter of the *task* table. This is explored in more detail in further sections.

The target variables for this problem are the task duration and task total effort. The duration is not a variable directly stored in the database. It is expected that, whenever a project manager is defining and planing each task for a project, after the *start* month of the task is defined manually by PM then with the duration forecast made by the models the *finish* month is quickly predicted by a

simple calculation of *start* + *duration* - 1 [1]. With that being said, the target variable *task_duration* is calculated similarly to the *wp_duration* seen in equation 5.1. The task duration is then calculated as seen in equation 5.2. On the other hand, the other target variable, task effort, has a parameter in the *task* table to store this value which is called *expected_effort*. Unfortunately, things are not so simple and this parameter was only added to the database recently which means that is empty (*null*) for every record of task in the database. With that being said, some feature engineering is needed to calculate this effort and this is where the table *Allocation* is actually useful for. Since *Allocation* data is associated with the *User* table and there is no way to know beforehand which allocations a task will have, it will only work as an auxiliary table to calculate the task total effort for the existing 54153 tasks. From the table *Allocation* only the parameters detailed below will be considered to aid in the calculation of the effort:

- **task_id** - the task id which the selected allocation belongs to;
- **start** - the month in which the allocation started;
- **finish** - the month in which the allocation ended;
- **value** - the effort in man-months the allocation took per month.

With the *Allocation* data the total effort can be calculated as seen in equation 5.3.

$$task\_duration = finish - start + 1 \tag{5.2}$$

$$task\_effort = value * (finish - start + 1) \tag{5.3}$$

Note that variables *finish* and *start* from both equations 5.2 and 5.3 are different, with these variables in the task_duration equation being taken from the *Task* table while the others from *Allocation*.

Finally, a view was created in the MySQL database to help retrieving the task_effort for each task. More details on the creation of this view can be seen in the Appendix code listing A.1. This view will retrieve all the task ids with the corresponding total effort observed to complete the task.

To wrap everything up, a total of 25 variables from 4 different tables were chosen as initial predictors to predict the 2 target variables as can be seen in table 5.1.

Although there were some parameters that for the data specialists had more weight than others, like the *wp_duration* that one can see the obvious correlation with the *task_duration*, with some features raising big question marks about how they may influence the target variables, an attempt to include the most parameters was made, since when in doubt it is rather preferred to include than to exclude to not lose any possible relationship between the available data.

[1] The minus 1 is needed because the *start* always begins at day 1 of a month while *finish* is related with the last day of that month

	Project	Baseline	WorkPackage	Task
Predictor Variables	type_mask; aicos_funding_rate; equipment_funding_type_mask; backlog_type_mask; revenue_distribution_mask; department_mask; business_field_mask; business_subfield_mask; priority_mask	type_mask; duration; cost_type_mask; stage_mask; status_mask; funding_total; budget_indirect_model_mask; funding_total_costs; aicos_partner_number; overhead_funding_type_mask; overhead_funding_rate	type_mask; number; wp_duration	number; task_type
Target Variables	-	-	-	task_duration; task_effort

Table 5.1: Predictor and Target variables from different tables chosen to feed the ML models

5.2 Data Mining and *task_type* predictor

As discussed in the last section, there was an opportunity to extract knowledge through data mining from the *name* parameter of the *Task* table. As Koehrsen says in his article about how to improve a model performance: "Assuming that there are relationships in the data giving the model more data will allow it to better understand how to map a set of features to a label" [25]. Also Halevy et al. from Google [2] support the idea that most of the times is better to gather more quality data than increasing model complexity [19]. With that in mind, after some discussion with data specialists from the company Fraunhofer [3], a new predictor was settled to capture more information about a specific task. This predictor will be called *task_type* and will start being stored in the database as well. The new predictor will represent the type of task to be handled. The possible values are web, mobile, results, interface, ethics, business, warranty, literature, requirements, specification, software, hardware, firmware, algorithm, network, maintenance, management, tests, pilot, design, data, thesis, supervision, dissemination, others, and any type of combination between these two of these values (e.g. Web;Software). Nevertheless, since it does not exist any record with this information in the database yet, a python script was developed to extract this information from the task *name* string parameter.

The script can be consulted in the appendix code listed in A.1. A brief description of how it was done is described below to better understand the process to acquire each *task_type*.

The script starts by calling the function *label_tasks()* which will open a csv file and save in a list *labels* all the possible values of *task_type* and their correspondent wildcards which are the words expected to be found in a task name that will pair them to a specific value. Then in line 31 it will open another file, *promessa_tasks.csv*, that contains all tasks with their names and then process each task in line 39 with the help of the function *process_task()*. The process of each task is made from line 44 to 48 where each existent wildcard stored in the list *labels* will be compared with each word found in the task name to retrieve its correct task type. Finally, after this the tasks

[2] https://about.google/
[3] https://www.aicos.fraunhofer.pt/

and their correspondent types will be exported to another csv file (*output.csv*), in the *export_tasks()* function called in line 42, to be later consumed and merged together with the rest of the data before feeding it to the models for training.

Since there will be a wide variety of different names in the whole set of tasks not all of them will have a paired type value which means that some records will be empty. This will derive in a problem of having some missing values that will be further analysed and solved. Besides that, since some ML models can not handle pure categorical variables an encoding of *task_type* will be needed to transform this predictor into numerical. Many possible encodings can be applied, but they behave differently depending on the data and the problem being considered. Several encoding methods are explored in either Kumar's and João Moreira's et. al research [27, 43], but some stand out in both papers with the *Target*, *CatBoost* and *LeaveOneOut* encoding, that take into consideration the target value for the transformation, to present the best results for most cases. These three encoders can be easily implemented in Sklearn [4] with the help of the module *category_encoders* [5] and the three classes *TargetEncoder*, *CatBoostEncoder* and *LeaveOneOutEncoder*. However, since they all transform a categorical variable considering the target variable values, then they can be defined using only the training data instead of the whole dataset to avoid target "leakage" problems on the testing samples, as having an input that varies depending on the target could potentially hinder the true accuracy of a model. Other encoders will also be explored such as the label, binary and one hot encoder.

In the end, as expected by Koehrsen [25], getting more data will prove to be very useful and enhance successfully the model's performance.

5.3 Pre-processing

From 25 possible predictors and 2 targets seen in table 5.1 to be fed the models, only one predictor had missing values, the *task_type*. In 54153 tasks only 36973 had a value for this parameter, resulting in 17180 missing values. To handle this, the standard method used was just to impute these with the generic value "Others", which in this case makes sense regarding that if no match value was found for these tasks then they should be considered as "Others". Nonetheless, other techniques such as removing the records, imputing them with the median, mean, and using the k-Nearest Neighbors with the sklearn class *KNNImputer* from the *impute* [6] module were used to achieve the best result. To handle outliers the unbiased interquartile method discussed in section 3.5 was used.

More tests will be made on the 25 possible predictors, such as correlation studies, to do a feature reduction that could achieve the best input set for each target model.

A skewed distribution is a non-symmetric distribution of the data. If the predictor distribution is roughly symmetric, the skewness values will be closer to zero, if more right skewed then the

[4] https://scikit-learn.org/stable/
[5] https://contrib.scikit-learn.org/category_encoders/
[6] https://scikit-learn.org/stable/modules/impute.html#impute

values are larger, and if left skewed the values will be negative. ML models normally tend to perform better on unskewed data (closer to normal distribution) [26, p. 31–33]. The skewness can be easily calculated in python with the pandas dataframe function *skew()* [7]. Another possible way to quickly detect the skewness is through data histograms. To solve this issue, transformation methods like the Box-Cox or Yeo-Johnson can be applied to approximate any distribution to a normal distribution [26] as seen in figure 5.2. Beside the skewness problem, different variable scales can also affect negatively some ML models such as SVM and neural nets. To give every input the same chance to influence the target variable in the same manner, techniques such as standardization (centering and scaling) and normalization should be applied to increase numerical stability between inputs. These methods can be applied in sklearn with the help of the class *StandardScaler*[8] (standardization) and *MinMaxScaler*[9] (normalization).

Finally, due to the fact that between the 25 predictors there might be some that are not able to help the model predict and only increase model complexity or, even worse, are just adding noise which decreases performance, a simple automatic feature selection will be made with the help of the Random Forest (other tree-based models could have been used).

5.4 Regression Models and Parameter Tuning

After the data preparation and pre-processing, a stack of 8 different regression models was developed, tuned, and benchmarked against each other to find the best models and parameter settings to be applied in the final average ensemble proposed in section 4.2.

Each model was then implemented following the below algorithm 1.

Algorithm 1 Delivery Forecast Model Implementation

```
1: function MODELX(df_tasks,encoding)
2:     test_size ← 0.25
3:     cv ← 5
4:     train, val ← train_test_split(df_tasks, test_size)
5:     param_grid ← initialize with specific model tuning parameters
6:     model ← parameterTuning(model,param_grid,metric,cv,val)
7:     scores ← cv_score(model, train, cv, encoding)
8:     return scores
```

[7] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.skew.html

[8] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html?highlight=standardscaler#sklearn.preprocessing.StandardScaler

[9] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html?highlight=minmax#sklearn.preprocessing.MinMaxScaler

[10] https://scikit-learn.org/stable/modules/preprocessing.html#mapping-to-a-gaussian-distribution

In each model, the project control dataset was provided as a pandas dataframe[11] (*df_tasks*) and an encoding function, to transform the categorical variables into numerical, was also provided through the function parameters. Then, at line 4, the data is split into 75% for training and 25% for parameter tuning. After defining the settings for each method as established in the table 5.2, the parameter tuning is done with a 5-fold cross-validation, higher than 3 to increase confidence but less than 10 to reduce processing time, using the sklearn class GridSearchCV [12]. As observed in line 6, a *metric* is used to evaluate each method settings performance and since this is a regression problem the popular RMSE was used. Finally, with the best tuned parameters the model is trained in the rest of the 75% of the data, using again a 5-fold cross-validation strategy and the performance scores are returned.

To define the possible values for each model settings to realize the parameter tuning, the sklearn documentation [13] was consulted as well as other references like the Max Khun's predictive model book [21] [26]. In the following table 5.2 each model parameters are described as their possible values to find the best possible settings. To conclude, the final average ensemble proposed beforehand is then implemented with the best performing models and compared against them.

[11] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[12] https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html

[13] Documentation for ensemble models: https://scikit-learn.org/stable/modules/ensemble.html

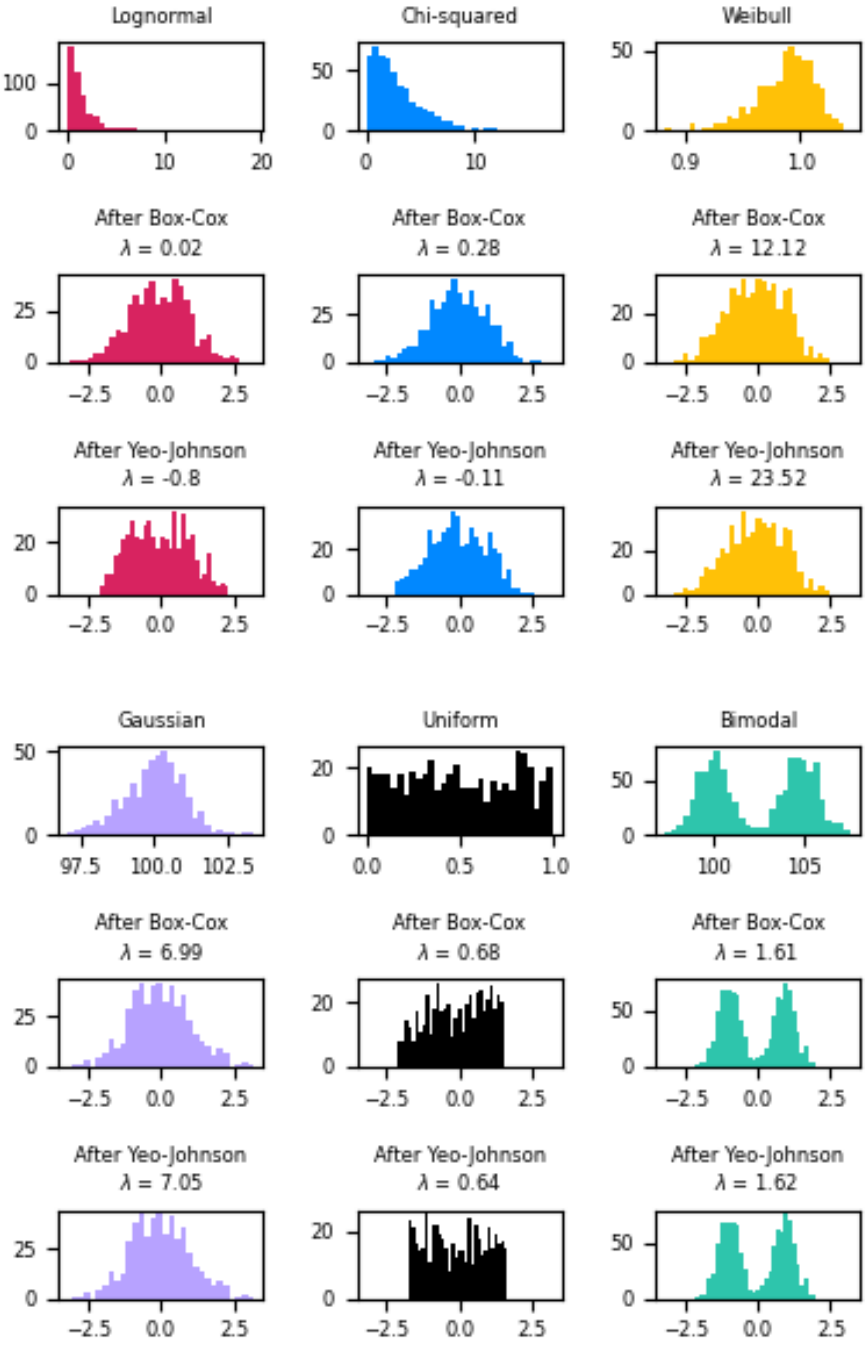

Figure 5.2: Examples of Box-Cox and Yeo-Johnson applied to various probability distributions with the *PowerTransformer* class taken from the sklearn documentation [10]

Model	Parameters	Values	Description
RF & extraTrees	n_estimators	100-1000	Nr. of trees in the forest
	max_features	'sqrt','log2','auto'	Nr. of features to consider when looking for the best split. Can be the sqrt/log(n_predictors) or auto which includes all predictors
	max_depth	20-100 & None	The maximum depth of the tree. None means that it is expanded until all leaves are pure
gbm	n_estimators	100-1000	Nr. of boosting stages to perform
	learning_rate	0.0001-0.2	Shrinks contribution of each tree by learning_rate
	max_features	'sqrt','log2','auto'	Nr. of features to consider when looking for the best split. Values equal to RF/extraTrees.
	max_depth	20-100 & None	The maximum depth of the tree. None here is 3 by default
XGBoost	learning_rate	0.0001-0.2	Shrinks the contribution of each tree by learning_rate
	min_child_weight	1,4,7	Minimum sum of weights of all observations required in a child best split. Values equal to RF/extraTrees.
	max_depth	20-100 & None	The maximum depth of the tree. None here is 6 by default
	subsample	0.5-1	Subsample ratio of the training instances
	gamma	0-0.4	Min loss reduction required to make a split
	colsample_bytree	0.5-1	Subsample ratio of columns for each tree
Lasso	alpha	0-1	Constant that multiplies the L1 term
KNN	n_neighbors	3-10	Number of neighbors
SVR	C	0.001-1000	Regularization parameter
	gamma	0.001-1	Kernel coefficient
	kernel	'poly','rbf','sig'	Specifies kernel type used
MLP	activation	'relu', 'tanh', 'logistic'	Activation function for the hidden layer
	hidden_layer_sizes	P([50,100,150],1-3)	Nr. of neurons in the ith hidden layer. Values are all permutations in a list with 1-3 hidden layers & 50,100 or 150 neurons
	alpha	0.0001-0.1	L2 penalty (regularization term)

Table 5.2: Parameters to be tuned for each of the 8 ML algorithms

Chapter 6

Results and Discussion

In this chapter, every result will be presented and discussed from the pre-processing to the implementation of the final heterogeneous ensemble. In the end, research questions proposed in section 1.4 will be discussed such as validity threats present in this study.

6.1 Preparing data for models and Establishing Baseline

Before starting the model's implementation, data must be cleaned and prepared to be fed the models in best way possible. As seen in table 5.1 of section 5.1, a total of 25 inputs are to be used to predict the task effort and duration. However, first it will only be considered 24, as the extracted *task_type* variable will only be explored further in this section. Notice that, these 24 variables were added solely based on data expert opinions. Not all of these predictors are expected to contribute the same, moreover there might be some that are just adding noise and thus must be found and removed. With this in mind, a baseline model with automatic feature selection will be implemented and tested in different scenarios with varying input sets to achieve the best combination in either accuracy and processing time before implementing the remaining stack of ML models. To start as a baseline the random forest model was picked from the stack. This has been a versatile and very used model in the machine learning community as it provides automatic feature selection and does not need much pre-processing while still providing good performance.

Within the 24 inputs to be used no missing values were found. In order to reduce most of noisy data a shallow analysis was done in the database. Most of tasks that presented extreme effort values (>100 man-months) were identified as errors since they were all related with management and belonged to projects that did not pass the proposal phase. Hence, all these samples were considered as data quality issues and thus removed from the dataset. A deeper analysis should be done in the future to further identify and remove more possible hidden noise in data. Regarding outliers, these will be explored and their effect tested further in more detail.

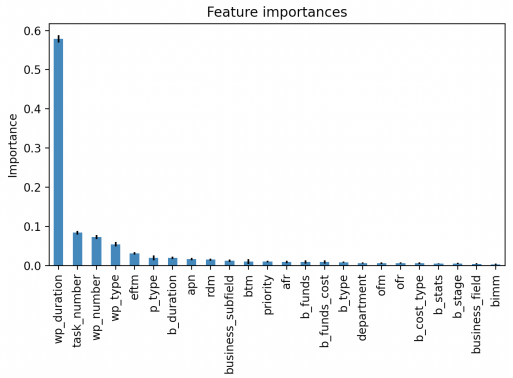

Figure 6.1: Feature importance by the RF with the initial 24 inputs

```
Mean Absolute Error: 1.02
Root Mean Squared Error: 2.41
R2: 0.94
Mean Magnitude of Relative Error: 0.25
Percentage Relative Error Deviation (PRED(0.3)): 0.8
Mean Train Time: 7.64
Variable: wp_duration          Importance: 0.58
Variable: task_number          Importance: 0.084
Variable: wp_number            Importance: 0.073
Variable: wp_type              Importance: 0.054
Variable: eftm                 Importance: 0.031
Variable: p_type               Importance: 0.02
Variable: b_duration           Importance: 0.018
Variable: apn                  Importance: 0.016
Variable: rdm                  Importance: 0.015
Variable: business_subfield    Importance: 0.012
Variable: priority             Importance: 0.011
Variable: btm                  Importance: 0.01
Variable: b_funds              Importance: 0.01
Variable: afr                  Importance: 0.009
Variable: b_funds_cost         Importance: 0.008
Variable: b_type               Importance: 0.008
Variable: ofr                  Importance: 0.007
Variable: department           Importance: 0.006
Variable: b_cost_type          Importance: 0.006
Variable: business_field       Importance: 0.005
Variable: b_stats              Importance: 0.005
Variable: ofm                  Importance: 0.005
Variable: b_stage              Importance: 0.005
Variable: bimm                 Importance: 0.003
```

Figure 6.2: Performance and detailed feature selection of RF on the first experience with 24 inputs

A first iteration forecasting task duration with the baseline model (random forest) having the default settings and using the whole 24 inputs was done. A 5-fold cross validation strategy was used to avoid overfitting problems and enhance confidence in results. In the left figure 6.1, we can observe the feature selection done by RF on these 24 inputs and on second figure 6.2 the RF performance and a list of importance for each feature in this forecast.

From first experience and from what can be seen on figure 6.2, a pretty decent performance was achieved for the random forest with a MAE of 1.02 months, a RMSE of 2.41 months and a R^2 of 0.94. Besides that, automatic feature selection from the RF shows on figure 6.1 that the most outstanding features for this forecast were work package (WP) duration by quite a margin (0.58), task and WP number (together had $\sim$0.16 importance), and WP type had an importance of 0.05. The other 20 variables only explained for about 0.2 of the result with some not even having 0.01 of importance. This means that some of these 20 variables are not helping the final forecast with some probably only adding noise and decreasing the model performance. One important thing to notice from the feature importance graph is the tiny black lines in the middle of each blue bar that indicates the standard deviation of each importance. Since most of these importance values have little standard deviation we can trust these values do not variate much from the value defined, giving an increased trust on conclusions made.

From the previous conclusion, a new test was run with the same random forest but only counting all features that accumulated together an importance of 0.95 as observed in figure 6.3. This resulted in a model that counted only with the highest importance inputs until, and including, the *b_funds_cost* predictor. Which means it only trained with the 15 predictors with higher importance (all predictors left of blue vertical line in fig. 6.3). This new model actually improved the performance with a MAE of 0.92, a RMSE of 2.33 and the R^2 staying the same at 0.94. The importance of each predictor stayed very similar compared with last model and mean train time improved from 7.64 to 4.73 seconds, which was expected as the model complexity decreased with the input set being reduced to 15 predictors.

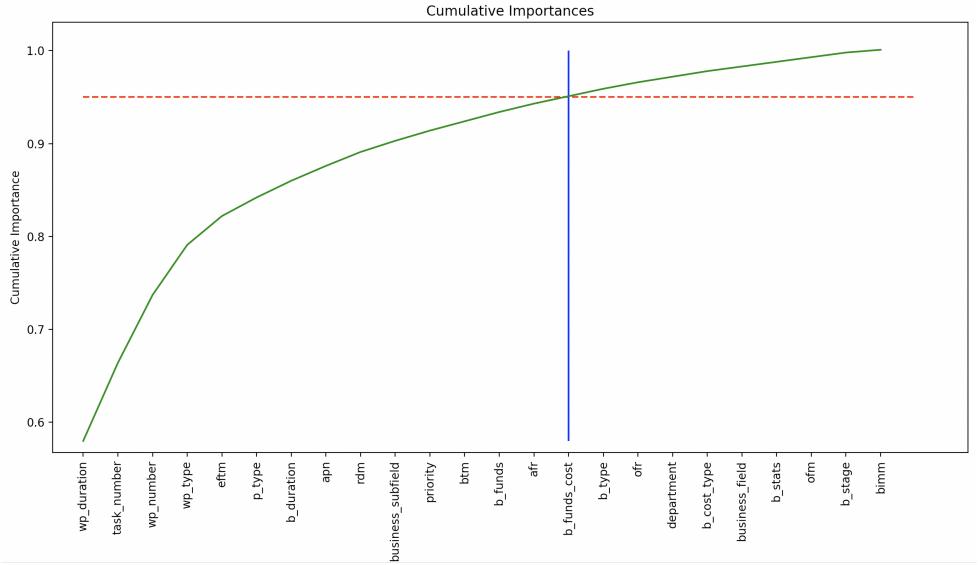

Figure 6.3: Cumulative Feature Importance for RF duration forecasting with initial 24 inputs

However, from the 9 inputs removed there are no certainties that all 9 were only adding noise so a few more tests with these left out inputs were done. It seems that the guess was right and the results actually improved when variables *department* and *b_cost_type* were added back again. With the remaining 17 predictors the model was able to improve the performance to a MAE of 0.82, a RMSE of 2.2 and a R^2 of 0.95. The overall improvement in performance confirms the previous affirmation that some predictors were just adding noise to the model or simply adding unnecessary complexity.

Nonetheless, noisy predictors are not the only thing that can decrease model accuracy. Multicollinearity within predictors can also become harmful and might have to be dealt with. This is when predictors have a high correlation between each other which might increase model variance and decrease its performance. However, this is not always the case, as stated by Frost on his book [16], since most times it does not influence the predictions. Nevertheless, even if it does not affect model performance it still increases complexity, turning the results and coefficients harder to interpret and, consequently, enhance the processing time unnecessarily [16]. Furthermore, a correlation study was made between variables which resulted in the correlation matrix observed in figure 6.4. With this matrix some high correlated variables, with more than ± 0.7 correlation, can be easily identified. From here, it is easy to see that *afr*,*btm* and *rdm* are the three correlated between each other. Besides that, the *b_funds* and *b_funds_cost* are also strongly correlated with 0.87 correlation, and lastly the *b_duration* and *wp_duration* with 0.77. So, before removing and testing the model without these variables, an analysis was made with predictors against the target variables through scatter plots as seen in figure 6.5. It is now pretty clear from the plots that these variables are indeed high correlated since the aforementioned identified predictors also present very similar plots between each other when run against the target variables.

Starting with *afr*,*btm* and *rdm*, it has been seen in RF feature selection that the *rdm* is the one

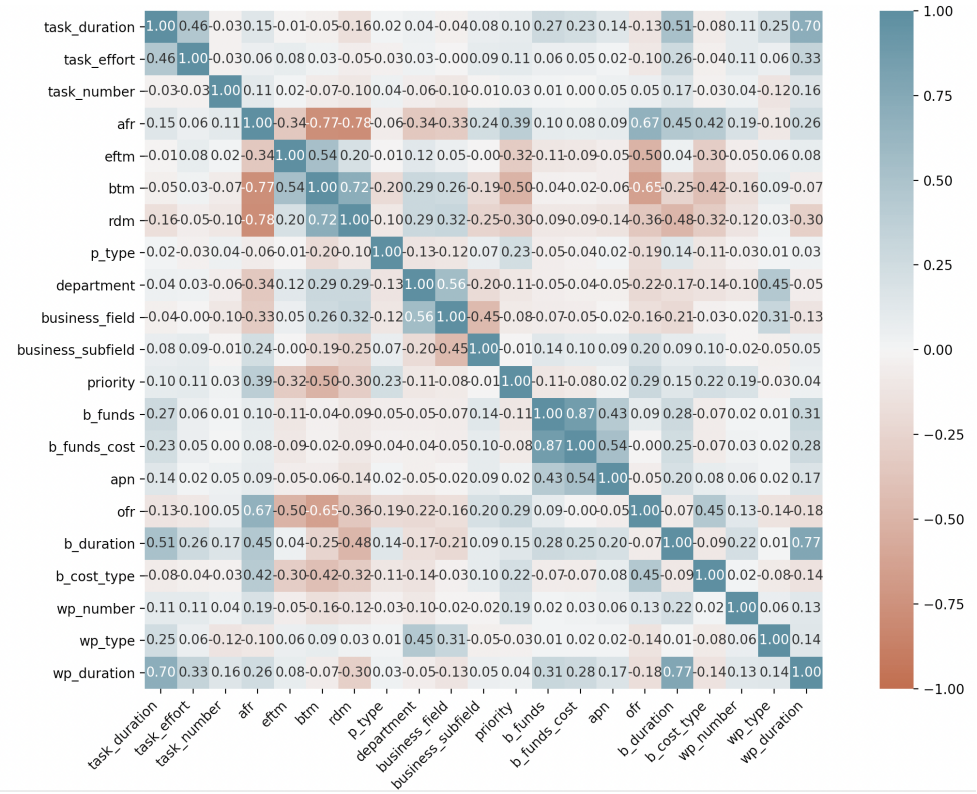

Figure 6.4: Pearson correlation matrix for the remaining inputs selected from the RF for task effort and duration forecasting

with the highest importance (0.15). It is also clear from its scatter plot that it is very similar with the *btm* plot with which shares a 0.72 correlation. Consequently, the model was rerun without this *btm* predictor. As expected, results did not change. Again the *rdm* and *afr* also share a high correlation of -0.78. Notice now that this correlation is negative, this does not mean they are not correlated it just means that when the *rdm* increases the *afr* decreases similarly and vice-versa. If we compare the 2 scatter plots of these variables it is not so obvious that they are similar. This is because, first, the scale of both variables is not the same and, second and most importantly, since the correlation is negative the graphs are supposed to mirror each other which is actually what it seems to happen. Again the model was rerun without the *afr* now and the performance stayed the same. Other experiments were made removing the *rdm* instead and keeping one of the other 2 predictors. Again, since the *rdm* had a higher importance for the RF in prior runs, which means it was a more significant predictor for the model, as expected performance decreased a bit. Even though this was not a big decrease in performance, in a commercial environment each decrease might be meaningful in the long run for the company. Following the same logic, now for *b_funds* and *b_funds_cost* predictor, where both plots turn this correlation very obvious, given their similarity *b_funds_cost* was removed. And with no surprise, model performance stayed the exact same. At last, comparing plots of remaining *b_duration* and *wp_duration* predictors it is very clear the similarity from each. Since *wp_duration* had an enormous difference in importance,

as observed in fig. 6.1, an experiment was done removing *b_duration* predictor. Although it was expected that performance stayed the same or close, it actually decreased accuracy by a piece. Without *b_duration*, the model performed a bit worst. Indeed this decrease was not very significant, but it connotes that *b_duration* can not be completely replaced by *wp_duration*, and it can also help explain some variance in model results. With that being said, it was decided to only cut the following predictors: *btm*, *afr* and *b_funds_cost*. The final RF model with remaining 14 predictors obtained a **0.82 MAE**, **2.2 RMSE** and a R^2 **of 0.95**, with a record train time of 4.57 seconds. This just confirms that most of collinear predictors were not making any difference for final prediction but were instead increasing model complexity, hence could be removed.

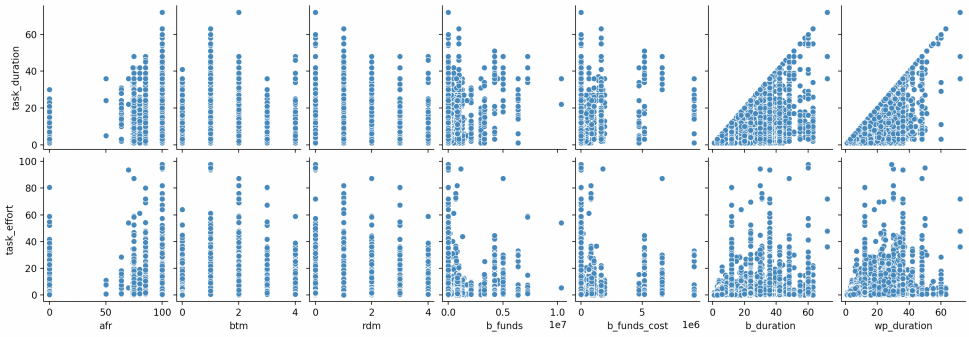

Figure 6.5: Scatter plots of the high correlated predictors against the target variables

Now the same process must be replicated for our baseline model but forecasting the total effort instead. The first run of RF for effort came out with a performance of 0.84 man-months MAE, 2.4 man-months RMSE and R^2 at 0.82. It is a good starting point, but before improving the model with the aforementioned reduction of predictors another possibilities were explored and found. As Koehrsen and Halevy et al. refer in their articles [25, 19], the best way to improve a model performance is to feed it with more new quality data. After some discussion with data experts, it was clear that every PM, when defining a new project and its tasks, always predicts first the task expected duration and only then the effort is forecast. With that in mind, the model was rerun with the target variable *task_duration* as a new predictor for the effort forecast.

As seen in figure 6.7, the new model fulfilled expectations and improved performance from a 0.84 MAE to 0.73 (15% improvement) and from a RMSE of 2.4 to 2.18 (10% improvement). Consequently, adding the *task_duration* to input set proved to be successful. Note also that previous RF model (without *task_duration*) gave its top importance with 0.19 to *wp_duration*, while this new model crowned the *task_duration* as the new top predictor with 0.26 importance and with *wp_duration* going down to 9^{th} position with a mere 0.04 importance. This is explained by the fact that both variables are very similar as they both represent the duration of either a work package or a task. Also, since *task_duration* is closer to the tasks in the database hierarchy it makes sense that the *wp_duration* is "swallowed" by task duration influence. The fact that correlation between effort and both predictors is higher for *task_duration* (0.46 > 0.33 fig.6.4) also supports the previous sentence. An interesting observation from feature selection graph in fig.6.6 is that the deviation

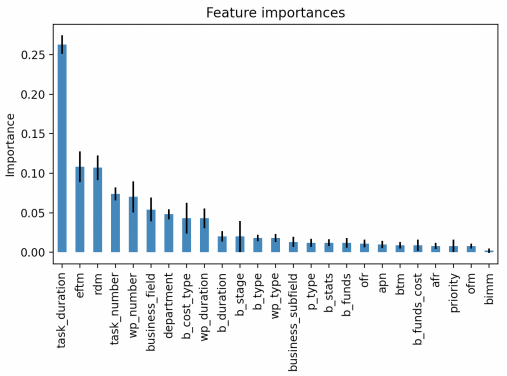

Figure 6.6: Feature importance by the RF with the initial 24 inputs plus the *task_duration* for the effort forecast

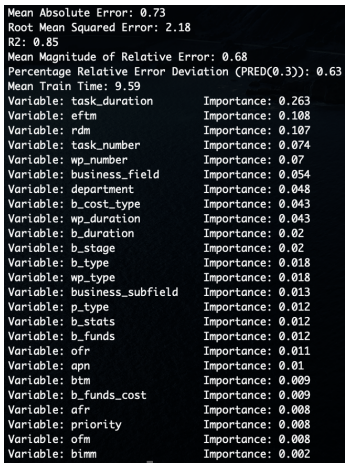

Figure 6.7: Performance and detailed feature selection of RF on with the 24 initial plus *task_duration* inputs for the effort forecast

of importance values is not so stable when compared to the one in fig.6.1 for the task duration problem. In fact, there is some predictors with a considerable importance that might variate so much that in some cases reach a 0 importance for the forecast. The most obvious predictor of such instability is *b_stage*. A new model was then rerun several times without this predictor and it actually improved the performance to 0.72 MAE and 2.17 RMSE. This just proves the importance of analysing the standard deviation of each importance value, since it can aid in finding not so obvious noisy predictors.

After the removal of *b_stage*, a new test was run with the same random forest but only counting the features that accumulated together an importance of 0.95 as observed in figure 6.8. This time the performance of the RF, with the 17 predictors on the left of the blue vertical line, remained unchanged. More experiences were made on remaining 17 predictors to guarantee that no predictor was still adding noise and it was found that by removing the *b_stats* and *b_type* the model improved from a 0.72 MAE to an impressive 0.62, 2.17 RMSE to 2.04 and R^2 from 0.84 to 0.87. This is why it is always important to test different subsets of the data since even if the predictor is able to reduce some impurity within each tree, hence increase its importance, this does not necessarily mean it will help in final prediction. Nonetheless, the same conclusion as before was reached, as some predictors just add noise to the model or simply add unnecessary complexity and must be removed.

Finally, considering the multicollinearity issues, an analysis is done again with help of the correlation matrix visible in figure 6.4. From there it is possible to see a strong correlation between *b_duration* and *wp_duration*(0.77), and another between the *wp_duration* and *task_duration*(0.7). No other high correlations (bigger than ± 0.7) were found for the task effort predictors. Similar to what was done before, a new battery of tests was run, removing the *b_duration* or *wp_duration* and both got slightly worst results. This means that none of these variables can fully replace the others and hence none was removed.

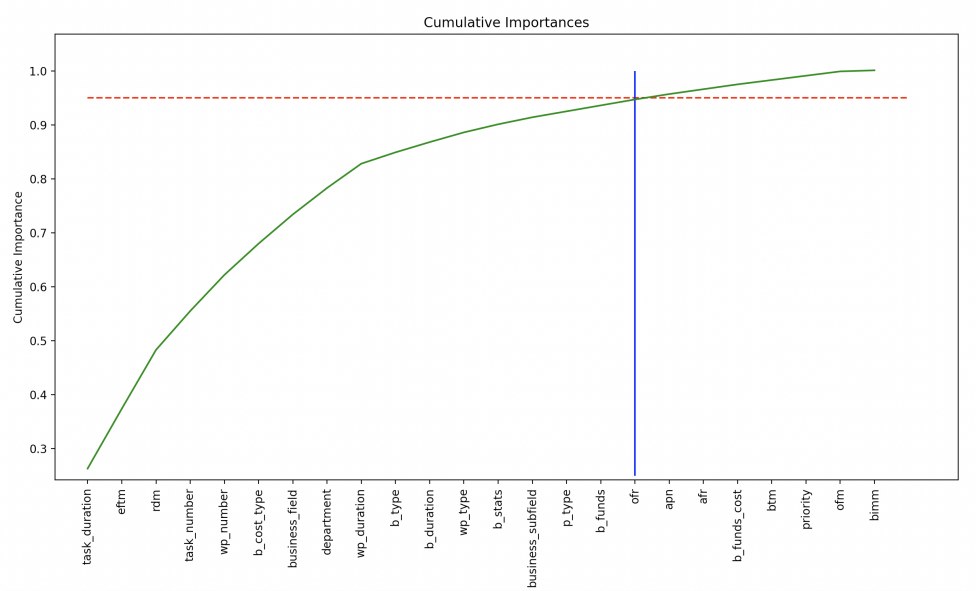

Figure 6.8: Cumulative Feature Importance for RF effort forecast

6.1.1 *Task_type* predictor

The selected data for the baseline model defined earlier showed some promising results for both effort and duration forecast. And, according to Koehrsen [25], the best way to improve a model is by adding more quality data, which has already been proven by using the target *task_duration* for effort forecasting. Following the same idea, new ways to find quality data will be explored further.

As discussed in section 5.2 of the methodology, an opportunity arose to extract new knowledge through data mining from the *name* parameter of the *Task* table. The newly created predictor was called *task_type*. From the 54153 tasks only 36973 had a value for this parameter, resulting in 17180 missing values. To handle this, a variety of techniques were used such as removing records, imputing these with the generic value "Others", imputing with the median, mean, and using k-Nearest Neighbors. Also, since this is a categorical predictor and most ML methods require it to be numerical, the standard target encoding technique was used. After running a considerable amount of tests, target encoding was considered the best overall with the cat boost and leave one out encodings having a slightly worse performance. Other simpler methods like label and one hot encoding were also experimented but did not reach the before mentioned performance. More experiences for both the effort and duration models were done with this new predictor and various techniques to handle its missing values. Results can be seen in the following table 6.1.

Looking at the results it is possible to observe that from all techniques used to remove or impute the missing values they all had similar results but the one that obtained best results in either models was the imputing with the generic value "Others". This result makes sense, as most values that were not attributed a value are supposed to be considered generic tasks. Comparing previous results against models without the *task_type* predictor, it is possible to see that Koehrsen [25] was right again since these last models improved considerably. The *task_type* predictor turned to be a

Model	Metric	Remove rec.	Mean	Median	K-NN	Generic 'Others'
Task_duration	MAE	0.7	0.72	0.72	0.72	0.67
	RMSE	2.0	2.01	2.02	2.01	1.95
	R^2	0.96	0.96	0.95	0.96	0.96
Task_effort	MAE	0.57	0.58	0.57	0.59	0.57
	RMSE	1.98	1.94	1.92	1.92	1.92
	R^2	0.85	0.88	0.88	0.88	0.88

Table 6.1: Duration and Effort baseline RF model results with the *task_type* predictor and several techniques to handle the missing values

successful adding for both models as the importance for task duration model was 0.15 (becoming second best before *wp_duration* and for the effort was 0.8 (becoming the 4^{th}), explaining why models improved so much from the previous ones. The next section will now explore the influence of outliers and how to handle it.

6.1.2 Effect of outliers in the baseline model

It is known that outliers can dramatically affect a models performance. However, outliers are not always noise. Their existence might actually mean that a new group of extreme samples is starting to appear and must be explored. Or they could just be values that even though only happen rarely they do happen and are serious cases that must be considered. To analyse the outliers for our dataset a several box plots were made for every predictor selected in the last section and can be seen in figure 6.9.

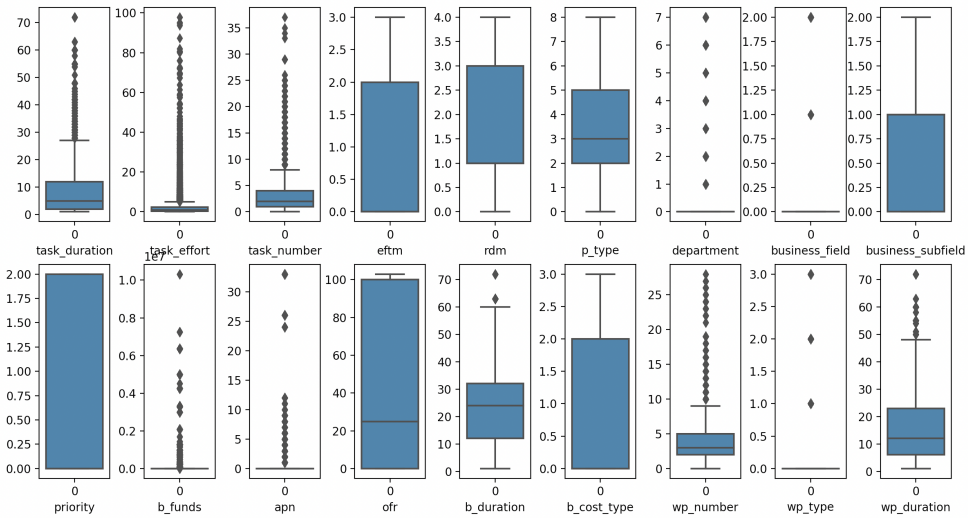

Figure 6.9: Box plots of the several selected predictors from the RF baseline model to detect outliers

From the figure 6.9 results, it is clear that there are some outliers and these appear in bigger number on the numeric variables. After a review of these data points with data specialists, some outliers were removed, since they were considered noise, others were considered valid points and the rest needed a deeper analyse which is not always possible in business environment due to tight schedules and the large amount of data to cover. Nevertheless, just to test the effect of the outliers, a test was made where every outlier (except *b_funds* and *apn*) in the numeric variables was removed from the training sets through an interquartile method. If we removed the outliers of categorical variables such as the *department* this would be limited to a single value since most values rely on 0 and hence this predictor would become useless. The results show that the new task duration model even got worse with a 0.87 MAE and a RMSE of 2.64. As we can notice here, the RMSE is especially high and this means that bigger errors are happening. This was an expected outcome as by removing outliers, we are also removing the model ability to predict extreme values that might actually be, and probably are, valid data points. Hence, if it is not obvious that the outliers are errors/noise than it is rather preferable to keep these instead of removing.

To conclude, the data preparation and cleaning turned out to improve the performance of our baseline RF model (with default settings). This will now serve as a baseline for next sections where a stack of different models, with the best input sets identified until now, will be trained and compared with these results. Parameter tuning will also be done to try to improve even more the models performance. In the end, a discussion will be made and the best models will be used for the final proposed ensemble.

6.2 Duration and Effort Forecast

Before starting the implementation of the stack of models, Khun et al. refer in their book [26] that data skewness can negatively affect some models performance. To check if our continuous variables follow a normal distribution or if they are somewhat skewed, an analyse is made with the help of some histograms as seen in figure 6.10. From here we can observe that some variables look a bit more normal distributed like the *b_duration* and *wp_duration*,the *ofr* looks like it has something closer to a trimodal distribution, and the rest are more or less right skewed. With *task_effort*, *b_funds* and *apn* being more notably right skewed variables. To confirm this the skew values for these variables were calculated the same way as in section 3.2, page 31 of Khun et al. book [26]. The same source says:

> "If the predictor distribution is roughly symmetric, the skewness values will be close to zero. As the distribution becomes more right skewed, the skewness statistic becomes larger. Similarly, as the distribution becomes more left skewed, the value becomes negative."

And to no surprise the *b_duration*,*ofr* and *wp_duration* got a skew of 0.25, 0.66 and 0.90 respectively, confirming a more symmetric distribution. Again, *task_duration*, *wp_number* and *task_number* that were a bit more right skewed got values ranging from 1.97 to 2.9. Lastly, the

more extreme right skewed remaining variables obtained values from 5.9 to 9.2, confirming the graph interpretation. To try to fix this issue, tests will be made with the skewed data having a Box-Cox and Yeo-Johnson transformation.

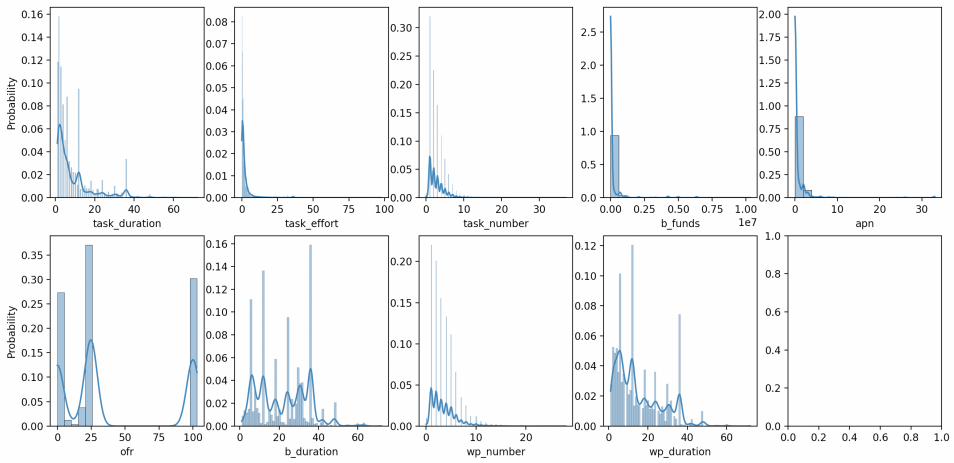

Figure 6.10: Continuous variables distributions

Besides skewness, data scaling can also affect some models performance. As seen in section 3.4.7, table 3.1, models like neural nets, lasso and others benefit from centering and scaling techniques. These models are usually susceptible to different scale predictors, and as we can observe from fig. 6.10 there are continuous variables that might only have a maximum value around 30 but others like *b_funds* can reach numbers with 8 digits. Again, tests will be made with the data suffering centering and scaling or normalization transformations to try and solve this issue.

One more thing to do before implementing the proposed stack is to quickly plot every dependent and independent variable to analyse their relationship and what to expect from the models. This can be seen in fig. 6.11.

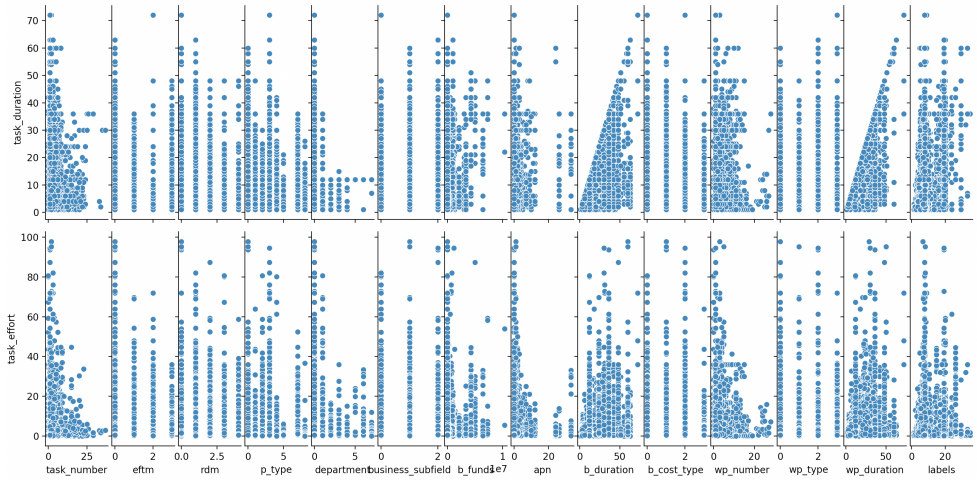

Figure 6.11: Scatter plots with the relations of dependent and independent variables [1]

From the above plots we can clearly see that most relationships, if not all, look non-linear. The closest to a somewhat linear relationship could be the plots for *task_number* and *wp_number*, but even those look more curvillinear than a straight linear relationship. Either way, from these plots we can expect the only linear model picked (Lasso) to not have the same performance as others in the stack. Some plots for variables with less importance, as indicated by the RF baseline model seen in last section, were excluded to ease visualization, but even with those the conclusions would be the same.

Recalling what was done in the previous section 6.1, a baseline RF model was established to serve as a guideline for implementing the stack. From there the data was cleaned and prepared to forecast both target variables. The following figures 6.12 and 6.13 show the best input sets for both targets and their importance values calculated by the baseline model. These input sets will be picked to feed the whole stack.

```
Variable: wp_duration          Importance: 0.376
Variable: b_duration           Importance: 0.152
Variable: task_type            Importance: 0.149
Variable: task_number          Importance: 0.062
Variable: wp_number            Importance: 0.058
Variable: wp_type              Importance: 0.041
Variable: b_funds              Importance: 0.03
Variable: p_type               Importance: 0.023
Variable: eftm                 Importance: 0.022
Variable: rdm                  Importance: 0.02
Variable: apn                  Importance: 0.018
Variable: business_subfield    Importance: 0.014
Variable: priority             Importance: 0.012
Variable: b_cost_type          Importance: 0.012
Variable: department           Importance: 0.01
```

Figure 6.12: Best predictors set selected by the RF for. *task_duration* and their correspondence importance

```
Variable: task_duration        Importance: 0.2
Variable: wp_duration          Importance: 0.094
Variable: wp_number            Importance: 0.093
Variable: eftm                 Importance: 0.092
Variable: task_type            Importance: 0.081
Variable: task_number          Importance: 0.068
Variable: b_duration           Importance: 0.06
Variable: b_cost_type          Importance: 0.059
Variable: business_subfield    Importance: 0.055
Variable: rdm                  Importance: 0.052
Variable: department           Importance: 0.034
Variable: ofr                  Importance: 0.034
Variable: business_field       Importance: 0.021
Variable: p_type               Importance: 0.02
Variable: b_funds              Importance: 0.02
Variable: wp_type              Importance: 0.017
```

Figure 6.13: Best predictors set selected by the RF for *task_effort* and their correspondence importance

These baseline models proved to be pretty efficient, and techniques like data mining to extract more information from the data available just played a big part on this success. It is true that getting good quality data can make or break a model and, just as Koehrsen referred [25], it is the best way to improve performance. But, it is known that, this is not the only way to improve performance. Something that has not been done in the last section for the baseline model and that can optimize its accuracy is model tuning. In the following table 6.2, we can observe every stack model optimal settings for both targets. All these models were run with a 5-fold cross validation strategy and tested through out all values as proposed in table 5.2 of the methodology section (5.4). The scoring function (or evaluation metric) used to optimize this settings was the RMSE, due to the fact that this metric has the advantage of picking the small portions of large errors while other metrics like MAE only do the mean of residuals and hence a small group of large errors will not affect this metric as much as it does on RMSE. Besides, for the problem at hand it is way more affordable for Fraunhofer to have it optimized in favor of RMSE since it is preferable to have more small errors than a bigger portion of large errors that could be more harmful for the company in the long way.

[1]The last plot from fig. 6.11 was called 'labels' but is the equivalent to 'task_type' predictor

Model	Task_duration	Task_effort
RF	n_estimators: 600 max_features: 'sqrt' max_depth: 60	n_estimators: 800 max_features:'log2' max_depth: 100
extraTrees	n_estimators: 600 max_features: 'auto' max_depth: 80	n_estimators: 100 max_features:'auto' max_depth: 60
gbm	n_estimators: 900 learning_rate: 0.01 max_features: 'sqrt' max_depth: 20	n_estimators: 800 learning_rate: 0.01 max_features: 'sqrt' max_depth: 20
XGBoost	learning_rate: 0.1 min_child_weight: 1 max_depth: 40 subsample: 1.0 gamma: 0.0 colsample_bytree: 0.7	learning_rate: 0.1 min_child_weight: 1 max_depth: 60 subsample: 1.0 gamma: 0.0 colsample_bytree: 0.7
Lasso	alpha: 0.01	alpha: 0.1
KNN	n_neighbors: 3	n_neighbors: 3
SVR	C: 100 gamma: 0.1 kernel: 'rbf'	C: 100 gamma: 0.1 kernel: 'rbf'
MLP	activation:'tanh' hidden_layer_sizes: (50, 100, 150) alpha: 0.001	activation:'tanh' hidden_layer_sizes: (150, 50, 100) alpha: 0.0001

Table 6.2: Optimal parameters tuned for each of the 8 ML algorithms

After tuning, the results of every model, using a target encoder to convert the *task_type* predictor into numerical, for five different test cases: not considering any type of pre-processing, using centering and scaling (CS), using normalization, using Box-Cox with CS and using Yeo-Johnson with CS; will be demonstrated in table 6.3.

Model		**Task_duration**					**Task_effort**				
		No-Pre	CS	NZ	BC	YJ	No-Pre	CS	NZ	BC	YJ
RF	MAE:	**0.69**	0.69	0.69	0.69	0.69	**0.55**	0.55	0.55	0.55	0.55
	RMSE:	**1.9**	1.9	1.9	1.9	1.9	**1.86**	1.86	1.86	1.86	1.86
	R^2:	**0.96**	0.96	0.96	0.96	0.96	**0.89**	0.89	0.89	0.89	0.89
	MMRE:	**0.17**	0.17	0.17	0.17	0.17	**0.5**	0.5	0.5	0.5	0.5
	PRED(0.3):	**0.87**	0.87	0.87	0.87	0.87	**0.72**	0.72	0.72	0.72	0.72
extraTrees	MAE:	**0.46**	0.46	0.46	0.46	0.46	**0.45**	0.45	0.46	0.45	0.46
	RMSE:	**1.79**	1.79	1.79	1.79	1.78	**1.81**	1.81	1.8	1.81	1.81
	R^2:	**0.96**	0.96	0.96	0.96	0.97	**0.9**	0.9	0.9	0.9	0.9
	MMRE:	**0.11**	0.11	0.11	0.11	0.11	**0.36**	0.36	0.36	0.36	0.36
	PRED(0.3):	**0.91**	0.91	0.91	0.91	0.91	**0.8**	0.8	0.8	0.8	0.8
gbm	MAE:	**0.48**	0.48	0.48	0.48	0.48	**0.46**	0.46	0.46	0.46	0.46
	RMSE:	**1.78**	1.78	1.78	1.79	1.79	**1.77**	1.77	1.79	1.78	1.79
	R^2:	**0.96**	0.96	0.96	0.96	0.96	**0.9**	0.9	0.9	0.9	0.9
	MMRE:	**0.12**	0.12	0.12	0.11	0.11	**0.37**	0.37	0.37	0.37	0.37
	PRED(0.3):	**0.91**	0.91	0.91	0.91	0.91	**0.8**	0.8	0.8	0.8	0.8
XGBoost	MAE:	**0.51**	0.51	0.51	0.51	0.51	**0.46**	0.46	0.46	0.46	0.46
	RMSE:	**1.78**	1.78	1.78	1.78	1.79	**1.78**	1.78	1.78	1.79	1.78
	R^2:	**0.96**	0.96	0.96	0.96	0.96	**0.9**	0.9	0.9	0.9	0.9
	MMRE:	**0.12**	0.12	0.12	0.12	0.12	**0.39**	0.39	0.39	0.39	0.39
	PRED(0.3):	**0.9**	0.9	0.9	0.9	0.9	**0.8**	0.8	0.8	0.8	0.8
Lasso	MAE:	**4.34**	4.34	4.34	4.35	4.35	**2.31**	2.3	2.27	2.31	2.31
	RMSE:	**6.06**	6.06	6.06	6.05	6.05	**4.84**	4.85	4.92	4.86	4.85
	R^2:	**0.6**	0.6	0.6	0.6	0.6	**0.26**	0.26	0.24	0.26	0.26
	MMRE:	**1.06**	1.06	1.06	1.06	1.06	**2.96**	2.94	2.75	2.96	2.96
	PRED(0.3):	**0.32**	0.32	0.32	0.31	0.31	**0.17**	0.17	0.18	0.17	0.17
KNN	MAE:	**1.0**	1.04	1.12	1.03	1.03	0.76	**0.72**	0.77	0.72	0.73
	RMSE:	**2.78**	2.84	3.07	2.82	2.81	2.41	**2.41**	2.59	2.41	2.41
	R^2:	**0.91**	0.91	0.90	0.91	0.91	0.81	**0.82**	0.79	0.81	0.81
	MMRE:	**0.23**	0.23	0.25	0.24	0.23	0.63	**0.57**	0.61	0.59	0.57
	PRED(0.3):	**0.8**	0.8	0.79	0.79	0.79	0.65	**0.68**	0.67	0.67	0.68
SVR	MAE:	**0.92**	1.54	2.99	1.1	1.1	**0.62**	0.75	1.23	0.68	0.68
	RMSE:	**2.77**	3.33	5.15	2.97	2.99	**2.28**	2.45	3.54	2.36	2.35
	R^2:	**0.92**	0.88	0.7	0.9	0.9	**0.83**	0.81	0.6	0.82	0.83
	MMRE:	**0.23**	0.37	0.69	0.27	0.27	**0.65**	0.66	1.03	0.65	0.65
	PRED(0.3):	**0.84**	0.69	0.47	0.8	0.8	**0.65**	0.6	0.4	0.63	0.63
MLP	MAE:	2.43	**1.08**	1.42	1.41	1.39	1.18	**0.86**	1.20	1.14	1.15
	RMSE:	4.97	**2.34**	2.63	2.65	2.66	2.79	**2.11**	2.57	2.40	2.39
	R^2:	0.73	**0.94**	0.92	0.92	0.92	0.75	**0.86**	0.79	0.81	0.81
	MMRE:	0.56	**0.29**	0.38	0.36	0.36	1.59	**1.11**	1.43	1.28	1.28
	PRED(0.3):	0.59	**0.74**	0.65	0.67	0.68	0.32	**0.41**	0.29	0.36	0.36

Table 6.3: Model stack results, for both task duration and effort, with target encoding for categorical variables and 5 different test cases: No-Preprocessing; Centering ans Scaling (CS); Normalization (NZ); Box-Cox (BC) transformation with CS; Yeo-Johnson (YJ) with CS

Starting with the ensemble tree based models like RF, extraTrees, gbm and XGBoost, it is clear from the results that these methods got the best accuracy within the stack for both duration and effort models. Also, since tree based methods do not require any pre-processing, every type of feature transformation did not change the results of these models. Despite the RF having better results with parameter tuning in comparison with the baseline RF presented in last section, it still was not able to beat the other 3 tree ensembles that achieved the best results very closely between each other. Another positive aspect of the tree based ensembles is that they all achieved pretty high R^2 scores for both models which means the results variance is explained by the models and not by mere chance. On the other hand, lasso, the only linear model on the stack, got not only very poor results, but was the worst performing model by far. This just confirms the dependent versus independent variable scatter plots (fig. 6.11) interpretation, where no signs of any existing linear relationship was found, hence lower expectations were built for this model.

About KNN, SVR and MLP, it was expected, according to research made and table 3.1 (section 3.4.7), that centering and scaling would improve their performance, same as normalization for KNN and MLP, but this was not the case for SVR in both models and even KNN in the task duration model. This unexpected and interesting results could be explained though. It turns out that the variables that have most importance for calculating the targets are the ones with higher scale, such as *wp_duration*, *b_duration*, *task_number*, *wp_number*, etc., which can be confirmed in fig. 6.12 and 6.13. Then, when centering and scaling the input set one is actually taking weight away from these variables, and knowing SVR and KNN are distance based methods they would benefit more, in this case, of having the most important variables with the highest scale, hence with more weight than other less important variables. Either way, KNN, SVR and MLP, considering their best case scenario (e.g. for SVR is with no pre-processing), also presented good performances not so distant from the tree-based models.

In general, all MAE values are significantly lower than the RMSE ones. Due to the fact that RMSE is a better metric to detect larger errors, this confirms the existence of a small portion of big errors in the models. Moreover, probably means that, even after some cautious data preparation and cleaning, there still might be noise in the data to take care of. Either way, most of the models generated substantially precise estimates for both effort and duration. In fact, almost all the algorithms (except lasso for the reasons mentioned earlier) produced at least one model that surpassed or was close to surpass the threshold defined by Conte et al. [12] and Shepperd et al. [22] that consider an accurate model when MMRE $\leq .25$ and PRED (.3) $\geq .75$.

From this point, the final average ensemble is ready to be implemented. Since, for the exception of lasso, all algorithms presented models with relatively good accuracy and not much distance between each other, besides, the more methods included in final ensemble the more robust it will get, the whole stack was picked (except lasso) to construct the final ensemble for both targets.

The results of the ensembles are demonstrated in table 6.4.

Relative to the duration ensemble, even though its performance is slightly worse than the boosting ensembles (gbm, XGBoost) and the extraTrees it still provides a good performance beating all other models from the stack inclusively the random forest. Nevertheless, it is also a more

Metrics	Duration	Effort
MAE:	0.67	0.52
RMSE:	1.88	1.82
R^2:	0.96	0.9
MMRE:	0.16	0.46
PRED(0.3):	0.88	0.74

Table 6.4: Final average ensemble results for both task effort and duration models

stable model, being more resilient to residual outliers and overfitting due to the nature of the ensemble. This can be proved when comparing fig. 6.14 with fig. 6.15, where the ensemble has fewer outliers showing its robustness to high variance while, on the other hand, the MLP alone shows more larger errors, hence the higher RMSE when compared with the ensemble.

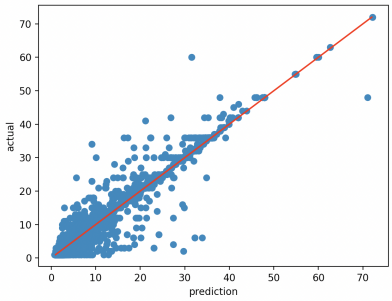

Figure 6.14: Scatter plot with duration predictions versus actual values for ensemble

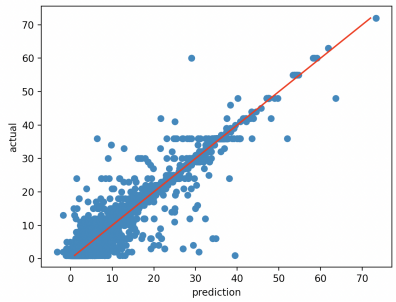

Figure 6.15: Scatter plot with duration predictions versus actual values for MLP

As for the effort ensemble, it got similar RMSE to extraTrees, only losing against the boosting ensembles. Again fig. 6.16 and fig.6.17, help concluding that even though not being the most accurate model it definitely compensates in stability, maintaining a good variance and bias trade off which is essential for the success of any ML method.

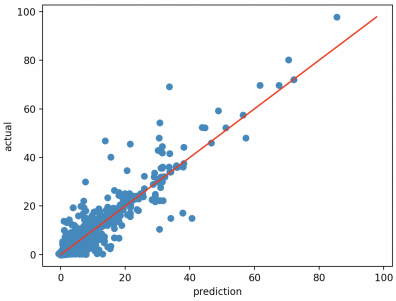

Figure 6.16: Scatter plot with effort predictions versus actual values for ensemble

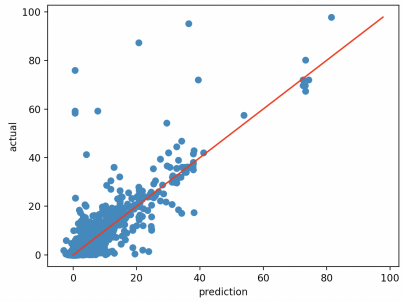

Figure 6.17: Scatter plot with effort predictions versus actual values for MLP

6.3 Discussion

In this section, it is going to be discussed the research questions and their main conclusions achieved from all experiences. In the end, the hypothesis will be validated based on the previous conclusions.

6.3.1 RQ1: What predictors can be used for accurate forecasts and why?

As studied through out sections 5.1 and 6.1, a long set of experiences were made in order to find the best predictor set from the available data in the Project Control database from the PROMESSA project. The best set would obviously be the one with highest correlation and significance to predict the target variables *task_duration* and *task_effort*. With that in mind, the first step was to filter the initial database and pick the most relevant tables and data related with the tasks. From this process resulted table 5.1 from sec. 6.1 with the initial promising 25 inputs. These inputs were mainly related with duration of projects/work packages, funding and revenues, areas of interest like business or R&D, stages of project, order of tasks, etc. A mindset of rather include than exclude were practiced as there was no certainties about what could actually weight in the prediction. Even data from task names was extracted through data mining to attempt to include the maximum knowledge possible to predict the targets. But not all filtered data is expected to help the same or help at all. Other methods must be used to dig further in the data and find relationships with the target variables.

As from the Pearson correlation matrix constructed with these filtered predictors and seen in fig. 6.4 of sec.6.1, one could see the obvious variables that would affect the targets, such as work packages and baseline duration, which make sense since the only difference for the task duration is that they are higher in the database hierarchy and thus represent another dimension like the projects that contain these tasks. But, since Pearson's correlation indicates mostly linear relationships, there could be some important predictors that were not showing a decent correlation, even though the relationship existed (just not so linear). For those reasons, other feature selection options were explored like intrinsic algorithm feature selection such as LASSO and Tree based models which are very reliable options, but even wrapper methods could have been used. In the end, the random forest model was picked due to its flexibility allied with reliability in finding any type of relationship.

After some iterations of tests and with the help of correlations and random forest intrinsic feature selection, the best input set was found and can be seen in fig. 6.12 for duration and fig. 6.13 for the effort. Again for the duration the most relevant and selected predictors were not surprisingly the other duration variables like the *wp_duration* and *b_duration*. After all, the fact that every task duration has to be contained in the span of a work package or baseline duration clearly shows some strong relationship between these variables. The other important predictors were the parameters closer to the tasks like the *task_type* and *wp_number*. One can not stop to notice that the less relevant predictors were actually the ones related with the projects and baselines which are farther away from the tasks when compared with the task itself and the work packages.

For the task effort, normally one can not completely dissociate effort from time, which, in this case, is pretty obvious their strong relationship since the top 2 predictors were duration ones. It is curious to notice that for effort the predictor *eftm* - equipment funding type mask from the projects table, also plays a very strong role in its prediction coming at 4^{th}, really close to the other top predictors. This just proves that no assumptions can be made beforehand and that it is very important to understand the data deeply, prepare it and clean it the best way possible, before using it for any forecasting method as its success depends on it.

6.3.2 RQ2: Which methods can be used for accurate forecasts and how are the results comparable with the state of art?

From the research made, with special focus on the articles from Minku and Yao [33], Tin Ho [50] and Pospieszny [39], which supported the idea that models should not rely on individual algorithms but instead act as an ensemble, different from the regular homogeneous, but rather an heterogeneous ensemble, that would strengthen the model to noisy data, and problems such as outliers and overfitting. Besides that, the lack of consistent results associated with either bad practices or poor data were the main responsible for the lack of ML implementations in practice. Pospieszny [39] attempted to overcome this issue with a small heterogeneous average ensemble (with only 3 different algorithms) in an investigation environment with a public heterogeneous dataset. However, the problem still remains as there is still lack of sufficient proof that this can be applied in a commercial environment. To surpass this issue the same heterogeneous ensemble was developed in the Project Control dataset for Fraunhofer, but to increase confidence a much bigger stack (of 8 different models) was implemented and tested against other research works. However, first, to select an appropriate stack of the latest and most promising regression models, Fernández et al. [15] survey was consulted and 8 of the best models from different families were selected.

After a smart data preparation and pre-processing the 8 model results were compared in section 6.2, table 6.3. From there it was concluded that all models, except for the lasso which suffered from the problem not being linear, had sufficiently good results and were used to build the final heterogeneous ensemble. The best individual models for both duration and effort were the boosting ensembles and the extraTrees which agrees with Fernández et al. [15] survey pointing these algorithms to be within their global top 6 regression methods for a collection of 83 datasets. The final ensemble model obtained the results presented in table 6.4 from section 6.2. Comparing other research results:

- **Berlin et al.**[7] obtained, for a similar ensemble approach, but only with ANN and LR (linear regression), using log transformation for both effort and duration, MMRE=0.22 and PRED(0.3)=0.81 for effort, which gets better results than our effort ensemble, but on the other hand, gets completely "swallowed" by the duration ensemble which Berlin only got a poor MMRE=0.88 and PRED(.3)=0.27;

- **López-Martín and Abran** [30] compared different ANN algorithms only for duration forecast with log transformation of target variables and cross-validation obtaining MMRE=0.18 and PRED(.25)=0.8, which were slightly worse than our ensemble;

- **Pospieszny** [39] with a similar ensemble approach composed by a SVR,MLP and GLM, for both duration and effort, obtaining for the first MMRE=0.21 and PRED(0.3)=0.76 and for the second MMRE=0.17 and PRED(0.3)=0.74. With those results, our duration ensemble wins by a considerable difference but the effort ties in PRED but looses in MMRE. However, Pospieszny used log transformation on dependent variables and maintained them log transformed which biased the results since they are not on their natural unit;

- **Barcelos Tronto et al.** [6] using ANN and linear regression models for effort estimation with average MMRE 0.45 for ANN, similar to our effort ensemble MMRE but winning against their linear regression models that got worse MMRE scores;

- Others like **Wen et al.** [54] made a systematic review for effort prediction with various individual models such as SVM with MMRE=0.34 and PRED(.25)= 0.72 and ANN with MMRE=0.37, PRED(.25)= 0.64, which got similar results when compared with our effort ensemble, which slightly gets worse MMRE but at the same time a higher PRED.

These results show that duration ensemble model dominated the other researches individual and ensemble models. On the other side, effort managed to loose only a couple of them, although if the effort ensemble instead of using 7 models used the 3 or 4 best seen for effort in table X, than the results would not be the same, but on the other hand the ensemble would loose some robustness to data noise and overfitting. This type of trade-off might be desirable in environments where precision a key point but to keep a more stable model the effort ensemble with 7 models is rather advised.

Concluding, both ensemble models proved to be very competitive in performance against the state of art results, but again with the obvious advantage of being way more stable and reliable than individual models.

6.3.3 Hypothesis

"Machine learning methods for software project forecasting can accurately be applied, bringing benefits in terms of reduced forecasting errors and resources management"

From the above research questions, it is clear that when the data is well understood, prepared correctly, and a good stack of different algorithms is tried for a specific problem, the chances of successfully applying a ML model get substantially higher. Taking as an example the ensembles implemented in this dissertation, not only it was proven that the hypothesis can be true, but also that it can be true in an industrial environment where until now only resided inconclusive results and very few implementations.

6.3.4 Threats to validity

In this section, an analysis will be done on dissertation assumptions and possible validity threats:

- The established baseline model was the Random Forest. Even though, being a widely adopted method with good accuracy and not much need for pre-processing, other feature selection models could have been used such as the LASSO or Elastic Nets which could have changed the input set and hence the final results.
- From the results observed from the state of art, most of those papers implemented their methods on databases such as ISBSG, COCOMO, or other public and shared datasets that are rather heterogeneous and sometimes very noisy. This can negatively affect the researchers results which might invalidate the comparison made in the discussion section and hence reduce the reliability of this dissertation results. Also from the papers chosen, even with systematic reviews and surveys, the sample of these papers is not high enough to guarantee that other papers would not get the upper hand and achieve better results than the ones presented which could invalidate the conclusions.
- Implementing machine learning models for software project forecasting might be dependent of factors such as data availability, diversity and size. For this dissertation tens of thousands of records were used to obtain the models, but it is understood that small organizations might not have this capacity. Knowing the importance of the size but also the quality of the data, this type of implementations might be more appropriate for medium and large sized companies which also tend to get more affected by wrong software project estimations. Lastly, a balance between the benefits and losses must be done in order to decide whether to go further or try a simpler approach such as analogy methods or others.

Chapter 7

Conclusion and Future Work

In this chapter, some last conclusions regarding the dissertation are made as well as considerations about the expected contributions to arise from this work.

7.1 Conclusion

Through the last decades, researchers have been trying to successfully implement software project forecasting to better manage resources. This however, has not been successfully due to the many reasons aforementioned. Because of that, few if none machine learning implementation is seen in practice. This dissertation aims to fight that by successfully implementing an heterogeneous ensemble with 7 different models in an industry environment to contribute, not only for Fraunhofer, but also for similar projects that might benefit of an heterogeneous ensemble approach. This type of algorithms, due to their nature, usually trade high precision predictions for robustness, although still being able to deliver good performance. The main advantage of these ensembles are their stability and ability to handle outliers, noisy data and overfitting problems. Situations where small errors are tolerable but big errors not, such as the task effort and duration forecast, is where this model is ideal. Another main expected contribution is to extend the PROMESSA module with both effort and duration machine learning ensembles developed in the dissertation which will allow every project manager to save time, manage resources better and decrease forecasting error. This module will aid tools like the Project Control, to better manage resources and expectations.

7.2 Future Work

The results proved to look promising, but still many things could be done to improve even further:

- From figures 6.14 to 6.17, it is clear that some residual outliers are constant between the plots. This allied with the fact that the algorithms generally present a RMSE for around 3 times the MAE, which signifies that a small portion of big errors is still happening. This

is probably caused by noise that is still present in the database. A deeper analysis for this points and for the maximum number of possible rows should be done to decrease this errors.

- On the same plots from figures 6.14 to 6.17, it is noticeable that most data points still reside aggregated on the left side of the plot, meaning that the plot is composed mainly by small values. This could mean a problem of unbalanced data if outlier points prove to have value for the organization. To solve this, sampling approaches such as smote for regression as suggested by Torgo et al. [9, 51] could prove to be an efficient way of dealing with this which would increase performance and residuals stability.

- From Fernández et al. [15] research, the rule based models *cubist* and *M5* were mentioned as the algorithms with the best overall performance for that study. Unfortunately, since these methods are not yet implemented for sklearn, but instead in *R* and other languages, this algorithm could not been explored. A vanilla implementation of this algorithm in python can be tried to test it in the proposed ensembles. Besides that, other algorithms from sklearn could also be tested to find the best possible stack of models.

- A simple average of results technique was used for this experience ensemble models. However, there are other ensemble approaches with dynamic selection as proposed by João Moreira's research [31] that could be tried to improve the ensembles performance.

Appendix A

Appendix

In this section will be annexed other types of figures or information that are not necessary to be posted in any of the other sections and that can be referred here from anywhere.

A.1 Data Preparation Scripts

Relevant scripts and snippets of code concerning data preparation will be posted here for reference.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `task_effort_view` AS
    SELECT
        `allocations`.`task_id` AS `task_id`,
        ROUND(SUM((`allocations`.`value` * ((`allocations`.`finish` - `allocations`.`start`) + 1))),
            2) AS `effort`
    FROM
        `allocations`
    GROUP BY `allocations`.`task_id`
```

Listing A.1: Task effort view in SQL

```
import csv

tasks = []
headers = ''
labels = []

class Task:
    def __init__(self, identifier, name):
```

```
9           self.identifier = identifier
10          self.name = name
11          self.label = []
12
13      def __iter__(self):
14          return iter([self.identifier, f'{";".join(self.label)}'])
15
16      def add_label(self, value):
17          if value not in self.label:
18              self.label.append(value)
19              self.label.sort()
20
21  def label_tasks():
22      with open('labels.csv', mode='r') as label_file:
23          csv_reader = csv.DictReader(label_file)
24          label_count = 0
25          for row in csv_reader:
26              labels.append(row)
27              label_count += 1
28          print(f'Loaded {label_count} labels')
29
30      with open('promessa_tasks.csv', mode='r') as csv_file:
31          csv_reader = csv.DictReader( (line.replace('\0','') for line in
        csv_file) )
32          line_count = 0
33          for row in csv_reader:
34              if line_count == 0:
35                  global headers
36                  headers = list(row.keys())
37                  headers.append('label')
38              process_task(Task(row['id'], row['name']))
39              line_count += 1
40          print(f'Processed {line_count} lines.')
41          export_tasks()
42
43  def process_task(task):
44      for label in labels:
45          if label['wildcard'] in task.name.lower().split() and len(task.label) <
         2:
46              task.add_label(label['name'])
47      tasks.append(task)
48
49  def export_tasks():
50      tasks_label = 0
51      with open('output.csv', 'w') as csv_file:
52          wr = csv.writer(csv_file, delimiter=',')
53          wr.writerow(['id',
54                       'task_type'])
55          for task in tasks:
```

```
56              if len(task.label) > 0:
57                  tasks_label = tasks_label + 1
58              wr.writerow(list(task))
59
60      print(f'Tasks with labels: {tasks_label}')
61
62  if __name__ == '__main__':
63      label_tasks()
```

Listing A.2: Script to extract the *task_type* from the task *name* variable

References

[1] Project evaluation review technique (pert). Available at https://corporatefinanceinstitute.com/resources/knowledge/other/project-evaluation-review-technique-pert/.

[2] Multivariate adaptive regression splines. Available at https://support.bccvl.org.au/support/solutions/articles/6000118097-multivariate-adaptive-regression-splines, January 2019.

[3] P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi. Effort prediction in iterative software development processes – incremental versus global prediction models. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 344–353, 2007.

[4] Fatima azzahra Amazal Ali Idri and Alain Abran. Analogy-based software development effort estimation: A systematic mapping and review. *Elsevier*, 07 2014.

[5] D. Azhar, P. Riddle, E. Mendes, N. Mittas, and L. Angelis. Using ensembles for web effort estimation. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 173–182, 2013.

[6] Iris Barcelos Tronto, José Silva, and Nilson Sant'Anna. An investigation of artificial neural networks based prediction systems in software project management. *Journal of Systems and Software*, 81:356–367, 03 2008.

[7] Stanislav Berlin, Tzvi Raz, Chanan Glezer, and Moshe Zviran. Comparison of estimation methods of cost and duration in it projects. *Information and Software Technology*, 51:738–748, 04 2009.

[8] Barry W. Boehm. *Software Engineering Economics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[9] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Comput. Surv.*, 49(2), August 2016.

[10] Frederick Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering*. 01 1995.

[11] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[12] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software Engineering Metrics and Models*. Benjamin-Cummings Publishing Co., Inc., USA, 1986.

[13] Egemen Ertugrul, Zakir Baytar, Cagatay Catal, and Can Muratli. Performance tuning for machine learning-based software development effort prediction models. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING COMPUTER SCIENCES*, 27, 03 2019.

[14] Richard E. Fairley. Recent advances in software estimation techniques. In Tony Montgomery, Lori A. Clarke, and Carlo Ghezzi, editors, *Proceedings of the 14th International Conference on Software Engineering, Melbourne, Australia, May 11-15, 1992*, pages 382–391. ACM Press, 1992.

[15] M. Fernández-Delgado, M.S. Sirsat, E. Cernadas, S. Alawadi, S. Barro, and M. Febrero-Bande. An extensive experimental survey of regression methods. *Neural Networks*, 111:11–34, 2019.

[16] J. Frost. *Regression Analysis: An Intuitive Guide for Using and Interpreting Linear Models*. James D. Frost, 2020.

[17] Evans M. Galorath, D. *Software Sizing, Estimation, and Risk Management*. Auerbach Publications, 2006.

[18] James Grenning. Planning poker or how to avoid analysis paralysis while release planning. Available at http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf, April 2002.

[19] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12, 2009.

[20] Jianglin Huang, Yan-Fu Li, and Min Xie. An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Information and Software Technology*, 67:108–127, 2015.

[21] Aarshay Jain. Complete guide to parameter tuning in xgboost with codes in python. Available at https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/, Mar 2016.

[22] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, 2007.

[23] Magne Jørgensen. Forecasting of software development work effort: Evidence on expert judgement and formal models. *International Journal of Forecasting*, 23(3):449–462, 2007.

[24] E. Kocaguneli, T. Menzies, and J. W. Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 38(6):1403–1416, 2012.

[25] Will Koehrsen. Improving the random forest in python part 1. Available at https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd, 2018.

[26] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, New York, NY, 2013.

[27] Satyam Kumar. Feature engineering — deep dive into encoding and binning techniques. Available at https://towardsdatascience.com/feature-engineering-deep-dive-into-encoding-and-binning-techniques-5618d55a6b38, 2020.

[28] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.

[29] Troy Magennis. Managing software development risk using modeling and monte carlo simulation. 05 2012.

[30] Cuauhtémoc Martín. Predictive accuracy comparison between neural networks and statistical regression for development effort of software projects. *Applied Soft Computing*, 27:434–449, 02 2015.

[31] João Mendes-Moreira, Alípio Mário Jorge, Jorge Freire de Sousa, and Carlos Soares. Improving the accuracy of long-term travel time prediction using heterogeneous ensembles. *Neurocomputing*, 150:428–439, 2015. Special Issue on Information Processing and Machine Learning for Applications of Engineering Solving Complex Machine Learning Problems with Ensemble Methods Visual Analytics using Multidimensional Projections.

[32] Darko Milicic. *Applying COCOMO II - A case study*. PhD thesis, Blekinge Institute of Technology, SE – 372 25 Ronneby, 2004.

[33] Leandro L. Minku and Xin Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, 55(8):1512–1528, 2013.

[34] Pedro Miranda. *Lean Forecasting In Software Projects*. PhD thesis, Engineering University of Porto, R. Dr.Roberto Frias, Porto, 2020.

[35] Pedro Miranda, J. Pascoal Faria, Filipe F. Correia, Ahmed Fares, Ricardo Graça, and João Mendes Moreira. An analysis of monte carlo simulations for forecasting software projects. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, SAC '21, page 1550–1558, New York, NY, USA, 2021. Association for Computing Machinery.

[36] Ali Nassif, Mohammad Azzeh, Luiz Capretz, and Danny Ho. Neural network models for software development effort estimation: a comparative study. *Neural Computing and Applications*, 27, 11 2016.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[38] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[39] Przemyslaw Pospieszny, Beata Czarnacka-Chrobot, and Andrzej Kobylinski. An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software*, 137:184–196, 2018.

[40] M. T. Rahman and M. M. Islam. A comparison of machine learning algorithms to estimate effort in varying sized software. In *2019 IEEE Region 10 Symposium (TENSYMP)*, pages 137–142, 2019.

[41] Da Ruan, Guoqing Chen, Etienne Kerre, and Geert Wets. Intelligent data mining: Techniques and applications. *Studies in Computational Intelligence*, 5, 01 2010.

[42] Christopher Schofield. *An Empirical Investigation into Software Effort Estimation by Analogy*. PhD thesis, Bournemouth University, 1998.

[43] Diogo Seca and João Mendes-Moreira. Benchmark of encoders of nominal features for regression. In Álvaro Rocha, Hojjat Adeli, Gintautas Dzemyda, Fernando Moreira, and Ana Maria Ramalho Correia, editors, *Trends and Applications in Information Systems and Technologies*, pages 146–155, Cham, 2021. Springer International Publishing.

[44] Sumeet Kaur Sehra, Yadwinder Singh Brar, Navdeep Kaur, and Sukhjit Singh Sehra. Research patterns and trends in software effort estimation. *Information and Software Technology*, 91:1–21, 2017.

[45] Martin Shepperd and Steve MacDonell. Evaluating prediction systems in software project estimation. pages 820–827, 2012. Special Issue: Voice of the Editorial Board.

[46] Dr. Amit Sinhal and Bhupendra Verma. Software development effort estimation: A review. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3:1120–1135, 06 2013.

[47] Andrew Stellman and Jennifer Greene. Applied software project management: Estimation. Available at https://www.stellman-greene.com/LectureNotes/03%20estimation.pdf, May 2020.

[48] K. Strike, K. El Emam, and N. Madhavji. Software cost estimation with incomplete data. *IEEE Transactions on Software Engineering*, 27(10):890–908, 2001.

[49] I. C. Suherman, R. Sarno, and Sholiq. Implementation of random forest regression for cocomo ii effort estimation. In *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*, pages 476–481, 2020.

[50] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[51] Luís Torgo, Rita P. Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. In Luís Correia, Luís Paulo Reis, and José Cascalho, editors, *Progress in Artificial Intelligence*, pages 378–389, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[52] Adam Trendowicz, Jürgen Münch, and Ross Jeffery. State of the practice in software effort estimation: A survey and literature review. In Zbigniew Huzar, Radek Koci, Bertrand Meyer, Bartosz Walter, and Jaroslav Zendulka, editors, *Software Engineering Techniques*, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[53] Muhammad Usman, Emilia Mendes, Fran Neiva, and Ricardo Britto. Effort estimation in agile software development: A systematic literature review. 09 2014.

[54] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[55] Aishwarya Wuntkal. Support vector machine - kaggle competition. 2020.

[56] R.K. Wysocki. *Effective Project Management: Traditional, Agile, Extreme, Industry Week*. John Wiley & Sons, 2014.