POLITECNICO DI TORINO

Department of Electronics and Telecommunications

Master's degree programme in
ICT For Smart Societies

Master Thesis

Data mining applied to vehicle usage in construction sites

Supervisors

prof. Marco Mellia
Luca Vassio

Candidate

Dena Markudova

December 2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First of all, the biggest Thank you to my supervisor, mentor and navigator, prof. Marco Mellia. Throughout the whole process of working on this thesis he was informative and helpful. His door was always open to me and he had a positive and friendly attitude. He let me be independent, yet gently guided me at the same time. Thank you for all your time and support.

A warm Thank you to Luca Vassio, my co-supervisor who inspired me and treated me as a friend.

I am grateful for all the professors of "Smart data@Polito", prof. Luca Cagliero, prof. Elena Baralis and prof. Francesco Vaccarino for their selfless guidance throughout all the long meetings.

A special gratitude to the partners from "Tierra S.p.A" for providing me with the essential part of the thesis – the data. I would especially like to thank Lucia Salvatori and Elvio Amparore for their immense hospitality and constant availability while I was a guest in their office.

And last, but by no means least, I would like to thank my family, my mother, father and sister for their everlasting encouragement and cheer in all goings-on in my life. Also, my boyfriend Marko who patiently and lovingly supported me through the emotional roller-coaster of working on the thesis.

1 INTRODUCTION

1.1 IOT AND DATA ABUNDANCE

In our modern-day society, everyone is connected. On our phones, our computers, we are one click away from our friends, from news, from information. But, in the last decade, there has been a revolution in the connectivity concept – even everyday objects are becoming connected - our home appliances, our cars and even the machines used in the industry.

This is facilitated by computing devices, such as sensors, actuators and microcontrollers embedded in these objects, enabling them to send and receive data. The process is also called making the objects "smart". How smart they are depends on the technology embedded in them. The lowest level is if they are only equipped with sensors that are only able to emit signals – just send data. Then, they can be equipped with sensors that are able to both send and receive data, making them an efficient communicator. Going higher on the scale, they can hold actuators, which enable them to be controlled remotely. Finally, they can be equipped with microcontrollers or more intelligent computing devices which make them able to process data, send more elaborate data to the end-user, make decisions themselves (programmed from before) and act independently by actuating. The interconnection and exchange of data between these "smart" objects is called a sensor network.

All of this falls under the umbrella of the Internet of Things (IoT). It is becoming an important topic and it is spreading fast. IoT systems are gaining popularity because they are easy to deploy, energy efficient and very useful. This technology has two main uses:

- Monitoring – data collection and visualization of what is going on inside a system. Owners can have a deep insight of the process via analytics and human-readable plots.
- Actuation - remote control of the system. If we know what is happening in the system, we can act from afar.

What is the driving force of these systems and at the same time the side-product? What is the residual of all this connectivity? The answer is **data**. An enormous amount of data. In order to

operate, to do monitoring or actuation, sensor networks gather huge amounts of data. And memory not being a problem for today's computers, all this data is stored.

The natural next step here is to explore this data and try to use it somehow. Data is powerful, it can be used to solve many practical problems. It can help give a deeper insight into what is happening in our system, find problems in the process that are not easily visible and do diagnostics. This way, systems can be better optimized and that leads to lower costs and higher quality.

Data mining is a field that allows us to do exactly that – examine datasets in order to generate new information. It is a computer-assisted process of analyzing enormous amounts of data and extracting meaning from them. It can help businesses make knowledge-driven decisions. A few notable uses of data mining are:

- Find hidden patterns: Data often holds information that even experts do not expect.
- Anomaly detection: Detect outliers in the data that may cause problems inside the system.
- Learn customer purchase habits: This is an example from the retail area – find out which items are frequently bought together and use this knowledge to boost revenue.
- Predictive maintenance: Instead of doing scheduled maintenance, predict when equipment might fail and perform maintenance to prevent it.
- **Predict behaviour and future trends**: Knowledge of the past can give us insight on what will happen in the future.

In this thesis, we explore exactly this application of data-mining. We take data collected remotely from a real-life IoT platform and we process it to get a valuable outcome. This outcome is – a prediction of the future.

A field that is tightly coupled with Data mining is **Machine learning**. It is hard to explicitly define Machine learning, but (Faggella, 2018) grasps the concept with this definition:

"Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions."

Machine learning is the science of getting computers to act without being explicitly programmed. Computer systems are given a dataset and they "learn" by themselves what are the correct answers,

so they progressively improve on accomplishing a certain task. Machine learning is becoming ever-present in the modern day. It is the key technology to self-driving cars, practical speech recognition, effective web search etc. It is precisely Machine learning algorithms that enable Data-mining and that give the answers we are looking for in big datasets. In this thesis, we use machine learning algorithms to do prediction.

1.2 IN THE BACKGROUND OF THIS THESIS

This thesis was developed as part of a partnership between Politecnico di Torino and a company for IoT solutions – Tierra S.p.A. The part of the university involved in this partnership is the Smart Data Center, which is a Data Science and Big Data research center.

Tierra designs and develops advanced telematics & IoT solutions for geo-localization, management, maintenance and remote diagnostics of assets. It was born in 2008 as a joint venture of the Japanese Topcon Corporation and the Italian Divitech. It offers services of monitoring and control of assets, with remote diagnostics and reports in the fields of Industry, Agriculture and Construction. These services help reduce costs of maintenance and increase productivity.

What we are interested in is the part of Construction. Tierra provides solutions for tracking vehicles at construction sites. This is done with the help of an onboard device that is installed inside the vehicles and gathers data about their activity: their location, movement, fuel consumption, are they ON/OFF, in which state of workload are they (Idle, Moving, High workload), what is their RPM etc. This data is then sent to a server, which is part of a cloud infrastructure, where it is processed and transformed into human-readable analytics (charts, plots, text) and reports. These analytics are then showed to the owner of the vehicles in real-time, via a web-interface. This means that the owner can have an insight of their fleet from the comfort of their office.

On Figure 1.1 (Tierra presentation, 2018) we can see a sketch of the whole process.

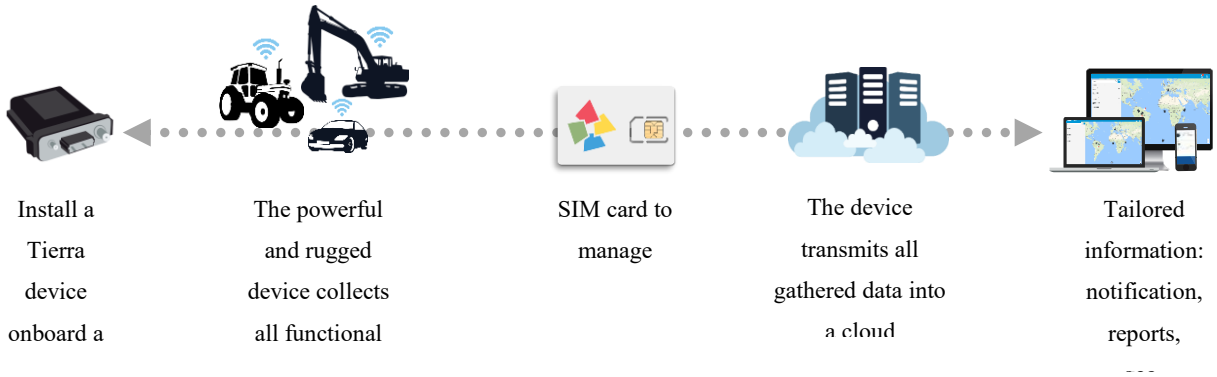

Figure 1.1: How Tierra's telematics works – as seen through the eyes of a customer

For now, what is shown to the owner is real-time observations with some basic analytics and statistics. However, no advanced analysis or usage prediction is offered. This information exists inside the data collected, it just needs to be found. This is where the idea for this thesis comes from. The data analysis is done on activity data coming from construction vehicles of companies that are customers of Tierra. The data is anonymized for privacy reasons.

1.3 THE PROBLEM AND THE GOAL

The problem that we are trying to solve is defined simply: Given the usage pattern of a vehicle in the past days, predict the utilization of the same vehicle for the next period (the next day, next working day). Basically, we use the data collected from the past, to try to forecast the future.

What we try to predict is: Utilization hours of a vehicle – how many hours it will be used in a certain point in the future (ex. tomorrow).

Which data we use to get there: The number of utilization hours in the past days along with some other indicators that may help us reach a precise prediction.

This problem falls under the category of **time series forecasting**. In the classical statistical handling of time series data, it is also called extrapolation. It is a different problem from usual

machine – learning problems, because it adds an explicit order dependence between observations: time. This additional time dimension is both a constraint and a source of additional information.

"Forecasting is the art of saying what will happen, and then explaining why it didn't. "

- Anonymous (communicated by Balaji Rajagopalan)

A time series is a vector of data points listed in time order. Usually, the time intervals between two successive points are equally spaced. Time series are vastly used in statistics, telecommunications, signal processing, weather forecasting, stocks forecasting etc. There are two separate paradigms connected to time series: time series analysis and time series forecasting and they have different goals.

- Time series analysis

  Comprises of methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. It describes and interprets the time series. It takes into consideration the fact that data points taken over time may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for.

- Time series forecasting

  It uses the information inside a time series plus maybe additional information from the dataset or the domain itself to predict future values of that series. There are many variables to be set about the prediction – what is the window in the past that will be used as information, what is the point or set of points in the future that will be predicted etc.

Even though time series analysis and forecasting are separate fields and the latter can be done without the former, it is always good practice to examine the dataset first and know what kind of data we are working with.

To sum up, the goal of this thesis is to explore the time-series made of Utilization hours of a construction vehicle and do a forecast of future values. In order to reach this goal, to make our

prediction, we use machine learning algorithms, specifically Regression techniques. The concept of regression and the details on the algorithms are better explained in Chapter 3: Methodology.

1.4 THE STEPS OF BUILDING A PREDICTIVE MODEL

Building a good forecasting model is a long process that is subject to a lot of planning and trial and error experiments. It is usually organized in a few steps:

1. Problem definition

   There must be a problem so that we can find a solution. What kind of forecast do we need? Who requires this forecast? How will it be used? The answers to these questions should be well defined. This step is problem specific, so the analyst should get acquainted with some domain knowledge.

2. Data collection

   The gathering of historical data to analyze and model. In the general sense of the word, data can be collected in many ways - through surveys, case studies, documents, records etc. With the development on IoT, a lot of data is collected through IoT platforms, which is also the case for this thesis. Domain knowledge would also help here to best interpret the historical data.

3. Data cleaning

   Raw data can come in any form and any frequency and is often messy and difficult to work with. It needs to go through a step of data-cleaning to become useful. This includes filtering outliers, handling missing data, formatting data, normalizing etc. This is a very important step since the accuracy of the prediction also depends on the quality of the data. In time series analysis this can consist of noise reduction and smoothing.

4. Exploratory data analysis

   After the data is ready for handling, we explore it to get a general idea of what is going on in the dataset and find basic characteristics. The data undergoes statistical tests and is visualized.

Statistics is the part of mathematics that deals with collection, organization, analysis, and interpretation of numerical data. This step helps us understand the data better. In our case of time series data, we can summarize and note obvious temporal structures, like trends and seasonality.

5. Model construction

In this step we model our problem and figure out how we are going to solve it. We decide what part of the data is going to be used to solve it and in what way. What is going to be the input and output from the algorithm? For example, what is the window in the past that we are going to use as historical data? Which features are we going to feed to the algorithm? What granularity of data points are we going to use? How far into the future do we want to predict? Here we answer questions of this kind. We have to make some assumptions in order to apply a specific model, but our choices are backed up by the data and/or domain knowledge.

6. Applying Machine learning algorithms

Finally, when our model is ready, we can apply machine learning algorithms and evaluate the outcome. We pick a few algorithms of varying types. They are configured, the parameters are tuned, and they are fitted on the historical data. This is where we obtain our prediction. Then their performance is evaluated by back-testing with the available data.

In this thesis we focus on some of these steps. We already have a well-defined problem and we have data gathered through an IoT platform, as mentioned in the previous sections. So, we concentrate on steps 3 to 6 – from data cleaning, through exploratory data analysis and building a model, to doing machine learning and acquiring a prediction.

1.5 MOTIVATION

In this Section we answer the questions: Why this goal? What is the purpose of predicting the utilization hours of a vehicle and how does this contribute to the business model of the owner?

By analyzing the behaviour of each separate vehicle, we can find out:

- The productivity of the vehicles: see if they work more than necessary or less than expected, if they stay with the engine ON but are not moving
- What is their pattern of usage and how does it vary between different vehicles
- How correlated are different vehicles of the same model, is there a pattern that vehicles of the same model usually follow, but some vehicles are not into this pattern?
- Are there outliers, is there any unusual activity and does that mean that something is wrong with the vehicle
- Is there any period of the year where they work more/less and why is that so?

The motivation behind predicting utilization hours is:

- Knowing future patterns can help the owner of the fleet plan ahead budgets for the usage of the vehicle, like amortization
- By predicting the utilization hours, we can easily calculate the fuel consumption and plan refueling in a more optimized nature, we can also include the fuel forecast in the budget planning
- Predictive maintenance: if we know how much a vehicle is going to work, then we know with a certain confidence when a vehicle is going to break down, so we can do maintenance before to prevent the failure
- Fleet management: predictions can help in overall fleet management

These are all valuable information to the owner of the vehicle and from a business perspective, they would pay for this kind of information.

1.6 LITERATURE REVIEW

The literature review showed that a lot of research has been done for vehicles in general, in terms of transport of people (usage patterns of cars, transport planning inside the city, public transport efficiency), but not a lot of papers have been written on vehicles in construction sites. Some papers that use data – mining approaches on construction vehicle data are quoted here and they are grouped by different topics.

One problem that is tackled in the construction field seems to be the ecological impacts of the vehicles. They all use fossil fuels, which makes them not environmentally friendly. Two papers were found that suggest data – mining approaches to limit these negative impacts.

- Bakhiet (2017) aims to reduce carbon dioxide emissions from construction vehicles. It talks about what kind of data should be collected so that we can clearly and efficiently find the greatest sources of emissions and the problems that arise from not having one unified standard for collection of this type of data (formats, classification, inaccuracy, requires additional administrative work etc.)
- Rasdorf *et al.* (2010) also propose standard procedures for field data collection for construction vehicles to conquer emissions. The methodology is based on second-by-second measurements of in-use activity and air pollutant emissions using a portable measurement system. They argue that the use of your own instrument insures a reliable collection of data, clean and of good quality.

Research has also been directed towards data-mining for more accurate simulation and modelling for construction operations.

- Akhavian and Behzadan (2012) note the importance of combining historical data (traditional approach) with real-time data from construction sites (novel approach) to make 3D simulations on what is happening on the site. These simulations would help construction companies make decisions on fleet management.
- Akhavian and Behzadan (2013) further their research by introducing an innovative approach for data capturing, fusion and mining for the purposes of knowledge-based

simulations of construction vehicle operation. The approach consists of using weight sensors and doing K-means clustering to distinguish the state of the vehicles (idle/busy).

Two papers were found that apply machine-learning algorithms to construction vehicle data in order to do predictive data – mining.

- Fan *et al.* (2008) use Autoregressive Trees to assess the residual value of heavy construction equipment. The motivation is on the contractor side - to make the right decisions on equipment repair, rebuilding, disposal, or equipment fleet optimization to maximize the return of investment. This data – mining approach is an improvement over the traditional statistical regression methods that cannot adequately capture the relationship between the residual value of a piece of heavy equipment and its influencing factors.
- Kaya *et al* (2014) use Decision Trees and Association Rule Mining to predict construction crew productivity in the ceramic tiling industry. They explore the impact of crew size, age and experience on productivity.

It is visible that studies on forecasting utilization hours of vehicles in construction sites using data – mining approaches have not been conducted. In general, IoT and data analysis in the field of Construction seems to still be a young topic, yet it holds a lot of potential. Data – driven approaches can be used to make the construction process faster and better optimized, thus cutting costs for the investors and potential users. They can also be applied to minimize negative environmental impacts, seeing as emissions are an issue among construction fleets.

Looking at the Methodology, techniques have been used to do time-series forecasting with Machine learning algorithms, that can help us make a better decision about the algorithms to use and how to tune their parameters. However, these techniques have mostly been used in Financial prediction problems and not industrial ones.

1.7 CONTENTS OVERVIEW

Here we provide a Short Summary of the Thesis organization.

Chapter 1 offers an Introduction to the problem that we are posing and our goal: predict utilization hours of vehicles in construction sites using a data-centric approach. It provides a general idea about the solution we are providing, it states the motivation for this kind of work, the previous research done on such topics and introduces Tierra Telematics, the company with which the thesis was developed.

Chapter 2 is an exploration of the vast Dataset mostly by using different kinds of plots to give an idea about the data. It first offers an explanation of the generation of data, giving a peek into the company workflow of Tierra and then proceeds to give interesting insights on the Data. One of the goals of the chapter is also to provide reasoning behind the choices done in the Methodology. Finally, it narrows down the Data to the part which is used in the development of the thesis.

Chapter 3 discusses the Methodologies used to solve the problem. First it talks about formally formulating the problem, then about preparing the data for machine learning and finally describes the algorithms used.

Chapter 4 reports the various Results and insights got by running the algorithms. It first discusses the tuning of general parameters and offers a comparison between five machine learning algorithms and two benchmark algorithms, evaluating their performance on a selected part of the dataset. Then it reports on the hyperparameter tuning of three of the machine learning algorithms and the improvement in the performance afterwards. It also explores the results in more detail. In the end it reports on the same experiments done with a change introduced in the problem formulation and compares the two scenarios.

Chapter 5 finishes the thesis with a recap of the study, points out the most important conclusions and offers suggestions for future work.

2 THE DATA

2.1 GENERAL OVERVIEW OF THE DATASET

For the purpose of this study, as mentioned in 1.2, we used activity data coming from construction vehicles of companies that are customers of Tierra. All the vehicles that are equipped with on-board devices from Tierra, send messages to a server at certain time-intervals. To do so, they use the CAN-bus standard. CAN stands for Controller Area Network and it is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol. More specifically, inside the vehicle, the SAE J1939 protocol is used, which is a higher-layer protocol based on CAN that provides serial data communications between microprocessor systems (also called Electronic Control Units) in any kind of heavy-duty vehicles.

The available data for each vehicle is the following:

- Information from the customers: asset info, maintenance services
- Information from embedded devices: unit/asset code, timestamp, geographic location of vehicle, digital inputs report
- **Information from CAN-bus**: Engine ON/OFF, CAN parametric messages, Diagnostic Messages and Status Reports

We are interested in the messages that specify the CAN parameters. The CAN messages are generated at a high frequency (up to 100 Hz) and are gathered by a controller, where they are collected and processed. Then an aggregated report is sent to the server every 10 minutes (this reporting frequency is governed by a policy in agreement with the customer and can change). For each vehicle, this report contains data on the momentary situation in the engine and the vehicle itself, such as: Fuel level, Engine oil pressure, Engine coolant temperature, Engine fuel rate usage, Engine speed, Engine hours, Engine percent load, Digging press, Pump Drive temp, Oil tank temp etc. The data is timestamped and the number of samples taken is specified, so we can calculate the exact number of hours the vehicle was used.

These aggregated reports are kept in a database in a cloud environment. This database is a copy of the operational database and it is used for Data warehousing. We are not using the real-time database. The data in this warehouse is normalized and can be used for analytics.

Then, using an SQL-query on the CAN-messages data, the data is again aggregated on a daily or weekly basis. The result is a table with all the data we need for our study on Utilization hours. We take this table as it is on the first week of September. The Table contains 601462 records. A very small sample of it (8 records) is shown on Table 2.1., as an example of what the Table looks like.

Table 2.1: An example of the Table of CAN messages aggregated per day

Unit type	Model id	Unit id	Date	ON time [h]	Fuel consumed [l]	Latitude	Longitude	Distance traveled [m]
Single Drum Roller	1328	4397	05-08-15	9.26	109.2107	51.4311166	-2.63635191	9954
...								
Tandem Roller	1366	6003	10-07-18	7.04	32.6819	52.6654062	10.75177909	4358
Tandem Roller	1366	6003	12-07-18	6.57	43.3815	52.6631542	10.75284204	20807
Tandem Roller	1366	6003	16-07-18	5.18	17.2123	52.4396025	10.78516819	4352
...								
Refuse Compactor	1390	4138	04-02-15	0.02	0	-11.9841032	-77.1331558	0
...								
Refuse Compactor	1254	5282	17-03-17	1.82	26.3454	30.0251196	-91.9586677	312
Refuse Compactor	1254	5282	21-03-17	5.56	133.5375	30.0233106	-91.9539994	0
...								
Paver	1290	4499	25-08-16	4.24	30.8916	53.8818782	27.60290141	0
...								

As we can see, the aggregation is per day. This means that for each day that a vehicle has worked, there is one record in the table. The first column is **Unit type** – this is the name of the type of construction vehicle. There are 10 types overall, the most common one being Refuse Compactor. Other types include Single Drum Roller, Tandem Roller, Coring Machine, Paver, Recycler, Cold Planner, Grader etc. For each type of vehicle there are several **models**, for example there are 44 different models of Refuse compactors, 65 models of Single Drum Rollers, 10 models of Recyclers, 5 of Graders etc. Each model has a unique Model ID. Then, each model holds a number of vehicles. For example, there are 66 vehicles of the model 1254, of the type Refuse Compactor. The vehicle is the leaf of the hierarchy and we work on vehicle level, meaning we do the prediction

per vehicle. On Figure 2.1, we visualize the hierarchy of a Refuse Compactor, for better understanding.

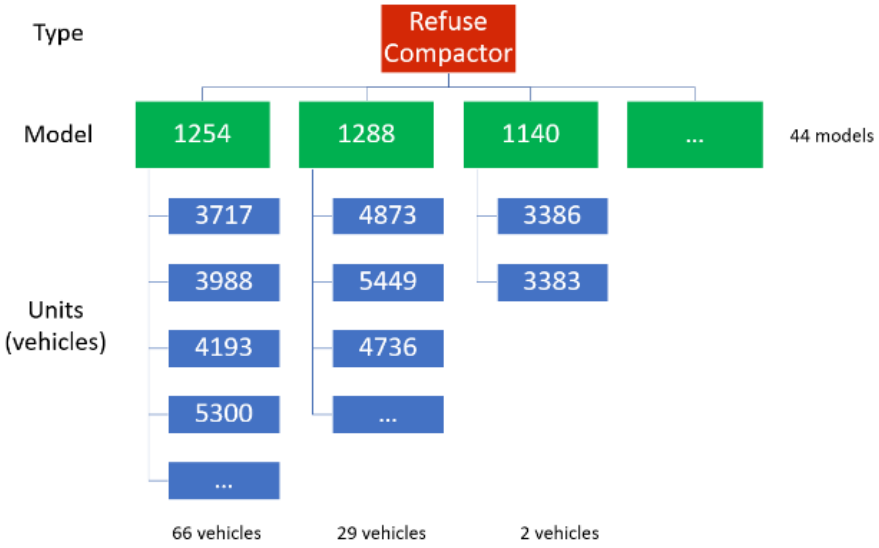

Figure 2.1: Type, model and vehicle units Hierarchy

The next field in Table 2.1 is the Date. We work with almost 4 years of data – from January 2015 to September 2018. Then we have the "ON time" in hours, which is the most important field for us: Utilization hours of the vehicle. This is what we use as historical data and the target of our prediction problem. Then, there is the Fuel consumed that day in liters and the estimated position of the vehicle that day – it is a mean of all the positions where the vehicle was that day, since in most of the cases it was moving around. For our problem, we do not need a precise position of the vehicle, just the country and city, so we can include local information, like the holidays in that country or other country-specific data. The distance traveled (in meters) is a field that can be used to find out if the vehicle was ON, but it was not moving or some other potentially suspicious behaviour.

Using the position data, we can take a look at the geographical location of the vehicles. Figure 2.2 shows a map of the vehicles working on 07.11.2017, using an average of all their locations from that day.

Figure 2.2 Positions of all vehicles active on 07/11/2017

We see a high concentration of vehicles in Europe, especially in Germany. Then there is a lot of vehicles in the USA, especially the east coast and also a high number is visible in Peru. The colours of the circles indicate the number of hours they worked that day, according to the legend shown in the top right corner of the Figure. Most of the dots are green and yellow, meaning they worked between 0 and 10 hours. This includes all the vehicles of all models of all types.

Next into the Chapter, we dive into the dataset and explore it on various levels. First, we analyze the amount of data we have per vehicle, then we explore the patterns of utilization among different types, models and finally vehicles. The analysis is supported by tables and numerous plots.

2.2 AMOUNT OF USEFUL DATA

The first step in exploring the data is to see how much data we actually have. On Table 2.2, we specify, for every type:

- No. vehicles: how many vehicles there are (counting all the models)

- No. records: how many records there are (from all the vehicles from all the models) in this Unit type – these are all the days they have worked
- Average data per vehicle: This is the important metric – how much data we actually have on the vehicles. It is simply the number of records divided by the number of vehicles.

What we need is good "density" of data, amount of data – since we do the prediction per vehicle, we need a lot of data on vehicle level. An average of 492 records per vehicle means that there is data on 492 days. What is interesting to notice here is that the unit type with most records is the Single Drum Roller (182620 records), but the one with most data per vehicle is the Refuse Compactor.

Table 2.2 Amount of data per unit type

Unit type	No. vehicles	No. records	Average data per vehicle
Refuse Compactor	311	153240	492
Grader	8	2454	306
Single Drum Roller	646	182620	282
Tandem Roller	412	99524	241
Material Feeder	8	1788	223
Paver	206	43710	212
Rubber Wheel Roller	10	1970	197
Recycler	114	20841	182
Cold planner	522	95216	181
Coring Machine	2	99	49

However, this average is on all vehicles of type Refuse Compactor. If we make the same kind of table, but grouping by Model ID, we can find some models that contain more data.

On Table 2.3, we can see a part of this table that is grouped by Model ID, showing the models with most data. For each Model ID, we can see the type of vehicles, the number of vehicles, the number of records found for that vehicle and the metric of Average data per vehicle. We again sort by the Average data per vehicle.

Table 2.3 Amount of data per model

Model ID	Unit Type	No. vehicles	No. records	Average data per vehicle
1269	Refuse Compactor	9	8037	893
1262	Refuse Compactor	3	2367	789
1199	Refuse Compactor	2	1534	767
1114	Refuse Compactor	4	2899	724
1133	Refuse Compactor	1	714	714
1295	Refuse Compactor	2	1426	713
1255	Refuse Compactor	1	697	697
1175	Refuse Compactor	2	1387	693
1116	Refuse Compactor	8	5299	662
1218	Refuse Compactor	1	624	624
1129	Refuse Compactor	5	2901	580
1316	Refuse Compactor	16	9261	578
1253	Refuse Compactor	2	1153	576
1200	Refuse Compactor	3	1716	572
1151	Cold planner	1	566	566
1254	**Refuse Compactor**	**66**	**35424**	**536**
1089	Cold planner	1	529	529
1312	Single Drum Roller	6	3171	528
1289	Refuse Compactor	23	11870	516
1168	Grader	1	512	512
1250	Refuse Compactor	13	6455	496
1179	Cold planner	1	485	485
1348	Grader	1	484	484
1288	Refuse Compactor	29	13796	475
1434	Single Drum Roller	1	473	473
1195	Refuse Compactor	13	6144	472
1331	Refuse Compactor	15	6851	456
1292	Refuse Compactor	7	3055	436
1343	Tandem Roller	1	410	410
1165	Cold planner	1	405	405

It is visible now that we have found a higher density of data, since we can see higher numbers in the last column. What is interesting to notice is that most of the models with high Average data per vehicles are Refuse Compactors. This gives us an incentive to work more closely with Refuse

Compactors. We later on take particular interest in the model with ID 1254 of the Refuse Compactors, since it shows a good trade – off between the three columns: the number of vehicles, the absolute number of records and the average data per vehicles. It holds a lot of vehicles and also has good amount of data per vehicle.

Another important aspect about the amount of data is that it changes over time. One way in which this can be explored is by checking how many vehicles are active at a certain time interval over the whole time that we have data. Figure 2.3 shows exactly that – a monthly representation of the number of vehicles that have some records of being used that month.

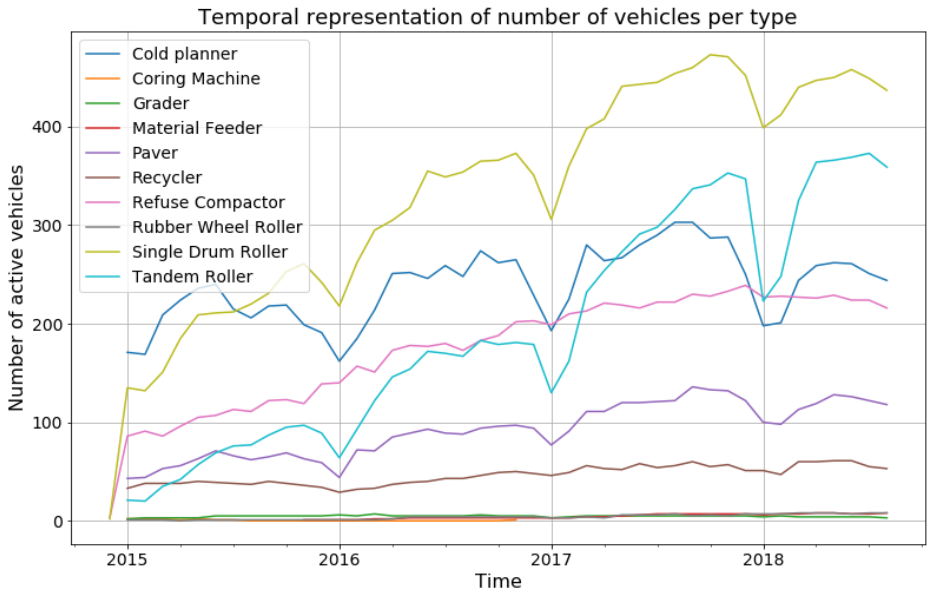

Figure 2.3 Change of number of vehicles being used by each type in time

A few things are notable from the plot. First, the number of vehicles being used changes over time and there is an increasing trend. This means more and more vehicles of each type are being used. Second, in most of the types there is a drop of the number of vehicles used in January. This is probably due to holidays and cold weather which are factors that affect open-sky construction sites. Interestingly, these drops are not present for the types Refuse Compactor and Recycler, probably due to the fact that they handle waste, which needs to be collected at all times. Another thing is

that some types have a really low number of vehicles present throughout all the three years. These types are Coring machine, Grader, Material Feeder and Rubber Wheel Roller, but this data is also visible in Table 2.2.

2.3 EXPLORATORY DATA ANALYSIS

2.3.1 Analysis on the different types of vehicles

The next thing in the analysis is to explore the utilization hours of vehicles on Type level. This means examining how the different types behave, considering all their vehicles, so using all the data we have in the dataset.

On Figure 2.4, we can see the CDFs (Cumulative Distribution Functions) of the usage of the different types of vehicles. The CDF of a real-valued random variable X is given by the function:

$$F(x) = P(X \leq x)$$

where the right-hand side represents the probability that the random variable X takes on a value less than or equal to x. On the plot, the x-axis holds x, all the possible utilization hours and the y-axis represents this probability given by $F(x)$, where for us the random variable X is the utilization hours of every day of every vehicle of a certain type.

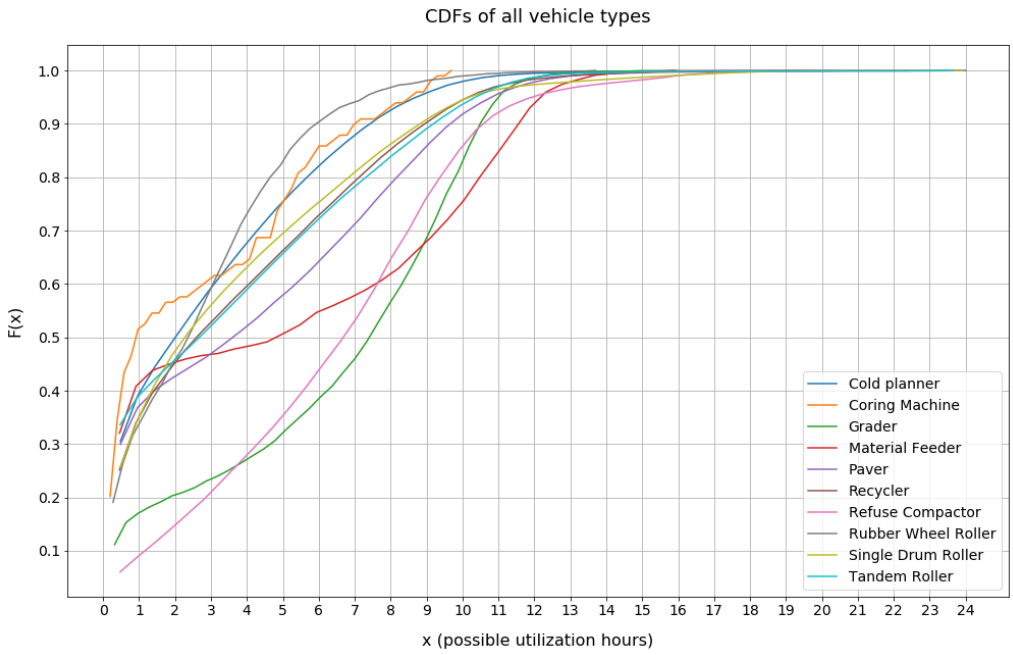

Figure 2.4 CDFs of utilization hours of all vehicle types

Looking at the CDFs of the utilization hours of the different types, we can clearly see the average usage patterns. The types Grader and Refuse Compactor are being used with the highest number of hours, while the Rubber Wheel Roller is used the least number of hours. Some of the types expose a long tail in the CDF, meaning there are days where they have been turned ON for 24 hours. These situations are very rare and they could be true or they may be outliers in the data, which can be accounted as errors in the data collection. Overall, the number of records that exceed 20 hours in the dataset is 384 out of 601462, which is 0.064% of the data. This is a negligible percentage. What is more important is that a lot of the types start with a steep slope, meaning they have a lot of values close to 0. We can see some types not passing the 1-hour threshold even until the 40^{th} or 50^{th} percentile. If we check this empirically, it turns out that 30% of the overall records show a value of utilization hours lower than 1. One type that does not show this kind of behaviour is the Refuse Compactor, where indeed, only 9.14% of the data shows a value of utilization hours lower than 1 hour, which results in the flatter CDF curve.

On Figure 2.5, we can see the same data, but on a boxplot. Each type is represented by a box. The bottom x-axis labels the name of the vehicle type, while the top x-axis labels the number of vehicles

of that type present in the dataset. The box plot (also called box and whisker diagram) is a standardized way of displaying the distribution of data based on the five-number summary: minimum, first quartile, median, third quartile, and maximum. The central rectangle spans from the first quartile (Q1) to the third quartile (Q3) – meaning from the 25th percentile of the CDF to the 75th percentile. This is called the interquartile range or IQR. The orange line inside the rectangle shows the median value. The vertical lines above and below the rectangle are called whiskers and, in this plot, and all other boxplots that are shown in this study, they span from the first data point greater than Q1 - 1.5*IQR to the last data point less than Q3 + 1.5*IQR, where IQR = Q3 – Q1 and Q1 and Q3 represent the first and third quartile respectively. The dots represent all values outside of these limits – the outliers.

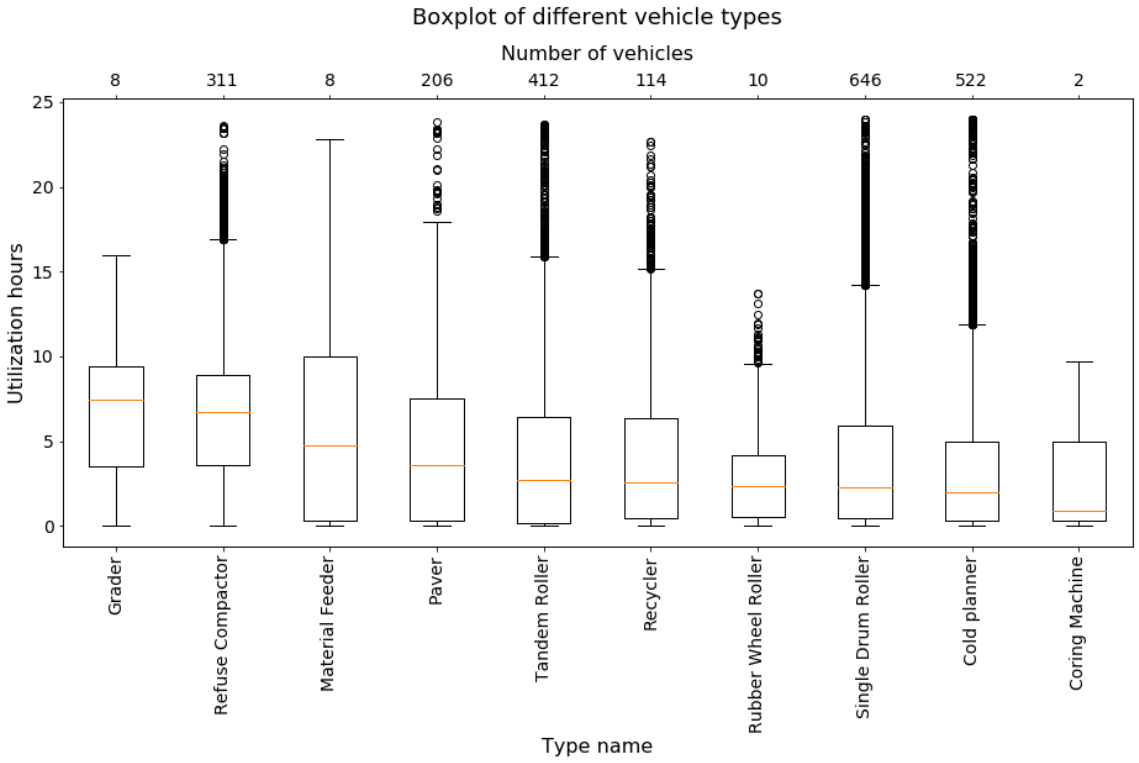

Figure 2.5 Boxplot of the distribution of utilization hours of all vehicle types

The boxes are ordered in descending order according to the median value. What is interesting is that most of the models' boxes are placed low on the scale and have the median at under 5 hours

of usage per day. This is not the case only for the Grader and Refuse Compactor, but the Grader only has 8 vehicles, which is a very low number in comparison to the Refuse Compactor. No similar pattern is visible among the different types, they each tell their own story. A lot of the types also present outliers but only as surprisingly high values with respect to their distribution.

Figure 2.6 shows a temporal representation of the utilization hours per type. It is the average monthly utilization hours. The value is obtained as the sum of all records of all vehicles of that type during the month divided by the number of vehicles of that type active in that month. For example, the Refuse Compactor evolves around 120 hours, which means that a vehicle of the type Refuse Compactor works for 120 hours per month on average.

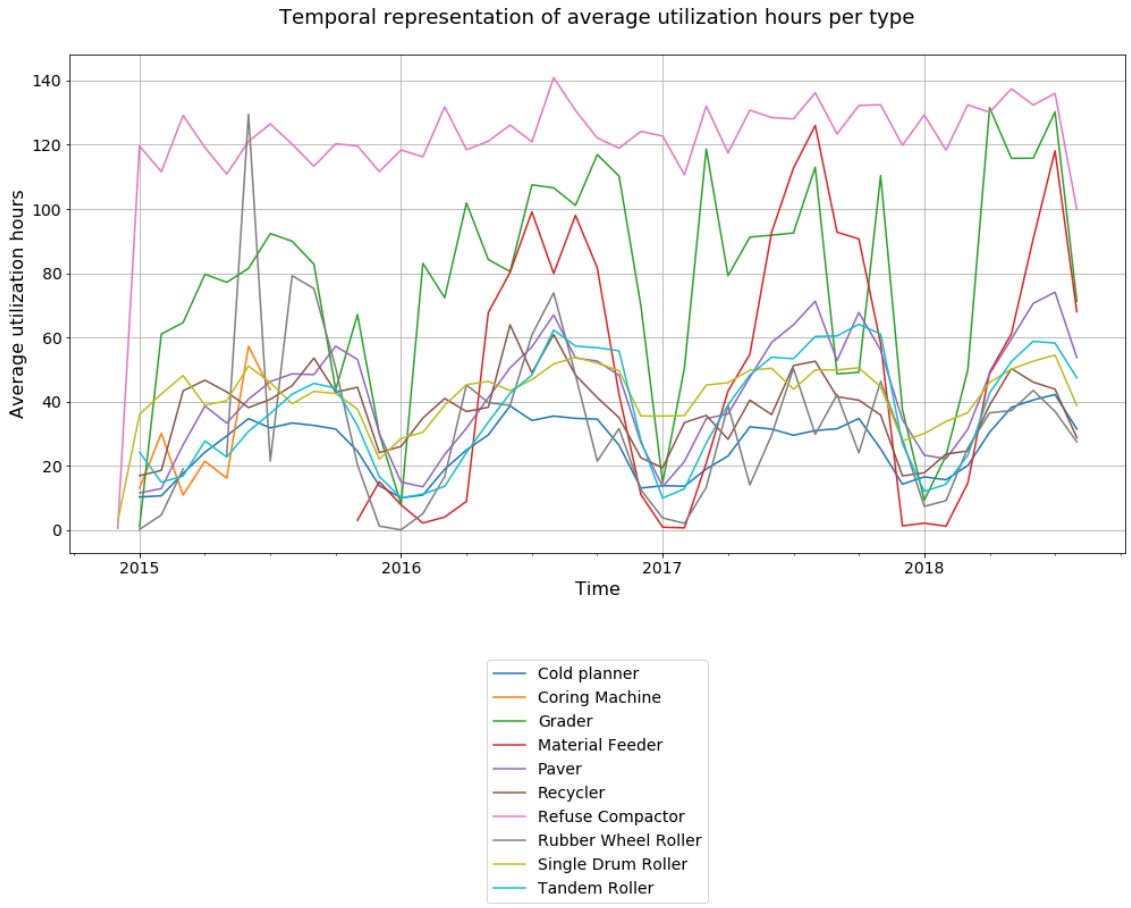

Figure 2.6 Average utilization hours per vehicle per month

From Figure 2.6 it is visible that most of the types present a seasonality during the year. They have a really low or no utilization during the winter months of December, January and February and highest utilization around the summer months. Refuse Compactors, with respect to the others, do not present a clear visible seasonality nor trend. We have to mention that this plot is done on all vehicles of a model, some of which are situated in the southern hemisphere, where the summer and winter are not at the same time as in the northern hemisphere. However, their percentage is really small with respect to the other vehicles, so we can still draw this kind of conclusions.

2.3.2 Analysis on the different models of vehicles

In this chapter, we dive inside the different types of vehicles and take a look at all the models that comprise them. For the sake of repetitiveness, we only present plots of the models of a Refuse Compactor and a Cold planner, as the most used and least used type, respectively.

On Figure 2.7, we can see a boxplot of the utilization hours for all 44 models of the type Refuse Compactor, sorted in ascending order according to the median. This is the type that is mostly used according to all previous data analysis. The top x-axis labels the number of vehicles of the corresponding model. It is obvious that most of the models have a really low number of vehicles. Only 9 out of 44 models have more than 10 vehicles. However, most of the models are being regularly used, since their rectangles are not close to 0.

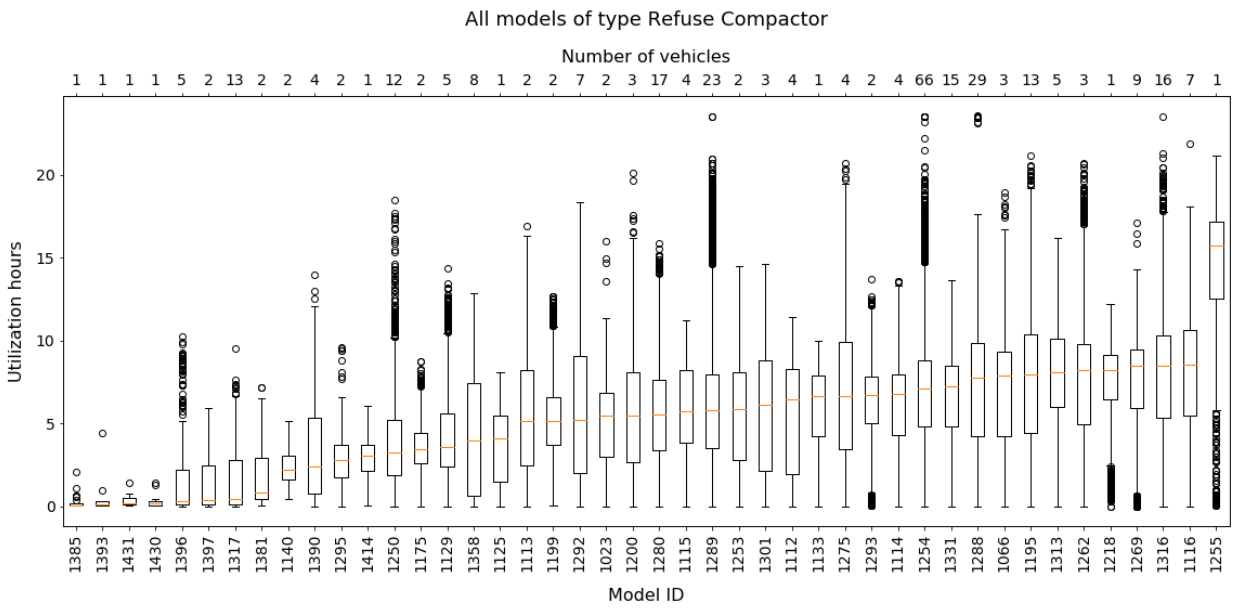

Figure 2.7 Boxplot of the utilization hours per model for all models of type Refuse Compactor

Another noticeable thing is that there is no pattern among the different models. They have different rectangle sizes, different whisker sizes and a different number of outliers. But it is not a difference that we can exploit, but rather a random one. This tells us that, looking at the different models, we cannot infer anything about their utilization hours. Dividing all the vehicles of type Refuse Compactor in models is not an indicator on how they will behave in utilization.

Going back to the number of vehicles, the model with the highest one is the model with ID 1254. We later on take particular interest in the vehicles of this model.

On Figure 2.8, on the other hand, we can see one of the least used types, the Cold Planner.

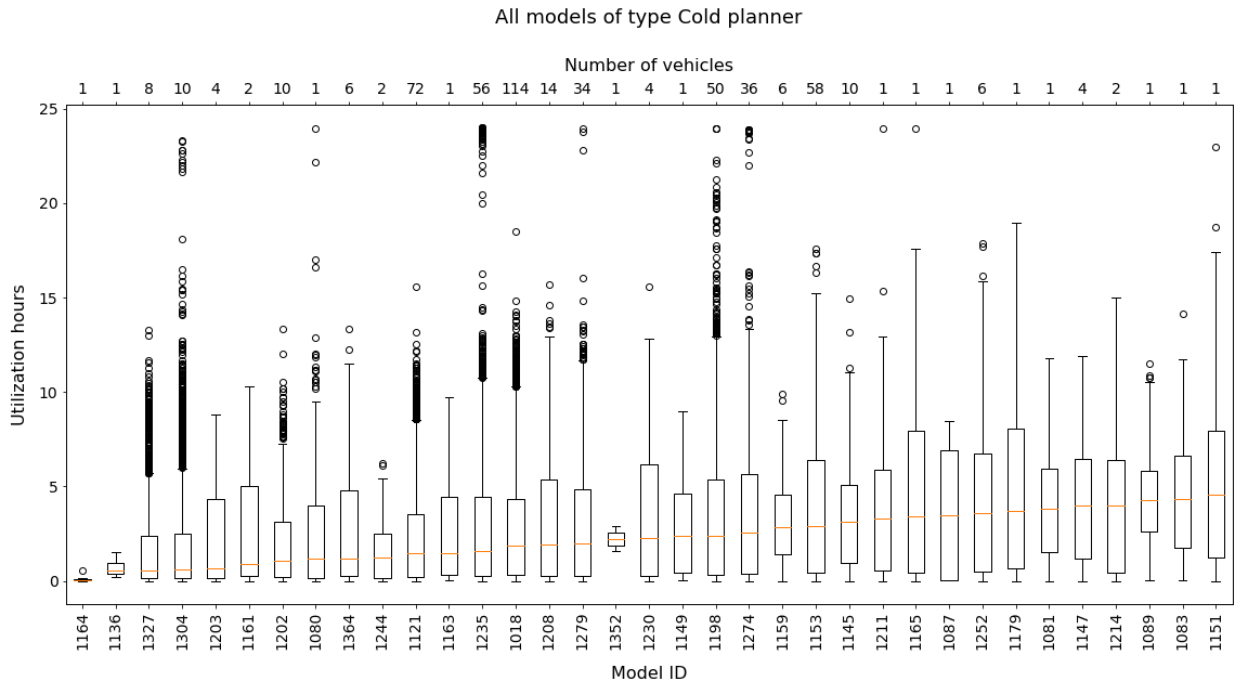

Figure 2.8 Boxplot of the utilization hours per model for all models of type Cold planner

The Cold Planner also has a lot of models, 35 in total, and also has a lot of models with a low number of vehicles, but it has more vehicles in total (522 vehicles versus the 311 of Refuse Compactor). What we see here is that almost all the rectangles are close to 0 and very low medians. This confirms the previous conclusions about the low utilization of this type. Also, there is no particular similarity or particular difference between the models.

2.3.3 Analysis on the different vehicles

In this section, we concentrate on the utilization per vehicle, taking as an example the vehicles of the Model 1254 of type Refuse Compactor. As mentioned before, this model is chosen because it has a lot of vehicles and a good amount of data per vehicle. We explore the utilization of every vehicle using different types of plots.

On Figure 2.9, we can see a boxplot of the utilization of all 66 vehicles of this model, ordered in ascending order by median. Here we can see, that the vehicles also differ a lot between themselves in terms of how much they are used. Some of them have short IQRs (rectangles) and some have really long ones. Also, the position of the median inside the IQR changes. What is different here than on Model level is that there are a lot of outliers on the bottom, meaning the values close to 0 are rare. They are outliers and not general behaviour. The overall conclusion is that every vehicle tells its own story.

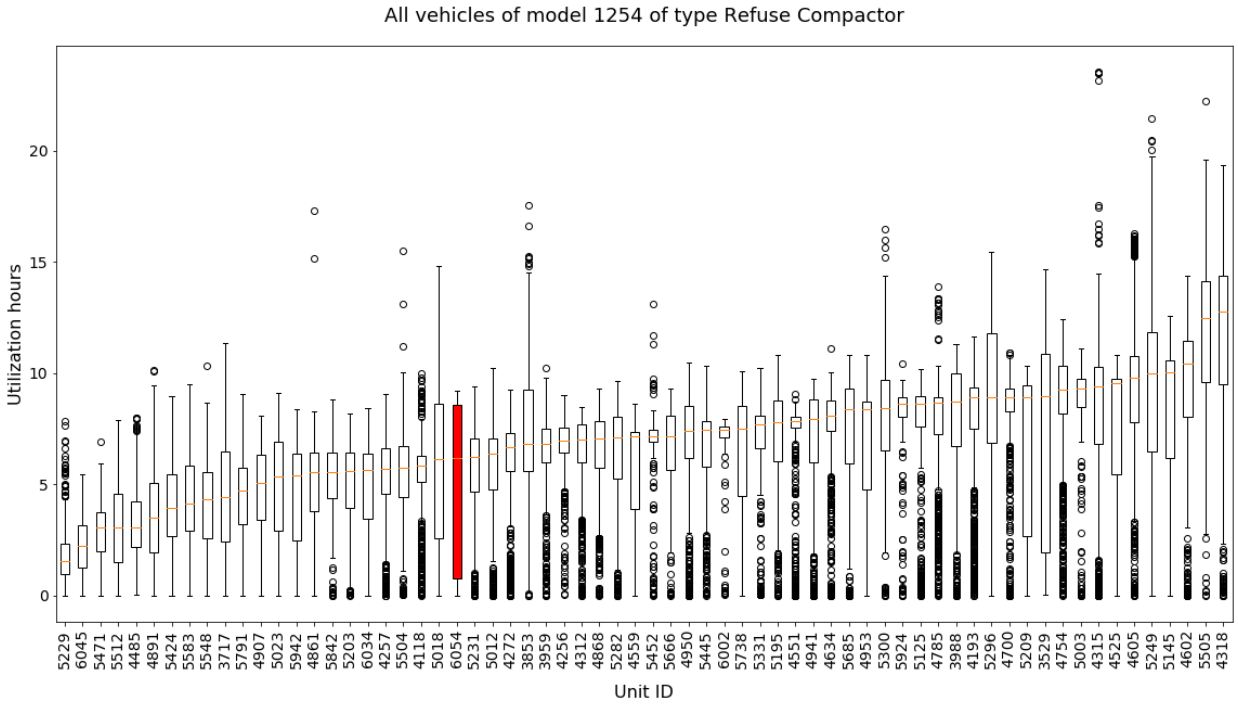

Figure 2.9 Boxplot of the utilization hours of all vehicles of model 1254 of type Refuse Compactor

Not all 66 vehicles have enough data to be representative. To make a prediction, which is our goal, we need a certain amount of data to be able to tell how the vehicle will behave in the future. For reasons explained in Chapter 3: Methodology, we choose a threshold of 135 days as the minimum data required to obtain a valuable prediction. All the vehicles that do not meet this requirement (do not have data for more than 135 days), are marked red in Figure 2.9. In this case it is only one vehicle, 6054. Its behaviour does not seem special with respect to the other vehicles, it seems as if

it were randomly picked. The real reasons why some vehicles are not used enough to gather data are probably company-specific. From this point on, we remove this vehicle and all the remaining plots and analysis are done without it.

The next three Figures take a better look at these 65 vehicles. Figure 2.10 is a histogram of the utilization hours of all the vehicles of the model.

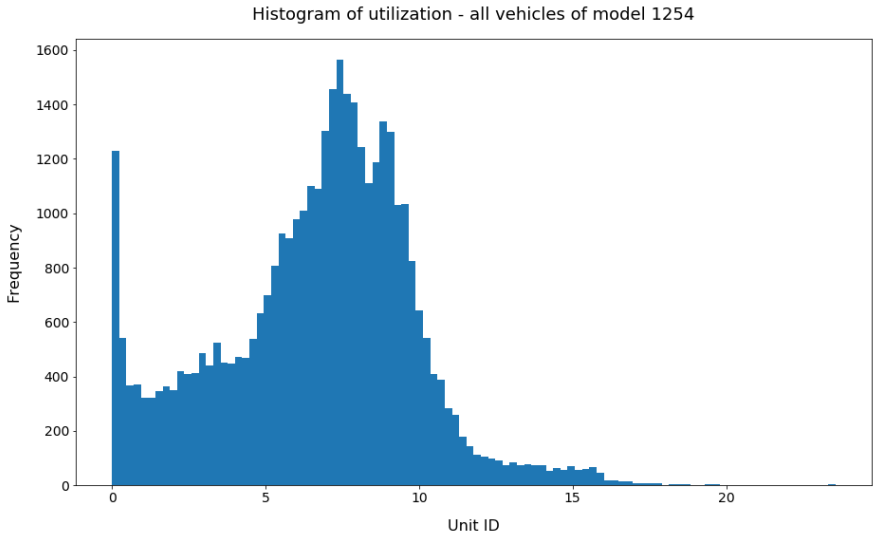

Figure 2.10 Histogram of the utilization hours of vehicles of model 1254 of type Refuse Compactor

What can be observed here is that the distribution is not simple. It seems like a sum of two or three distributions and there is also a peak between 0 and 1 hours. The mean is 6.67 hours and the standard deviation is 3.2 hours. If we try to fit the histogram to the normal distribution, using the Kolmogorov–Smirnov test for goodness of fit, we get 0.82 of certainty.

Figure 2.11 shows the same data, but on a violin plot. A violin plot is just a histogram (a smoothed variant like a kernel density) turned on its side and mirrored. This kind of plot allows us to see the distribution of every vehicle separately and not only the quartiles, like the boxplot. The width represents the frequency of samples and the line in the middle of the violins is the mean. In our case, what can be observed is that, generally, the vehicles with low utilization (left part of the plot), have distributions that are more spread and tend to be multimodal. The vehicles in the middle have

much tighter distributions and the mean is generally in the lower part of the distribution. This means they have vastly spread low values and then a peak at high values. The vehicles in the right part of the plot, the ones with higher utilization present both of these characteristics.

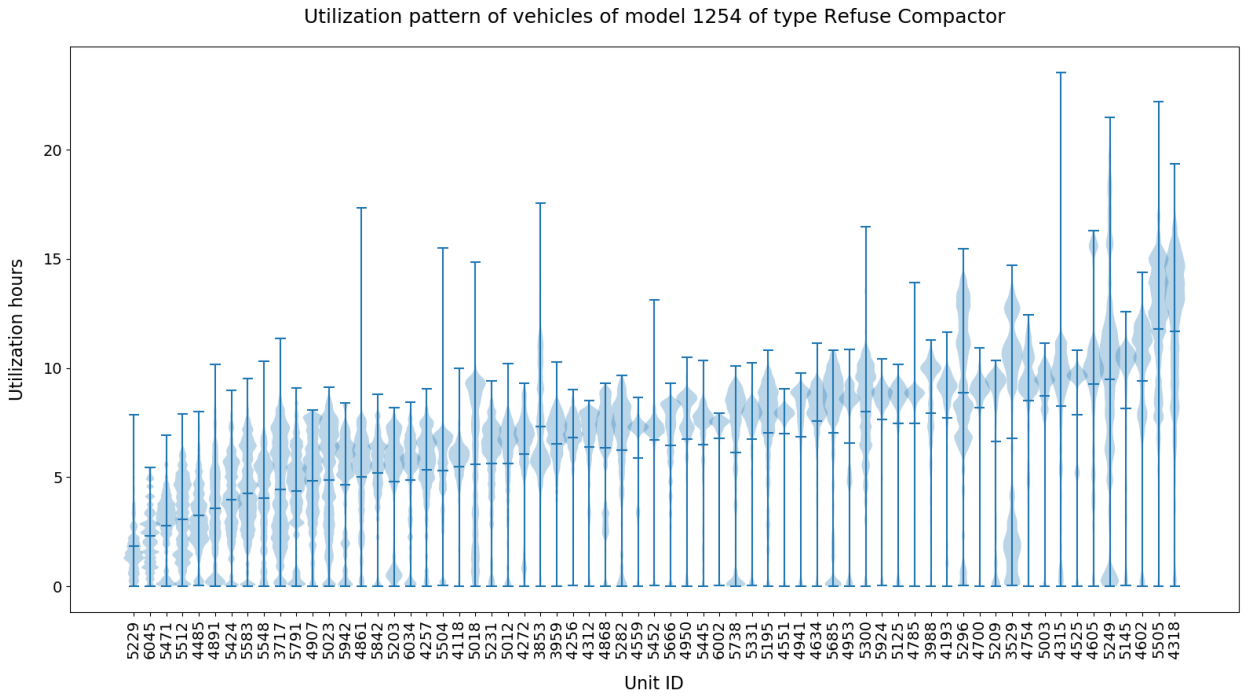

Figure 2.11 Violin plot of utilization hours of vehicles of model 1254

The next basic plot to observe the distributions of the different vehicles is the CDF plot. On Figure 2.12, we present the CDFs of the same 65 vehicles of model 1254.

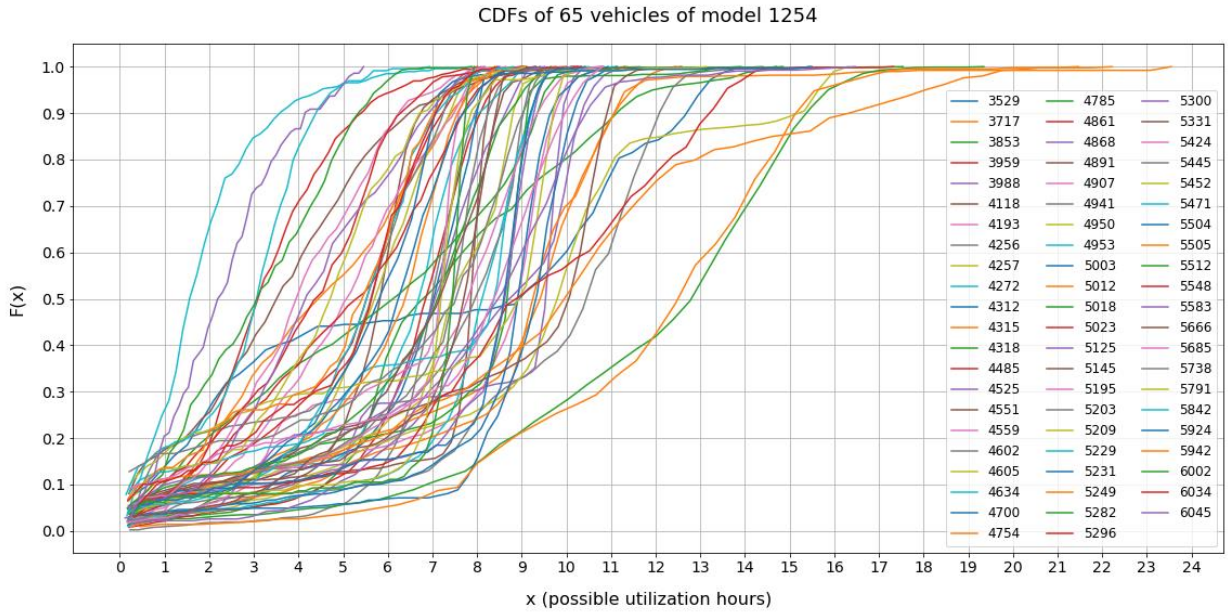

Figure 2.12 CDFs of utilization hours of the 65 vehicles of model 1254 of type Refuse Compactor

It is obvious that the vehicles are used in different manners. There are those on the left side of the plot that are less used and those on the right side that are more used. The CDFs in the middle show some similarity in the way the vehicles are used.

2.3.4 Utilization hours of a vehicle as a time-series

While characterizing the types, models and vehicles in terms of distribution of utilization hours is extremely important for us and gives us a deep insight into the problem we are trying to solve, how we model the problem in the end is by observing the utilization hours in time. The next plots show the usage of the vehicles in time and model them as time series.

Figure 2.13 shows the daily utilization hours of 6 randomly picked vehicles from the model 1254. Here we can observe a few things. First, every vehicle has a different behaviour in terms of the number of hours it is being used per day. Second, every vehicle starts collecting data at a different time. We can see vehicle 4318 (green curve) joining around March 2015, vehicle 4605 (red curve)

joining in September 2015 and vehicle 5145 (purple curve) joining even later, around July 2016. What is common between all joining later is that they have a short period of working only a few hours or not working at all before they start their usual pattern. This is because the device that collects data is turned on and tested in a safe environment, before actually being mounted on the vehicle.

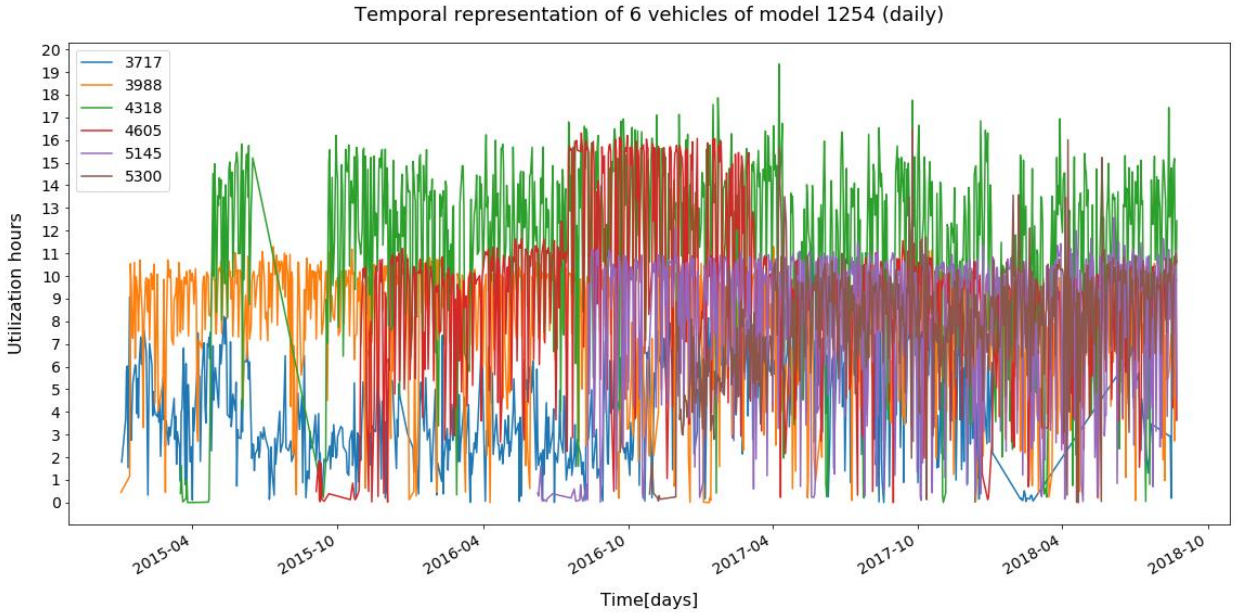

Figure 2.13 Daily utilization hours of 6 vehicles of model 1254 as a time series

Next, some vehicles have data missing in the middle of the time period, for example the vehicle 4318 (green curve) takes a break between July and October 2015. It is interesting that when they start working, they mostly follow the same pattern, except 4605 (red curve) that has a jump for a few months in late 2016 and early 2017. This regular usage was also visible on Figure 2.6 of the previous section, as a characteristic of the Refuse Compactors. Lastly, the data on daily granularity is very noisy.

On Figure 2.14, we can see the same data (utilization hours in time), on the same 6 vehicles of model 1254, but this time the granularity is weekly. The hours per vehicle are summarized per week, so the values on the y-axis are higher.

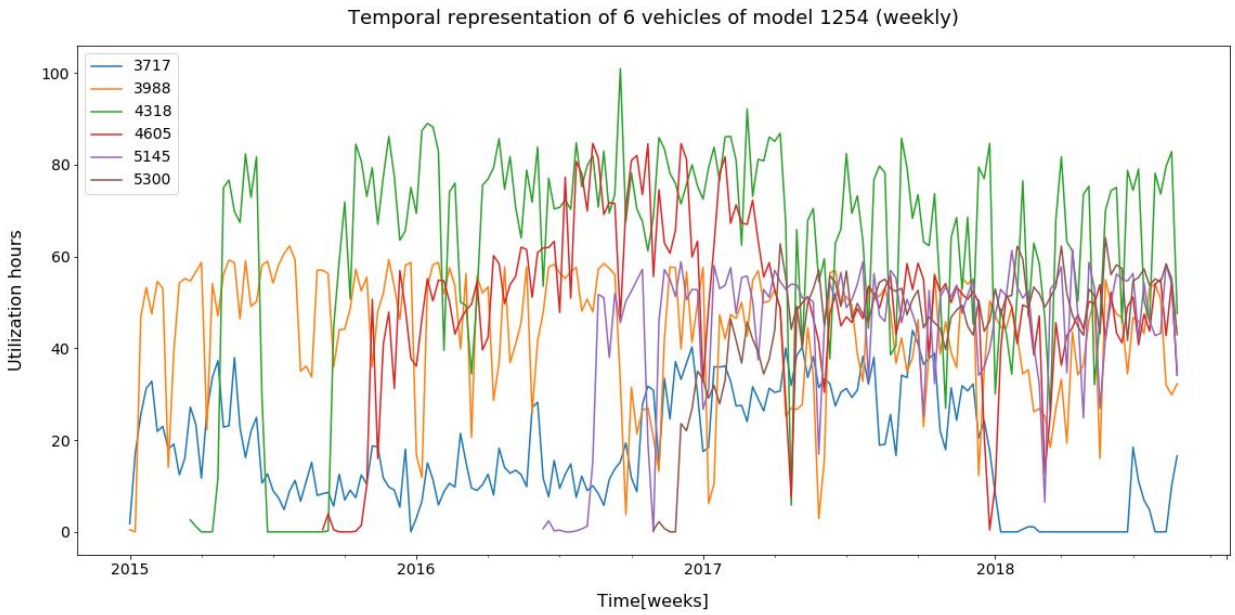

Figure 2.14 Weekly utilization hours of 6 vehicles of model 1254 as a time series

This is simply a "clearer" version of Figure 2.13 and we can more easily see the pattern of utilization between the different vehicles. It is still noisy, though not so much. Here we can see that vehicle 3717 (blue curve) has a jump in utilization in 2017 and is shut down in 2018. Also, the curves of vehicles 4605, 5145, 5300 and 3988 all seem to overlap for the better part of 2017.

In time-series analysis it is common to decompose a time series into four components:

- Level – the baseline value for the series if it were a straight line
- Trend (optional) – an increase/decrease over time
- Seasonality (optional) – repeating patterns or cycles of behaviour over time
- Noise (optional) – variability in the observations that comes from externalities

$$y = level + trend + seasonality + noise$$

Where y is the time-series. These components are often used to model a time-series using traditional statistical methods. They may or may not be important for the forecasting of future values. However, it is always useful to characterize a time-series using them.

Looking at our examples from Figure 2.13 and Figure 2.14, we can see that our time-series (seen one by one) have a somewhat stable level, they do not present a visible trend, nor an obvious seasonality, but they do present a lot of noise – they are very random.

However, the time series as seen until now is unevenly spaced – there are no values present for every day, but only on those days when a vehicle was active. It is normal that a lot of vehicles are not active during the weekend, holidays or just randomly not active some days.

A common approach to analyzing unevenly spaced time series is to transform the data into equally spaced observations using some form of interpolation. In our case, we suppose that when there is no record in the data, the vehicle was not active. So, to make it an equally spaced time series, we add zeros as the utilization hours on the days that are not present in the dataset. This is done in the following manner: we take all the raw values of utilization - from the beginning when the vehicle started collecting data until the moment it stopped collecting data and we only fill zeroes in the middle. This does not mean that now every vehicle has 3 years and 10 months' worth of data, but it depends on when it started / ended working. Different vehicles have different number of samples, since they start or end collecting on a different date, but they are all equally spaced. Most of the vehicles are still collecting data, so their end date is in August 2018.

On Figure 2.15, we can see the time series of vehicle 4318, constructed in this manner.

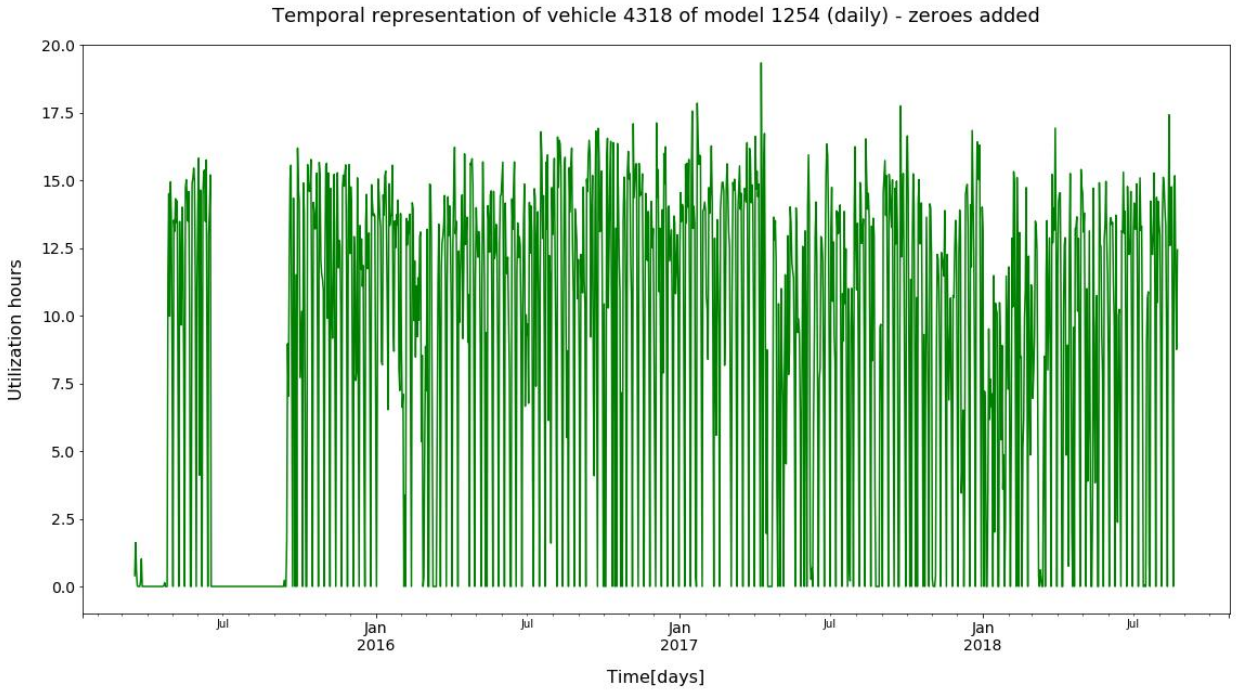

Figure 2.15 Time series with added zeroes - equally spaced: Utilization hours of vehicle 4318

It is obvious that now the zeroes prevail the time series. The values are not spread. They are either zero or above 10 hours (mostly). That means that when the vehicle works, the hours are pretty regular, but often it does not work.

Figure 2.16 is a violin plot of the distributions of the data that now includes zeroes. It is done on all vehicles of model 1254. Looking at this figure, we can see that a higher density is now present around 0 utilization hours. They all become multimodal and their peaks are somewhat drowned due to the zeroes. This is also visible if we compare it with Figure 2.11.

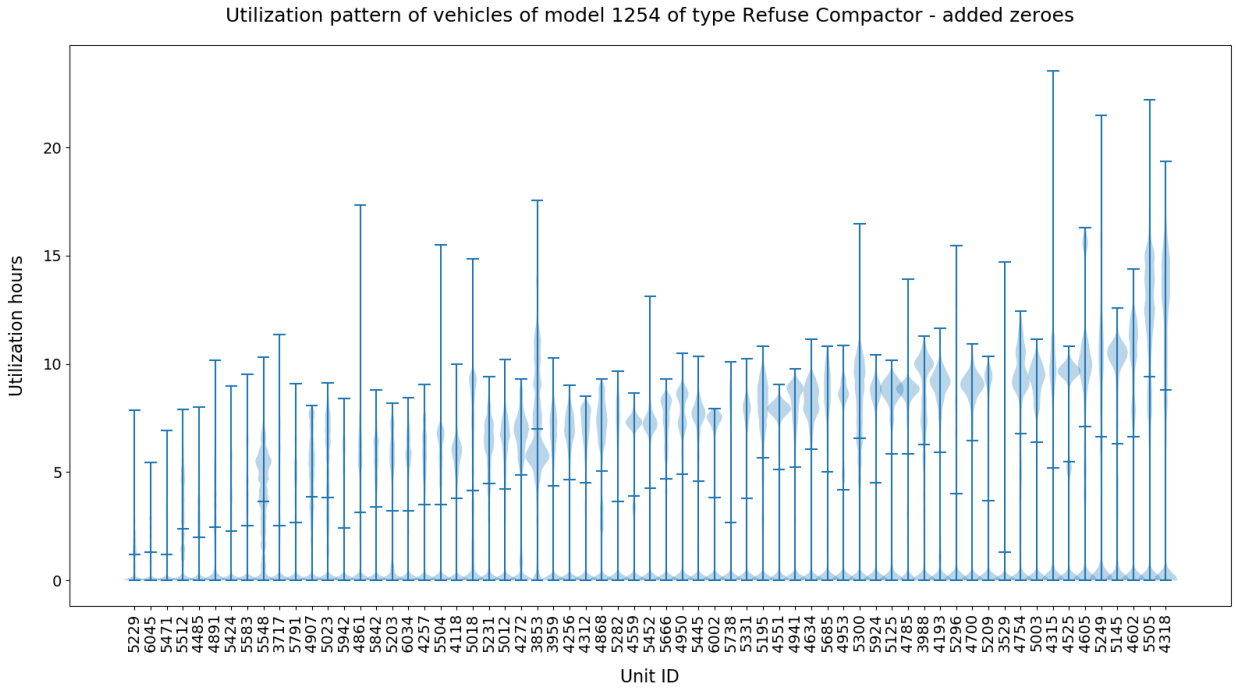

Figure 2.16 Violin plot of the utilization hours of all vehicles of model 1254 with added zeroes

2.4 CONCLUSIONS FROM OBSERVING THE DATASET

In the previous few chapters we explored the dataset in depth, starting from the highest level – the type, to the lowest, most spread level – the vehicle. This analysis led to a few important conclusions. The first takeout is the diversity of the data. Every type of vehicle is unique in the way it uses the vehicles. Every vehicle is different, even if they belong to the same model or the same type. This classification of vehicles in models and types, even though normal and very logical for other things, does not say much about their utilization hours.

This leads us to the decision to treat all the vehicles separately and try to solve our problem vehicle by vehicle. We devise a separate prediction model for each vehicle and train it for that vehicle only. We predict the utilization of a certain vehicle using data only from that same vehicle.

This means that we need a lot of data **per vehicle**. Even though in general we have a lot of data, data on specific vehicles is sparse. The only type that has acceptable rates for data per vehicle, as shown on the Tables and Figures from Chapter 2 is the Refuse Compactor. That is why in the next chapters, all experiments are done on vehicles from this type, with special attention to model 1254 as the model with the highest number of vehicles.

Another thing that has to be addressed is the sampling frequency of the dataset. When we add zeroes to make it an equally spaced time series, we change the whole structure of the data and we monopolize it with zeroes. That is why in Chapter 3: Methodology we define two scenarios, or two points of view for our problem, that have a direct effect on the results.

3 METHODOLOGY

In this chapter we talk about how we formulated the problem conceptually, represented it mathematically, how we adapted it to fit a machine learning problem and which algorithms we used to solve it. Throughout the whole development of the thesis, the software used was Python, with its pandas and numpy libraries for data handling, scikit-learn for the machine learning part and matplotlib for plotting.

3.1 PROBLEM FORMULATION

As mentioned in the Introduction, our problem is: Given the usage pattern of a vehicle in the past days, predict the utilization of the same vehicle for the next time instant (the next day/ next working day).

Let us represent the model mathematically:

$$y = g(x_{t-1}, x_{t-2}, x_{t-3} \dots x_{t-n}, \quad f_1, f_2 \dots f_l)$$

(Equation 1)

Where:

- y is our target variable – the prediction
- x_i (i = t-1 ... t-n) are values of past observations, n is the window size of past observations we use to do the prediction (they are known), t signifies a time instant
- f_i (i = 1 … l) are additional values (features) connected to y that can help us reach a better prediction, l is the number of additional features we include in the model

We approach the problem in 2 different ways based on what we want to predict, and they define x and y differently:

1. Predict usage on next day

Here vector $\boldsymbol{x}$ is a time series with zeroes included for the days where there is no data – an equally spaced time series. The target variable $y = x_t$, which is the next calendar day.

2. Predict usage on next working day

Here vector $\boldsymbol{x}$ is an unevenly spaced time series, comprised only of the days where there is data present in the dataset and this data marks utilization hours > 1h. 1 hour was chosen as a threshold of time above which we can consider that the vehicle was really working. It was an arbitrary choice decided along with professionals from Tierra. The target variable is $y = x_t$, which is the next day the vehicle is going to work (it supposes we know which calendar day that would be).

The additional features f could be anything that we feel is connected to the target variable y. They could be mathematical operations of the vector $\boldsymbol{x}$, like an average, minimum, maximum etc. of the past values. They could also be information that we know on y: Is y a working day? Is it a holiday? What will the weather be like on that day? What is the season of the year?

Then there is the function g. It is a function that describes the relationship between the variables x,f and y and tries to approximate y. What kind of function it is depends on the algorithm we use for prediction. In linear regression it is simply a weighted sum. In the other algorithms the situation is more complicated. The algorithms are explained in detail later in this Chapter, in Section 3.4.

On Figure 3.1, we visualize the model of the problem. The scenario explained is the first one – Predict usage on next day, but the model is the same for the second scenario, just instead of "tomorrow", we would have "next working day".

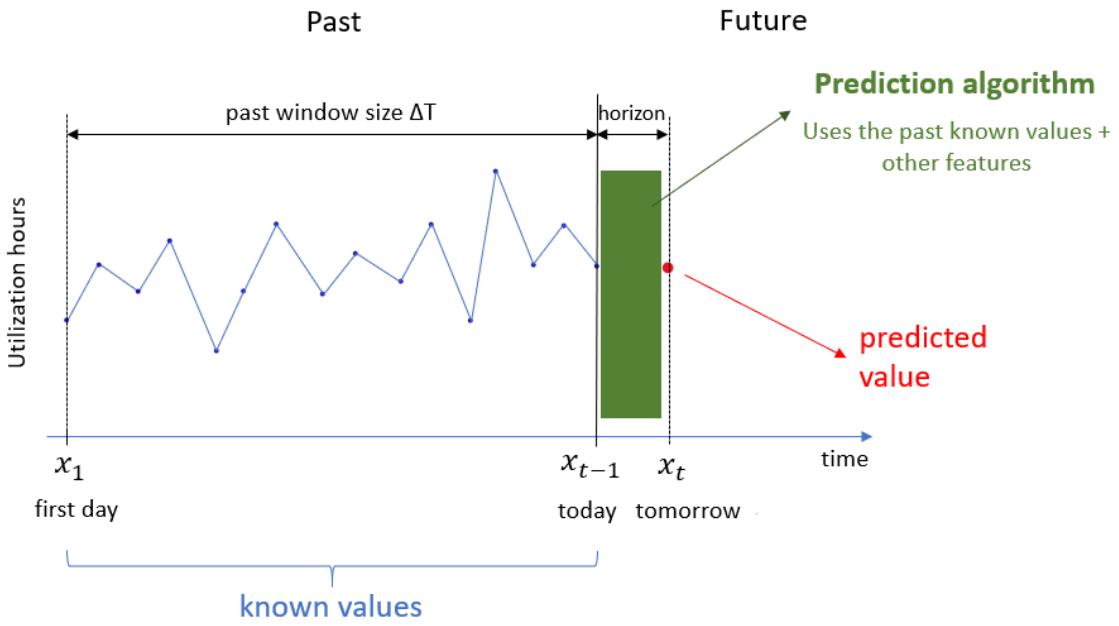

Figure 3.1 Sketch of the problem

As seen on Figure 3.1, there are some parameters in the model that need to be fixed, like the past window size and the horizon. The horizon can be seen as a kind of "gap" measured in number of time instances (in our example days) between the known data and the data to be predicted. In our problem formulation, the horizon is 1, since we only predict one time instant away in the future (next day, next working day). This also means that we have one predicted value per window of past values. The past window size, on the other hand, can vary and its length is discussed in more detail in Chapter 4: Results.

3.2 TIME SERIES TO SUPERVISED MACHINE LEARNING PROBLEM

To be able to use machine learning (ML) to solve the problem, we have to adapt it to fit the concept. Our problem falls in the category of supervised learning problems and in particular, a regression problem, since we want to predict a continuous variable. In this Section we introduce the concepts of supervised learning and regression and how we turned our time series into an input for a machine learning algorithm.

Within the field of ML, there are two main types of tasks: supervised, and unsupervised.

- **Unsupervised learning**: we have a set of data points and we want to find out if there is some natural structure present in the data – to find patterns. The algorithm does not know if it is right or wrong, it just tries to identify commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data.
- **Supervised learning**: we use a ground truth, we have prior knowledge of what the answers (output values) should be. The goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data.

There are two main types of problems that concern supervised learning:

- **Classification** – we want to predict a categorical variable, or a class. Our dataset is labeled with different classes and we want to be able to predict the class of unseen data.
- **Regression** – we want to predict a continuous variable. It is basically a statistical approach to find the relationship between variables and predict an outcome of an event based on that relationship. We will talk more about Regression and its algorithms later in Section 3.4: Regression algorithms

If we go back to Section 3.1: Problem formulation, we can see that our devised formulation fits this description and what we want to find out is the function g. So, our problem can be treated as a Regression problem.

3.2.1 Vectorization of the problem & Feature Construction

To use machine learning, we need to organize our data into matrices and define the input and output data. In ML terminology, the input is a matrix X and the output a vector y. Then we map a function $y = g(X)$. The matrix X, or input matrix has as columns the **Features.** The features are basically all the things that our prediction depends upon. For example, if we want to predict the price of a house that is about to be on sale, we can put features like: the square footage of the house, the number of rooms, is there a yard or no, the number of bathrooms etc.

In time series forecasting using machine learning, the main part of the features is the **past values.** We feel that the utilization hours in the past will have a direct effect on the utilization hours in the future, that is why as features we put exactly the utilization hours of a past period. If we again go back to (Equation 1), x_i are these past values.

The process is visualized on Figure 3.2. The vector on the left is the original time series, the daily utilization hours of a vehicle and on the right, we have the same data transformed as an input to the ML algorithm. X is the feature matrix (the independent variable) and y is the target (the dependent variable). Every row of X is lagged values of the original time series. For example, the first four values of the time series that are marked with a blue square, compose the first row of the feature matrix X. The next value in the time series, marked with a green square, is the first value of the target vector y. This means that we use the past 4 values to predict 7.31 hours, the value of the fifth day. Building X in this way, in the columns we get, for each day, the values of the previous day (t-1), the values of the day before (t-2) that and so on. For the purposes of explaining the process we used a window of 4 lagged values, but in the real scenario we use a different past window size.

Original time series

Day no.	Utilization hours
Index	on_time_h
0	9.84
1	9.65
2	6.48
3	8.04
4	7.31
5	10.15
6	10.17
7	10.27
8	8.84
9	6.26
10	8.92
11	5.06
12	6.42
13	10

X – Feature matrix

Index	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)
4	9.84	9.65	6.48	8.04
5	9.65	6.48	8.04	7.31
6	6.48	8.04	7.31	10.15
7	8.04	7.31	10.15	10.17
8	7.31	10.15	10.17	10.27
9	10.15	10.17	10.27	8.84
10	10.17	10.27	8.84	6.26
11	10.27	8.84	6.26	8.92
12	8.84	6.26	8.92	5.06
13	6.26	8.92	5.06	6.42

y – Target

var1(t)
7.31
10.15
10.17
10.27
8.84
6.26
8.92
5.06
6.42
10

Figure 3.2 Turning a time series into an ML algorithm input

3.2.2 Smart Feature Selection

In the above example, we used the values of the previous 4 days to predict today. In the same way, in our problem we could use the values of the past week, or past month to predict today, But, sometimes, it is not the events of all past days that influence the present day. For example, if we want to predict the utilization hours on a Monday, we may be more interested in what happened last Monday, than what happened on Thursday. Thursday may even have no correlation whatsoever to Monday.

Having this in mind, we devised a "smart" feature selection method, where we take as features only the utilization hours of the days that are correlated to today. This is done by checking the autocorrelation function of our original time series and deciding which lagged observations are most worthy to become part of our Features. On Figure 3.3, we can see an example of an Autocorrelation plot: the values of the autocorrelation function of the time series of utilization

hours of vehicle 4605 (one of the vehicles of model 1254 of type Refuse Compactor that is included in our analysis).

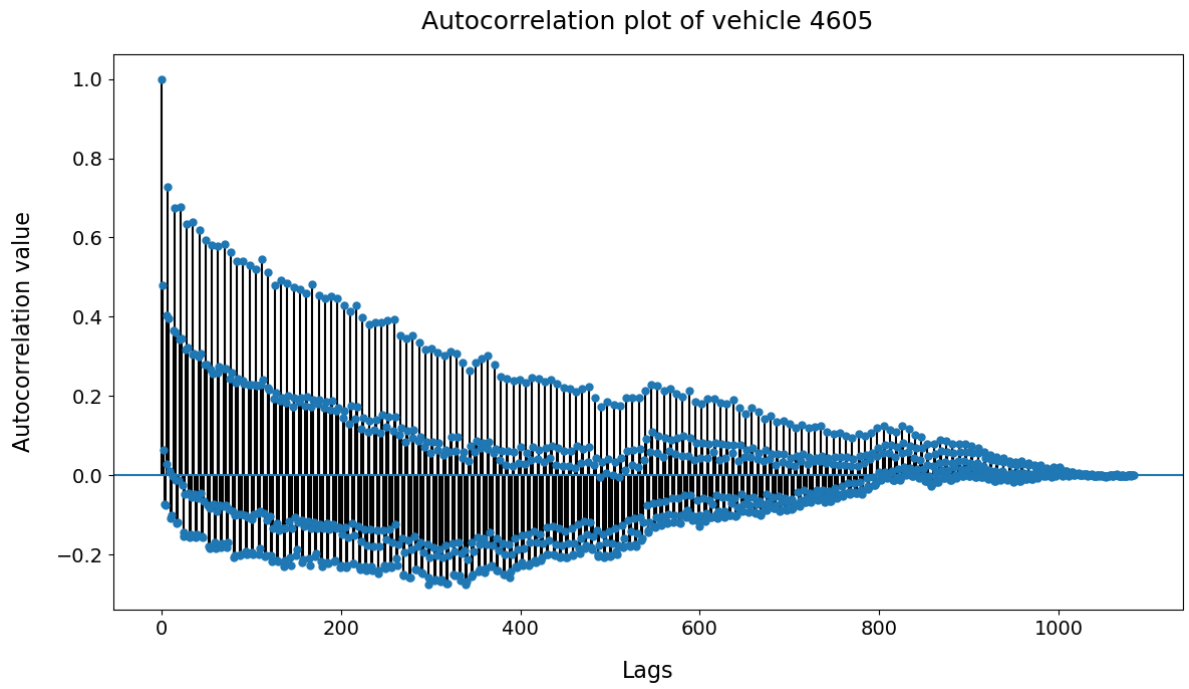

Figure 3.3 Autocorrelation plot of vehicle 4605

On the x-axis there is the lags and on the y-axis the value of the autocorrelation. Of course, the value for the zero lag is 1, since we are comparing an observation with itself. The other values help us see how correlated a value is with its lagged values. As expected, the further into the past we go, the less correlation we find. This kind of analysis can also give us an incentive on how far we should go with the past window size. However, there are higher and lower values mixed up throughout the plot, which means that not every past day is equally correlated with the present day.

If we zoom in for only 30 days, we get Figure 3.4. It is visible from this plot that the values for t-1, t-6, t-7, t-8, t-13 etc. are much more correlated to today than all the other values.

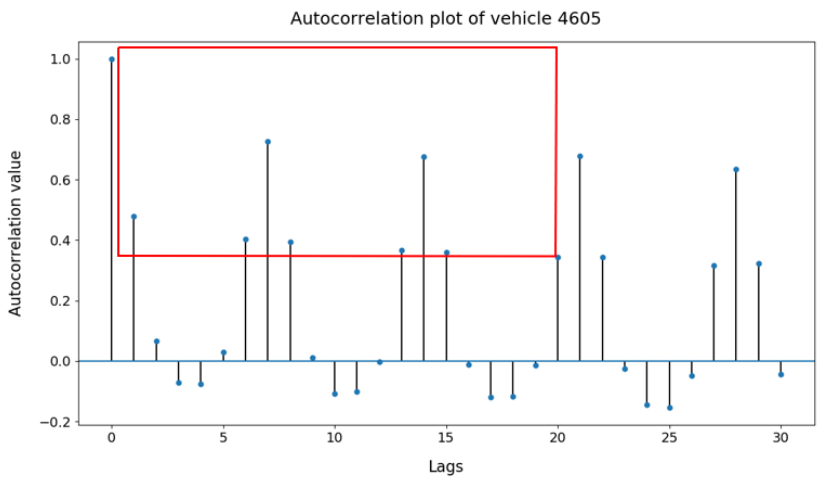

Figure 3.4 Autocorrelation plot of vehicle 4605: zoomed to 30 lags

This observation can be incorporated into our model using the Features. We will take as Features only the past days which are highly correlated. Which values we consider highly correlated? The method we use is the following: We take a window of S lagged values starting at the beginning (ex. The first 10 lags) and we take the mean of the absolute values of all of them. Then the values that make it as Features are the ones that have an absolute value larger than this mean. Only the first 2*S values are considered (in this example the first 20 lags) and they can be both positive or negative. The optimal size of S is discussed in more detail in Chapter 4: Results. Basically, in this example, the days that are taken as features are the days with lags that fall into the red rectangle.

Part of the Feature matrix and target vector for vehicle 4605 are shown on Figure 3.5. Now the columns resemble the most correlated past days and this can also be seen on their headers.

X – Feature matrix

var1(t-15)	var1(t-14)	var1(t-13)	var1(t-8)	var1(t-7)	var1(t-6)	var1(t-1)
4.68	10.19	10.91	10.39	9.31	10.86	10.81
10.19	10.91	10.63	9.31	10.86	10.76	10.64
10.91	10.63	10.07	10.86	10.76	9.36	11.44
10.63	10.07	7.44	10.76	9.36	4.96	11.63
10.07	7.44	0	9.36	4.96	0	11.28
7.44	0	10.39	4.96	0	10.81	6.25
0	10.39	9.31	0	10.81	10.64	0
10.39	9.31	10.86	10.81	10.64	11.44	11.33
9.31	10.86	10.76	10.64	11.44	11.63	11.38
10.86	10.76	9.36	11.44	11.63	11.28	11.23
10.76	9.36	4.96	11.63	11.28	6.25	11.41
9.36	4.96	0	11.28	6.25	0	11.66
4.96	0	10.81	6.25	0	11.33	4.54

y – Target

var1(t)
10.64
11.44
11.63
11.28
6.25
0
11.33
11.38
11.23
11.41
11.66
4.54
0

Figure 3.5 ML algorithm input using Smart Feature Selection

3.2.3 Adding Additional Features

As mentioned in the Problem formulation, the primary Features are lagged values of our time series, but there are other phenomena that are connected to the utilization hours at the day we want to predict that can also be input as Features.

One additional column we add to the X (Feature) matrix is a column called “isWorkday”. It is basically a column that contains 1 if that day of the target (y) is a working day, or 2 if it is a weekend or a holiday. This column is very helpful in the scenario “Predict usage on next day”, where we maintain the temporality by adding zeroes in the data, since these zeroes are the ones that are hardest to predict, and they often mark the weekend.

3.3 WALK – FORWARD VALIDATION

Now that we know our input and desired output from the ML algorithm, we introduce the concept of training and testing the model, how it is done in normal ML problems and how it is handled in our case.

An important paradigm of supervised machine learning is the split of the dataset into training and testing sets. We take part of the dataset to "train" the machine: giving it inputs and correct outputs so that it can map a function $y = g(X)$ to best describe this relationship. The machine finds a function based on different logic depending on the algorithm used. Usually, the bigger the training set, the better, since with every new training sample the accuracy of the function is improved. Then, we try the already trained machine on unseen data: we apply the function it found to the test set. The machine now does not know the correct answers and predicts them, but we know the correct answers, so we can check how accurate the developed function is.

In a normal ML problem, where the time of event occurrence is not important, the data can simply be split in two: a training and testing set. As an example, using Figure 3.5, we would take the first 7 rows of the matrix X and the first 7 rows of the matrix y as (X, y) training pairs and the last 3 rows of X and y as testing pairs. This is the simplest scenario. Then, there are more sophisticated ways to do the train-test split to avoid undesirable behaviour, like overfitting. Some of them are using a third validation set or doing k-fold cross-validation, where we split the dataset into k subsets and use one as test, the other k-1 as training, interchanging the test one and then averaging the error between all the k test sets. This is the best way to do cross-validation, since we really use up the dataset and not overfit the data on one small test set.

However, in our case we cannot use k-fold cross-validation because there is the time factor. We cannot randomly pick training and testing samples since time has to be preserved. This means that the testing samples should come later in the dataset than the training samples. Another important thing to consider is that the data collection is from an IoT system, so there is new data available every day. The original Table (Table 2.1) is updated and this new data can and should be used in our model. So, we want to refresh the model every day.

Having these two things in mind, **we construct the validation in the following way**: Starting at the beginning of the time series, a number of samples (past window) is used to train a model. This "model" is in fact the mapping equation $g = f(X)$. Then the model makes a prediction for the next time step, the next day (test phase). This prediction is evaluated against the known value and the error is recorded. Then we move on to the next day, including in the window the known value, thus moving forward one step, and we repeat the process.

Because this methodology involves moving along the time series one-time step at a time, it is often called Walk Forward Testing or Walk Forward Validation. The process implies that we build not only one model of our problem, but many models, since a new model is created and trained every day. This way, we can provide a much more robust estimation of how the chosen modeling method and parameters will perform in practice, if we assume the data will change its distribution in time and the dynamic of the system is fast. However, this improved estimate comes at a computational cost of creating so many models.

To clarify and quantify the process, we specify the matrix sizes of the input and output using walk-forward validation on our data. First, we define N and F:

N (past window size) – number of past samples (days) used for training (ex. N=100)

F (feature window size) – number of features: the lagged values constructed either simply, or by using the Smart feature selection, plus any other features that we might incorporate (ex. F=7)

For each model the X_train matrix has the size NxF and the y_train vector is of size Nx1. x_test is a vector of size 1xF and y_test is of size 1x1 – it is a real number. This says that for every model (one model per day), we train with N samples and we test with 1 sample.

Finally, we have L models (and predictions): $L = M - N - F$

L is equal to: M number of days in the dataset for that vehicle, minus N for the first window minus F for the feature window size.

Let us call our model of L models a mega-model. To evaluate the mega-model accuracy, we average out the error of all the L models. Which error measurements we use is explained later in this chapter.

One more important concept about the walk-forward validation that we need to address is the way we build the past window. Since we are moving ahead in time, we have more and more data for training. Should we use all this data in our next model, expanding our window every day or fix a past window size and slide it? It is a plausible question. The two paradigms are the following:

- Expanding window

  The past window size grows as we go further in the future. With every new day we use more and more data as training. In the above definitions of the sizes, this means that N would start from a value and grow with every new model. So, we only have to fix the starting N and it is not so crucial to the performance. The positive side is that we have a lot of training samples, which in basic machine learning is always good. But how dependent is the present from the far past? Does the past just stop being meaningful for predicting the future at some point? And we have to consider that as the data grows, the computational time also grows.

- Sliding window

  In this case the past window size (N) is fixed. It is always the same, on every new model. It appears as though it slides with the data, hence the name. This type of window better captures the behaviour of the data at the moment, getting rid of the far past. It is much less computationally expensive, but is limiting the training data size. Also, we have to carefully choose the window size N.

Figure 3.6 illustrates the two window types in terms of training and testing portions of the dataset.

Figure 3.6 Expanding vs. Sliding window

It is clearly visible what is the difference between them and gives an accurate notion on how we do the training and testing in general and how we put together all the separate test samples to evaluate the accuracy of our mega-model.

In conclusion, what is really important to distinguish from this Section is the **past window size** and the **feature window size.** The past window size is how many samples we use for training. They are values of the past, that is why we call it the past window. The feature window size is how many samples we use as features (which are these samples is for the normal/smart feature construction algorithm to decide). This number is always less than the past window size. Basically, the feature window size is the width of our matrix X, while the past window size is the height of it.

Another important thing is the **walk-forward validation** method: the fact that we do a new model every day, which gives us many models per vehicle. In the end, we are not giving a specific model as a solution, but more a manner of choosing the perfect model tomorrow.

As we can see, there is a number of parameters to fix even before we start with the algorithm choice. We have to decide which past observations to use as Features, are we going to construct the Feature matrix simply or by doing the Smart Feature Selection, if the selection is making our results better, how are we going to fix its parameter (the number of lagged values that are included

for taking the mean), then we need to decide the past window size and whether we are going to use a sliding or expanding window. When we have decided all these parameters, we have a scenario and we can apply an ML algorithm. In the next section we discuss the 5 algorithms that were tried in the scope of this thesis.

3.4 REGRESSION ALGORITHMS

As mentioned in Section 3.2, Regression is a supervised ML technique that is used to predict continuous variables. There are many algorithms that fall under the umbrella of regression, each trying to find the best mapping function $y = g(X)$, that links its input to its desired output. We also said that regression fits the formulation of our problem perfectly and so it is regression algorithms that we use to solve our problem. In the previous few sections we described how to prepare our data for regression and in this section we describe all the algorithms that were used to predict our variables.

All of the algorithms that are described, except Linear regression, have hyperparameters that need to be tuned. (DataCamp, 2018) defines a model hyperparameter as a configuration that is external to the model and whose value cannot be estimated from data. They have some default values that generally work well with the algorithms, but they are often specified by the practitioner. In this study, we tune some of these hyperparameters.

3.4.1 Linear Regression

As a first, simple algorithm we try Linear Regression. We do not know if the relationship between our input and output can be described as a linear combination, so it is a justified choice to try a simple linear algorithm. We use the implementation of the Python library scikit-learn, (Scikit-learn.org, 2018).

Linear regression tries to map the input to the output using a linear function. Each feature is attributed a weight in the equation, which accounts for the importance of that feature for predicting the output. The general form of the equation is the following:

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \ldots + w_F x_{NF}$$

Where y_i is the i-th output (i = 1 … N). N is the number of samples and F is the number of features.

In matrix form the equation is:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$$

Where $\boldsymbol{y}$ is the target variable vector (Nx1), $\boldsymbol{X}$ is the feature matrix (NxF) and $\boldsymbol{w}$ is a vector of weights (Fx1).

The goal is to estimate the weights $\boldsymbol{w}$ and this is done through the method of Least Squares.

Least Squares means that we are trying to find a vector of weights that minimizes the square error:

$$e(\boldsymbol{w}) = ||\, \boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2$$

Getting the gradient of this error $e(w)$ to zero gives us the solution for w:

$$\hat{\mathrm{w}} = [\boldsymbol{X}^T\boldsymbol{X}]^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Linear regression is good for exploring the model. The different weights (coefficients) can tell us in a simple manner how the different features impact the model - if they are important or not. It was exactly this analysis that led us to the idea for the smart feature selection. It is also a good way to see how any new features we might add reflect on the model. The coefficient analysis is also a good tool to explore the impact of the sliding and expanding window on the model. More on this is discussed in Chapter 4: Results.

3.4.2 Lasso Regression

Lasso regression uses the same principle as linear regression, in that it tries to map a linear function to the problem and aims to find the weights (coefficients) of the different features in this function. However, aiming to perform better than normal linear regression, it adds a regularization parameter to the weights, a kind of penalization, so that they do not grow very high. The name Lasso is an abbreviation of Least Absolute Shrinkage and Selection Operator. We again use the scikit-learn implementation of the algorithm, (Scikit-learn.org, 2018).

The difference from Linear regression comes in the objective function, what we are trying to do in this case is minimize the following error:

$$e(\boldsymbol{w}) = ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2 + \alpha|\boldsymbol{w}|$$

Where **y**, **X** and **w** are defined in Section 3.4.1 and α is an L1 regularization parameter. If α is zero, we get exactly Linear regression as above. Controlling α controls the penalization of the features. Then minimizing the error and solving for $\boldsymbol{w}$ we get:

$$\hat{\mathrm{w}} = (\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Lasso is counted as one of the simple techniques to reduce model complexity and prevent overfitting which may result from simple linear regression. The type of regularization (L1) it uses can lead to zero coefficients, meaning some of the features are completely neglected for the evaluation of the output. So, Lasso regression not only helps in reducing overfitting, but it can help us in feature selection.

There is one hyperparameter that needs to be fixed and that is the regularization parameter α. The default value of α by the Python implementation sklearn.linear_model.Lasso is 1.

3.4.3 SVR – Support Vector Regression

SVR, or Support Vector Regression is a regression algorithm that is derived from SVM (Support Vector Machine), which is a classification algorithm. The main idea in SVR is the same: to minimize an error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated. The model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction. We use the scikit-learn (Scikit-learn.org, 2018) implementation of SVR (sklearn.svm.SVR).

Linear SVR tries to find the function:

$$f(x) = \boldsymbol{x}^T \boldsymbol{w} + b$$

Training the original SVR means solving the minimization problem:

$$J(w) = \frac{\mathbf{1}}{\mathbf{2}} ||\boldsymbol{w}||^2$$

subject to subject to all residuals having a value less than ε:

$$\forall n: \ |y_n - ({x_n}^T w + b)| \leq \varepsilon$$

where x_i is a training sample with target value y_i. The inner product plus intercept $\langle w, x_i \rangle + b$ is the prediction for that sample, and ε is a free parameter that serves as a threshold: all predictions have to be within an ε range of the true predictions. It is possible that no such function $f(x)$ exists to satisfy these constraints for all points. To deal with otherwise infeasible constraints, slack variables ξ_n and ξ_n^* are introduced for each point. This approach is similar to the "soft margin" concept in SVM classification, because the slack variables allow regression errors to exist up to the value of ξ_n and ξ_n^*, yet still satisfy the required conditions.

Including these slack variables leads to the objective function, also known as the primal formula:

$$J(w) = \frac{\mathbf{1}}{\mathbf{2}} ||\boldsymbol{w}||^2 + C \sum_{n=1}^{N} (\xi_n + \xi_n^*)$$

Subject to the following constraints:

$$\begin{cases} \forall n:\ y_n - ({x_n}^T w + b) \leq \varepsilon + \xi_n \\ \forall n:\ ({x_n}^T w + b) - y_n \leq \varepsilon + \xi_n^* \\ \forall n:\ \xi_n^* \geq 0 \\ \forall n:\ \xi_n \geq 0 \end{cases}$$

The constant C is the box constraint, a positive numeric value that controls the penalty imposed on observations that lie outside the epsilon margin (ε) and helps to prevent overfitting (serves for regularization). This value determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than ε are tolerated.

The optimization problem previously described is computationally simpler to solve in its Lagrange dual formulation. The solution to the dual problem provides a lower bound to the solution of the primal (minimization) problem. The optimal values of the primal and dual problems need not be equal, and the difference is called the "duality gap." But when the problem is convex and satisfies a constraint qualification condition, the value of the optimal solution to the primal problem is given by the solution of the dual problem.

To obtain the dual formula, a Lagrangian function is constructed from the primal function by introducing nonnegative multipliers α_n and α_n^* for each observation x_n. Then we minimize a new function that includes these multipliers and we constrain them to be lower than C.

Now the function used to predict new values depends only on the support vectors:

$$f(x) = \sum_{n=1}^{N} (\alpha_n - \alpha_n^*)({x_n}^T \mathrm{x}) + b$$

Nonlinear SVR:

Some regression problems cannot adequately be described using a linear model. In such a case, the Lagrange dual formulation allows the previously-described technique to be extended to nonlinear functions. We can obtain a nonlinear SVR model by replacing the dot product $({x_1}^T x_2)$ with a nonlinear kernel function $K(x_1, x_2) = < \varphi(x_1), \varphi(x_2) >$, where $\varphi(x)$ is a transformation that maps x to a high-dimensional space.

The three kernel functions that we use are:

$K(x_1, x_2) = {x_1}^T x_2$ – Linear (dot product)

$K(x_1, x_2) = (\gamma({x_1}^T x_2) + r)^d$ – Polynomial, d is the degree

$K(x_1, x_2) = (-\gamma||x_1 - x_2||^2)$ – Radial basis function (RBF)

That said, there are a few hyperparameters to be chosen when using SVR:

- the kernel type
- the C parameter
- the range ε
- the multiplier γ if we use polynomial or RBF kernel
- the degree d if we use polynomial kernel

SVR also requires standardizing the data to have mean 0 and variance 1.

3.4.4 Random Forest Regression

Random Forest Regression is an ensemble method. The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. In particular, it is an averaging/bagging method, meaning it builds several independent estimators (weak learners) and then averages their predictions. We use the implementation of the Python library scikit-learn (Scikit-learn.org, 2018), given by sklearn.ensemble.RandomForestRegressor.

The core of the Random Forest, its base estimator, is the Decision Tree. The Decision Tree is an algorithm which forms a tree structure, calculating the best questions to ask in order to make the most accurate estimates possible. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The observations about

an item are represented in the branches and the conclusions about the item's target value are represented in the leaves.

Random Forest consists of multiple trees - in ensemble terms, the trees are weak learners and together they make the Random Forest, which is a strong learner. Each decision tree in the forest considers a random subset of features when forming questions and only has access to a random set of the training data points. This increases diversity in the forest (each tree is slightly different) leading to more robust overall predictions and gives it the name "random forest". When the time comes to make a prediction, the algorithm takes an average of all the individual decision tree estimates. As a result of the randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance decreases, which usually more than compensates for the increase in bias, hence yielding an overall better model.

The main hyperparameters to adjust when using Random Forest Regression are:

- **n_estimators**: The number of trees in the forest. The larger the better, but also the longer it will take to compute. It is expected that the results will stop getting significantly better beyond a critical number of trees.

- **max_features**: The size of the random subsets of features to consider when splitting a node. The lower it is, the greater the reduction of variance, but also the greater the increase in bias. Empirical good default value is max_features=n_features for regression problems.

- **max_depth**: The maximum depth of the decision trees. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- **min_samples_split**: The minimum number of samples required in a node to be considered for splitting. This parameter is used to prevent overfitting.

- **min_samples_leaf**: The minimum number of samples required at each leaf node.

- **max_features:** The number of features to consider when looking for the best split. These are randomly selected.

Max_depth and min_samples_split control the size of the trees. The default values are max_depth=None and min_samples_split=2. They lead to fully grown and unpruned trees which

yield good results, but can potentially be very large on some data sets. To reduce memory consumption, the complexity and size of the trees should be controlled by setting those parameter values.

Using scikit-learn's Random Forest we can explore the features with the attribute feature_importances_. Features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features. In scikit-learn, the fraction of samples a feature contributes to is combined with the decrease in impurity from splitting them to create a normalized estimate of the predictive power of that feature. This estimate is stored in feature_importances_.

The pros of using Random Forest is that, being an averaging ensemble method, it reduces variance and thus conquers overfitting. It is also good for exploring the features. Theoretically, we can also find out what is going on inside it by looking at the separate decision trees, but this process is messy since there are many trees for every model and we are making a new model every day. It is also more computationally expensive than the other algorithms we try and it has a lot of parameters that need to be tuned.

3.4.5 Gradient Boosting Regression

Gradient Boosting Regression is another ensemble method, like Random Forest, but this one is a boosting method, rather than an averaging (bagging) method. Boosting is an ensemble technique in which the predictors are not made independently, but sequentially. We once again use the scikit-learn implementation of the algorithm given by sklearn.ensemble.GradientBoostingRegression. On Figure 3.7 (Medium, 2018), we visualize the difference between these two ensemble methods.

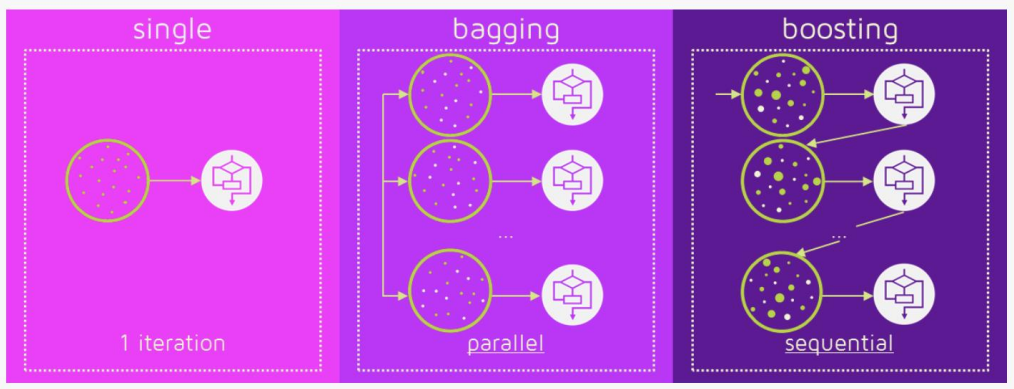

Figure 3.7 Difference between bagging (averaging) and boosting ensemble methods

(Medium, 2018): This technique employs the logic in which the subsequent predictors learn from the mistakes of the previous predictors. Therefore, the observations have an unequal probability of appearing in subsequent models and ones with the highest error appear most. (So, the observations are not chosen based on the bootstrap process, but based on the error). The predictors can be chosen from a range of models like decision trees, regressors, classifiers etc. Because new predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach actual predictions. But we have to choose the stopping criteria carefully or it could lead to overfitting on the training data.

The advantages of Gradient Boosting are that it can handle heterogeneous features and is robust to outliers in output space, while the disadvantage is that it is hard to scale due to its sequential nature.

(Learning and Python, 2018) splits the parameters of Gradient Boosting into three categories:

1. Tree-Specific Parameters: These affect each individual tree in the model.

   They are similar to those of Random Forest and are explained in the previous section: **min_samples_split, min_samples_leaf, max_depth, max_features**.

2. Boosting Parameters: These affect the boosting operation in the model.

   There are three important parameters here:

- **Learning_rate**: A multiplier that shrinks the contribution of each tree by its value. Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
- **n_estimators:** The number of sequential trees to be modeled (the number of boosting stages to perform). Gradient boosting is fairly robust to over-fitting, so a large number usually results in better performance.
- **Subsamples:** The fraction of observations to be selected for each tree. Selection is done by random sampling.

There is a trade-off between learning_rate and n_estimators.

3. Miscellaneous Parameters: Other parameters for overall functioning.

There are a few parameters here but the most important one is:

- **Loss**: The loss function to be minimized at each split. The default is least squares. Some alternatives are least absolute deviation, huber and quantile.

3.5 PERFORMANCE EVALUATION

An essential part of the model construction and the whole prediction process is to know how accurate our solution is. We use a few metrics to judge the performance of the devised models - we measure the error in the prediction.

First, we analyze the error in absolute terms. There are two very common error metrics that come into play here: The Mean Absolute Error and the Root Mean Squared Error.

We define the vector of true values for the target **y** as y_{true} and the vector of predicted values as y_{pred}. With this said we define the two errors in absolute terms:

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_{true_i} - y_{pred_i}|$$

- Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_{true_i} - y_{pred_i})^2}$$

Although these are good and accurate error measurements, they are absolute, meaning they measure the average error we make over all predictions in hours. Since we make a different mega-model for every vehicle and the vehicles have very different utilization magnitudes (the number of hours they are usually used varies a lot from vehicle to vehicle), we need a relative error measurement that would allow us to compare performance of predictors between different vehicles. This is why we also measure the Percentage Error.

- Percentage Error (PE)

$$PE = \frac{\sum_{i=1}^{n} |y_{true_i} - y_{pred_i}|}{\sum_{i=1}^{n} |y_{true_i}|} * 100$$

This error tells us on average, how much our prediction is wrong relative to the utilization hours of that vehicle. In the further text, this error can also be found as Error in %.

Other data about the error is also collected upon the execution of the models for every vehicle. This includes some insights on the residual error $\boldsymbol{r}$ (a vector):

- Residual Error $\boldsymbol{r}$

$$\forall i: r_i = y_{true_i} - y_{pred_i}$$

We analyze the distribution of the residual error values, its mean and standard deviation – we want a normally distributed residual error, with a mean of 0 and a small standard deviation. This tells us if the model is biased or not.

4 RESULTS

In this Chapter, we report the Results we obtained, using the Methodologies described in Chapter 2, on the Dataset explored in Chapter 1.

We begin with the first scenario, Predict usage on next day. We want to predict the next calendar day, so we fill the days for which there is no data with zeroes, as mentioned in the previous Chapters. This makes it a difficult prediction problem since the zeroes are throwing off the predictor.

We start the Result analysis by tuning the general parameters that are used in the methodology, regardless of the algorithm. In this section we also choose the best setup for running the algorithms - which type of window are we going to use, whether smart feature selection is generating better results, whether our additional feature is really helping the algorithm etc. Then we proceed to run the algorithms with some coarse hyperparameter tuning and generate some first results. After this phase, we narrow down the algorithms to the best ones and we do a broader and proper hyperparameter tuning. We also go into more detail on the results, showing plots and interesting insights about the working of the different algorithms.

Then we move on to the second scenario, Predict usage on next **working** day, which simplifies the problem. We repeat the valuable experiments in this scenario using only the best algorithms, having in mind the difference in the problem formulation. Then we analyze the results of this scenario and we compare the two.

4.1 CHOICE OF FEATURE CONSTRUCTION METHOD AND TRAINING SET SIZE (ALGORITHM INPUT)

A lot of the methodologies explained above require tuning of parameters, even before we start to apply algorithms. We need to decide some general values like:

- the past window size

- usage of sliding or expanding window (past window size important for usage of sliding window)
- the feature window size when we use normal feature construction
- the threshold of lags considered from autocorrelation function to form the mean, if we use smart feature selection

To fix these values we use Linear regression, since the algorithm itself does not require any parameter tuning and is thus good for comparison between setups.

4.1.1 Fixing Past window size and Feature window size

We start with tuning the Past window size and Feature window size. Tuning the feature window size means we are using the "normal" feature construction, where we take consecutive past days as features. The past window size is the number of samples used for training, while the feature window size tells us how many consecutive days in the past shall be used as features. These two parameters define the size of the **X** matrix (the training matrix that is an input to the algorithm, as explained in 3.2.1) – the past window size is N and the feature window size is F. This makes them coupled values that should be picked together as a combination.

The choice of the past window size is really important because it is the size of the sliding window.

The tests are done on 10 representative vehicles of model 1254 and the error is averaged. Their IDs are: 3959, 3988, 4256, 4551, 4700, 4891, 5003, 5229, 5504, 6034. The tried values for the feature window size are: 5, 7, 10, 20, 30, 40 and 50 days. The tried values for the past window size are: 70, 90, 100, 110, 120 and 130. The results are shown on Figure 4.1.

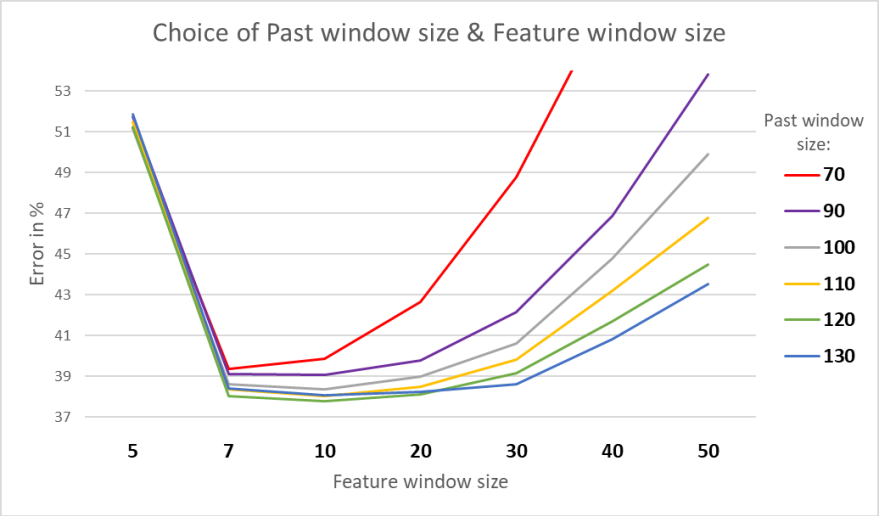

Figure 4.1 Choice of past window size and feature window size

On the x-axis we have the different values for Feature window size, on the y-axis we have the average error in % of Linear regression run with the use of these parameters. The different curves represent different Past window sizes and they are listed in the legend.

All the curves have a somewhat similar behaviour with respect to the Feature window size. The lowest errors are obtained by using a past window size of 120 or 130 (green and blue curve), depending on the Feature window size. However, the best result is a Feature window size of 10 and a Past window size of 120.

4.1.2 Fixing the Past window size and the parameter for Smart Feature Selection

If we want to use Smart Feature Selection, we do not have a fixed Feature window size. It varies from vehicle to vehicle according to the autocorrelation of the data. As mentioned in Section 3.2.2 of Methodology, there is a parameter there that we have to tune and that is: the size of the vector of lagged observations from which a mean is taken, so that we include as features only the lagged days that have a higher absolute value of correlation than this mean. In the previous chapter this value was defined as S. Let us now call this value *acf_value* (autocorrelation function value). Then,

we are taking only the indexes of the autocorrelation function that are less than $2 * acf\_value$. Putting this value low, means we are introducing a higher mean, since usually the lags close to 0 are more correlated and thus higher in value. This stricter mean, with the combination of a low number of indexes considered, lets in a lower number of features. On the other hand, putting acf_value high means setting a lower threshold and a longer range of considered indexes, thus allowing more lagged values to become features.

This value is also coupled with the past window size, so we again search for the best combination. Also, we want to make sure that the value we choose works well for a wider range of window sizes, to give it more credibility.

The tests are done on the same 10 vehicles of model 1254. The past window sizes are of the same order as before, but a bit larger: 100, 110, 120, 130, 140, 150. The tried numbers for acf_value are: 5, 10, 15, 20, 25, 30, 40. We have to keep in mind that the algorithm is letting in values from the range of $2 * acf\_value$.

On Figure 4.2 we show the results of the search. We can see that an acf_value of 15 is a good match for all the past window sizes, but as these sizes grow, larger values for acf_value also become competitive and for past window sizes of 140 and 150, they perform better. It is important to notice that overall, we are not dealing with a large change in the error, since the y-axis is very narrow, from 34 to 38 %. Taking into consideration only acf_value of 15, 20 and 25 and past window sizes larger than 120, we are dealing with only 0.6 % of improvement.

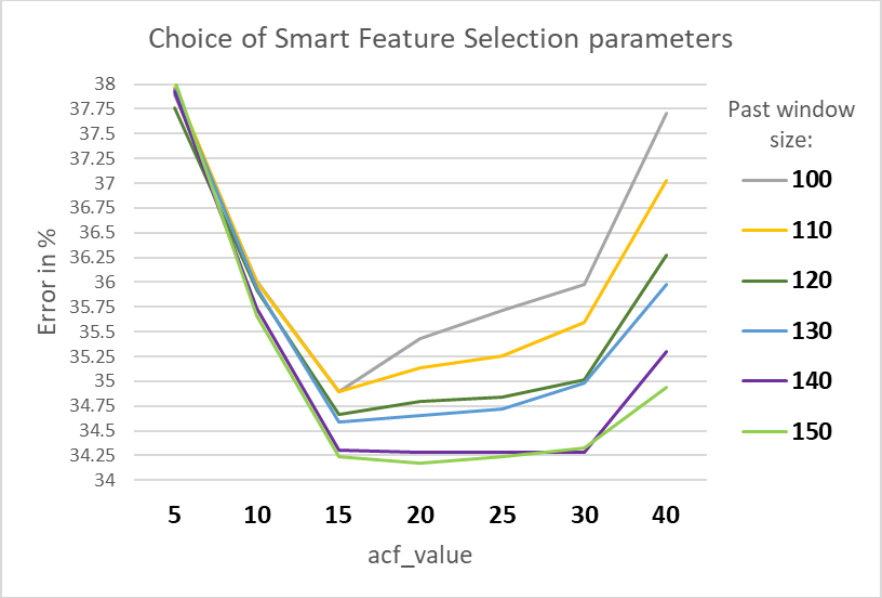

Figure 4.2 Choice of Smart Feature Selection parameters

Another thing that is visible here is that, the larger the window size, the smaller the error. This is expected, since we are using more and more training data. This opens the discussion on the Sliding and Expanding window, which needs to be discussed. We tackle this choice in the following section and we also conclude the choice of acf_value.

4.1.3 Sliding vs. expanding window

On Figure 4.3, we can see a comparison of a sliding window with length 150 (past window size) and an expanding window, which starts from 120 and changes the past window size with every model. The tests are done on the same 10 vehicles of model 1254 and the error is averaged. The comparison is again done in function of the acf_value.

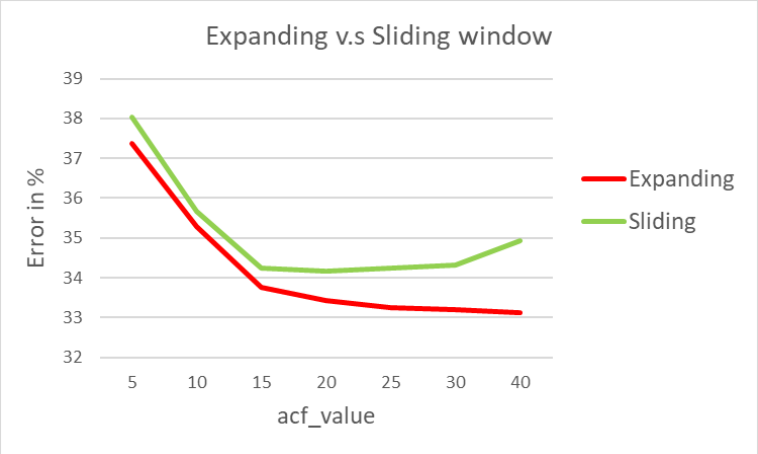

Figure 4.3 Comparison of expanding and sliding window in function of acf_value

It is visible that we can reach a lower error by using an expanding window. But, with a careful choice of the past window size and acf_value, we can get really close to this result by using a sliding window.

Even though the results are better by using an expanding window, for the simple reason of having more training data, we prefer the sliding window for a few reasons:

- Computational complexity

  Using an expanding window is much more computationally expensive since we increase the training data with every new model.

- Captures the temporary dynamics

  The sliding window is focused on a shorter period and can better approximate the model in that period of time.

To visualize how the sliding window captures the temporary dynamics, we offer a plot of the linear regression coefficients (weights) for all the models of all the days of vehicle no. 4272, done twice: once using an expanding window and then using a sliding window. We used the Smart Feature Selection, with a past window of 140 and acf_value of 20. This means that the sliding window is of size 140 and for the expanding window we start with 140 and expand with every new model. The coefficients are a metric of how important a feature is and they are the weights that multiply

each feature value in the linear equation of Linear Regression. The behaviour of the coefficients when using the expanding window and sliding window are shown on Figure 4.4 and Figure 4.5, respectively.

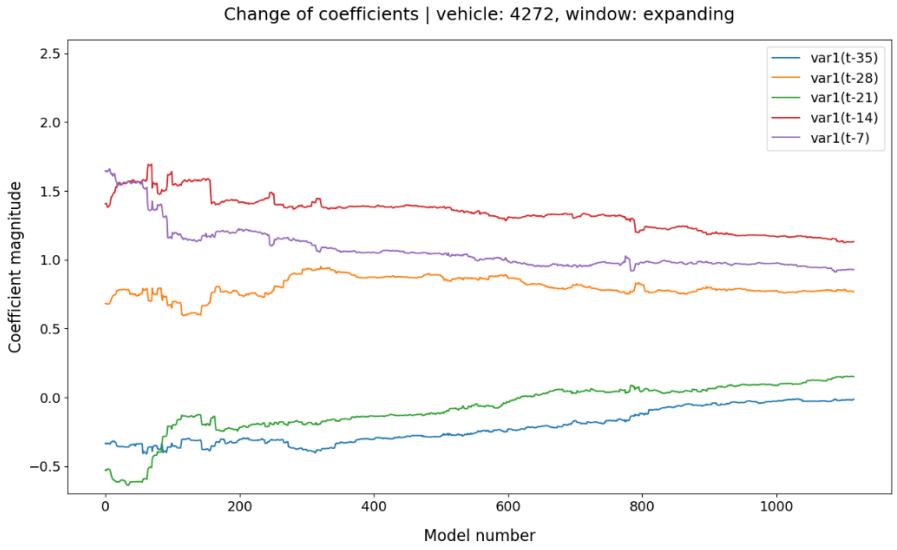

Figure 4.4 Change of coefficients of Linear regression with every new model using an expanding window

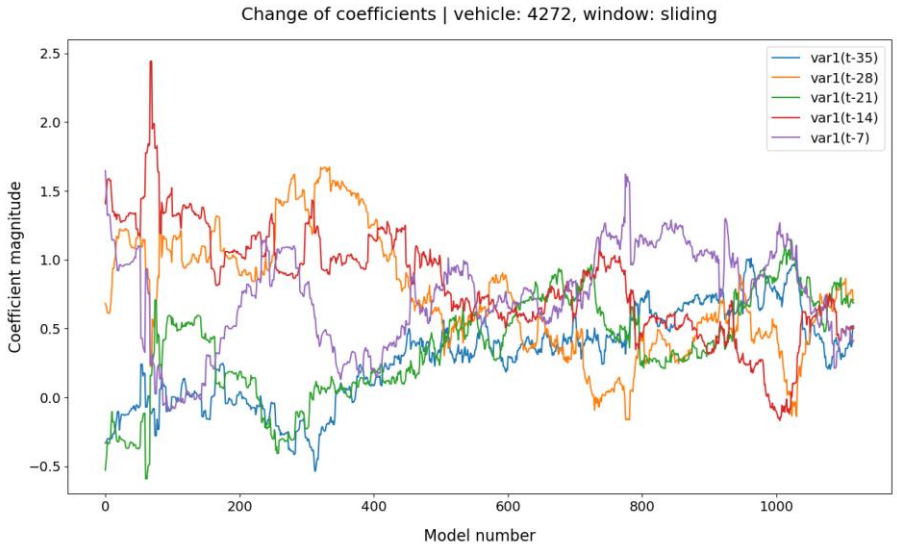

Figure 4.5 Change of coefficients of Linear regression with every new model using a sliding window

It is clearly visible from the plots how the coefficients behave differently in time for the two scenarios. Using an expanding window results in calm, almost linear change in coefficients, meaning the importance of each feature (lagged day for us) stays the same in all the models that are generated. In contrast, using a sliding window makes the coefficients change with every new model and dynamically capture the situation in the system. Let us take the utilization hours of 14 days before the day we are predicting (t-14, red curve), as an example. At the beginning this value is really important, granted a coefficient larger than 1, and as the time goes on it becomes less and less important, going towards a coefficient of 0.

There is another trade-off to keep in mind when choosing the sliding window size: we want to get close to the results of the expanding window and as we make the past window size larger, we get closer and closer to this result. On the other hand, we do not want to use a really big sliding window size, because we want to be able to do predictions also for vehicles that do not have that much data. As Future work, it may be interesting to try a dynamic past (sliding) window size, that depends on how much data there is for the vehicle. However, in the thesis we fix this parameter to obtain fair comparisons of the algorithms and scenarios.

Taking all these things into consideration and also making some experiments that involve 57 vehicles of model 1254, we decide to use, when dealing with the Smart Feature Selection scenario, **140** samples for the **sliding window size** (past window size, training samples count) and **20** for **acf_value**.

4.1.4 Improvements by adding additional Features

The last general paradigm that needs to be discussed is the adding of additional features, besides the values of utilization hours of the past days. As mentioned in Section 3.2.3, we add a Feature that is called “isWorkday”, which, as the name suggests, specifies if the target day is a workday or a weekend/holiday.

Testing the scenario proves that this Feature is helpful to the algorithm and decreases the average percentage error over 57 vehicles of model ID 1254 by 2 %. Moreover, if we take a look at the

coefficients of linear regression now, using also this feature, we can see that it is awarded a high weight, which means it is important to the algorithm. Figure 4.6 shows a stem plot of the coefficients of model no. 870 out of 1116 models of the mega-model trained for vehicle 4272. We can see that the new feature (rightmost part) has a higher coefficient (-1.5), with respect to the other features. This is not always the case, nor is the difference between this coefficient and the other ones always big, but in general the feature is given importance.

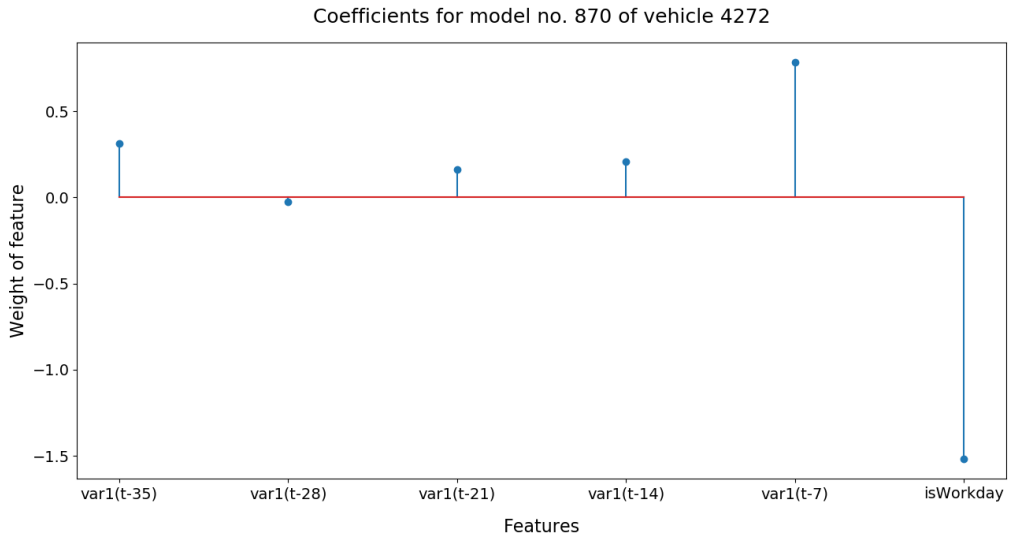

Figure 4.6 Linear Regression coefficients of the 870th model of vehicle 4272

4.1.5 Overall comparison and choice of best setup

To sum up, in this Section we show the 6 main scenarios with a bar chart of the average error in % calculated over 57 vehicles of model 1254. The tests are done using Linear Regression with a change in the setup – normal feature construction v.s smart feature selection; sliding v.s expanding window and usage of the additional feature that specifies if it is a working day or not. The "Normal Feature Construction" scenario (in short, "normal") uses a past window of 120 values and a feature window size of 10. The "Smart Feature Selection" scenario (in short, "smart") uses a past window

of 140 samples and an acf_value of 20 samples. These are the values that were discussed and decided in the past few Sections. The resulting barchart is shown on Figure 4.7.

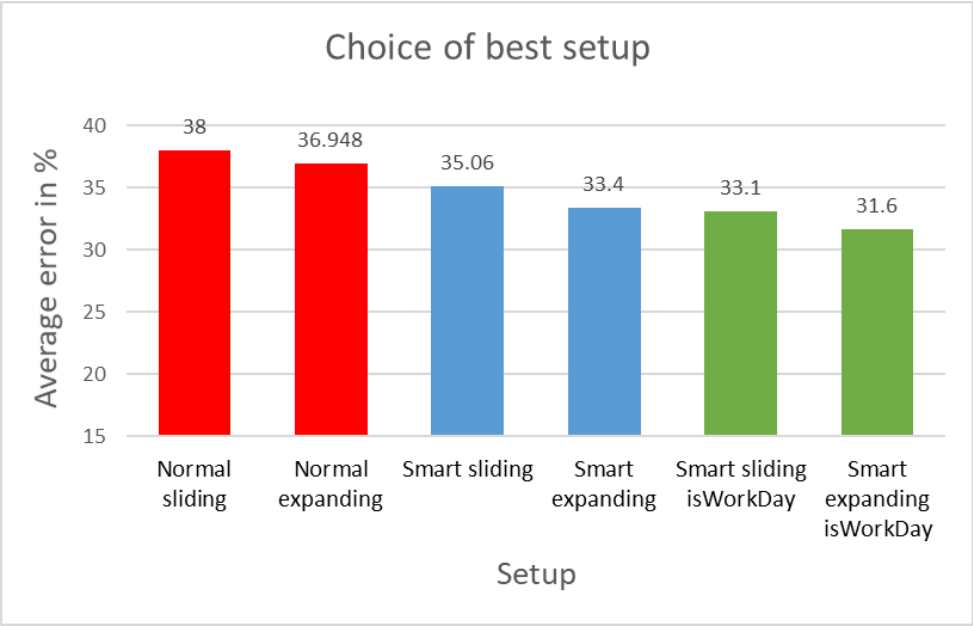

Figure 4.7 Error in % of Linear regression over different problem setups

The red bars represent the "normal" scenario, using a sliding and expanding window respectively. The blue bars represent the "smart" scenario with again a comparison between sliding and expanding window. Finally, the green bars represent the "smart" scenario with the added feature "isWorkday". The bars are sorted in descending order according to the error in %. It is visible that the "normal" scenario performs worse than the smart scenario and the smart in combination with the added feature performs the best of all three. With respect to the sliding and expanding window, the expanding window performs better in all three cases. However, for reasons discussed in Section 4.1.3, we prefer to use a sliding window, so we are willing to take the trade-off in terms of the slightly worse performance.

To conclude, based on the searches executed in this Section, in all the experiments reported in the next sections, we use the following scenario: Smart Feature Selection with acf_value = 20, combined with a sliding window of size 140 and the additional feature "isWorkday".

4.2 ALGORITHM COMPARISON

Having set all the parameters from the previous section, we are now able to proceed with the algorithms and gathering results. However, most of the algorithms require parameter tuning of their own. In this section, we do a coarse hyperparameter tuning, just a few experiments with different values with the goal to see the general behaviour of the different algorithms. Then we choose two algorithms and in the next section we concentrate on them, doing a proper hyperparameter tuning and giving detailed insight into the results we obtain with them.

The other parameters of the algorithms (explained as part of describing the algorithms in Section 3.4) are the following:

- Lasso
  - alpha = 0.1
- SVR:
  - kernel = "rbf"
  - C = 10
  - γ = 1 / n_features
  - ε = 0.1
- Random Forest
  - n_estimators = 50
  - max_features = "sqrt"
  - min_samples_split = 5
  - min_samples_leaf = 5
  - max_depth = 10
- Gradient Boosting
  - learning_rate = 0.1
  - n_estimators = 100
  - max_depth = 1
  - loss = "lad"

It is usual when testing an algorithm performance to add a benchmark algorithm to compare with. The benchmark algorithm is something really simple and the proposed algorithm should perform much better than it. For our problem, we chose two benchmark algorithms which are usual for time-series forecasting. They are:

- Last Value algorithm

  The predicted value is equal to the last observed value – for us this means that the utilization hours of tomorrow are the same as those of today.

- Moving Average

  The predicted value is equal to an average of the features. For this scenario we use acf_value=30, since after a few experiments it yielded the best results.

The algorithms were then tested on 57 vehicles of the model 1254 (Refuse Compactors) and the Error in % was measured for each vehicle. Figure 4.8 shows a boxplot of the distribution of this error for each algorithm. As in the boxplots of Chapter 2, the whiskers span to $\pm$1.5*IQR and the lines in the middle of the boxes show the median. In this plot there is the addition of x marks that label the mean.

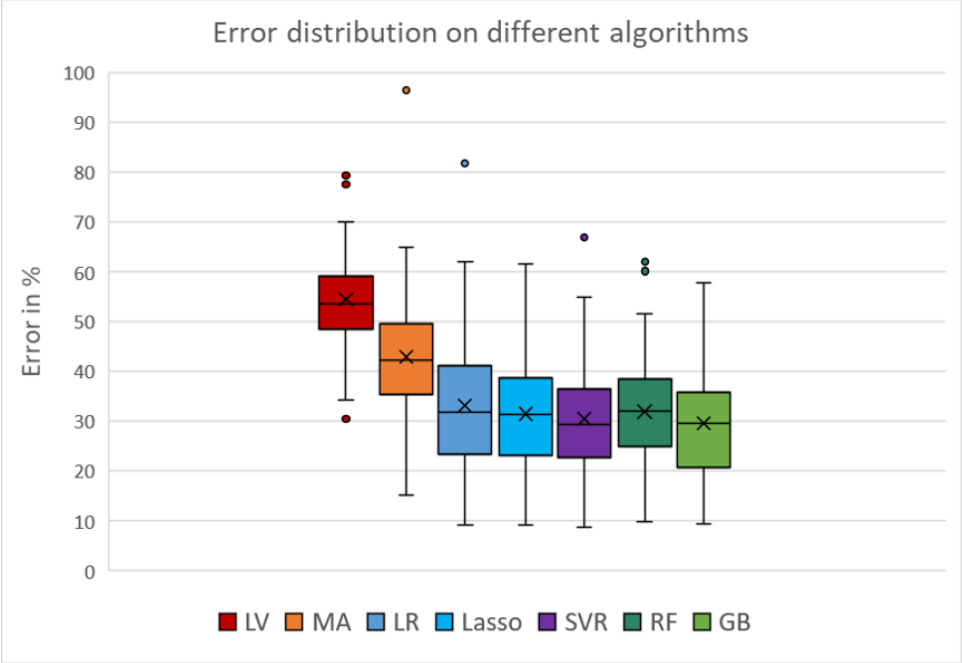

Figure 4.8 Distribution of Percentage error on different algorithms

The first obvious thing from Figure 4.8 is that the machine learning algorithms we try, perform significantly better than the benchmark algorithms Last Value and Moving Average. So, it is worthy to continue working on them. Of the benchmark algorithms, the Percentage errors obtained by Last Value have a tighter distribution than the other algorithms and the mean is 54.46%. The errors obtained by Moving Average are more spread, going from 15% to 65%, with a mean of 42.93%. This is because our data does not have a constant level, it hits extremes when it goes down to zero and this happens often. Moving Average cannot capture well this explosive behaviour, but it still performs better than Last Value.

The second obvious thing is that the distributions of the errors are regular. The mean is almost aligned with the median in every box, which means the distributions are centered around the mean and are not long-tailed. We have to bear in mind that the plotted distribution is made of 57 samples, so we cannot derive any strong conclusions about it, but we can get a good general idea about the algorithm performance.

About the five ML algorithms that were tried, it is visible that they are not that different in the performance. They all have the IQR between 20% and 40% with little difference in the mean and standard deviation. Random Forest shows a smaller variance and this is expected of an averaging ensemble method. However, the mean and median value are the second worst from all the algorithms. Table 4.1 reports the mean and standard deviation of the errors for every algorithm.

Table 4.1 Mean and Standard deviation of the Percentage error over 57 vehicles

Algorithm	Mean of the Errors	Standard deviation of the Errors
Last Value	54.46	8.87
Moving Average	42.92	12.87
Linear Regression	33.1	13.11
Lasso Regression	31.49	10.98
SVR	30.49	10.73
Random Forest	31.98	10.40
Gradient Boosting	29.64	10.76

It is difficult to evaluate which of these algorithms perform the best. If we consider only the mean and standard deviation of the errors, then the best choice is Gradient Boosting and next to it is SVR. Then with 1% worse performance is Lasso, close to it is Random Forest and finally Linear Regression. But, there are other metrics that should be taken into consideration and one of them is the computational time. Figure 4.9 shows the computational time of each algorithm to be applied to 57 vehicles, in seconds. The experiments that yielded these times were done on a laptop with an Intel(R) Core(TM) i7-8550U CPU with 16 GB of RAM.

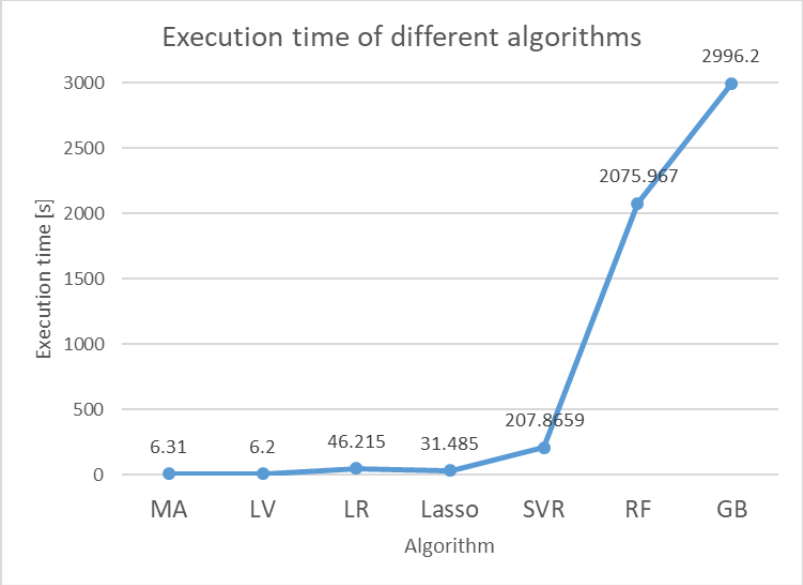

Figure 4.9 Execution time of the algorithms

It is obvious that for the ensemble methods, Random Forest and Gradient Boosting, the computational time explodes. This is because they build a number of trees for each model in the mega-model. SVR has a really good performance and a low computational time. Lasso and Linear regression also take a really short time, since they are simple algorithms.

Looking at Figure 4.7 from the previous Section and Figure 4.8 of this section, we can say that we started from an average error of 54% with Last Value, lowered it first to 38% by using a "Normal Feature Construction" configuration, then to 33% by coming up with the "Smart Feature Selection" paradigm and using it with Linear regression and then applying all of this to more sophisticated algorithms, managed to get to 29.6% of average error. The question is, can this error become even lower? In the next section we discuss the attempt to further reduce the error by doing a proper parameter tuning on Lasso, SVR and Gradient Boosting.

4.3 HYPERPARAMETER TUNING OF THE ALGORITHMS

In this section we report the results of a deeper parameter tuning done on three algorithms: Lasso, SVR and Gradient boosting. These three were chosen because they had the lowest average error and because they are different in their core. Lasso is linear regression with a regularization parameter, SVR is kernel-oriented and Gradient boosting is an ensemble method that uses sequential learners. Even though with a carefully tuned Random forest we also expect better results, since it is similar in its nature to Gradient boosting (both being ensemble methods), and takes a really long computational time, we choose only one of them.

Why is hyperparameter tuning important? (DataCamp Community, 2018) says that: "The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as you might turn the knobs of an AM radio to get a clear signal." These hyperparameters are free, so that the algorithms can be better applicable and moldable to different problems and scenarios.

4.3.1 Hyperparameter Tuning of Lasso Regression

Lasso regression has only one parameter to tune – α. It is the parameter that quantifies how much the coefficients of the algorithm (the weights of the different features) are being regularized. We perform the tuning over 10 vehicles of model 1254 (the same 10 ones that were used for the general parameter tuning). The result of the search is shown on Figure 4.10. We do a denser check in the area of least error. The best value for α turns out to be 0.065, yielding an error of 30.855% for the 10 vehicles. When we try this value on all 57 vehicles, we get an error of **31.38**%, which is not significantly less than the 31.49% we had obtained before.

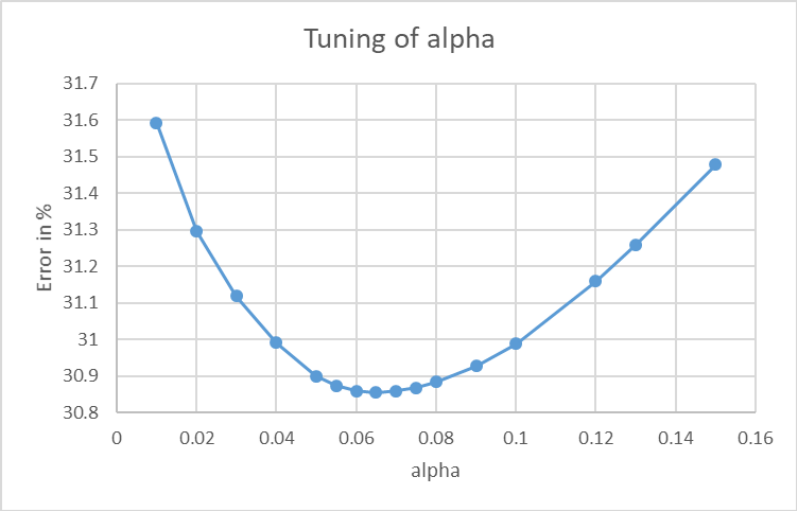

Figure 4.10 Tuning of the alpha parameter of Lasso regression

4.3.2 Hyperparameter tuning of SVR

The next algorithm to undergo proper parameter tuning is SVR. This is again done on the same 10 vehicles of model 1254 of type Refuse Compactor and the error is averaged. Here the parameters in question are the kernel, the cost coefficient C, epsilon (ε) and γ, if we use RBF kernel. The constant C is the box constraint, a positive numeric value that controls the penalty imposed on observations that lie outside the epsilon margin (ε) and helps to prevent overfitting.

We try 3 different kernels: linear, polynomial and RBF and first tune the parameters for each to reach the best option by using that kernel, then compare the three best options and find the optimum configuration.

The first kernel we try is the linear kernel. There we compare performance for different values of C and ε. The result is shown on Figure 4.11. As the plot may suggest, wherever we see a minimum approaching, we do a deeper search, so there are more points in this area. We try to minimize C for different values of ε. As ε decreases, we get a better performance and this converges at 0.01, then if we go lower with ε, the error increases by very little. The best result is obtained with an ε of 0.01 (orange curve) and a C of 0.15. The error in this case is 29.73% and this is the lowest error

that can be obtained on these 10 vehicles by using a linear kernel. However, the x-axis is very zoomed, in general, the difference in the error between configurations is very small.

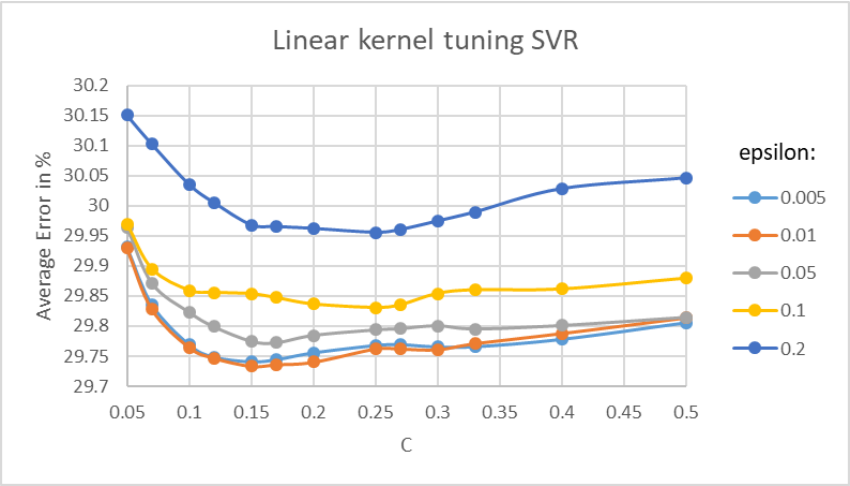

Figure 4.11 Hyperparameter tuning using linear kernel in SVR

The next kernel we try is the polynomial kernel. The polynomial kernel does not yield promising results. In spite of that, we try to optimize its performance to see how low the error can go by using a polynomial kernel. Figure 4.12 shows the process, where the variables changed are again C and ε. The degree of the polynomial is 3, which is the default one (other degrees were also tried and this one had the best performance).

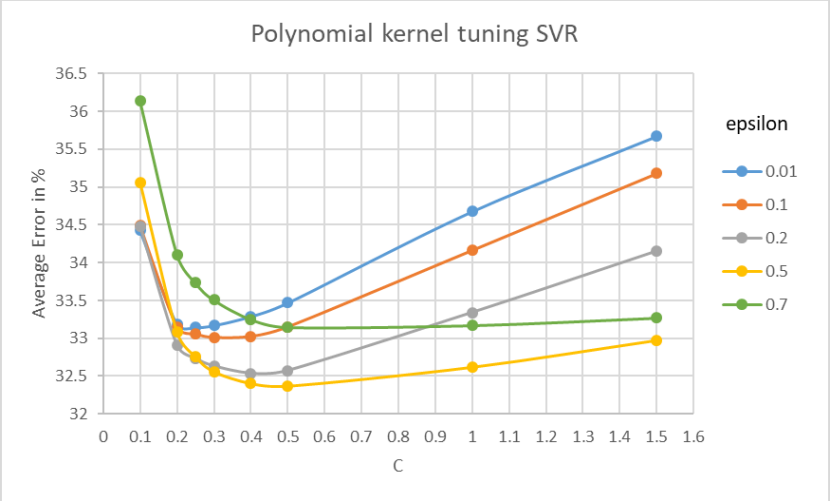

Figure 4.12 Hyperparameter tuning using polynomial kernel in SVR

The best results are yielded by using an ε of 0.5 (yellow curve) and a C of 0.5 and the error obtained is 32.37%, which is worse than using the default parameters of SVR on our problem..

We then move on to the RBF (Radial Basis Function) kernel, where things are more complicated since here, in addition to C and ε, we also have to tune γ (gamma). Scikit-learn has a default value of γ that it uses when left to "auto" and that is 1 / n_features. For us the number of features is different for every vehicle (in the order of 10-20), so this means γ is also different. The results using this kind of gamma are shown on Figure 4.13.

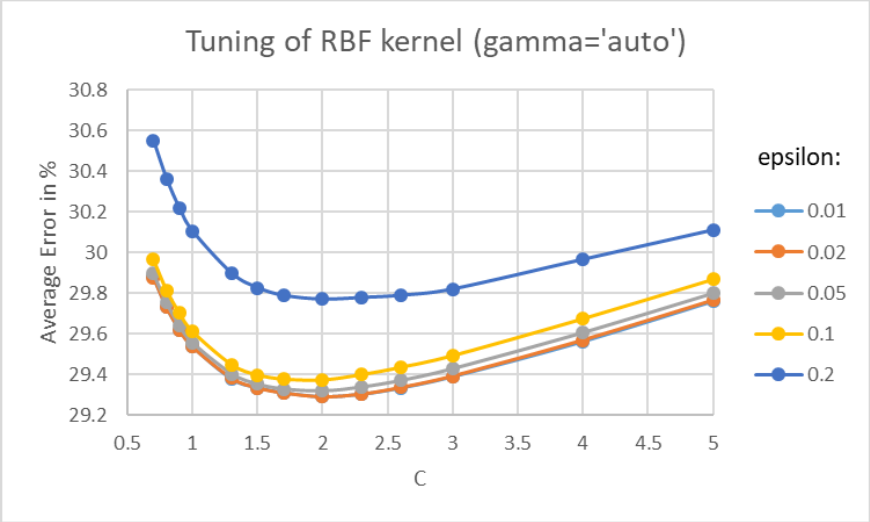

Figure 4.13 Hyperparameter tuning using RBF kernel and gamma "auto" in SVR

If we take a look at the legend and a closer look to the orange curve, we can see that the curves for ε=0.01 and ε=0.02 totally overlap as the ones with lowest errors. Also, an ε of 0.05 and 0.1 are really close to the least error. In this scenario, the best choice for C is 2 and for ε either 0.01 or 0.02. The end result is an average error of 29.29%, which is an improvement from the other kernels.

Then we try to also tune γ and see if that can help us reach an even better result. In this case, we fix ε to 0.02 and search for a good configuration between C and γ. The resulting curves are shown on Figure 4.14.

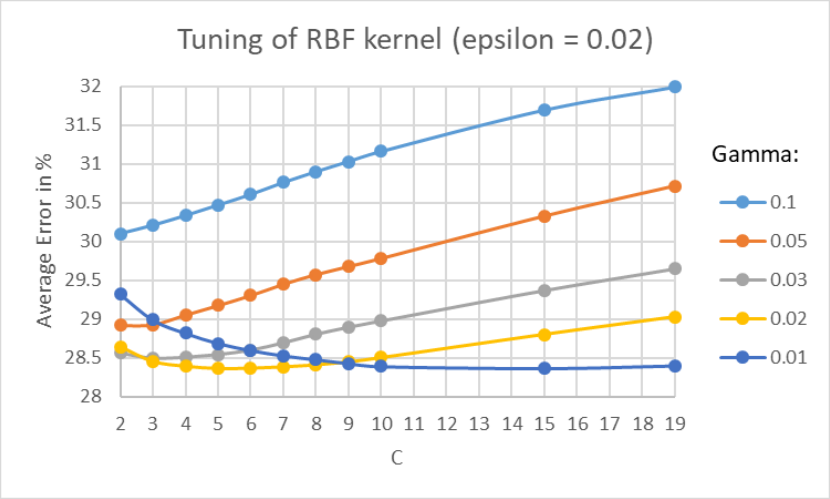

Figure 4.14 Hyperparameter tuning using RBF kernel and epsilon 0.02 in SVR

It is interesting to see that all the curves have the same behaviour, except the curve for $\gamma = 0.01$ (dark blue). This curve reaches the minimum at C = 15, while the other curves for a C lower than 6. The two best results here are: $\gamma = 0.01$ and C = 15 or $\gamma = 0.02$ and C = 5. The average errors in % are 28.3671% and 28.3669%, respectively. They are different only in the third decimal, so we use other indicators to make the decision. We choose the second one ($\gamma = 0.02$ and C = 5, yellow curve) because its minimum is on a lower C, since as C grows, so does computational time. Using the RBF kernel with this configuration, yielded the lowest error of the three kernels.

Finally, the hyperparameters that we use for SVR are the following:

- RBF kernel
- $\varepsilon = 0.02$
- $\gamma = 0.02$
- C = 5

Trying SVR with these hyperparameters on all 57 vehicles of model 1254, we get an error of **28.53%**, which is an almost 2% improvement from the results with the "coarsely" tuned hyperparameters.

4.3.3 Hyperparameter tuning of Gradient Boosting

The tuning is again done on the 10 vehicles mentioned in the previous sections, with IDs: 3959, 3988, 4256, 4551, 4700, 4891, 5003, 5229, 5504 and 5512. If we run the algorithm with the coarsely tuned parameters, as reported in Section 4.2, only on these 10 vehicles, we get an error of 29.15%. So, the goal of the tuning is to find a combination of parameters that yield a lower error than this on the 10 vehicles, which would mean that the error on all 57 vehicles will also be lesser. The hyperparameters that need to be tuned are described in Section 3.4.5 of Methodology.

Gradient Boosting is the most complicated of the three algorithms to tune, because it has a lot of hyperparameters and takes a long time to do the computation. That is why we approach the tuning of the hyperparameters here in a different way. We first determine some parameters that are more important using the default values for all others. Then, when the important parameters have been decided, we proceed to tune other parameters using the already configured ones. This process can be thought of as a tree. We visualize the process and the results obtained on Figure 4.15.

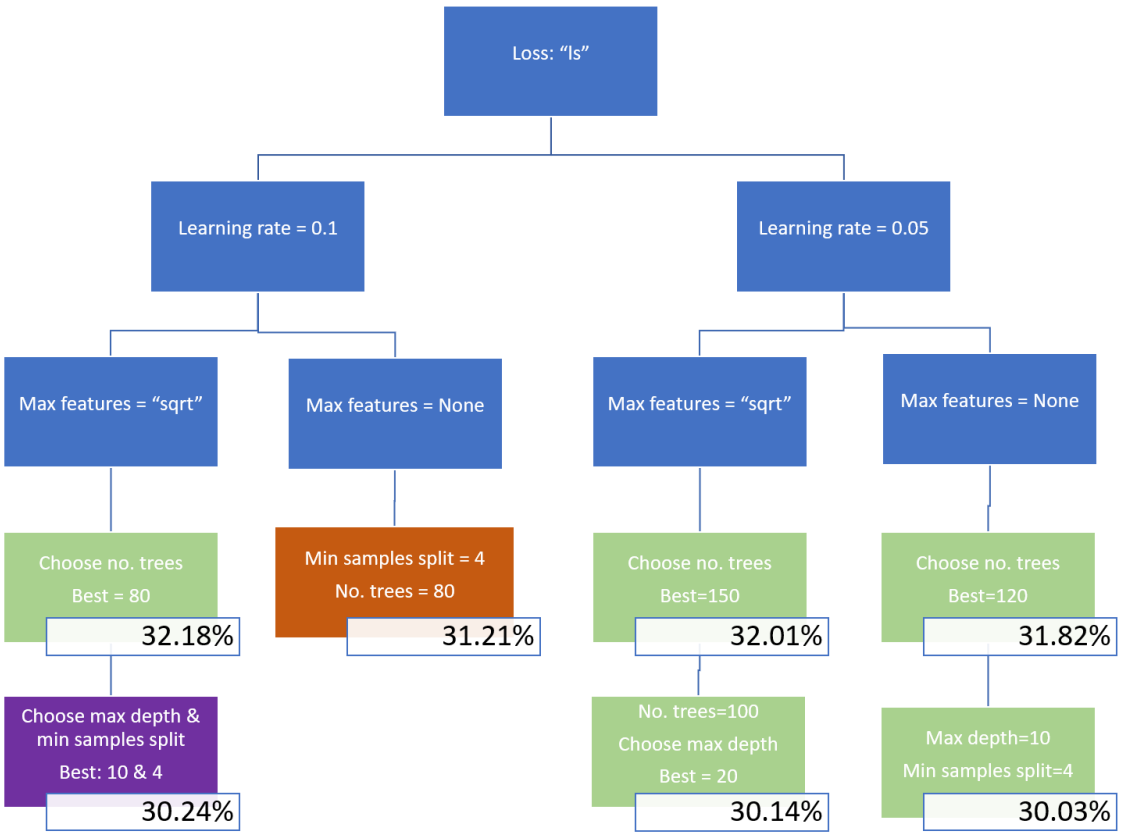

Figure 4.15 Hyperparameter tuning of Gradient Boosting using a Least Squares loss function

The parameters that we start with are the default parameters of Gradient Boosting defined by the scikit-learn Python Library (Scikit-learn.org, 2018), except subsamples (the fraction of samples to be used for fitting the individual base learners), which is set to 0.8, instead of 1, in the attempt to reduce variance and max_depth, which is set to 1 (the default being 3). The other parameters that are set to default, but are to be tampered with in the process of tuning are:

- min_samples_split = 2
- min_samples_leaf = 1
- max_depth = 1
- subsamples = 0.8

When max_features is set to None, it means that it is considering the total number of features when looking for the best split in the individual trees, while if it is set to "sqrt", it considers the square

root of the total number of features. The other variables are numbers, so they are self-explanatory. Of course, we also fix the random state for all experiments in this Section, because the algorithm has some randomicity and we need to keep this parameter the same, so we can really compare the scenarios.

The blue rectangles represent a change of one hyperparameter in the next scenarios – we set a hyperparameter that will be used in all the children of this branch. The green rectangles represent a search that was done with many values of the mentioned parameter and the best one (the one with least error) is reported in the tree. The purple rectangle represents a choice of two parameters and reports the best combination, while the orange rectangle represents one try with the specified parameters different from the parent scenario.

As we go down from the parents toward the children of the tree, there is an improvement in the error. The best scenario is the one that can be seen in the lower right corner of the Figure, which yields an error of 30.4% on the 10 vehicles tried. This is worse than the average error we had with the coarse tuning of hyperparameters, which is 29.15% if we run it on these 10 vehicles. So, we proceed by taking the hyperparameters from the coarse parameter tuning and trying different combinations of them. This scenario uses a loss function Least Absolute Deviation ("lad"), so that we start from this parameter and develop a tree, but this time a more sequential one. The other parameters we start from are all default, except max_depth, which is set to 1. The process is visualized on Figure 4.16.

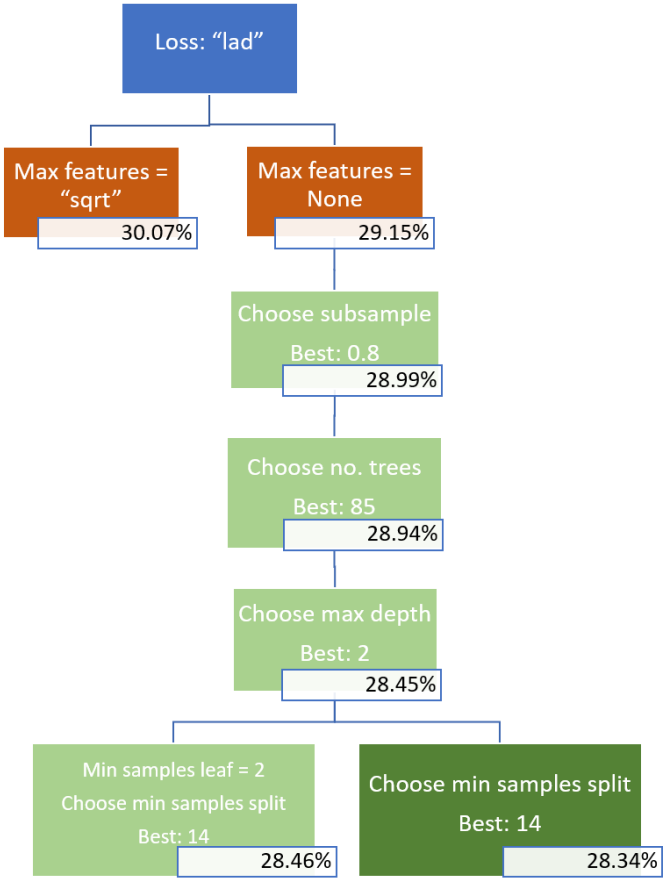

Figure 4.16 Hyperparameter tuning of Gradient Boosting using a Least Absolute Deviation loss function

The legend for the colours of the rectangles is the same as Figure 4.15 and the dark green now represents the best scenario. We start from the scenario that was used for the first run of the algorithms and then proceed to tune parameter by parameter. For completion, we try this scenario also using max_features = ”sqrt”, which yields a worse result, so we do not go on developing the tree in that direction. To visualize some of the processes inside the rectangles, on Figure 4.17 we plot the process of choosing the optimal number of trees (estimators) and the optimal min_samples_split.

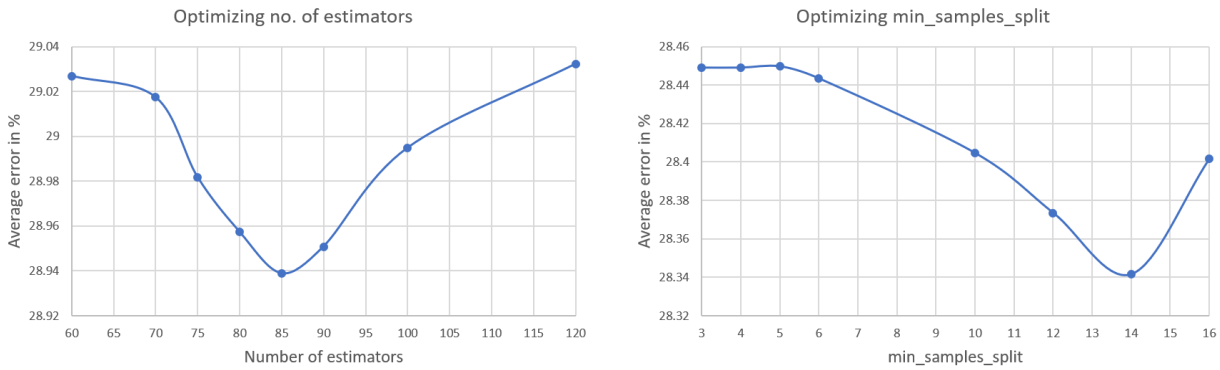

Figure 4.17 Optimizing individual hyperparameters of Gradient Boosting: number of estimators and min_samples_split

Again, as we go down towards the leaves of the tree, we get a better performance. In the end, the optimum parameters found for Gradient Boosting are:

- learning_rate = 0.1
- n_estimators = 85
- max_depth = 2
- loss = “lad”
- min_samples_split = 14
- min_samples_leaf = 1
- subsample = 0.8
- max_features = None

Using these hyperparameters, the Average Percentage error over all 57 vehicles of model 1254 is **28.42%**, which is an improvement of 1.22% over the 29.64% error reported with the coarse parameter tuning.

4.3.4 Conclusions from the Hyperparameter tuning

The tuning was done with the hope of further improving the accuracy of the models. Even though it ultimately did result in an improvement, this turned out to be marginal – only 0.1% for Lasso regression, 1.22% for Gradient boosting and 2% for SVR. However, these improvements were found on only a part of the dataset - 57 vehicles of one model of one type, so we cannot exclude that these improvements may be due to the choice of the dataset – for other vehicles we could observe a different set of optimal hyperparameters. All in all, the best we could do on this part of the dataset is an average Percentage error of 28.42%, obtained by using a carefully tuned Gradient Boosting algorithm and SVR as a very close second with 28.53%.

4.4 RESULTS AND INSIGHTS

In the previous sections, all that is shown is a numerical evaluation of the algorithms, using the Percentage error averaged over many vehicles, or looking at its distribution using a boxplot. The hyperparameter tuning is also done with the objective to lower this Percentage error on global level.

However, our models are done vehicle by vehicle, so it is important to know what is going on at vehicle level - take a look at one vehicle at a time and how the algorithms perform on specific vehicles. Taking a closer look allows us to better see the performance of the algorithms and the methodology in general and to understand why the errors are such.

That is why in this section we break an example of a few vehicles and report some plots and insights on the algorithm performance for these vehicles.

4.4.1 Detailed results of vehicle 4272

Vehicle no. 4272 is of model ID 1254 and it is a Refuse Compactor. It is located in Czech Republic. It is one of the vehicles that were included in the 57-vehicle analysis and it has a pretty regular utilization pattern, which makes it easy to predict. In this section we look at the results of running SVR (with optimized hyperparameters) on this vehicle.

Vehicle 4272 is used from February 2015 to August 2018. To get an idea about its behaviour, on Figure 4.18 we plot 2 months of data, more specifically April and May of 2017.

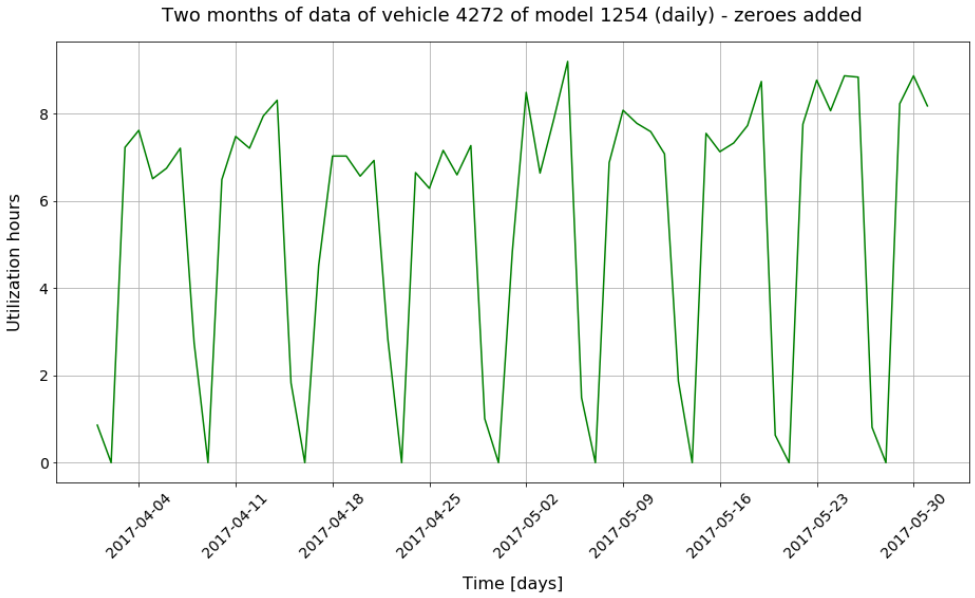

Figure 4.18 Temporal representation of 2 months of daily data of vehicle 4272

We can see that there is a somewhat regular behaviour, the vehicle works around 7 hours in April and around 8 hours in May. The zeroes are also regular in this period and they are all located on Sundays. The minimum value that is found in the time series of this vehicle is 0 and the maximum is 9.29 hours. The mean is 4.87, with a standard deviation of 3.1 hours. Obviously, the zeroes bring down the mean of the whole time series. However, if we remove the zeroes and then compute the mean, we get a more realistic value of the average utilization hours. This value is 6.48 hours with a standard deviation of 1.61 hours. This is also visible on the histogram on Figure 4.19.

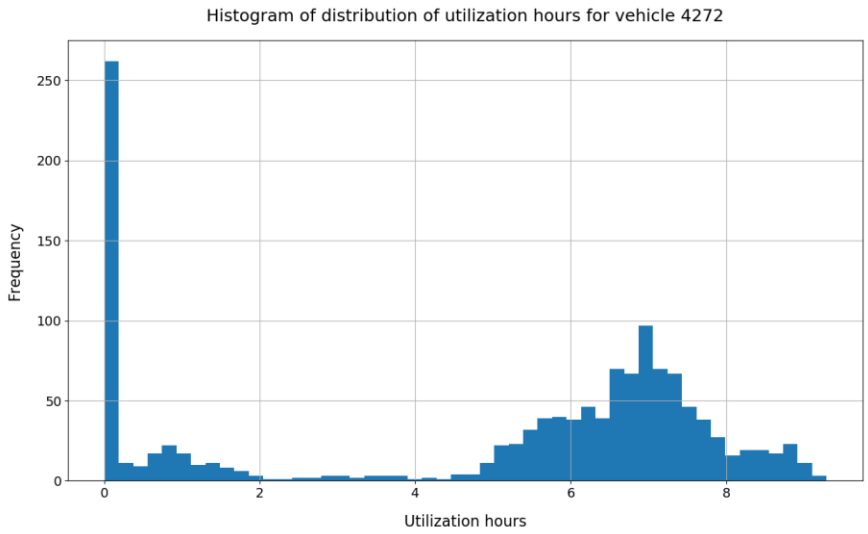

Figure 4.19 Histogram of utilization hours of vehicle 4272

The histogram shows a multimodal distribution, but actually most of the values follow a normal distribution with an additional very high peak at zero and a small peak between 0 and 2 hours.

Then we run SVR on this time series and the results are reported in the next part of this section.

On Figure 4.20, we plot a temporal representation of the real data and the predicted data. Vehicle 4272 has data for 1291 days in total, out of which 253 are zeros. Using the first 140 days for training and 35 days as features, we start getting results from the 175-th day until the end. In total there are 1116 resulting values. The true values are plotted with green and the predicted values with blue.

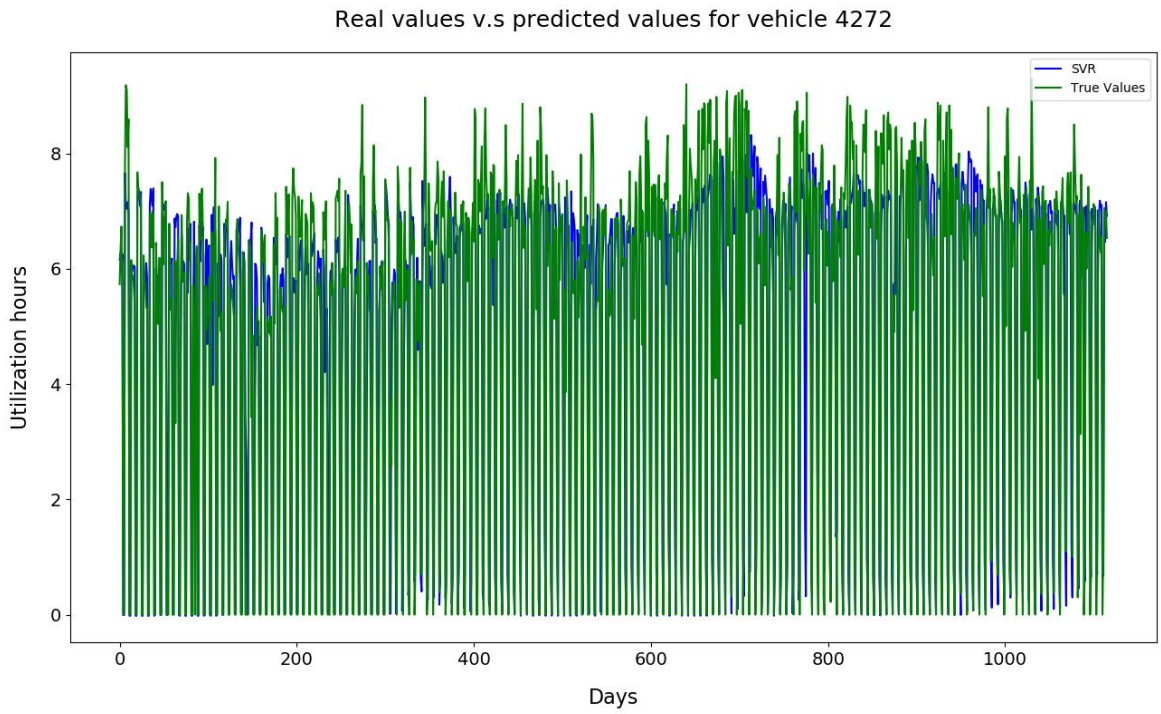

Figure 4.20 SVR performance on vehicle 4272, temporal plot

It is visible that the algorithm can capture the behaviour in general, but cannot get the exact peaks in the data. To be able to see this more clearly, Figure 4.21 shows a zoomed version of Figure 4.20.

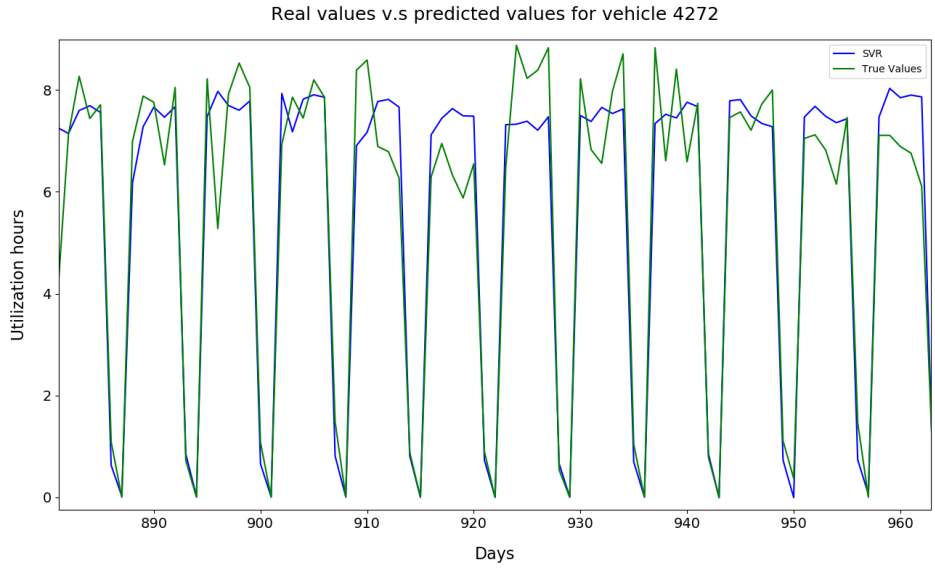

Figure 4.21 Zoom of the temporal representation of SVR on vehicle 4272

When the zeroes are regular, SVR is able to almost perfectly predict them. The peaks, on the other hand present a harder challenge in this case, since they are more variable. All in all, the prediction of 4272 is of high accuracy in comparison to other vehicles. The Percentage Error is 13.32%, which is far below the average of all vehicles for SVR (28.53%). The MAE is 0.67 hours and the RMSE is 0.99 hours. This means that the predictor is on average wrong for this number of hours. Of course, the squared error RMSE is stricter, so the value is bigger.

Another good way to inspect the accuracy and behaviour of the algorithm is with a scatterplot. Figure 4.22 is exactly that. The x-axis represents the real values for y, the target variable and the y-axis labels the predicted values by the algorithm. Obviously, both are measured in hours. The blue line is just a diagonal. If a dot is on the diagonal it means that the true y and the predicted y are the same, which is ideal. So, we want the dots as close to the diagonal as possible.

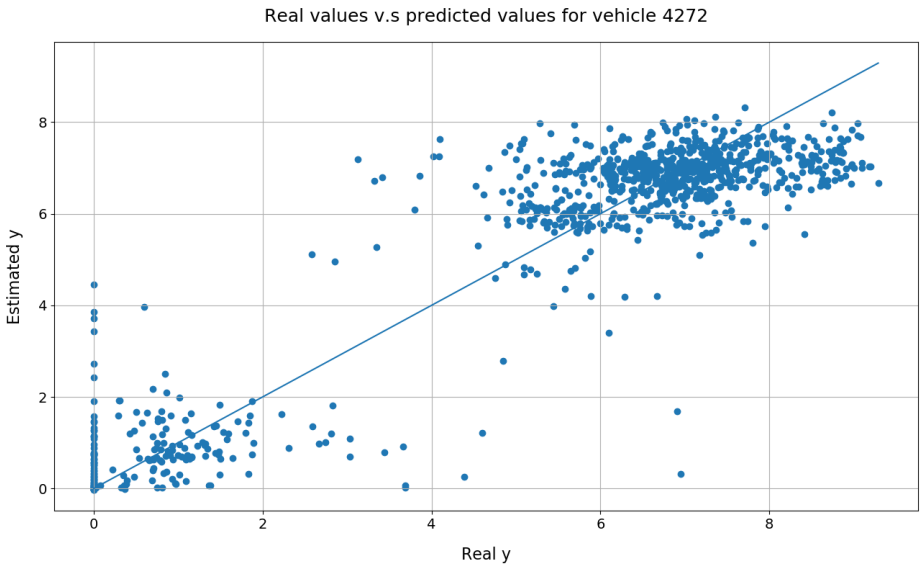

Figure 4.22 Scatterplot of true values and values predicted by SVR on vehicle 4272

There are a few notable things to be seen in this plot. First, the separation of the dots in two groups: it shows that most of the values are either close to zero or between 6 and 8 hours. This is also visible on Figure 4.20 (the temporal one). Second, the vertical line made of points in the lower left corner shows that zeroes in the real data are often mistaken with higher values (real y is 0, while

estimated goes from 0 to 5 hours). This is understandable, since there is a somewhat regular pattern that suddenly drops to zero and the algorithm cannot catch it. Third, the number of overestimates and underestimates. The points the top left half of the plot are the values that have been overestimated, while the points on the lower right half are those that have been underestimated. In this configuration (vehicle 4272 solved with SVR), there are 560 underestimates and 556 overestimates. When these numbers are close it means we have an unbiased estimator, which is good.

Next, we take a look at the distribution of the residual error. As defined in Section 3.5, the residual error $\boldsymbol{r}$ is simply a vector of the absolute value of the differences between each y_{true} and $y_{predicted}$. A histogram of its distribution is shown on Figure 4.23.

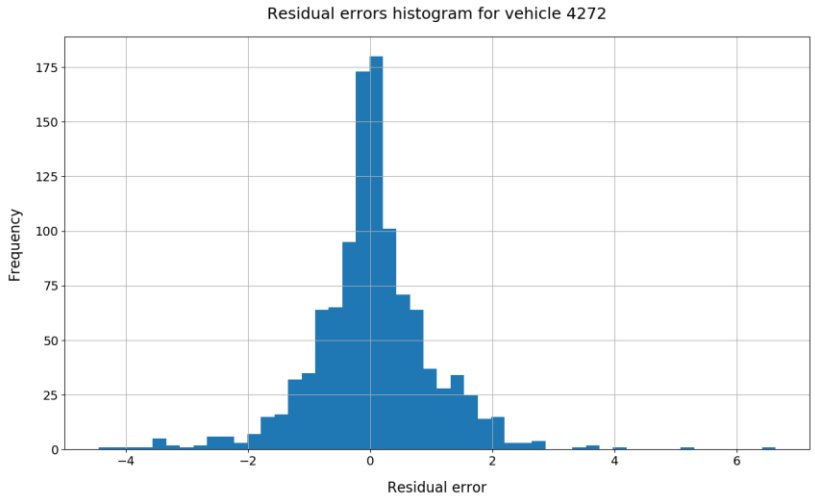

Figure 4.23 Distribution of the residual errors

The distribution of the residual error is a regular normal distribution with mean 0.04 (very close to 0) and standard deviation 0.99 (very close to 1). This is again an indicator that our predictor is unbiased. The distribution spans from -4.45 hours to 6.63 hours of error, but these are extremely rare occurrences. Most of the errors are from -2 to 2 hours and 75% of the errors are within 0.5 hours. As mentioned above, the vehicle usually works for around 7 hours and 0.5 hours offset present around 13%, which is exactly equal to the Percentage error for this vehicle.

4.4.2 Algorithm performance on vehicles with long periods of zero usage

Sometimes, a vehicle would stop working for a while. This is a hard transition for the algorithms to do and they usually make bigger errors in these phases. For some algorithms this is more difficult than for others, for example Linear regression has a hard time grasping these sudden changes of behaviour, while the other ML algorithms find it easier, but still troubling.

An example of such a case is vehicle 3717, which suddenly stops working for a period near the end and then resumes working again. Figure 4.24 shows how both SVR and Linear Regression handle this, with a zoom of the temporal plot. The green curve represents the true values, the red curve the predicted values by SVR and the blue curve the predicted values by Linear regression.

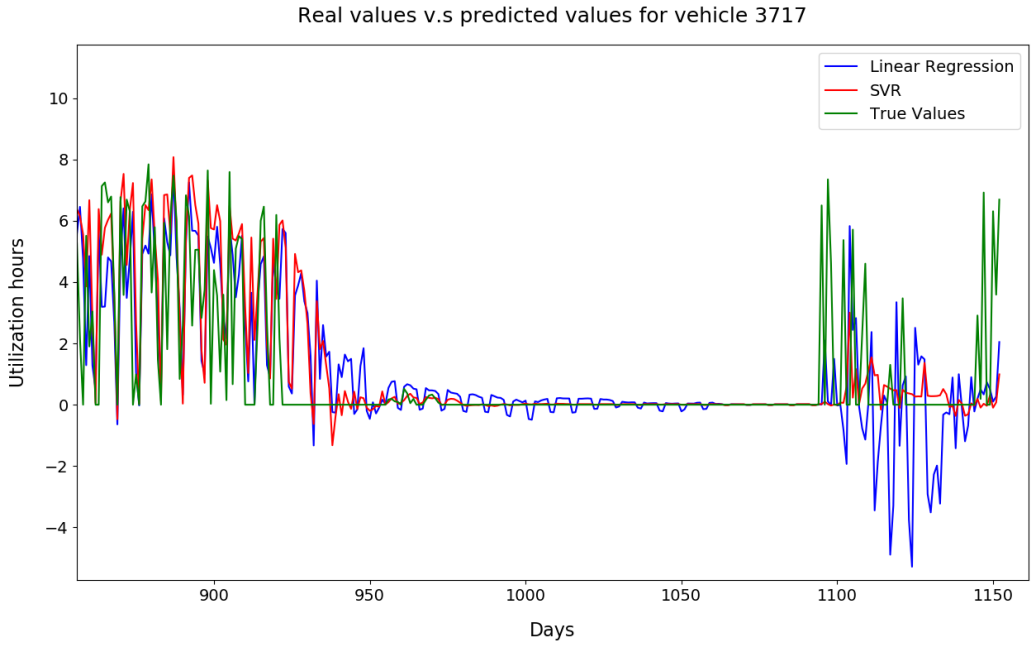

Figure 4.24 SVR and Linear regression handling a period of zero usage

There are two situations to observe here and how the two algorithms handle them: The first one is when a vehicle stops working, suddenly going to a long period of no usage and the second one is when a vehicle starts working again. Both algorithms have some trouble adjusting to both situations, but SVR handles them much better.

In particular, when the vehicle stops working, SVR "converges" to zero faster and needs a shorter time to understand that the vehicle is not working anymore. It means that SVR is more responsive. Linear regression, however, needs 140 days to get to this conclusion. When the vehicle starts working on the other hand, SVR again handles the situation more smoothly, but still needs time to resume regular behaviour. Linear regression performs really badly in this situation, more so than when the vehicle stops working. It starts predicting a lot of negative values with a lot of variability. This is one of the main disadvantages of Linear regression and a reason why it performs worse than the others and cannot handle complicated cases such as this one.

This behaviour of the algorithms is an indicator that algorithm performance should not be evaluated only on average for the whole period, but also in a temporal sense. These zero periods should be detected and the the algorithm should be deemed unreliable for a period of time following a sudden change of usage.

Another notable thing is that both algorithms generate negative values, while we know for a fact that the utilization hours cannot be negative. So, as a future adjustment to the methodology, it should be specified that the target has to be positive or zero.

4.4.3 Algorithm performance on rarely used vehicles

Throughout the whole research process, between all the experiments and trying different algorithms, the question that remained was: What makes a vehicle hard to predict?

First and foremost, the zeroes. The zeroes in general and the zeroes in a sequence – so, when a vehicle is not being used. We cannot infer anything about the usage of the vehicle if it is not being used often and this is a given. Of course, a regular usage pattern is easier to predict than a very noisy one with a high standard deviation, but still the first ingredient is to have enough **usage** data.

Second, unpredictable changes in patterns and sudden very high peaks make a vehicle's usage hard to predict accurately.

On Figure 4.25 we plot the worst-case vehicle of these 57 vehicles that are in the core of our experiments. SVR with tuned hyperparameters exhibits an error of 57% on this vehicle. The vehicle in question is no. 5229.

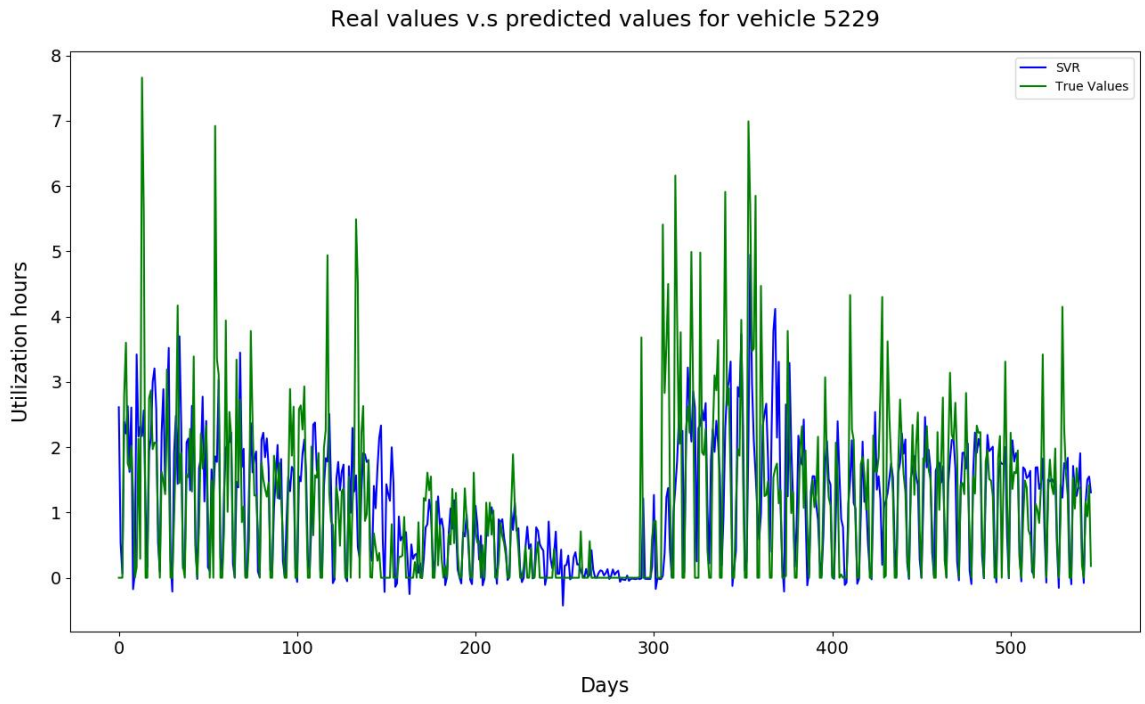

Figure 4.25 Temporal plot of the real and predicted values by SVR of vehicle 5229

In general, the vehicle has low utilization, with a mean of 1.19 hours and if we remove the zeroes, a mean of 1.85 hours. But that is not the problem, the problems are:

- Sudden high peaks of utilization exhibited often

  They can be seen throughout the whole plot and their timings seem random. High peaks like this are hard for any algorithm to catch and they cost it a lot in the average error.

- Changes of pattern of usage

  This vehicle has a really irregular pattern of usage. We can split it into 4 periods: first a regular usage with a mean of around 1.5 hours (0-150th day of prediction), then a drop in the usage

leading to a halt and the halt itself (150-300th day), then a sudden start of very high usage (300-380th day) and finally a stable period with a mean of 1.5 hours again (380th day until the end).

The zeroes play a big part in both of these problems: if it is not for them, the peaks would not be that much higher with respect to the other values. The pattern would still be irregular, but more reasonable.

All of the reasons mentioned in this section and the previous one, lead us to construct Scenario 2: Predict usage on next working day, where we remove the zeroes and predict utilization where there actually is utilization.

4.5 A SIMPLER PROBLEM: PREDICT USAGE ON NEXT WORKING DAY

It is visible from the previous Chapter that the zeroes which make our time series evenly spaced, are very hard to predict. They seem random to the algorithms and the models cannot capture this behaviour. Moreover, a period of consecutive days where the vehicle was for some reason not used are further confusing the algorithm. As mentioned in Section 3.1: Problem formulation, there is another way to approach the problem that would make it simpler and easier to predict and that is to keep the unevenly spaced time-series. We assume that the user of the solution (the construction company) would know which day will be the next working day (the date), so we can proceed to predict the utilization hours for that day.

The goal of this problem formulation is not only to remove the zeroes that were added for temporality, but to generally remove all values from the dataset where it seems that the vehicle was turned ON, but was not actually utilized. For example, it was only re-parked on the jobsite, or driven a few meters for some reason. This is not considered as utilization, so we remove these values from the dataset.

In particular, what is removed is all the data samples (days) in the dataset that mark utilization hours lower than 1h. Construction vehicles are heavy and of high power and some of them stay

ON without moving for approximately 20 minutes to warm up the engine. So, the value of 1 hour was decided along with professionals from Tierra as a reasonable threshold of time above which we can consider that the vehicle was really working.

An example of a time series of one vehicle with this kind of problem formulation is shown on Figure 4.26. The vehicle is 4256 of model 1254 of type Refuse Compactor.

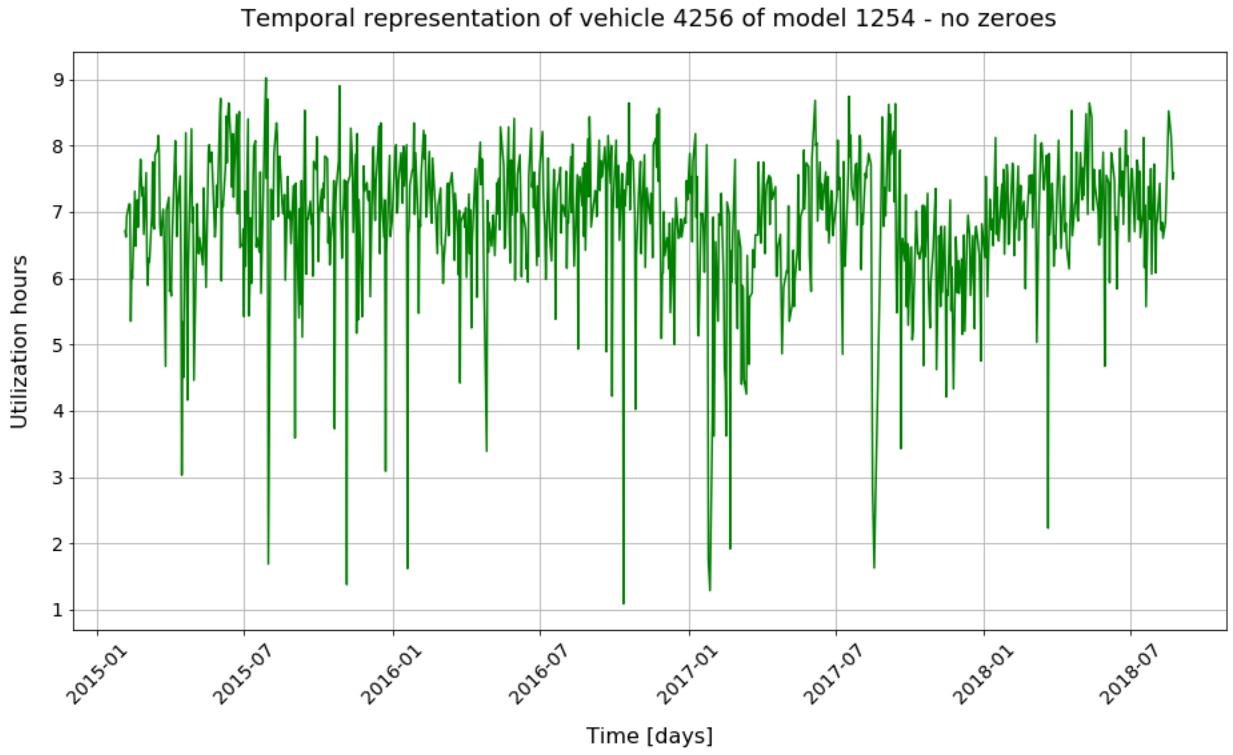

Figure 4.26 Temporal representation of vehicle 4256 with all values for utilization hours less than one excluded

This way the usage pattern is much more regular. There are still some values that go as low as 1 hour, which should not be considered as usage and this gives us the notion that the threshold should maybe be dynamic and depend on the mean and standard deviation of the usage of each vehicle. However, in most cases 1 hour is a reasonable choice and the outliers are rare, so for simplicity we continue using 1 hour.

This change of scenario afflicts a change in the system. We have to tune some general parameters again, such as the sliding window size and the feature window size and take a look at the autocorrelation function to decide whether using Smart Feature Selection is in this case an improvement. We are not doing experiments using Expanding window, since we have already decided to use a sliding window cross-validation approach, for reasons explained in Section 4.1.3. Also, we are using the additional feature "isWorkday", since it has proven helpful for the prediction. This time, to do the tuning, we use SVR, as a fast and effective algorithm. The hyperparameters used for SVR are those that were already tuned in Section 4.3.2. Then we try the rest of the algorithms and evaluate their performance on this slightly different time-series.

The tuning is done on the same 10 vehicles from the previous Sections. First, we try the Normal feature construction, where we take past consecutive days as features and we fix the past window size (sliding window, training window size, length of training matrix X) and the Feature window size (how many past consecutive days to use as features, width of training matrix X). For the past window size we test values of: 90, 100, 110, 120, 130, 140, 150 and for the feature window size, values of 4, 5, 6, 7, 8, 10, 20. The results are shown on Figure 4.27.

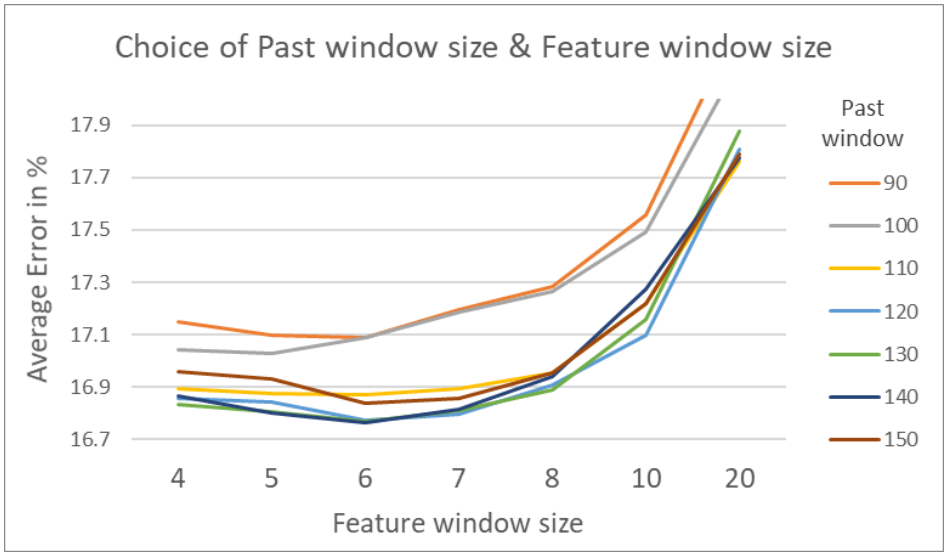

Figure 4.27 Choice of Past window size & Feature window size: No zeroes scenario

Past window sizes of 120, 130 and 140 all have a good performance and are almost equal in the minimum. We choose a past window size of 140 (dark blue curve) and a feature window size of 6.

Another notable thing from this plot is the y-axis values: for this problem formulation the average errors are around 16-17%, which is half of the average error magnitude for the first scenario that includes an equally spaced time series full of zeroes (Predict usage next day).

Next, we try the Smart Feature construction and tune the past window size along with the acf_value for this case. For the past window size, we again try the values 90, 100, 110, 120, 130, 140, 150 and for acf_value we try 4, 5, 6, 7, 8, 10, 15 and 20. The results are shown on Figure 4.28.

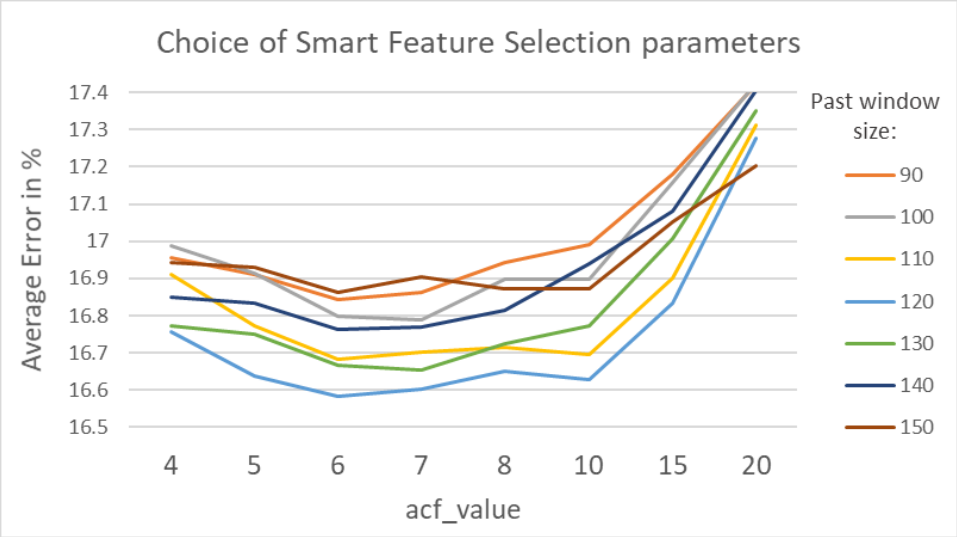

Figure 4.28 Choice of Smart Feature Selection parameters: No zeroes scenario

The optimum values in this case are a past window size of 120 (light blue curve) and an acf_value of 6. This value is much less than the optimum acf_value for the previous scenario (Predict usage next day), which was 20.

What is interesting here is that the Smart feature selection does not perform better than the Normal feature construction, since the values are generally only correlated with the first few lags, that are

also captured by the Normal feature construction. This is why the Normal feature construction performs almost the same and even a bit better.

To prove this, we run SVR, Gradient Boosting and Linear Regression (for comparison) with both scenarios (normal and smart). We also run one of the benchmark algorithms, Moving average to check if the ML algorithms are performing better. For it, we set the window of past values for taking the average to 30, in both the normal and smart scenarios (in the smart scenario this means we set acf_value=30). The average Percentage errors over all 57 vehicles are shown on Table 4.2.

Table 4.2 Average Percentage Error of different algorithms for Normal and Smart feature construction

	Mean of the Errors	
Algorithm	Normal	Smart
Moving average	21.628	21.829
SVR	16.633	16.648
Gradient Boosting	17.142	17.190
Linear Regression	17.304	17.354

The table reports that the results of using the Smart feature construction are almost equal to using the normal one, differing only after the first decimal. Since we can say that the performance is the same, it is better to use the Normal feature construction because it is computationally less demanding. In the next experiments, we use Normal feature construction.

Another interesting thing from the Table is the performance of the benchmark algorithm Moving average. The results it yields are close to the ones of the ML algorithms. This shows that the dynamic of these kinds of time series can be somewhat accurately captured by a moving average.

Figure 4.29 shows a boxplot of the error distributions of the algorithms on this scenario.

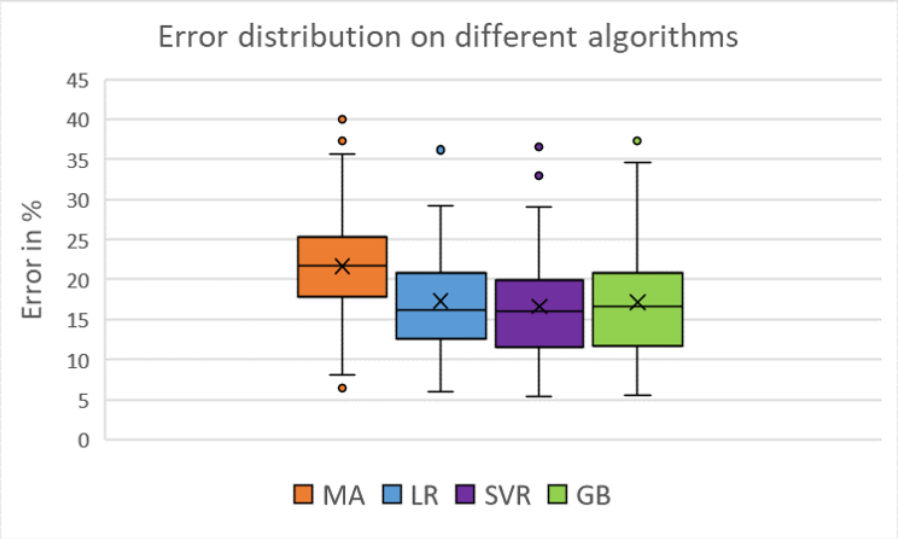

Figure 4.29 Error distribution on different algorithms: No zeroes scenario

Again, we can see that in this scenario, Moving average comes closer to the other algorithms with performance, than in the previous scenario (Figure 4.8). When predicting usage on the next day, the mean error yielded by Moving average was around 14% higher than those yielded by the ML algorithms, while in this case it is just 5% higher.

In general the errors by the algorithms here are lower than the previous scenario (Figure 4.8), spanning from 5% to 30% for the ML algorithms (note that the y-axis spans from 0% to 45%), with a few negligible outliers. In the previous scenario the span of the errors in general was from 10% to 60%, so this scenario also has tighter distributions – lower variance. The best performing algorithm is SVR, having the lowest mean and median and a low variance, but all the algorithms show a more or less similar performance.

Going into the details of the results obtained with this scenario (Predict usage on next working day), we again report the results from one vehicle, this time vehicle no. 4256 of model 1254, of type Refuse Compactor. The time series of this vehicle is shown earlier in this Section, on Figure 4.26. The histogram of utilization of it, however, is shown on Figure 4.30.

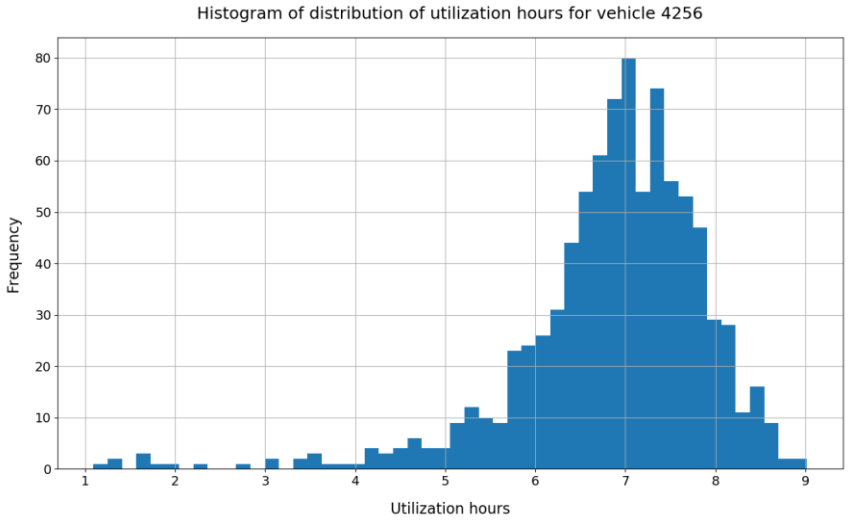

Figure 4.30 Histogram of utilization of vehicle 4256: No zeroes scenario

Now the histogram is clearer than in the previous scenario – the peak at zero is not there and we can better see the distribution of the utilization hours. This vehicle is pretty regular – follows a normal distribution, with a long tail up to 1h, which is understandable and also visible from the temporal plot (Figure 4.26). Even in the scenario with zeroes, there is usually some usage in the lower hours, seen for example on the histogram of utilization hours of vehicle 4272 with zeroes (Figure 4.19).

Then we run SVR on this time series and the results are reported in the next figures.

On Figure 4.31, we plot a temporal representation of the real data and the predicted data.

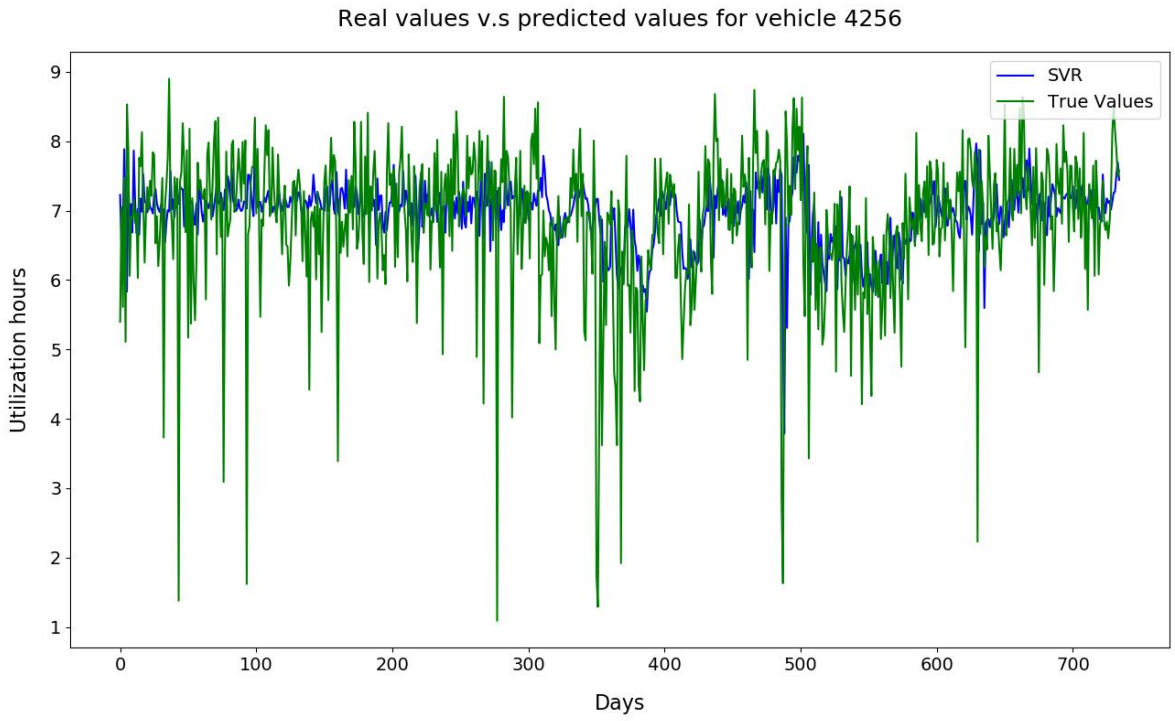

Figure 4.31 Temporal representation of real vs. predicted values by SVR

The algorithm is much more capable of capturing the behaviour in this case. The data is noisier than the prediction and there are some peaks that are hard to catch by any algorithm, but in general the performance of SVR is good. This can also be inspected by seeing the scatterplot of the real vs. predicted values on Figure 4.32.

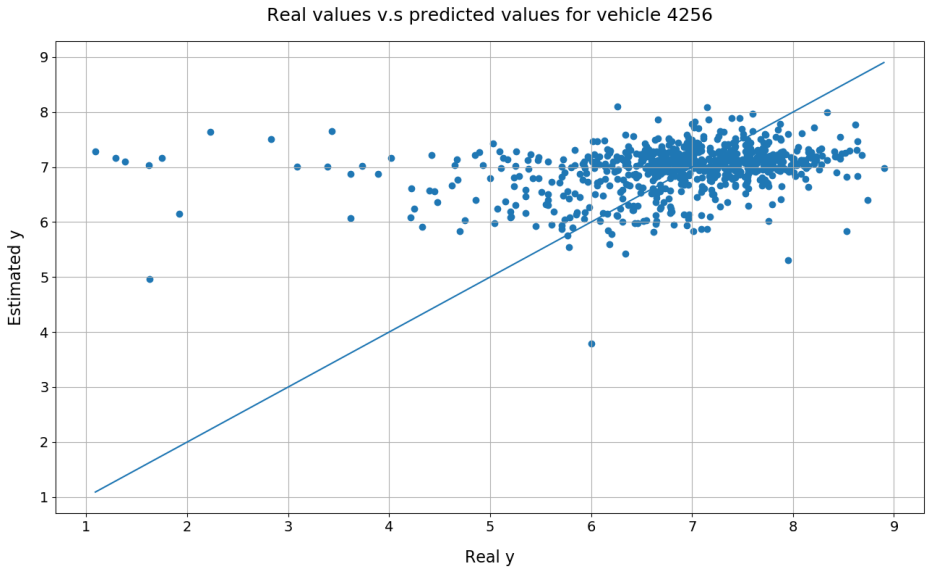

Figure 4.32 Scatterplot of real vs. predicted values by SVR

The values are generally close to diagonal, with some outliers. Something that can be noticed is that the algorithm tends to overestimate the small values and underestimate the high values, and this is also visible on the temporal plot on Figure 4.31.

To sum up, Scenario 2: Predict usage on next working day is issuing an average error over all 57 vehicles of around 16%, while Scenario 1: Predict usage on next day is yielding an average error of circa 28%. This means that the problem in Scenario 2 is much easier to solve. However, which scenario will be used depends on the customer's preference.

5 CONCLUSIONS

In conclusion, the thesis posed a problem of predicting future utilization hours of construction vehicles and offered methodologies to solve it using time series forecasting with machine learning algorithms. First, the problem and the background were posed, then the Data was examined in detail, after which a formal problem formulation was derived and methodologies were suggested to solve it. Finally, some results were reported on the performance of these methodologies.

The problem proved a difficult one to solve and was yielding high errors. Moreover, making the time series temporal (equally spaced) constructed a complicated problem. Relieving the data of these zeroes and setting a threshold that detects if the vehicle was really working that day simplified the problem and much better results were obtained.

In terms of the performance of different ML algorithms, they were much better than the Benchmark algorithms, but did not differ a lot in performance between themselves. With a bit of fine hyperparameter tuning we were able to lower the error, but there was only so much that we can do from the side of the algorithms. The more important part, we found, was the input to the algorithm, in particular, the features.

In the study we used only temporal features, meaning features related to the working hours of the previous days (historical values) and information about the day to be predicted - is it a working day or not. The features can be further enriched by other perspectives, like the season of the year, the weather on that day, information about the project a vehicle is working on, geographical information about the jobsite and the kind of work done there etc. These are only some ideas of how to improve the accuracy of the models by using more features as input.

Furthermore, due to the very different usage patterns between vehicles, we decided to do a different model per vehicle. Another approach may be to first cluster the vehicles according to their usage patterns and then try to make group predictions where we could use historical data of more vehicles to predict the future utilization of one. This could be done with the help of unsupervised machine learning techniques.

Another issue that should be considered is a better performance evaluation of the algorithms. We do an average prediction error on all models (one per day) of a vehicle. But, as we could see, the

vehicles express different periods of usage patterns through the years and this should be accounted for in the error evaluation. If a vehicle has a period of no usage, for example, there should be a transient of a number of days where we say the prediction will be unreliable, until the predictor gets back on track.

In any dataset, there is a certain margin of unpredictability that cannot be extracted by any data mining technique. Using the tested techniques, we can say that we found a soft upper bound of the unpredictability of our dataset: we know that there exists a technique that can yield an error below 29% (17% considering the non-zero scenario) on the tested 57 vehicles.

In general, as we said in the first chapter, where there is data, there is power. We can find many ways to handle the data and explore various interesting insights. We can formulate and solve different problems and help the customer better exploit their product. It can lead to scientific conclusions for the researchers and economic improvements for everyone. The key is data.

6 REFERENCES

Akhavian R. and Behzadan A. (2012). “An integrated data collection and analysis framework for remote monitoring and planning of construction operations”. *Advanced Engineering Informatics*, 26, 749–761

Akhavian R. and Behzadan A. (2013). “Knowledge-Based Simulation Modeling of Construction Fleet Operations Using Multimodal-Process Data Mining”. *J. Constr. Eng. Manage.*, 139(11), 04013021

Bakhiet O. (2017). “Data collection for management of fuel consumption in vehicles and machinery: A study on the challenges and strategic possibilities in the construction industry”. Degree in environmental engineering, KTH Royal Institute of Technology, Stockholm, Sweden.

Fan H., AbouRizk S., Kim H. and Zaïane O. (2008). “Assessing Residual Value of Heavy Construction Equipment Using Predictive Data Mining Model”. *J. Comput. Civ. Eng.*, 22(3), 181-191

Kaya M., Keles A. and Oral L. (2014). “Construction Crew Productivity Prediction By Using Data Mining Methods”. *Procedia - Social and Behavioral Sciences*, 141, 1249 – 1253

Rasdorf W., Frey C., Lewis P., Kim K., Pang S. and Abolhassani S. (2010). “Field Procedures for Real-world measurements of emissions from Diesel construction vehicles”. *Journal of infrastructure systems*, 16(3), 216-225

Tierra introduction presentation. (2018).

Beaulieu, K. and Dalisay, D. (2018). Machine Learning Mastery. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/ [Accessed 26 Nov. 2018].

DataCamp Community. (2018). Hyperparameter Optimization in Machine Learning. [online] Available at: https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models [Accessed 30 Nov. 2018].

Faggella, D. (2018). What is Machine Learning? - An Informed Definition - Artificial Intelligence Companies, Insights, Research. [online] Emerj. Available at: https://emerj.com/ai-glossary-terms/what-is-machine-learning/ [Accessed 3 Dec. 2018].

Learning, M. and Python, C. (2018). Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/ [Accessed 1 Dec. 2018].

Medium. (2018). Gradient Boosting from scratch – ML Review – Medium. [online] Available at: https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d [Accessed 26 Nov. 2018].

Python.org. (2018). Welcome to Python.org. [online] Available at: https://www.python.org/ [Accessed 30 Nov. 2018].

Scikit-learn.org. (2018). scikit-learn: machine learning in Python — scikit-learn 0.20.1 documentation. [online] Available at: https://scikit-learn.org/stable/ [Accessed 30 Nov. 2018].

Scikit-learn.org. (2018). 3.2.4.3.6. sklearn.ensemble.GradientBoostingRegressor — scikit-learn 0.20.1 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor [Accessed 2 Dec. 2018].