Master Thesis

Development of a NO_x Emission Model Using Advanced Regression Techniques

August 16, 2020

Juul Romkema

Faculty of Electrical Engineering, Mathematics
and Computer Science (EEMCS),
University of Twente

Examination Committee:
dr. D. Bucur (UT)
Prof. dr. M.E. Iacob (UT)
ir. A. Smit (Emission Care BV)
ir. M. Gent (Emission Care BV)

Abstract

Polluted air is a big problem nowadays. As air pollution has many downsides, governments have laws in place to minimise the amount of these substances in the air. One of the measurements is that gas turbines need to know the amount of emission they produce. One way to achieve this is by measuring the amount of emission with the aid of measurements tools. However, the measurements tools are expensive and require much maintenance. In recent years, we have therefore seen the use of prediction instead of measuring. A predictive emission monitoring system (PEMS) is one way to track the emission in gas turbines. A PEMS consists of an emission model which is able to calculate the emission based on sensors within the gas turbine. As it is important that the emission is accuractly tracked, the legislation is quite strict. PEMSs need to comply with many rules to be put into use. This makes building a PEMS an labour-intensive task where a lot of field knowledge is required.

Emission Care is an example of a company that builds PEMS for their clients. A big part of this job is the cleaning of the data and the selection of the input features. In their work they experience the amount of time it takes to retrieve those insights. Currently they select the input features based on physics and years of experience. To model their findings they license the software to build their neural network-based PEMS. However, the iterative strategy they apply now is time- and labour expensive and would benefit from a supporting tool.

In this research we focused on the creation of an emission model and how data-driven techniques and machine learning can be used to support this process. The aim was to support the modeller by finding appropriate input combinations as well as validating them with the use of our model in order to reduce the time-spent on building a PEMS. In this research we first did a literature study to understand PEMS better and to determine how other studies approached the development of a PEMS. Furthermore, we looked into various regression techniques and how we could incorporate historical information to models that are not built for time series.

The conducted research has resulted in a feature selection algorithm that is able to support the process of selecting feature combinations. The feature combinations proposed are tested with the current CEM software and similar scores, R^2-score of 0.97, are obtained as for model currently used. Furthermore, the proposed feature combinations are also applied to our own developed emission models. These models are based on linear regression, tree-based methods, support vector regression and neural networks. We found that the models based on data with historical aspects performed similar to the ones without the historical aspect. If we compare the scores to the CEM software scores, we find that results are comparable for the tree-based method and the support vector regression model which would make those emission models good candidates to substitute the current emission model. However, the emission model is only a small part of the PEMS and therefore, the emission models found in this research serve as proof that the current CEM software peforms well.

Contents

List of Figures

List of Tables

Acronyms and Terms

Acronym/Term	Meaning
ABC	Artificial Bee Colony
ACO	Ant Colony Optimisation
Adam	Adaptive momentum estimation
AK	Adaptive Kernel
BGD	Batch Gradient Descent
BP	Back Propagation
BPNN	Back Propagation Neural Network
BRT	Boosted Regression Trees
BT	Boosting Trees
CEM(S)	Continuous Emission Monitoring (System)
CN	Combination Number
CV	Cross-Validation
ELM	Extreme Learning Machine
EN	ElasticNet
FIS	Fussy Inference Systems
GA	Genetic Algorithm
GRNN	Generalised Regression Neural Network
GS	Grid Search
GSA	Gravitational Search Algorithm
iRPROP	Improved Resilient Backpropagation
IV	Interval
KRR	Kernel Ridge Regression
LASSO	Least Absolute Shrinkage and Selection Operator
LGOCV	Leave-Group-Out Cross Validation
LR	Linear Regression
LSSVM	Least-Squares Support Vector Machine
LSTM	Long Short-Term Memory
MAE	Mean Average Error
MAPE	Mean Absolute Percentage Error
MLP	Multi-Layer Perceptron
MLR	Multiple Linear Regression
MSE	Mean Squared Error
NARX-RNN	Nonlinear Autoregressive Networks with exogenous inputs
NN	Neural Network
PCA	Principal Component Analysis
PCR	Principal Component Regression
PCR_FS	Principal Component Regression with the scores added in a forward stepwise fashion
PEMS	Predictive Emission Monitoring System
PLS	Partial Least Squares
PSO	Particle Swarm Optimisation
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
R	Correlation between x and y
R^2	Coefficient of Determination
RF	Random Forest

Acronym/Term	Meaning
RFE	Recursive Feature Elimination
RFECV	Recursive Feature Elimination Cross-Validated
RMSProp	Root Mean Squared Propagation
RNN	Recurrent Neural Network
RR	Ridge Regression
RRS	Residual Sum of Squares
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SVR	Support Vector Regression
tanh	tangent hyperbolic
TLBO	Teaching-Learning Based Optimisation
VIF	Variance Inflation Factor

1 Introduction

Polluted air is a big problem nowadays. An air pollutant can be defined as a substance in the air that can potentially harm humans, animals or our ecosystem when the concentration is high enough. This substances can be either solid particles, liquid droplets, or gasses. Nine of out of every ten persons breathe polluted air. The World Health Organisation (WHO) even estimates that seven million people die anually due to exposure to heavily polluted air [3]. 90% of these deaths are in low- and middle-income countries in Asia and Africa. They are followed by Europe and America. In other words, air pollution is a global problem.

Most air pollution is caused by humans and directly results from human activity. The three most well-known emitted substances are CO_2, SO_x and NO_x. The x in SO_x and NO_x means that the number of oxygen compounds can differ. This research will mainly focus on NO_x emissions. 21% of NO_x emissions in Europe is caused by energy production and distribution [4]. NO_x can cause both health issues and environmental problems [5, 6]. NO_x, for instance, generates acid rain when it reacts with hydroxide (OH) in the air. Because of the low pH of acid rain, plants, aquatic animals and our infrastructure are affected. When the nitrogen oxides react with ultraviolet light from the sun it can also result in photochemical smog. This is a form of polluted air can cause adverse health effects for the people breathing it in. Altogether, NO_x is considered to be one of the heaviest air pollutants.

Polluted air not only causes many deaths, but also contributes to the climate change [5]. Both polluted air and global warming influence the quality of life on earth negatively. In order to improve, the amount of harmful substances in the air must be limited. To achieve a limited amount of substances in the air, agreements are needed. On global, continental, nationwide and local level laws are in place to limit the emissions. Most of these require plants and other big emitters to measure their emissions. As continuous measuring tools are expensive to purchase and maintain, several countries also allow prediction-based monitoring. One way to predict emissions is with the aid of a predictive emission monitoring system (PEMS).

Emission Care is one of the companies that develops PEMSs for their customers. Their customers are located in the Netherlands and Norway. In order to build a PEMS they license CEM software developed by Rockwell Automation [7]. This software uses neural networks in order to build a PEMS. One of the components of the PEMSs is an emission model. An emission model is built based upon input data, gathered by sensors in the gas turbine, and is responsible for estimating the emissions. They build PEMS by selecting input features based on physics of the process producing the emissions and start designing emission models. Selecting the input features and training emission models is done iteratively and based on the model performance it is judged whether another iteration is required and the input features are adapted accordingly. In this way Emission Care slowly moves towards a set of features providing a model that performs best in terms of uncertainty, robustness and maintainability. However, the process of selecting the input features and training the different emission models is very time-expensive. Besides that, as the current approach is based on field-knowledge, known feature combinations are often tried and tweaked. The main advantage of this approach is that the combinations have already proven themselves and are explainable in terms of physics which is an important requirement for the delivery of the PEMSs. However, this approach also creates blinkers, as a solution is mostly sought in the same direction as previous times.

This research focuses on providing support while creating a PEMS. Rather than looking at the physics, we focus on the characteristics of the data to create input combinations. With the aid of data-driven techniques an algorithm is designed that provides its users with insight into the data and suggestions with regards to possible feature combinations. In order to test this algorithm the CEM software is used, however as the machine learning field emerges fast we also examined a number of other machine learning techniques. In other words, the focus of this research was on providing insight and recommendations in

terms of the input features as well as the development of an emission model. This brings us to our main question:

> **M-RQ:** How can data-driven techniques and machine learning be applied to support the creation of an emission model?

The main question actually consists of two parts. The first part is focused on preparing the data for the emission model. Whilst the second part aims to investigate the various machine learning techniques available and to build an emission model. This results in the following research questions:

> **RQ1:** How can feature selection be applied to support the creation of an emission model?
>
> **RQ2:** Based on existing literature, what machine learning techniques are good candidates to be used for an emission model with time series input data?
>
> **RQ3:** Which machine learning technique is most suitable for an emission model in terms of uncertainty, robustness and maintainability?

The conducted research is distributed over several chapters. In chapter 2, the background, gives a more extensive explanation of what PEMSs are and their acceptance within Europe. It also includes a more detailed description of what the emission model entails and how it relates to the other components of a PEMS. Next, in chapter 3, the various machine learning techniques are explained. The chapter starts off with a short part about data pre-processing and then it explains the four machine learning techniques – linear regression, tree-based algorithms, support vector regression and neural networks – as used for this research. Then, in chapter 4, the related work is discussed. The focus of this chapter is on the implementation as done by other researches and their results obtained. Thereafter, in chapter 5, the data preparation for the emission model is discussed. Apart from creating insight and cleaning the data this also entails the feature selection algorithm as developed in this research. The results from chapter 5 can then be applied in chapter 6 where the implementation and results of the different machine learning techniques are discussed. Chapter 6 first briefly introduces the different hyperparameter tuning techniques as used in this research and next it discusses the various machine learning techniques. The method section of the every machine learning technique is directly followed by its results as acquired knowledge is applied for the following techniques. Then the limitations of this research can be found the discussion in chapter 7 and the documented is concluded by chapter 8 which entails a conclusion where the three research questions and the main question are answered. Furthermore an overview of the contribution of this research is given and future work is discussed.

2 Background

This chapter presents an overview of PEMS. It will provide background information about PEMS in order to give a better picture of this niche. First, legislation in Europe is discussed with a focus on the Netherlands. Followed by how PEMS actually work, what role they play in this research and their acceptance in Europe.

2.1 Emission legislation in Europe

Emission legislation and guidelines are often drafted on a European level. All European member states, plus Norway, Iceland and Liechtenstein (European Free Trade Association, EFTA states) are required to follow the EU environmental guidelines. In 2001 the EU issued a directive, the large combustion plant directive (LCPD, Directive 2001/80/EC), that specified limits for flue gas emissions for combustion plants with a thermal capacity over 50 MW. As of 2016 these large combustion plants must comply with the industrial emission directive (IED, Directive 2010/75/EU). This directive aims to control and reduce the impact of industrial emissions on the environment. [8]

One of the ways to achieve this aim is by raising the cost of emissions. For instance, in Europe there is a trading system for CO_2 emissions. Industrial companies need rights for the emissions they produce. Part of those rights can be obtained freely for every company, but to emit more additional rights have to be purchased. These rights can be bought at an auction. On top of the costs for obtaining emission rights, industrial company owners often have to pay taxes for the emissions they emit. Norway is one of the countries with an NO_x tax [9]. The government introduced it as an incentive for industrial companies to reduce their emissions. The NO_x tax income is used to subsidise investments in NO_x reduction measures in the industry.

Each EU member state implements the EU guidelines in national legislation which forms the legislative framework for companies and citizens of the member state. The implementation of the EU environmental guidelines in the Netherlands is given as an example: Small industrial companies in the Netherlands must satisfy the requirements as stated in the Activities Decree, Activiteitenbesluit in Dutch, which consists of general requirements with regards to emissions and environmentally harmful matters. However, for large industrial companies this is not sufficient, they need a specialised permit. These specialised permits consist, among other things, of emission limit values (ELVs) for air and water, but also of limits for the amount of waste and noise. Mostly these specialised permits are based upon the Activities Decree. The Activities Decree also refers to the European best available technique (BAT) guidelines. BAT guidelines state which technique is advised to protect the environment. Companies must use or implement this guidelines unless they can prove that it does not work or cost are exorbitantly high. However, in most situations BAT should be a sufficient method to satisfy the requirements as stated in a permit. [10]

EU guidelines and member state legislation refer often to standards, describing technical details for implementation and quality assurance of requirements given in the guideline/legislation. Two of the most well-known parties to write down standards are the international standards organisation (ISO) and the European norm (EN). Members can easily use these standards to incorporate the norm in their own environmental legislation. An example of a European norm written out as standard is EN 14181:2014[11]. This standard specifies the need for continuous monitoring for power plants with a capacity over 100 MW. The norm specifies quality assurances needed for automated measuring systems (AMS). It consists of three quality assurance levels and an annual surveillance test (AST). An AMS is a continuous emission monitoring system (CEMS) or a predictive emission monitoring system (PEMS). CEMS measures emissions directly from a stack. However, the measurements tools used are expensive to purchase and maintain and requires a monthly calibration [11]. PEMSs, on the other hand, determine the emission based on settings of the installation. Due to the specific nature of PEMS, PEMS shall also comply with CEN/TS 17198 (applicability, execution and quality assurance of PEMS). [10]

2.2 Predictive Emission Monitoring System (PEMS)

This section will describe PEMS in more detail. It will first elaborate on how PEMS work followed by an overview of the acceptance of PEMS across Europe and the rest of the world.

2.2.1 PEMS explained

Simply put a PEMS is a system that is established based on data measurements at the plant. The resulting system, in combination with combustion data from the plant, is able to calculate the emissions in the chimney. However, in practice there are many requirements that PEMS have to meet in order to be accepted as valid emission monitoring system.

Building a PEMS consists of four phases: (1) functional design, (2) stack testing, (3) PEMS engineering, (4) PEMS online [1]. The first step starts with drafting a document stating, among other things, the project scope. Furthermore, the variables that should be measured at the plant are determined. In the next phase stack testing is performed. Emission data from the stack and process data of the plant is collected under variable operating conditions of the plant. The stack test duration is typically three to five days. All operating modes of the plant that can influence the emissions are to be predicted are tested, such as plant capacity, air/fuel ratio, combustion air temperature, fuel type, and so on. A visual representation of stack testing can be found in Figure 2.1. These measurements are done using CEMS. The aim during these measurements is to cover normal operation conditions, but also a wide variety of scenarios so that the drafted PEMS is able to deal with boundary situations. Depending on whether available, historical data can also be used to determine a PEMS. With these two streams of data a PEMS is drafted. [12]

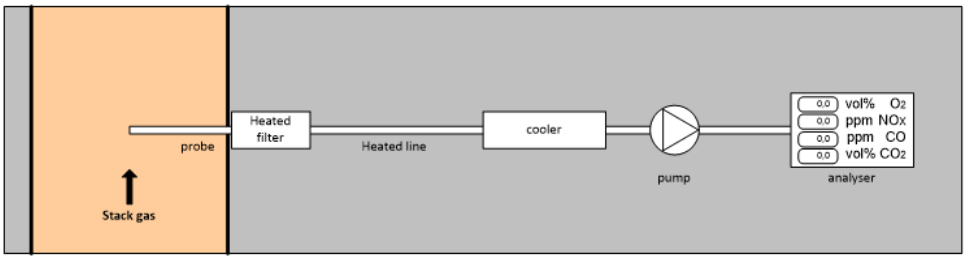


Figure 2.1: Stack testing [1]

When all measurements are done, we can move on to the third phase: engineering of the PEMS. Figure 2.2 actually consists of multiple models. Every sensor that is used as an input in the PEMS emission model is modelled in the sensor validation system. Whenever a sensor does not work properly the value can be replaced by a modelled parameter from the parameter model. Thanks to these modelled parameters the emission can be predicted, even when a sensor fails. Sensor validation contributes to the accuracy of the emission model and is therefore obligatory to include in an approved PEMS [12]. The emission model uses the validated input parameters to generate the output. On the emission model a daily integrity test is performed. This test does nothing else than making sure the model remains unchanged. The engineering of a PEMS, furthermore, entails the configuration of a PEMS data management system. This is where the process data is saved.

The last phase starts when the engineering of the PEMS is fully finished. During this step the PEMS is put online. In this phase the software is installed on a clients' computer to make sure the measurement data from the sensors is written to the PEMS data management system. Furthermore, the sensor data and PEMS is used to generate predictions. Normally, validation is performed against data collected during the stack test at the start of the project. However, if this is not sufficient for the authorities, a

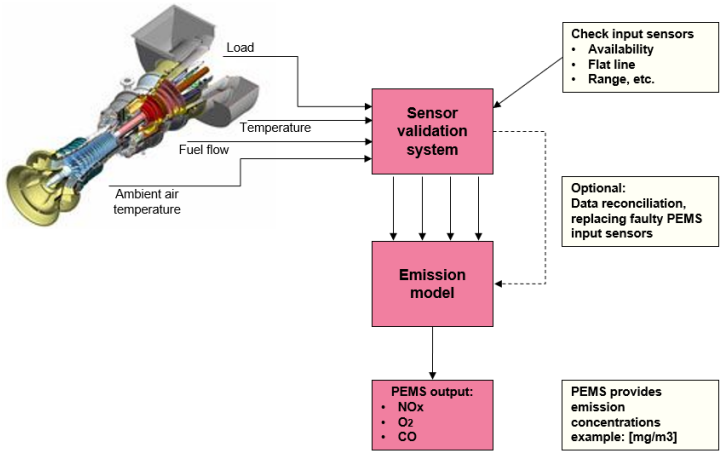


Figure 2.2: PEMS overview [1]

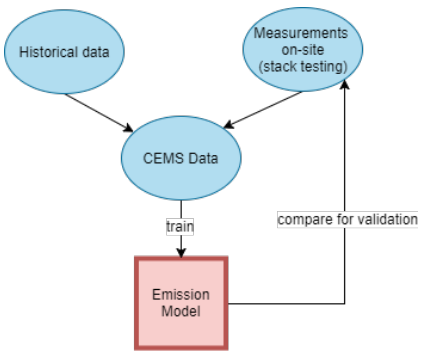
second stack test is performed from which the measurements are compared to the PEMS predictions. All these phases are also documented and will result in a document when the PEMS is build and validated.

2.2.2 PEMS in this research

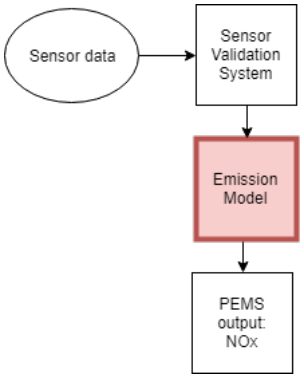
In this research we only focus on the emission model as part of the third phase of building a PEMS. There are many techniques that can be used to draft the emission model. The available techniques can be grouped in three kinds of models:

1. physical models,
2. statistical models and,
3. hybrid models.

Physical models are mostly focused on fitting variables so that the function of the inputs results in the desired output. Statistical models are more focused on selecting the variables. Examples of this technique are regression models and neural networks. And at last, we have the hybrid models which focus on the selection of variables through statistics and physical relations [1]. This research will mostly focus on the second type of models, statistical models. Statistical models enable us to build a PEMS without knowledge of physical relations. In this research the statistical models are developed with the aid of various machine learning techniques as introduced in chapter 3.



(a) Design of our emission model



(b) Our emission model incorporated in a PEMS

The focus of this research is to build the emission model for PEMS. In other words, the validation of the parameters is not taken into account and therefore the PEMS build is only valid in combination with other software that does validate the parameters of the model. Figure 2.3a gives a schematic overview with all the elements that are included in our emission model. Then, in Figure 2.3b one can see how our emission model for PEMS would function in an operational PEMS. Although an emission model only seems to entail building a model, the feature selection is a big part of it. This research also supports this task. Besides that, explaining the emission model is also from great importance. As we develop our own emission model this enables us an opportunity to give more insight in the process than is currently available.

2.2.3 Acceptance of PEMS

At their introduction PEMS were often used as back up of a CEMS. However, more and more countries nowadays accept the PEMS model as an alternative for a CEMS. The advantage is that their initial cost only consists of developing the model and there are no additional measurement instruments to maintain and install. In other words, the costs of PEMS in comparison to CEMS are much lower.

Before reading this section, it is important to keep in mind that part of this section is based on research from 2014 [13]. One should take into account that acceptance of technologies takes time and is subject to change. Despite the benefits that PEMS have over CEMS, not all countries allow the usage of PEMS. This is caused by a lack of trust in the predictive performance of PEMS. As mentioned before, the Netherlands allow the usage of PEMS by law. This same goes for Denmark. Also, the United Kingdom permits the usage of PEMS although this does not seem to be enshrined in law. Main condition is that the operator is able to demonstrate that the PEMS produce valid results. In practice this means that PEMS must regularly checked by CEMS. There are also countries, like Norway, that only accept the usage of PEMS on offshore locations. Maintaining and installing CEMS in offshore locations is expensive and therefore the Norwegian government sees PEMS as a cost saving alternative for these locations. In other countries like France and Sweden they were still experimenting with PEMS back in 2014. However, there are also European countries, like Germany and Italy, that are against PEMS. In other words, the acceptance of PEMS really differs per country. When it comes to PEMS acceptance outside of Europe we see that in most of the US states and in the middle east PEMS are accepted and used, because of their low purchase costs. [9, 13].

However, the acceptance of new technology can take time and often countries want to see proof of concept before implementing and accepting something themselves. The European pre-norm as published in August 2018, CENTS 17198-PEMS, might be an encouragement for more European countries to accept PEMS.

3 | Machine Learning (Literature)

In this section the machine learning is discussed as we encountered it in literature. This section starts with machine learning in its broadest sense. It will then scope based on our data. The machine learning problems that can be applied in our case will then be explained in the next sections.

In this research the aim is to develop a model that is able to predict the NO_x emissions. In order to develop the model, measurements are done on location for a timeframe of 3 to 5 days, up to 8 hours a day. In these days they test multiple scenarios – covering the boundaries to make sure the model does not exceed its training boundaries during regular use. In other words, the input parameters as well as the output parameters are measured and therefore known for this period. This indicates a supervised learning algorithm as a best fit for this research.

Supervised learning algorithms are the algorithms that require that input and output data are known beforehand. However, a distinction can be made based upon the form of the output data. The range of data can either be limited or unlimited. Whenever the number of outcomes is limited, we refer to it as categories and therefore this is known as classification. On the other hand, when the number of outputs is unlimited we call it regression. [14]

The aim of this research is to predict the NO_x emission. Although this number will most likely be within a certain range most of the time, this number can have virtually any value. In other words, this research aims to solve a regression problem.

Regression analysis is used to determine the relationship between two or more variables. Regression analysis is one of the most used techniques when it comes to analysing data with multiple factors [15]. It is often used for data description, parameter estimation, control and prediction & estimation.

If regression data is collected at a single time-period, this is called cross-section data. If the regression data is not collected within a single time-period, we refer to it as time series data [16]. This is also the case for the emission data as used for this research.

This chapter first continues with a short introduction to data pre-processing. Next it introduces the regression methods most often used: linear regression, decision trees, support vector regression, and neural networks [17]. In the following sections, respectively Section 3.2 till 3.5, these methodswill be discussed. Thereafter, Section 3.6 discusses ensemble learning which aims to use multiple machine learning algorithms to improve the predictive power.

3.1 Data pre-processing

Data pre-processing is a very important step to prepare data for machine learning algorithms. Data is often messy which makes it difficult to process for machine learning algorithms. Cleaning data can be a time expensive task, it can even take up to 80% of the project time, but it is proven to be effective for the machine learning results [18].

The messiness of data often already starts with the different ranges that every input variable has. Machine learning algorithms often have trouble distinguishing various situation, which affects the determination of a model. Therefore, it is better to standardise the data, before it is processed [19]. Chollet recommends, in his book [20], to use feature-wise normalisation. In case of feature-wise normalisation every variable is normalised separately by subtracting the mean and standard deviation of the applicable variable. Afterwards, all variables in a dataset are centred around zero, meaning that the ranges are comparable while the ratio of values within a variable are still intact.

Other challenges often observed are missing data points or data points with unusual values known as outliers. Missing data is derived from other variables when possible and otherwise deleted. The outliers, on the other hand, should always be deleted [21]. Depending on the size of the dataset one can also reduce the size by compressing the data. In this way the same machine learning techniques can be applied using less resources. As said, this is particularly useful for large datasets. For plants that only have measurements from 3-5 days this will not be necessary, however when a lot of historical data is present it might be worth considering compressing the data. An often-used method for data compression is principal component analysis (PCA). This method compresses the data by reducing its dimensionality. In other words, it reduces the number of features. The advantage is that it also removes the noise. Another way to reduce the number of features is by feature selection. Feature selection can be applied to make sure all features are relevant. Besides reducing the complexity of the model, this can also prevent overfitting and can therefore increase the model its accuracy. We will not go further into this topic, but in Section 4.6 one can find which feature selection methods other researchers have used in their work.

For this research, we also have some field-specific data pre-processing challenges. For instance, it is of great importance that the time stamps of the different variables are aligned. Furthermore, during the measurements days at the plant everything is measured from starting up the plant to shutting it down. However, for building an emission model we are only interested in the normal operations [12]. These normal operations do not include the process of starting up or shutting down a plant nor does it include the measurements when the load is adjusted. Another challenge to keep in mind is that the data must be physically probable, whenever physically improbable trends occur in the data, these sections also have to be removed from the dataset, since the accuracy of these data points cannot be guaranteed.

3.2 Linear regression

Regression analysis is a statistical technique for modelling and investigating the relationships between an outcome variable, also known as a response variable, and on or more predictor variables, also known as regressor variables. Regression analysis is often used to predict future response variables based on one or multiple predictor variables.

Simple linear regression is a linear regression model that contains only one regressor variable. A model of that kind is written as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where y is the response variable, x the regressor variable, β_0 and β_1 the unknown parameters, also known as regression coefficients, and ϵ the error term. To be more precise, β_0 is the intercept and β_1 is the slope. Both these parameters must be determined by estimation based on sample data. The error term, ϵ, accounts for the failure of the model to fit the data exactly. Oftentimes an assumption is made about its distribution. An example of this simple linear regression can be found in Figure 3.1.

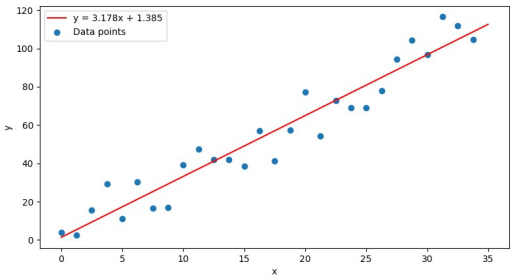


Figure 3.1: Example of linear regression

However, in most situations this model is too simple to capture everything. A more extended version of simple linear regression is general linear regression, also known as multivariable linear regression (MLR). MLR does not differ that much from a simple linear regression, however it contains more than only one regressor. The model is written as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + \epsilon$$

where the parameters $\beta_0, \beta_1, ..., \beta_k$ are referred to as partial regression coefficients. Both models as introduced in this section are linear regression models because they are linear in the unknown β's. The models do not tell us anything about the linearity between the regressor and the response variable(s) [16].

The formula for the model is adapted based on the kind of data it is fed. In this research we are dealing with time series data, as explained before. The regression model for time series data can be written as:

$$y = \beta_0 + \beta_1 x_t 1 + \beta_2 x_t 2 + ... + \beta_k x_t k + \epsilon \text{ with } t = 1, 2, ..., T$$

The unknown parameters, the β's, in a linear regression model are typically estimated using the method of least squares. One should take into account that autocorrelation can occur when working with time series. The presence of autocorrelation means that the data is correlated with itself at different time periods. The danger of this is that it affects the ordinary least-squares regression procedure [15]. In other words, it affects the adequacy of the model. In Section 3.2.1 autocorrelation and four other possible threads to model adequacy are named.

3.2.1 Model validity

It is important to check whether the regression model fit is accurate, rather than to assume it is. There are five different things that need to be checked in order to say something useful with regards to the adequacy of the regression model. Oftentimes graphical analysis of residuals is used to check this adequacy and the underlying assumptions it is based on.

Linearity is needed for (multi)linear regression; thus, it is important to check whether it is also actually there. This can be checked by a normal probability plot of the probabilities. This means that we need to plot the residuals against the regressor. This plots the cumulative normal distribution as a straight line. In the ideal situation all or most data point lies approximately on this straight line. If this is not the case, then a linear regression method is probable not suited. [15]

Multivariate normality entails checking whether the residuals follow a normal distribution. Two important characteristics are the mean of zero and a constant variance. Whether this is satisfied can be checked by a normal probability plot of residuals, this plot is based on the QQ-plot which can also be used to check normality. The x-axis, the scaling, remains unchanged when comparing to the QQ-plot, but the y-axis will now be the associated probability. The normal probability plot needs about 20 point to produce probability plots that are stable and easy to interpret [15]. One can also use the measurements skewness and kurtosis to prove normality. Skewness tells us something about the symmetry, or lack of it, of the data and the kurtosis is a measure the tell whether the data is heavy- or light-tailed relative to a normal distribution. For a normal distribution the skewness should be close to zero and the kurtosis should be three. [22]

No multicollinearity since this impacts the ability to estimate regression coefficients. Multicollinearity occurs when regressors are nearly perfectly linear related. One of the techniques to detect multicollinearity is with the aid of a correlation matrix. Examining the correlation between the regressors is helpful method to detect multicollinearity between pairs of regressors only. Unfortunately, when more than two regressors are involved in a near-linear dependence, there is no assurance that any of the pairwise correlations will be large. Therefore, it is recommended to also run experiments with variance inflation

factor (VIF). VIF quantifies the severity of multicollinearity in an ordinary least-squares regression analysis. One or more large, when it exceeds 5 or 10, VIFs indicate multicollinearity. In order to remove multicollinearity one can, remove the variable causing it or centre the data by subtracting the mean score from each observation for each independent variable. [15]

Homoscedasticity of variance basically means that data points should all have approximately the same distance from the line. There should be no clear pattern in the distribution. For this purpose, one can the plot of residuals against the fitted values. However, it is also possible to calculate whether the data is homoscedasticity. The rule of thumb here is that whenever the ratio of the largest variance to the smallest variance is equal or lower than 1.5 the data is homoscedastic. When the data is homoscedastic one can remove this by a non-linear transformation or an addition of a quadratic term. [23]

No autocorrelation which can be checked by plotting the residual in time sequence. Autocorrelation occurs when there is a correlation between model errors at different time periods. One can use the Durbin-Watson test, which is based on statistics, to test for autocorrelation. For uncorrelated errors r=0 the Durbin-Watson statistic should be approximately 2. If the value is bigger there is autocorrelation. Autocorrelation can be solved by adding one or more new predictor variables. When this does not work one can use the Cochrane-Orcutt method or the method of maximum likelihood to estimate the parameters. The maximum likelihood method is the preferable option when the autocorrelative structure of the errors is more complicated than a first-order autoregressive. [15]

3.3 Decision trees

Decision trees use a tree structure to specify decisions and their consequences. A decision tree aims to predict a response or output variable, γ, given a number of input variables, $X = x_1, x_2, \ldots, x_n$. A decision tree consists of a root, decision nodes, leaf nodes and branches. An example of a decision tree can be found in Figure 3.2. The decision nodes, coloured in red, represent a test on a variable and the branches, the black arrows, the decision made. The leaf nodes, coloured in green, also known as terminal nodes, are the nodes at the bottom of the tree which include the decision one should make [24].

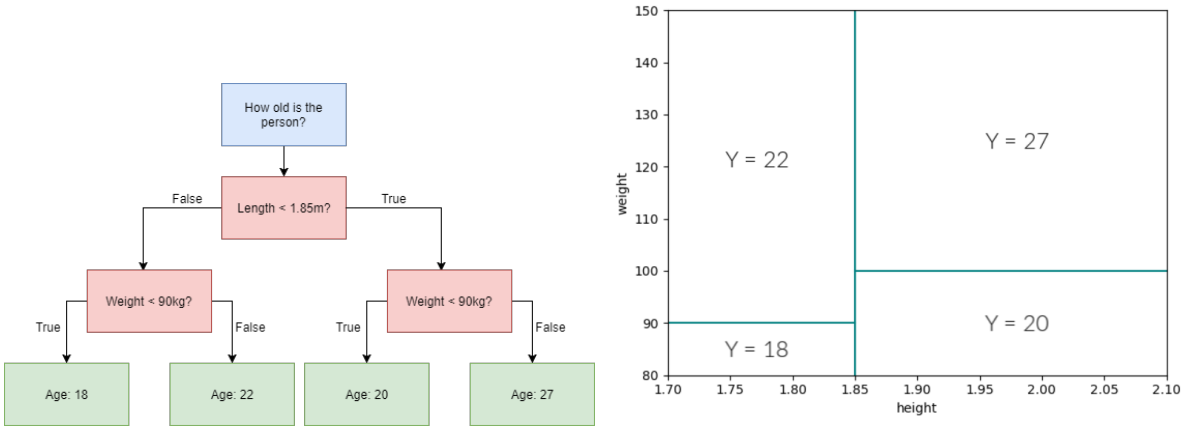


Figure 3.2: Example of a regression tree [2]

Decision trees are often used because of their clear visual representation of the decision-making process. Decision trees can produce both categorical and continuous outcomes, we respectively refer to them as classification trees and regression trees [25]. In practice decision trees are commonly deployed for classification purposes [24], while we see their usage less for regression problems. This is due to the fact that we cannot have a leaf node for every value in our training set and even if we could this would mean

the tree would be extremely overfitted. In order to solve a regression problem with a decision tree the problem needs to be generalised or categorised to some extent. How the regression tree is built, including how the features are chosen and split, can be found in Section 3.3.1 Thereafter, in Section 3.3.2 , we will describe how hyperparameters, pruning and multiple models can prevent the overfitting of regression trees.

3.3.1 Building a tree

A decision tree can handle continuous variables; however, the data becomes discrete to a certain extent. The most used option is to determine a threshold and determine to what side the data point belong based upon this threshold. Determining one threshold instead of making multiple bins for every variable makes more sense, since we can always decide to split the remaining part of the variable again after it has been split. The threshold can be randomly determined, for instance, so that an equal amount of data points is on both sides. However, the more usual way is by recursive binary splitting [26]. This is a greedy top-down approach of splitting, since it only takes into account the current split and works from the root till the leaf nodes. The splitting is based on the residual sum of squares (RSS). The average of RSS is the variance. If we would look at the variance, then the most useful split would be a split were the combined weighted variance of all child nodes is less than the original variance of the parent node. However, with recursive binary splitting one focuses on RSS. The idea behind is that it measures the variance for every split for every variable. In the end, the variable and belonging threshold that is chosen is the one that has the lowest RSS score of all. In the end, this results in the fact that a data set is split into multiple smaller regions. With the aid of the decision tree one is able to determine in which region its data point is located. However, this means that the tree is not able to return a continuous value, but rather gives a region back. This region is represented by the mean value of this region. In other words, the output value is a mean value of the region, rather than a specific value. One has a certain range, but not a concrete answer. These bins all have specified ranges, however this also means that trees have trouble predicting what they have not seen before. In other words, trees are good at interpolation but have trouble with extrapolation.

3.3.2 Avoid overfitting

With regression trees there is good chance of overfitting. The bins are, of course, not obvious for regression and therefore a regression tree will proceed till every leaf node consist of only one data point. However, this makes the leaf nodes too specific and only applicable to really specific data points, in other words the tree is overfitted. In order to prevent this overfitting one can constrain the tree size, prune the tree, or use multiple trees.

Constraint tree size: one can constraint the tree size by the tuning of the hyperparameters or by early stopping. The hyperparameters of a decision tree include, for instance, the minimum number of samples needed for a node split or for a terminal node, how many samples must be left in a bucket or the depth of the tree. Just like other cases of hyperparameter tuning, a grid search is mostly used to determine the suitable value for these parameters. However, one can also follow the rule of thumbs. A common rule-of-thumbs is, for instance, that at least 0.25% up to 1% of the data samples must be in each leaf [27]. The other option is early stopping, also known as pre-pruning. Just like hyperparameters it restricts the size of tree, but it does so by checking the cross-validation error at each stage. If the error does not decrease significantly enough, then the training of the tree is stopped. However, this leaves space for underfitting.

Tree pruning: pruning the tree happens after the training is finished, therefore it is also known as post-pruning. The idea is to remove the nodes that do not contribute any additional information. Two well-known approaches of tree pruning are minimum error and smallest tree. The tree is pruned back to the point where the cross-validated error, MSE, was at a minimum. In case of the latter method, smallest tree, the tree is pruned back one step further than with the minimum error. This method is

more intelligible, but at the cost of a small increase in the error.

The constraints and the pruning can also be used at the same time. The developer can decide whether to use the first, the latter, a combination or none. However, one can also decide to use multiple models, tree in this case.

Multiple trees: training more than one tree to 'merge' them in the end also prevents overfitting. Since multiple models, trees in this case, are used this is also known as ensemble learning. One can train multiple trees in parallel or sequential. A more detailed explanation of ensemble learning methods can be found in Section 3.6.

3.4 Support vector regression

Support vector machines (SVMs) are an often-used machine learning technique. SVMs aim to establish a solution based on a small subset of all training points. Compared to other machine learning approaches this gives an enormous computational advantage. In this research we are particularly interested in support vector regression (SVR). SVR is a generalisation of SVM into regression problems. It differs from a simple regression model as it tries to fit the prediction error within a certain threshold rather than for the minimal error.

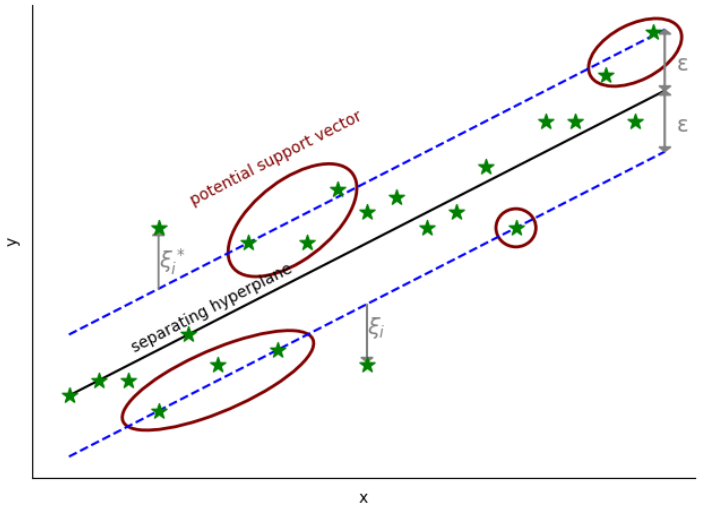


Figure 3.3: Example of a support vector regression model

SVMs were initially used to solve binary classification problems. The classification problem was for this purpose rewritten as convex optimisation problem. The aim of the initial SVMs was to find a line, called the separating hyperplane, that separates the two classes. This line should maximise the distance the distance to both classes. The closest points to the separating hyperplane are called the support vectors and they determine the margins on both sides. The margins on both sides must have an equal distance to the separating hyperplane. The area between these margins is known as the hyperplane. In case of SVR a ε-tube is introduced, as can be seen in Figure 3.3. This ε-tube is essentially the same as the hyperplane at SVM. Instead of keeping all data points outside the hyperplane, the aim is now to let most data point fall into the hyperplane. This tube is used to find the best approximation of the function. In addition, it tries to balance the model complexity and the prediction error. The value of ε, the margin, determines the width of the tube. The smaller the value of ε, the smaller the tube, the more training

points will fall outside and the wider the tube the more data point will be within the boundaries of the tube. The aim is to find the narrowest tube that contains most of the training instances. With SVR all training points outside of the tube are called the support vectors. These support vectors are the most influential instances that affect the shape of the tube. [28]. A simple multi-regression can be written as:

$$f(x) = w^T x + b$$

In case of SVR the aim is to find the narrowest tube, with the lowest prediction error. The lowest prediction error means the shortest distance between the predicted and the desired outcome. This aim is achieved by solving the following optimisation:

$$min_w \frac{1}{2}||w||^2$$

Where $||w||$ is the magnitude of the normal vector to the surface that is being approximated. To calculate the prediction error a loss function can be used. There are multiple loss functions that one can use in case of SVM. These same functions can be used for regression purposes, however the loss functions only penalise whenever a the threshold, ϵ, is exceeded. A selection of these loss function can be found in Section 3.4.1. However, whenever the function is not convex or contains outliers, using only a loss function, will not be sufficient. In this case one can instead regulate the model by using slack variables. Slack variables, ξ_i and ξ_i^*, enable us to make the margins of the tube softer. These slack variables determine how many points can be tolerated outside the tube. Furthermore, a constant, C, is established to determine how much affect these outliers should have on the results. The introduction of slack variables and the constant C results in the following (primal) optimisation function:

$$minR_{w,b} = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i + \xi_i^*$$

$$\text{Subject to } \begin{cases} y_i - w^T x_i \leq \epsilon + \xi_i^*, & i = 1, ..., N, \\ w^T x_i - y_i \leq \epsilon + \xi_i, & i = 1, ..., N, \\ \xi_i, \xi_i^* \geq 0, & i = 1, ..., N \end{cases}$$

C is one of the hyperparameters. C's value is a trade-off between the training error and the flatness of the solution. The larger C is the lower the final training error, however one risks losing generalisation properties if C is too large. The aim is to find a value for C where the training error is small, but the models its generalisation ability is also well. However, there is no best practice when it comes to determining C. In most cases C is therefore determined by a combination of grid search and cross validation. However, researchers have experimented a lot with the use of several other hyperparameter tuning methods. Which methods are used in our field of interest can be found in the related work on SVR in Section 4.3.

So far, we assumed that a linear function would be able to determine the tube. However, whenever a non-linear function is needed the data needs to be mapped into a higher dimensional space to achieve a high accuracy. A situation where a linear function is sufficient, as shown in the formula above, can be optimised with a primal formulation. For non-linear situations, a dual formulation is in most cases preferable. In order to deal with the non-linearity, the data has to be mapped into a higher dimensional space. This resolves in a quadratic function, which is expensive and time consuming to calculate. However, this mapping is not necessary one can also use a kernel function instead. This kernel function defines inner products in the transformed space. This results in the following function for a dual formulation:

$$max_{\alpha,\alpha^*} -\varepsilon \sum_{i=1}^{N_{SV}} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{N_{SV}} (\alpha_i - \alpha_i^*) y_i - \frac{1}{2} \sum_{j=1}^{N_{SV}} \sum_{i=1}^{N_{SV}} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(x_i, x_j)$$

$$\text{Subject to } \begin{cases} 0 \leq \alpha_i, \alpha_i^* \leq C & i = 1, ..., N_{SV}, \\ \sum_{i=1}^{N_{SV}} (\alpha_i^* - \alpha_i) = 0 \end{cases}$$

Where α_i and α_i^* are the Lagrange multipliers corresponding to ξ_i and ξ_i^*, and $k(x_i, x_j)$ the kernel function, and N_{SV} is the number of support vectors. There are multiple kernel functions that can be used, a selection of the most used once can be found in Section 3.4.2. After training the Lagrange multipliers become zero. Only the Lagrange multipliers that have nonzero values are called support vectors. Once the Lagrange multipliers are established one can also determine the weights of the regression and hence the optimal bias.

$$w = \sum_{i=1}^{N_{SV}} (\alpha_i, \alpha_i^*)\varphi(x_i) \sum_{i=1}^{N_{SV}} (\alpha_i \alpha_i^*) k(x_i, x_i)$$

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} (y_i - \varphi(x_i)^T w)$$

Where $\varphi(x_i)$ is the transformation from feature to kernel space ($k(x_i, x) = \varphi(x_i)\varphi(x)$). With this knowledge we can rewrite the simple regression function to a more explicit form. The more explicit expression is written as

$$y = w^T x + b = \sum_{i=1}^{N_{SV}} (\alpha_i^* - \alpha_i) k(x_i, x) + b, \;\; 0 \leq \alpha_i, \alpha_i^* \leq C$$

3.4.1 Loss functions

The loss function, a distance measure, determines the cost of the error between the predicted and actual outcome. In principle every convex, asymmetrical function can be used as loss function. The most used loss functions are shown in Table 3.1.

Which loss function to use depends on multiple factors. Three of these factors are the distribution of the noise that affects the data samples, the sparsity of the model one is looking for, and how complex the training computation may be [28]. The linear loss function is less sensitive for outliers than the quadratic function is. The linear loss function is the preferable option whenever the underlying data is distributed linearly, however it over emphasis small errors. Regularly the quadratic loss function is the standard method used in machine learning. However, its downside is that it has the tendency to overemphasise outliers. Huber is a combination of the linear and quadratic loss function. It is the smoothest loss function from the three. Huber penalises small errors quadratically and the large ones only linearly. The Huber loss function is a suitable loss function whenever the distribution of the data is unknown [29].

3.4.2 Kernel functions

When nonlinear data is used as input into an SVR the data has to be mapped into a higher dimensional space. However, this process increases the complexity of the model and therefore the computation time. Instead one can also use a kernel function. The kernel can be seen as a kind of shortcut which enables us to find the hyperplane faster than we would be able to if we had to calculate it into a higher dimensional space. In this section the most used, popular kernels are described and their pros and con [28].

Linear and polynomial kernels these two kernels are mainly used for modelling linear data. The advantage of the polynomial kernel is that it allows us to model feature conjunctions up to the order of the polynomial. Polynomial kernels are well suited for linear problems where the training data is normalised. Furthermore, the usage of either one of these kernels is quite fast.

Radial basis functions (RBF) are the most used kernels for nonlinear data. They allow us to find circled hyperplanes, also known as hyperspheres. RBF is actually a group of different kernels. RBF kernels are computationally heavier than linear and polynomial kernels, but they can achieve better predictive performances on nonlinear data then these other kernels. The downside, however, is that it is

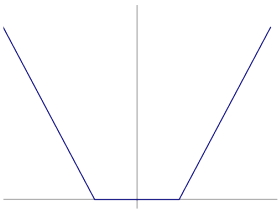
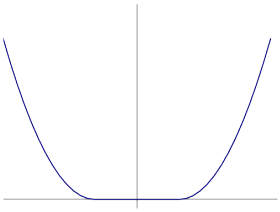
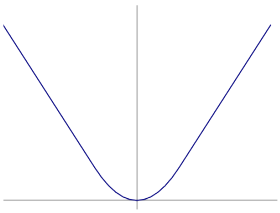
Loss Function	**Equation**	**Plot**
Linear	$L_\varepsilon(y, f(x,w)) = \begin{cases} 0 & \lvert y - f(x,w)\rvert \leq \varepsilon \\ \lvert y - f(x,w)\rvert - \varepsilon, & otherwise \end{cases}$	
Quadratic	$L_\varepsilon(y, f(x,w)) = \begin{cases} 0 & \lvert y - f(x,w)\rvert \leq \varepsilon \\ (\lvert y - f(x,w)\rvert - \varepsilon)^2, & otherwise \end{cases}$	
Huber	$L_\varepsilon(y, f(x,w)) = \begin{cases} c\lvert y - f(x,w)\rvert - \frac{c^2}{2}, & \lvert y - f(x,w)\rvert > \varepsilon \\ \frac{1}{2}\lvert y - f(x,w)\rvert^2, & \lvert y - f(x,w) \leq c \end{cases}$	

Table 3.1: SVR Loss functions

computationally heavy. As said there are multiple kernels that fall into the RBF group this is because the sigma in the RBF kernel can be chosen by the user. Examples of often-used RBF kernels are the Gaussian, exponential and ANOVA RBF kernel. Which RBF kernel to use depends on how your data is distributed. The ***Gaussian RBF kernel*** received a significant amount of attention and is the kernel most often used for SVMs. If we would compare it to a polynomial kernel than we could say that a Gaussian RBF kernel characterises a local mapping, whereas a polynomial kernel characterises a global mapping. An ***exponential RBF kernel*** can be quite effective when discontinuities in the data are acceptable. ***ANOVA RBF kernel***, on the other hand, tends to perform quite well in multidimensional regression problems.

Sigmoid kernel is also known as hyperbolic tangent kernel or MLP kernel. This kernel is inspired on the neural networks which sparked this kernels popularity. In fact, this kernel is equivalent to a two-layer perceptron neural network. In certain ranges it behaves like an RBF kernel. Although previous research shows that sigmoid kernels perform well, their behaviour is not fully known. For this reason, it is mostly advised to use the RBF kernels – at least next to it.

All these different kernels also come with hyperparameters. Depending on which kernel is chosen it ranges from one to multiple hyperparameters. These hyperparameters are, just like the constant C, determined by one of the hyperparameter optimisation techniques.

3.5 Neural networks

The full term is artificial neural network, but we often refer to them as neural network (NN). Neural networks are not algorithms but are rather a framework in which many different algorithms can be applied to retrieve the desired results. Originally NNs are inspired by the human brain and the aim is therefore to solve problems in a way the human brain would. That neural networks are powerful can be derived from the fact that neural networks have replaced both SVMs and decision trees in a wide range of applications [20].

The name neural network was introduced by Rosenblatt in 1958. He founded a perceptron. A perceptron is a binary classifier. Every input variable, $X = \{x_1, x_2, \ldots, x_n\}$, is given a weight, $w = \{w_1, w_2, \ldots, w_n\}$. The weight tells us something about the importance of the variable. In order to calculate the output, every variable is multiplied with its own weight and all outcomes of this process are summed. In other words, a perceptron is able to make simple decisions based on weighing input evidence. Then it is measured whether the exceeds the threshold or not, this enables us to identify a class for this data point. The threshold can also be seen as a bias. Furthermore, we replace the summation for a dot product, resulting in the following mathematical representation:

$$\text{output} = \begin{cases} 0 & if w \cdot x + b \leq 0 \\ 1 & if w \cdot x + b > 0 \end{cases}$$

where x is an input variable, w its weight and b the bias. The function measures whether the retrieved value is above the threshold. However, the perceptron as founded by Rosenblatt is only able to cope with linear input. A neuron, on the other hand, is able to cope with both linear and nonlinear input. The neuron differs from a perceptron in the fact that it applies an activation function over the output. A visual representation of a neuron can be found in Figure 3.4. Section 3.5.1 will tell more about the available activation functions. Furthermore, the weights have to be initialised to calculate the neurons output. How to determine the initial weights is explained in Section 3.5.2.

Oftentimes neurons are grouped into layers and neural networks consists of multiple layers. An example can be found in Figure 3.5. The output from the neurons in one layer can on their turn be seen as the inputs for the neurons in the next layer. The first layer in a neural network is the input layer and the last

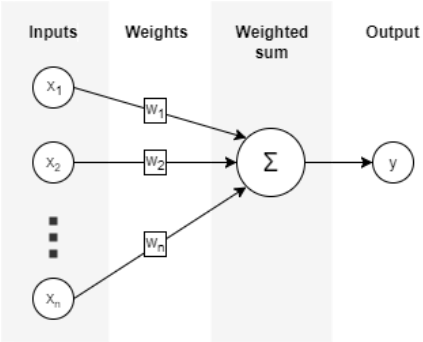


Figure 3.4: Neuron

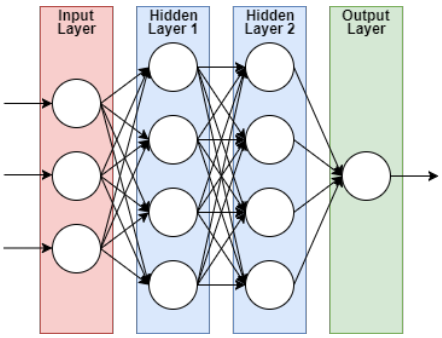


Figure 3.5: Neural network

layer is referred to as the output layer and all the other layers, the layer in between, are called hidden layers. The example, in Figure 3.5, shows a neural network where all neurons in the previous layer are inputs to all neurons in the next layer, however this does not have to be the case in neural networks. Furthermore, it is important to note that a neural network can have multiple outputs in the output layer.

Whenever the output is calculated it is time for a kind of feedback loop. In order to determine the quality of the network the loss is calculated. More about this can be found in Section 3.5.3. Furthermore, an optimiser is used to make sure the learning proceeds. The pace of learning is determined by the learning. More about the optimiser and the learning rate can be found respectively Section 3.5.4 and 3.5.5. In order to prevent overfitting, it is important that regularisation is applied. A selection of available methods can be found in Section 3.5.6. Lastly, an introduction into a specific neural network known as long-short term memory (LSTM) is given in section 3.5.7.

3.5.1 Activation function

In theory the activation function can be any continuous function that is differentiable. However, in practice in practice activation also perform better when they are bounded and monotonically increasing [30]. In this research we focus on the activation functions most often used and from which the usability is already proven. The formulas of these functions, their shape and boundaries can be found in Table 3.2. The linear function is also known as the identity function. As the name already suggests it is only used in case of linearity. In other words, this activation function will not contribute to the behaviour of the algorithm. All the other functions in Table 3.2 are nonlinear activation functions. The sigmoid activation function is often used for binary classification. The advantage of using the sigmoid function, over a binary activation function, is that due to the probabilities it also gives insight into the certainty of the outcome. The tangent hyperbolic function is usually used in the hidden layers of a neural network. It is actually a scaled version of the sigmoid function. It helps to centre the data, with a mean close to 0, making it easier for the next layer to learn. The computation cost of the last activation function in our table, the rectified linear unit function, is less expensive than the costs of the sigmoid and the tangent hyperbolic function. This is because the mathematical operations are simpler. The fact that the output of some neurons is zero and only part of them are activated, makes this function faster and easier to calculate.

3.5.2 Weight initialisation

Every input has a weight that determines that determines the importance of the input. The higher the weight the more impact this input variable has on the output of the neuron. But how does one determine with which weights to start at initialisation? In this section we go over methods regularly used nowadays.

Zero initialisation as the term already suggests, all the weights are initialised with a value of 0.

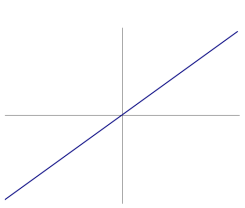
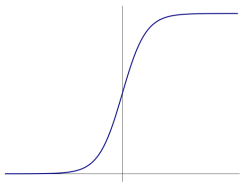
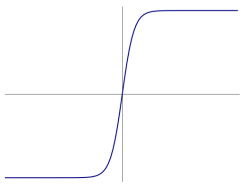
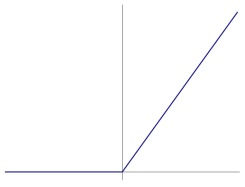
Activation Function	**Equation**	**Range**	**Plot**
Identity/-Linear	$\varphi(z) = z$	$(-\infty, \infty)$	
Sigmoid	$\varphi(z) = \frac{1}{1+e^{-z}}$	$(0, 1)$	
Tangent Hyperbolic (tanh)	$\varphi(z) = 2 \cdot sigmoid(2z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$(-1, 1)$	
Recified Linear Unit (ReLU)	$\varphi(z) = max(0, z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$	$(0, \infty)$	

Table 3.2: NN Activation functions

The downside of this method that since all the weights are equal, all the neurons will also have the same output. Then during the backpropagation, they will also all compute the same gradient, resulting in the same parameters. In other words, the zero initialisation causes a symmetry problem and therefore the recommendation is to not use it in any case.

Random initialisation as this term already suggests, the weights are initialised at random. However, it is important that the values initialised are not all the same as it would result in a similar issue as we observed at zero initialisation. Furthermore, the values should either not be too small or too large. In case the initialised weights are too small, the more epochs that are run, the more the variance is decreasing. As a result, the gradient saturates to 0.5 in case of sigmoid and to 0 in case of tanh. It does not affect ReLU as much since the gradient for negative value is 0 and 1 for positive inputs. In literature the saturating gradient is known as the vanishing gradient. When the weights initialised are too large, the absolute sum of the large weights multiplied by small the inputs will be very large as a result. This results in a large change in cost with large gradients as a result. This problem mainly affects the ReLU activation function. In literature this is called an exploding gradient. Both these phenomena, vanishing gradient and exploding gradient, are unwanted and therefore both these methods cannot be used in this shape. Important is to note that the gradient with respect to bias only depends on the linear activation of that layer and not on gradients of deeper layers. In other words, there is no diminishing or exploding gradients for the bias terms. They can be safely initialised to 0.

Xavier initialisation requires the gradient, Z-values and activations to be similar along all the layers. In order to use this technique, it is important to scale the data. Then, a formula for the variance is drafted where all three before-mentioned features remain equal in all layers. This gives the following formulas for the variance and the standard deviation:

$$var(w) = \frac{1}{fan_{in}} \text{ gives } \sigma(w) = \sqrt{\frac{1}{fan_{in}}}$$

In practice, the standard deviation, as written out in the formulas above, is multiplied by a random value from a normal distribution. A uniform distribution can also be used for this purpose, but since this comes with limits, we restrict ourselves to a normal distribution in this research. This way of initialising weights is known as Xavier or Glorot initialisation. This method is mostly used in combination with the tanh and the sigmoid activation function. Since they have a mean of 0 and 0.5. [31]

He initialisation differs from the Xavier initialisation by the fact that the variance needs to be multiplied by 2. This initialisation function is mainly used in combination with the ReLU activation function. The variance is multiplied by two, because halve of the Z-values - all negative ones -, turn zero in case of ReLU. In order to compensate for removing halve of the variance, the variance of the weights is doubled. This gives the following formulas:

$$var(w) = \frac{2}{fan_{in}} \text{ gives } \sigma(w) = \sqrt{\frac{2}{fan_{in}}}$$

In other words, the weights initialisation depends on the activation chosen. A random initialisation is seldomly used. Whenever a tanh or sigmoid activation function is used, normally a Xavier initialisation is used, while a ReLU activation function is combined with He initialisation [32].

3.5.3 Loss function

The loss function measures the quality of a network its output. Depending on the problem a different loss function can be chosen. For a two-class classification problem a binary cross entropy is used, while for a many-class classification problem a categorical cross entropy is used. In this research we want to predict a numerical value as output, which means that we have a regression problem. This is solved by

using the mean-squared error (MSE) as a loss function [20]. The equation for the MSE is:

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

where n is the number of data points, $\hat{y}_i$ are the observed values and y_i are the predicted values. If a neural network has multiple outputs, multiple loss functions can be used. The rule is one per output. However, gradient descent is only able to cope with one single scalar loss value. Therefore, in case of multiple they should be combined, averaged, too form one single scalar quality [20].

3.5.4 Optimiser

The optimiser determines how learning proceeds. It uses the loss function to update the network's weights. The aim is to find a combination of weight values that yields the smallest possible loss function. The can be done by solving the gradient. In other words, computing the gradient of the loss with respect to the network coefficients. Finally, one can move in the opposite direction from the gradient, thus thereby decreasing the loss. Written out the formula has the following form:

$$\theta = \theta - \eta \cdot \nabla J(\theta)$$

Where θ is the weight, η the learning rate and $\nabla J(\theta)$ the gradient of the loss function. Gradient descent is the most used technique for training and optimising neural networks. In case of batch gradient descent (BGD) the gradient is calculated only once and for the entire dataset. This makes this method slow and difficult for datasets that do not fit in memory. For this reason, stochastic gradient descent (SGD) is often used. With this method the parameters are updated for each training example. This method increases the speed tremendously, however the downside is that the frequent parameter updates have high variance. Due to the frequent updates and fluctuations it ultimately complicates the convergence to the exact minimum and will keep overshooting due to the frequent fluctuations. Furthermore, a lot of the functions are non-convex which means that they have more than one minimum and thus contain a number of suboptimal minima. The difficulty is that whenever SGD gets trapped into such a local minimum, it cannot leave anymore. To overcome all of these challenges SGD can be extended.

There are roughly two approaches. The first approach focuses on the acceleration of SGD. One of the methods belonging to this approach is momentum. Momentum is used alongside SGD. It accelerates SGD by steering into the relevant direction while softening the oscillations in irrelevant directions. It does so by adding a fraction γ of the update vector of the past step, $V(t-1)$, to the current update vector. This fraction γ is set to 0.9 by default. Lastly, we subtract the current update vector from the current weight to obtain the new weight. Written down mathematical this gives us the following equations:

$$V(t) = \gamma \cdot V(t-1) + \eta \cdot \nabla J(\theta)$$
$$\theta = \theta - V(t)$$

Where η is the learning rate and $\nabla J(\theta)$ the loss function of the current weights. The learning rate is set to 0.1 or 0.01 by default. The size of the learning rate can have a huge impact on the results. By momentum the learning rate has to be adjusted by hand. In the other approach learning rates play a major role. They are adapted to find the optimal solution. One of these methods is Root Mean Square Propagation (RMSProp). RMSProp does not require its users to adapt the learning rate themselves. Furthermore, it has a learning rate per parameter and updates are applied on individual parameters. By using different learning rates for different parameters, the focus is more on the importance of the individual parameters. The mathematical representation for this technique is:

$$V(\theta, t) = \gamma \cdot V(\theta, t-1) + (1-\gamma) \cdot (\nabla J(\theta))^2$$
$$\theta = \theta - \frac{\eta}{\sqrt{V(\theta, t)}} \cdot \nabla J(\theta)$$

Our final optimiser, the adaptive momentum estimation (Adam), is a combination between SGD with momentum and RMSProp [33]. Adam uses estimations of first and second momentums of the gradient to adapt the learning rate for each weight. The first momentum is the mean and the second momentum is the uncentred variance. Next, both these momentums are used to calculate the new weights. The mathematical representation for this technique is:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \text{ and } \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

Where ε is a small number to prevent division by zero and the β's are the forgetting factors for the gradients. By default, ε is set to 10^{-8}, β_1 to 0.9, and β_2 to 0.999. In practice Adam works well thanks to the fast conversion, the efficient and fast learning speed. Adam copes with the vanishing learning rate, slow convergence and the high variance in parameters updates. Although newer techniques are available, their usability has not been proven scientifically. For instance, AMSGrad which sometimes seems to outperform Adam, but sometimes performs less. Therefore, we will use Adam in this research since it has been applied widely and its usability is proven. [34]

3.5.5 Learning rate

The learning rate is a hyper-parameter that determines the speed in which the model learns. It does so by controlling how much the weights are adjusted. This is done with respect to the loss gradient. A large learning rate allows the model to learn faster but has a higher probability of ending up with a sub-optimal set of weights. While a smaller learning rate may allow for a more optimal result but will take significantly longer to train. The learning rate is often set to 0.01 for multi-layer neural networks. However, every value in the range of 1.0 to 10^{-6} is acceptable. But researchers do not recommend one to use a default value. Rather they emphasise that the learning rate hyper parameter is the most important parameter to tune. [35]

The difficulty is that there is no easy way of determining the best learning rate. In most cases trial and error is the best option. Below three methods to determine the learning rate are discussed and why they are probably better than a fixed learning rate.

Learning rate annealing starts with a high learning rate and lowers it bit by bit as the training progresses. The idea behind it is that one does not need to know all of the details from the beginning, hence the larger learning rate, while the details become more interesting in a later stadium, hence the smaller learning rate then. One of the most popular forms of learning rate annealing is a step decay. In this case the learning rate is reduced by some percentage after a set number of epochs.

Learning rate (LR) range test is the formalised form of the trail-and-error technique of finding the proper learning rate. The approach was formalised by Smith in 2015 [36]. The trial-and-error technique consists of running your model for a few iterations, while gradually increasing your learning rate with each iteration. At the meantime, you record the loss for each value of the learning rate. If you plot this against one other, the plot should consist of three areas. The first area is where the learning rate is too small and thus where the loss is barely decreasing. The second area is the area where the learning rate is just right, the loss converges quickly. This is the area that we are interested in. Lastly, the third area, is where the learning rate is too big and hence the loss starts to diverge. Although our best learning rate value lays within the second area, the other areas are also important. If not all three regions are displayed properly, for instance because there is a gap in the plot, this is a good indicator that there is either a bug in the model or an error in the data.

Cyclical learning rates is another technique to determine the learning rate also introduced by Smith [36].

The cyclical learning rate starts with a low initial learning rate and gradually increases it – either linearly or exponentially – at each iteration. As the learning rate increases, there will be a point where the loss stops decreasing and starts increasing again. Ideally a learning rate value just before the increase is selected, this is the lowest point on the learning rate against loss graph. This method might feel weird, because it approaches the problem the other way around of what we are just to. The rationale behind it is that increasing the learning rate might have a short-term negative effect but can yet achieve a longer-term beneficial effect. The variation between reasonable bounds would actually increase the accuracy of the model in fewer step. Appurtenant benefit is the ability to escape a saddle point in the loss landscape. A saddle point has a small gradient, so a small gradient rate makes model traverse these saddle point very slowly in later stages of training. By increasing the learning rate in a later stage, it will help to escape these saddle points more efficiently.

3.5.6 Regularisation

The challenge is not to make a model more complex than necessary. A too complex model contains often too many details of the data it was trained on. When a model performs well on the training set, but does not perform well on the test set, then the algorithm is most likely not able to generalise. This problem is also known as overfitting. Overfitting can be prevented by acquiring more data, however this is often an expensive and time-consuming task. Another method often used to prevent overfitting is regularisation. A regularisation technique has the aim to reduce the error on the test set which usually consists of reducing the variance while trying to keep the bias stable. Regularisation must always be unless the training set is very, very large meaning. In practice, that it exceeds tens of millions of rows [37]. There are multiple regularisation techniques from which we will highlight a few in this section.

Weight decay, also known as weight regularisation, is added to the loss function to penalise large weights. Large weights are undesired since they tend to cause sharp transitions which causes major changes in the output when the inputs are only slightly changed [38]. Due to the penalty for higher weights, the algorithm will choose small weights over the larger ones. This method is already used for decades and has been proven effective for simple linear models but also for more the more complex ones like neural networks. Examples of weight decay algorithms are L1- and L2-regularisation, respectively known as LASSO and RR. The difference between these methods is in the weight decay factor they add to the loss function. Whereas L1-regularisation adds the sum of absolute values of the weights as the weight decay factor, L2-regularisation adds the sum of the squared values of the weights as the weight decay factor. As already mentioned in Section 4.1, in practice this means that L2-regularisation penalises the smaller weights less than the larger weights resulting in no incentive for the smaller weights to reduce to zero. While L1-regularisation penalises all weights equally resulting in weights of zero, these variables can then be deleted. By bringing down the weights or even deleting certain variables from the equation, certain features can be deleted from the input. Elastic Net (EN) combines both L1- and L2-regularisation. EN has the ability to shrink some of the coefficient while also setting some of them to zero for a sparse solution. [39]

Dropout regularisation uses a random subset of the network at each iteration to train with. In the simplest version this random subset is established by setting a threshold and giving each neuron in the network a random value. Whenever the value exceeds the threshold, the node is kept and otherwise it will not be taken into consideration. Important is that the input and output layers remain untouched. By dropping certain neurons, the remaining neurons are forced to learn on their own without relying on the other neurons. The advantage of this technique is that it is computationally cheaper than weight decay and it has proven effective in the past [37]. Many different versions of this technique are available all focused on a specific kind of neural network. For instance, Semeniuta et al. and Kruegar et al. both made a version of dropout, respectively recurrent dropout and zoneout, that prevent the loss of long-term memory in RNNs and LSTMs [40, 41]. For optimal results dropout may be combined with other techniques.

Early stopping, as the name already suggests we stop the training of the model early in order to prevent the model from getting too complex. During the training process the error is monitored closely, whenever the error starts to increase the parameters and frozen and the training process is stopped. The idea is to start with small random weights and with every iteration the model gets more complex. In other words, the earlier the training process is stopped the less complex the model is. Early stopping has a similar effect as weight decay, however less training experiments and is therefore less time expensive. For this reason, Goodfellow et al. recommend to always consider the use of early stopping [37].

Noise is the addition of random input to current data in order to improve the generalisation ability of the model and its fault tolerance. Noise can only be added during training but can be added at multiple places. The most common place is to add it to the input data. Other options would be the addition of noise to gradients or activation functions. However, one has to be cautious about the usage of noise, Greff et al. added Gaussian noise to the input of their LSTM and concluded that it does not only hurt performance, but does also increase the computation time [42].

Regularisation techniques are often not used on their own, but multiple techniques are combined. For instance, early stopping combined with weight decay or early stopping combined with dropout regularisation. However, there is no golden rule on how to combine regularisation techniques it comes down to experimenting and prior knowledge.

3.5.7 LSTM

Long short-term memory neural networks are a specific kind of recurrent neural networks (RNNs). Unlike normal neural networks RNNs are able to cope with time series input thanks to the presence of memory. A memory is useful in case of time series as the previous steps can be used to predict the next. However, the downside of RNNs is that vanishing gradients often occur. Simply said this means that the weights of the neural networks do not update because the changes are extremely small. For this reason normal RNNs are not regularly used anymore. In this research we have therefore decided to favour LSTMs. As they do not suffer from this problem.

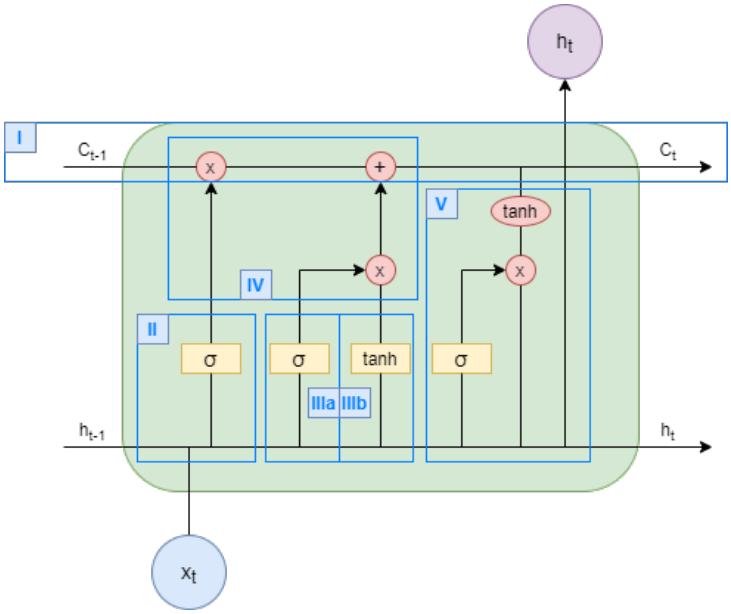


Figure 3.6: Example of an LSTM cell

When we speak of a LSTM neural network it is a neural network with LSTM cells incorporated. In practice oftentimes we see a combination of normal neurons and LSTMs cells. An example of a LSTM cell can be found in figure 3.6. The information flow through this cell, known as the conveyer belt, is shown in area I. Furthermore, a LSTM cell consists of an input, forget and output gate. The forget gate is shown in area II where the sigmoid function is used to decide what information to keep and what is thrown away. Then the next area, III, actually consists of two parts. This is where the storage of information is handled. In area a we see the input gate layer with again a sigmoid function to decide which information to keep. Then, in area b, a vector is created of the new candidate values. Next, the information obtained in area II and III flows back to area I where the information is updated with the aid of pointwise operations. This step is also marked by area IV. Then, lastly, the output is generated in area V. For this step the sigmoid function is used again to decide what information to keep and tanh is used to scale the output.

3.6 Ensemble learning

During ensemble learning multiple algorithms are used to retrieve better results than one single algorithm would be able to retrieve. These algorithms can be multiple times the same algorithm or a combination of different algorithms. There are three well-known ways of ensemble learning which will be discussed in this section. A visual representation of them can be found in Figure 3.7.

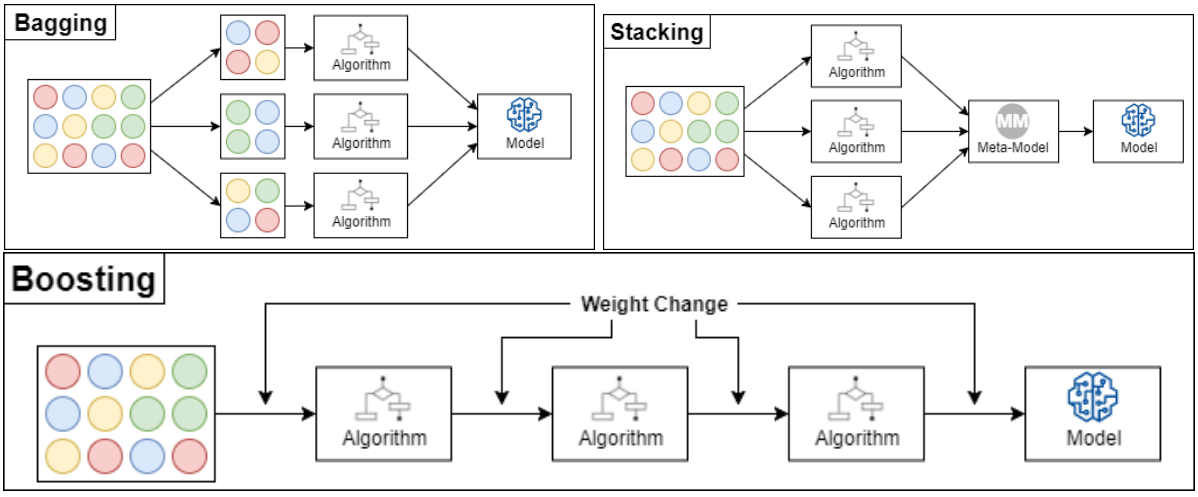


Figure 3.7: Ensemble learning techniques

Bagging, which is short for bootstrap aggregating, is a technique were algorithms are trained in parallel. To train every algorithm a dataset is made with the aid of bootstrap sampling. The next step is to aggregate the results, which is done by averaging in case of regression. An example of bagging is random forest (RF) which is an ensemble of decision trees. The advantage of RF over a single tree is that it significantly reduces the chances of overfitting.

Stacking also trains the algorithms in parallel together with a learning algorithm for a meta-model. All algorithms are trained using available data. When this process is done, the meta-model can be used at the end to combine the predictions of all trained algorithms.

Boosting trains all algorithms sequentially. The idea behind it is that when an input is misclassified by a previous algorithm, the weight is increased so the likelihood that the next classifies it correctly increases. In other words, every algorithm learns from the one before. The fact that the algorithm

learns from the one before means that chances of overfitting reduce. For this reason, neither pre- nor post-pruning is necessary. An often-used boosting technique is gradient boosting (GB). GB is essentially a combination between gradient descent and boosting. Another often used method is gradient boosted regression trees (GBRT) which is a regression tree that evolves until it reaches its optimum.

As can also be easily concluded from this section, ensemble learning is mostly used to increase the predictive performances of decision trees. However, stacking, for instance, can also be used to combine the results of multiple different algorithms. In this way, the strengths from every model can be included in order to retrieve the best possible results.

4 Related Work

This chapter contains an overview of the related work found for the various machine learning technique as introduced in the previous machine learning literature chapter. It first discusses the results for linear regression, tree-based methods, support vector regression, neural networks and ensemble techniques. Then, the last section, contains a table with a summary of the related work as found for this research.

4.1 Linear regression methods

A lot of related work have used linear regression to estimate emission. [43], [44], and [45] have used MLR to predict NO_x emission. However, in all three cases MLR was outperformed by a nonlinear method, either support vector machine or neural networks. However, normal MLR is fitted on the complete current data and can therefore be easily overfitted. For this reason, a lot of extensions of MLR exists. Most of these extensions are based on a form of regularisation in order to prevent overfitting. In this group we discuss a number of methods from two groups penalised methods and latent variable methods.

Penalised methods penalise models that have too many variables. Since these methods are also aimed at reducing the number of variables, these methods are also known as shrinkage methods. We can also refer to these methods as embedded methods. Embedded methods are feature selection methods that are used within a machine learning algorithm and are adapted based on the performance of the model. Two examples of penalised methods are L1- and L2-regularisation respectively known as Least Absolute Shrinkage and Selection Operator regression (LASSO) regression and Ridge regression (RR). LASSO penalises all weights, while RR penalises the smaller weights less than the larger weights. As a result, LASSO regression is able to set weights to zero after which an input parameter can be omitted, while RR keeps all features included. Elastic Net (EN) regression is a combination between the both of them which gives it the ability to shrink some of the coefficients while also setting some of them to zero for a sparse solution. The advantage of LASSO and RR is that they are both able to deal with multicollinearity. We see the usage of RR, Lasso and EN regression also back in related work. For instance, Cuccu et al. use the RR and Lasso regression to estimate the NOx emission of their engines of a gas turbine and Lepore et al. used it to estimate CO2 emission from a cruise ship [46, 47]. Especially in this last research the results of LASSO regression were promising. The shipping routes in this research were grouped based on their distance and for the short and medium length routes LASSO regression turned out to be the best estimator.

Latent variable methods are not directly observed but are determined based on other observed variables. An example of a method in this group is principal component regression (PCR). During PCR the dimensionality of the data is reduced, resulting in less variables with still most of the initial information. In PCR the principal components of the explanatory variables are used as regressors. Another latent variable method is partial least squares (PLS) regression. PLS regression performs least squares regression on principal components rather than on the original data. PLS usually outperforms PCR, since it makes use of the knowledge of PCR and extends it. Cuccu et al. and Lepore et al. both applied PCR and PLS during their research [46, 47]. Cuccu et al. research found that PLS indeed outperforms PCR (in their case), however better results were still obtained with RR and LASSO regression. Also, Liu et al. applied PLS to their NOx emission estimation, but found that PLS was not the most suitable method given the large number of outliers in their data [48].

4.2 Decision tree methods

Decision trees are seldom used on their own. This statement is confirmed when we search for related work with regard to emission monitoring with the aid of a decision tree. We could not find any work where they used only one decision tree model to calculate the emission. However, we were able to find work that exploits multiple decision trees. However, ensemble learning is not explained till Section 3.6 and

therefore we decided to discuss these papers afterwards in the related work section of ensemble learning in Section 4.5.

4.3 Support vector regression methods

Support vector regression is one of the most commonly applied methods for estimating the emission [43, 49, 50, 51, 48, 46, 47, 44, 52, 45, 53]. In most cases a normal SVR is used, however we also see the usage of least squares SVMs (LSSVMs) quite a lot. As the name already indicates LSSVMs use a least squares method to build the SVM. Where SVMs are based upon a quadratic programming problem, LSSVM are based on a linear problem. In LS-SVM there is no ε-tube or slack variables, rather those are replaced by error variables [54]. These error variables measure the distance to the regression function. Another difference between regular SVM and LSSVM is the fact that for LSSVM all variables are marked as support vectors, while in normal SVM this is only a selection of all data points. LSSVM has the advantage that it is reduces the time consumption. LSSVMs are therefore mostly used for large scale problems and although the same kernels can be chosen for LSSVMs as for SVMs usually the RBF kernel is chosen [55]. Most papers use either SVR or LSSVM, however Tang et al. compared both SVR and LSSVMs. In their paper the LSSVM outperforms the SVR [45].

With respect to the choice kernel we see that in almost all cases RBF kernels are used. For the LS-SVMs in our related work, we even see that an RBF kernel is always used [49, 52, 45]. Tang et al. focused in their research on the influence of kernel usage for the outcome. In their research they compared an SVR with RBF kernel with LSSVMs with an RBF kernel, ReLU kernel, sigmoid kernel and an adaptive kernel. This adaptive kernel (AK) is a combination of the ReLU, RBF and the sigmoid kernel. In the end, their adaptive kernel retrieved the best results of all. Besides using the kernel for SVR, we can also perform kernel ridge regression (KRR). KRR is a combination between RR and a kernel. RR is a form of penalised regression the RR can be seen as the loss function of the kernel. SVR and KRR therefore differ in the loss function they use. Fitting KRR is mostly faster for medium-sized datasets, however whenever the dataset is large SVR is recommended because it scales better [56]. Cuccu et al. tested KRR in their research, however there it was not able to outperform the normal SVR [46].

Another thing where we see a lot of differentiation between the papers is the method to tune hyperparameters. Traditionally we see grid search (GS) [50] and cross-validation (CV) [46, 47, 44] quite often, but the related work often uses population-based algorithms to tune hyperparameters. Zhou et al. use ant colony optimisation (ACO) to tune their hyperparameters [43]. ACO is inspired by the behaviour of real ant and in order to find the best hyperparameters the best path on a weighted graph has to be found. Another method used by Tan et al. and Azzam et al. is a genetic algorithm (GA) [51, 53]. GA solves the optimisation problem based on the principle of natural selection. In practice this means, random individuals from a population are selected to be parents and to produce children. In this way the problem "evolves" toward an optimal solution. Another method also used in related work [52, 45] is particle swarm optimisation (PSO). PSO is based on the social behaviour of animals that live in large colonies. The first generation is randomly generated, and generations afterwards are updated to find an optimum. However, since these researches are all applied to different data sets it is difficult to draw conclusions with regard to the best hyperparameter tuning technique. In [48] they did compare GS and GA which gave better results for GA and also when GS was compared to ACO, as they did in [43], the GS algorithm is outperformed. In the research of Li et al. the focus is on optimising the results of the LSSVM by the usage of different hyperparameter tuning techniques. In their paper, [49], they compared teaching-learning-based optimisation (TBLO) with ameliorated TBLO (A-TBLO), coupled simulated annealing (CSA), artificial bee colony (ABC), and gravitational search algorithm (GSA). They found A-TBLO as the best performing technique of all. A-TBLO is a quick version of TBLO which can speed up the convergence and can improve the quality. TBLO is inspired by the effect of the influence of a teacher on the output of learners in a class. It consists of two phases: (1) teacher phase, where the teacher teaches its learners all the knowledge he has himself, and (2) learner phase, where the learners

increase their knowledge by interactions among themselves. The difficulty is that the hyperparameter tuning techniques in this paper, [49], are not the same as used in the other related work papers, which still makes it hard to compare the different techniques.

4.4 Neural network methods

There are many different types of neural networks and many variations possible during training. One can vary the activation functions, how the weights are initialised, which learning rate is used and so on. However, all these options make it difficult to compare neural networks. What makes it even harder in this specific case is that every research also uses a different dataset. In order words, many variations are possible all with their own effect on the outcome, which makes neural networks really tricky to compare.

There is a lot of difference in how detailed the configuration of the neural networks is written down in the papers. There is even a number of papers where there is only mentioned that a certain type of NN is used without any further specifications [50, 48, 45]. In Tan et al. they used a 3-layer NN, where the number of neurons was determined by $log_2(T)$, where T is the amount of data samples [51]. Furthermore, they used an adaptive learning rate and gradient descent with momentum. However, this NN was not able to outperform the SVR model as used in this research. The same goes for a research conducted by Zhou et al [43]. They tested a generalised regression NN (GRNN) and a back propagation NN (BPNN), but both were not able to outperform the SVR. The GRNN in this research was based on the Euclidean distance between two input vectors, while the BPNN was a 3-layer MLP with 23 neurons in the hidden layer and it was trained with the Levenberg-Marquardt (LM) algorithm. This LM algorithm was also used by Korpela et al. for the training of their MLP with one hidden layer [44]. However, in their research the MLP was able to outperform the SVR. Also, Azzam et al. were able to configure a NN that outperformed SVR [53]. Their NN used Bayesian regularisation (BR) and the error of the network was determined by a combination of the training error and the generalisation error. Also, Cuccu et al. found that NNs perform well during their feasibility study [46]. For their research they looked into incremental BP, batch BP, improved resilient BP (iRPROP), and QuickProp. Especially the NN with iPROP seemed to be produce promising results for scaled data.

In research conducted by Kibrya et al. and Williams-Gossen et al. the predictive performance of multiple NNs was compared [57, 58]. In both their researches they compared MLP to an RBF NN, and a GRNN. They both found an MLP as best performing model, although they configured their MLP networks differently. Kibrya et al configured an MLP with two hidden layers with four neurons each with a sigmoid activation function for all, except the input, layer. Furthermore, the MLP was trained with BP, conjugate GD, and QuickProp. Williams-Gossen et al found an MLP consisting of only one hidden layer with two neurons and a hyperbolic tangent activation function also in this case with exception of the input layer.

Neural networks have proved capable of predicting emission quite well. This is proven in related work, but also by the fact that the current software used by Emission Care BV is based on NNs. However, all related work discussed so far did not have any memory. There are NN available that do have memory, these NNs are known as recurrent NN (RNN). RNNs can be thought of as multiple copies of the same network. In comparison with feed-forward NN they have an additional feedback loop. This additional loop makes them able to pass a message to the next copy. This message contains information about things learned in the previous versions and therefore it is often referred to as memory. This sequential information is preserved in the RNNs hidden state. Another advantage of RNNs is that they are able to take a series of input without a size limit. This makes them highly capable of dealing with time series for instance. A specific type of RNN is a long-term short memory (LSTM). The closest related work we could find on the use of RNNs was on the NOx emissions of engines, however the third page of this paper is blank in every version of the paper that we have found so far [59]. Another paper we found is more vaguely related and does not estimate the emission, but rather the fuel consumption of a vehicle using

smartphones and RNNs. This research was conducted by Kanarachos et al. and, to be more precise, in their work they compare the nonlinear autoregressive network with exogenous inputs (NARX-RNNS) with LSTMs [60]. The NARX-RNN outperformed the LSTM. The NARX-RNN was trained with a LM algorithm and the hidden layer had a log-sigmoid activation function and the output layer a linear one. Furthermore, the number of neurons in the hidden layer was varied even as the lag variables to find the number that yields the minimum error. This was found for five delay inputs and 47 neurons per hidden layer. Furthermore, the best results were achieved when the contrast-based fruit fly optimisation with group policy (c-FOA/g) was applied followed by BR-BP. This estimation error retrieved was beneath 6% of the actual fuel consumption value. The other research, conducted by Zhang et al, focused on the use of LSTMs and is more related than the other research since its aims is to forecast vehicle emission [61]. The input gate, forget gate, and the current cell state all use sigmoid functions, while the state of the current input can be calculated with the hyperbolic tangent function. Furthermore, the BP through time (BPTT) is implemented to optimise the network weights. The correlation coefficient, R, retrieved is 0.70 for the LSTM. However, when they apply a wavelet transformation in combination with the LSTM it gives an R of 0.97. The wavelet transform decomposes the original highly variable time series into several lowly variable sub-series. Each subseries is processed separate by the LSTM and later the results are summed.

4.5 Ensemble learning methods

Ensemble learning methods can entail all different sorts of machine learning models. In theory, stacking enables one to train a NN, a SVR, and MLR and combine it with the aid of a meta-model as created with stacking. However, in related work we mainly see the usage of ensemble learning for decision trees.

Lepore et al., for instance, use bagging of regression trees (BRT), RF, and boosting of regression trees (BT) in their research to predict CO_2 emission of a cruise ship [47]. BRT uses bagging and thus train multiple trees and takes the average prediction of all trees. RF does the same, however it uses bootstrap sampling to train on random datasets rather than on the whole dataset. BT is basted on boosting and trains the trees sequentially, where every tree learns from the previous one. For BT a fixed learning rate is used in this research of 0.02. In all cases the number of trees is determined by cross-validation. In the results of this paper we can see that the results for long distances can be best predicted with BT outperforming all other ensemble learning and linear models.

Another research that uses ensemble learning for estimating the emission is conducted by Pan et al [62]. They estimated real-driving emission for busses. In their initial dataset they did it for three different fuels and they predicted the emission of four different pollutants including NOx. They predicted it with the aid of VSP-GBRT. VSP stands for vehicle specific power and it calculates the engine power per unit mass. In the end, they compared their results against a VT-Micro model which is based on polynomial regression and found that in comparison their VSP-GBRT reduced the MAE by 27.3%, the MAPE by 33.4% and the RMSE by 22.1%. In other words, the VSP-GBRT clearly performs better for estimating the emission than the current often used VT-Micro model.

4.6 Related work summarised

This section does not include any new work, rather it presents already discussed work in a table. The aim is to make it a clear overview which makes it easily visible which methods are regularly used and which results are obtained. One should be aware of the fact that all results are obtained from different emitters and different datasets. In other words, one cannot simply compare the results from different papers one-on-one.

Ref	Pollutant emitter	Data specifics	Data scaling	Models tested	HT	FS	#IP	Validation technique	Results
[57]	Gas combuster	24 cases, 8 inputs	X	MLP, RBF NN, GRNN	X	-	8	X	MAE: 2.3%, Std: 1.97%
[43]	300 MW boiler	670 cases, 21 inputs, 67% train, 33% test	Scaled [-1,1]	ACO-SVR, GRNN, RBF NN, GRNN	ACO, GS	-	21	Hold-out, 5-fold CV	MRE: 1.597%, R: 0.934
[58]	Three gas turbine engines	2259 data points, 4 inputs, 83% train, 17% test	X	MLP, RBF NN, GRNN	X	Sensitivity analysis	4	X	R: 0.98
[49]	330 MW boiler	20 cases, 26 inputs	X	LSSVM	A-TBLO, CSA, TBLO, ABC, GSA	-	26	LOOCV	RMSE: 8.294
[50]	700 MW boiler	7200 patterns, 100 inputs, 85% train, 15% test	X	ELM, SVR, ANN	GS	X	20	X	MSE: 39.40, MRE: 1.13%, R: 0.998
[51]	1000 MW boiler	4371 measurements, 40 inputs	X	PCA-SVR, SVR, ANN	GA	PCA	5	5-fold CV	MSE: 131.6, MRE: 4.36, R:0.962
[48]	197 kW diesel engine	180 data points, 40 inputs, 33% train, 67% test	Normalisation: [0,1] and PCA	GA-SVM, SVM, NN, PLS	GS, GA	PCA	15	LGOCV, 7-fold CV	RMSE: 51.12, MAPE: 4.65%, R^2:0.98
[46]	Gas turbine	10 engines, 400k-670k data points, 50k clean, 165-171 inputs, 2.5% train, 97.5% test (10 unique test sets)	Normalisation: Z-score (iPROP scaled)	LR, RR, LASSO, PCR, PLS, KRR, SVR, Incremental BP, Batch BP, iRPROP, QuickProp	10-fold CV	Statistical analysis	34	10-fold CV	Deviance: ν-SVR: [-1.5,1.5], S-iRPROP: [-2,1.5]

Ref	Pollutant emitter	Data specifics	Data scaling	Models tested	HT	FS	#IP	Validation technique	Results
[47]	Ro-Pax Cruise Ship	1 year (CO_2), Short: 218, Medium: 85, Long: 235, 22 inputs, 80% train, 20% test	Autoscaling	RR, LASSO, EN, SVR, PCR, PCR_FS, PLS, BRT, RF, BT	10-fold CV	-	22	Double CV	S: LASSO RMSE: 0.58, M: LASSO RMSE: 0.86, L: BT ' RMSE: 0.91
[44]	Two 43 MW boilers	At least 7 inputs	Scaled [-1,1] (except MLR)	MLP, MLR, SVR, FIS	10-fold CV (with 10 MC loops)	X	2	10-fold CV	Boiler B: RMSE(abs): 5.08, RMSE(%): 3.58
[53]	Gas turbine	3 turbines, 395-472 instances, 4-6 inputs, 72% train, 18% test, 10% validation	Scaled [-1,1]	NN, SVM, One-layered NN	GA	-	4-6	X	R^2: 0.997
[52]	1000 MW boiler	1500 data points, 1117 clean, 33 inputs, 72% train, 28% test	X	LSSVM	PSO	Sensitivity analysis	7	LOOCV	7 inputs: MRE: 2.23%, R:0.9566
[45]	330 MW boiler	303 inputs	Standardised	LSSVM-AK, LSSVM-RBF, RBF NN, SVR, LSSVM-RELU, MLP, LSSVM-Sigmoid, MLR	PSO	Mechanism analysis, RF	12	X	RMSE: 2.0725

Ref	Pollutant emitter	Data specifics	Data scaling	Models tested	HT	FS	#IP	Validation technique	Results
[60]	Fuel consumption of vehicle	3693 samples, 70% train, 15% test, 15% validation	X	NARX-RNN, LSTM	c-FOA/g + BR-BP, c-FOA/g, BR-BP, etc	X	X	6000 evaluations	Estimation error $\approx$ 6%, R: 0.96
[13]	Vehicle	215,865 samples, 9 inputs	X	Wavelet-LSTM, LSTM	X	-	9	5-fold CV	R: 0.97
[62]	LNG bus	5243 NO_x samples, 75% train, 25% test	X	VSP-GBRT, VT-Micro Model (polynomial regression)	GS	GS	20-80	CV	$\underline{NO_x}$: $\overline{R^2}$: 0.80, MAE: 0.053, MAPE: 23.1%, RMSE: 0.073

Table 4.1: Overview of the most important related work

An overview of the most relevant related work for this research can be found in Table 4.1. The order of the studies in the table is based on their publication year. The oldest papers are at the top and the newest at the bottom. In this way it is easy to observe how method usage progresses over time.

Important is to note that the column of *data specifics* contains different values and properties for every paper. The image is drawn as complete as possible, but not in every case the same specifics, and level of it, were available. Except from choosing consciously to not write down specifics, some papers did not state it clearly. If an X is written out, it means that the authors of the paper did not include this specific information in their paper. In other words, it does not mean they did not do it, but at least it was not written down explicitly. Whenever we could find information implicitly this is included. For instance, the paper of Korpela et al. never explicitly state how many variables they have measured [44]. However, from the images and tables one can deduce that there are at least seven variables measured and therefore the table states "at least 7 inputs".

As one can see the amount of data points differs a lot among the studies. In a number of cases it was hard to establish whether the number of data samples was based on measured data or pre-processed data. Cleaning the data can sometimes reduce the data with as much as 90% which could be an explanation for the big differences. Furthermore, some datasets define their data in terms of samples and some in terms of cases. Because cases can exist of multiple data samples, this can also be a potential explanation. However, no real conclusions can be drawn and it just something to keep in mind will reviewing the results.

The column name #IP stands for number of input parameters, but this was abbreviated in order to keep the table width in proportion. The same goes for HT which stands for hyperparameter tuning. In order to keep the overview clear, things are simplified. The column with pollutant emitters, for instance, only specifies the emitter and when available its power, while there are many more specifics. However, these specifics would make the table more crowded while the specifics do not contribute for a better overview with regards to techniques used. For this reason, we decided for a shorter, more general version. Another thing simplified is the validation technique, models tested and their results. The models and validation techniques are ranked based on their performance. In other words, the model or validation technique mentioned first gives the best results, the second the second-best results, etcetera. Then, in the last column, the only the results of the best performing model and validation technique are given as these are most relevant. However, this ranking notation does not apply for the research of [46], [47], and [52], since the authors of these papers did not notate the performances on all models tested. In [46], this is due to the fact that is a feasibility study, while [47] has divided its results into three categories and [52] focuses on choosing the best input parameters rather than on comparing models. In these three cases it is clearly stated in the results column to which methods the results apply.

What stands out in Table 4.1 is the variety of feature selection (FS) algorithms used. Oftentimes, a sensitivity analysis or feature importance based on PCA is used as a selection criterion. However, we also see cases were random forests or grid search is used. In other words, it seems that there is no standard when it comes to determining a feature selection. The same seems to go for hyperparameter tuning where a variety of techniques is used. Besides grid search it seems that population-based optimisation techniques are a popular way to determine hyperparameters. However, comparing remains difficult because all studies are applied on different data. Therefore, there is no easy way to conclude which technique is best.

Another thing that strikes the eye is the fact that SVRs almost always outperform the other models tested. However, most of papers mentioned in the table have SVR as their main priority. Meaning that they have tried a lot of different techniques to optimise the accuracy of SVR, while the neural networks, for instance, are often only included at the last moment for comparison reasons. In other words, the amount of effort put in neural networks is disproportionate in those papers. In [53] a considerable amount

of effort is put in both SVRs and NNs and then we see that NNs perform slightly better than SVRs. However, difficulty is that when both models perform almost equally good, the SVR has the advantage of having lower computational costs which can be an attractive feature in PEMS applications. None of the closely related work has put effort into testing the new variants of neural networks like RNNs. The results obtained by [60] and [61] are promising and therefore neural networks should be given a fair share of attention while developing a new emission model.

The papers in the table are all based on their own dataset with unique characteristics and a variety of performance measures are used, which makes it complex to compare results. Furthermore, design choices made sometimes seem to prefer performance over the existing legislation. For instance, a PEMS may have no more than 15% uncertainty, to keep uncertainty low one wants to limit the number of features for this reason, however some of the papers in the table have many input features. However, a quick scan of the table learns us that, in most cases, R^2-scores around 0.94-0.96 are obtained. This is also in line with the score Emission Care currently obtained for our dataset. With their current approach and software they obtained a R^2-score of 0.97. As related work as well as Emission Care have retrieved this score, this seems like a reasonable score to aim for during this research as well.

5 Data Preparation

In this chapter the preparation of the data is discussed. In section 5.1, the specifics of the data are mentioned as well as how it is cleaned. Subsequently, in section 5.2, we elaborate on how data is selected and adapted. We then proceed with the feature selection in section 5.3 which focuses on the individual features and how selections are made.

5.1 Data exploration

The data used in this research was gathered on a Norwegian offshore production platform. It consists of 163 measured features excluding the target feature. However, this data also contains features that are not usable for modelling, like alarms. Those are used for other purposes like monitoring the process and the measurement equipment. Moreover, the data consists of 13 composed features which are created by arithmetic division of two features. The advantage of those features is that it reduces the number of input parameters which accelerates the modelling and results in a less complex model. The resulting model is aimed to predict the NO_x concentration. The dataset consists of multiple columns providing information about the NO_x concentration, i.e. the lower and upper range of and the values measured at different volume percentages. When we refer to the NO_x concentration in this research, we refer to the concentration at 15 volume percent.

All features mentioned so far are not cleaned yet. The cleaning entails, among other things, checking whether the values are within a reasonable range and whether the trends between features are as can be expected. As this cleaning requires domain knowledge we use the cleaned features from Emission Care BV. Their feature list consists of 21 features of which 10 are composed. The difficulty of the composed features is that most are assembled as combination of other features in the selection. This causes high correlations among features making the feature selection harder as the standard practice techniques often struggle to cope with dependencies between features. Additionally, the composed features make it more difficult to interpret results. Therefore, we decided to only utilise the 11 measured features throughout this research.

MIP columns	
All rows	15,853
- *Error*	3,420
- *Blank*	3,395
- *Break*	3,395
- *Cut*	3,395

(a)

String	**Occurrence**
Error	10,471
Blank	5,544
Break	4
Cut	3

(b)

Table 5.1: Data exploration in numbers

Data cleaning
The original dataset consists of 15,853 rows. Among these rows are also the faulty measurements and other irregularities. Besides numerical values the data contains strings as *Error*, *Blank*, *Break*, and *Cut*. These strings are partly added during the gathering of the data and partly while checking the data validity. Table 5.1b states how often each of those 4 strings occur in the data, whereas table 5.1a states the number of rows left when those strings are deleted. One row can contain multiple strings and hence a deletion of a string does not necessarily lead to the deletion of an equal amount of rows. Emission Cares' final dataset has 3,420 rows with 21 features and the target. The final number of rows in this research is slightly lower. In this research all rows that contained one or more of the previous mentioned strings were deleted, while Emission Care kept the *Blank*. As a result our dataset consists of 3,395 values of our

11 features and the target.

Feature name	Tag-name
F_1	EP____2660_MIP
F_2	FT____2601C_MIP
F_3	PDT___2626_MIP
F_4	PDT___2627_MIP
F_5	PT____2631_MIP
F_6	PT____2652_MIP
F_7	ST____2601_MIP
F_8	TT____2616_MIP
F_9	TT____2617_MIP
F_10	TT____2625_MIP
F_11	TZT___2658_MIP

Table 5.2: Features renaming

Features
The features all have tag-names that indicate what they measure. However, since these names can be hard to distinguish without domain knowledge we have simplified their names. The new feature names, along with their old tag-names, can be found in table 5.2. Next, every feature is plotted against the original number of rows. The feature plots of feature 1 and 7 can be found in figure 5.1, while an overview of all features can be found in appendix A. The grey areas in the plot represent the rows that have been deleted in the cleaning process, whereas the yellow areas are part of our data selection. The y-axis differs per feature plot and provides insight into the feature range of every feature. Although the ranges differ, fluctuations seem to follow a same kind of trend of increase and decrease for most features. This is also visible for the selected features: F_1 and F_7.

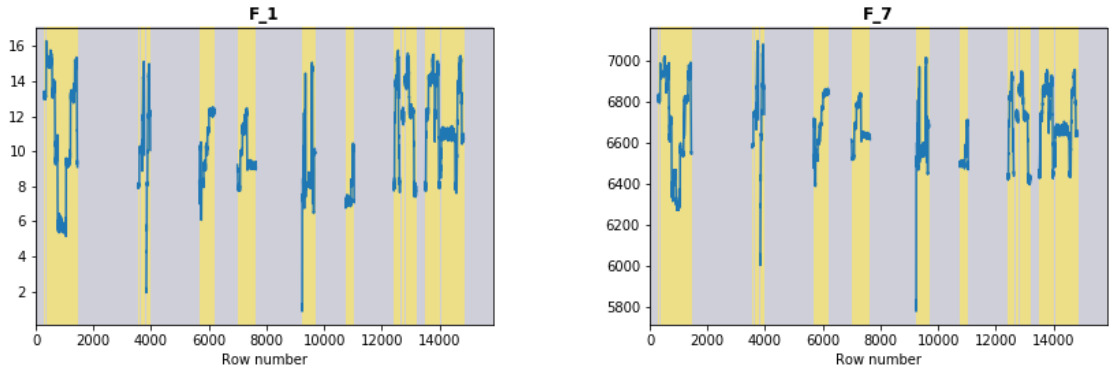


Figure 5.1: Behaviour of features, F_1 and F_7, over time

Target
The data is distributed over 38 time series. This number is partly based on measurements done and partly caused by our deletion of rows. Measurements were taken every minute. To retrieve more insight into the timeline of measurements taken the target feature is plotted against the time. The resulting plots show an overview of the time series measured in a day and indicate the number of measurements a time series consists of. Figure 5.2 shows an example of two measuring days. The figures of the other time series can be found in appendix B. Considering the dates it strikes that we can identify two measurements sessions. One in September 2018 and the other one in December 2018.

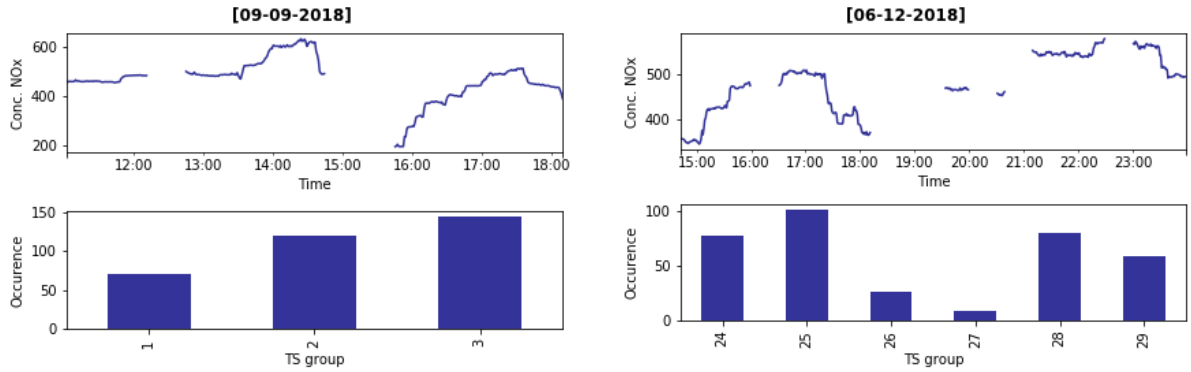


Figure 5.2: NO_x concentration over time

The figures display quite some variation in the number of data points per time series group and the range of the target feature. The lowest NO_x concentration is 292.2 against 629.8 as the highest concentration. The distribution of the NO_x concentration, figure 5.3, teaches us how the fluctuations add up in the total view. Important is to note that this is not necessarily how the NO_x concentration is usually distributed. When measurements are taken for a PEMS multiple scenarios are covered, while in practice the production platform usually operates in a more constant way. The fact that the data seems to be log normal or normally distributed is therefore not something we can utilise in this research, however it does add to our understanding of the data.

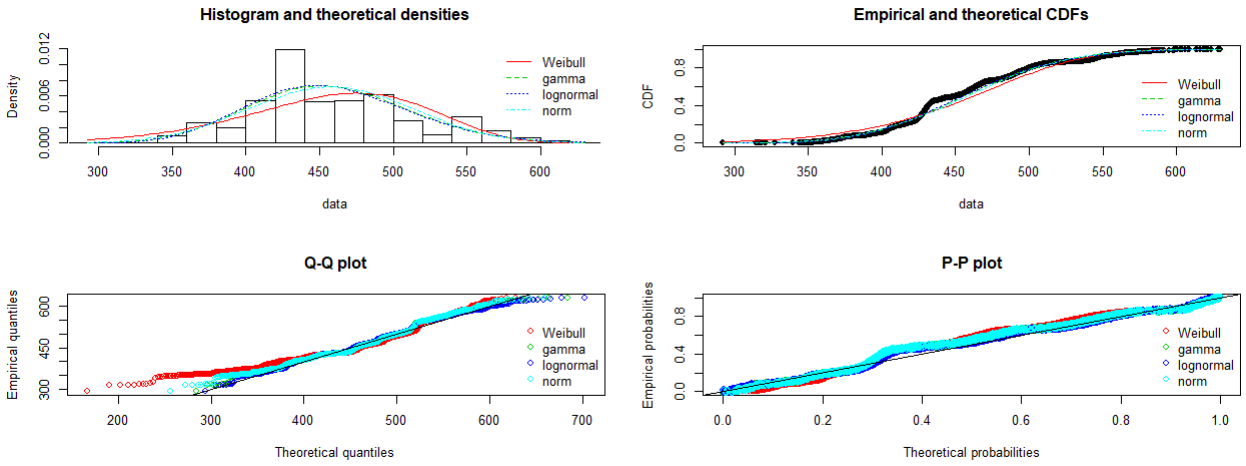


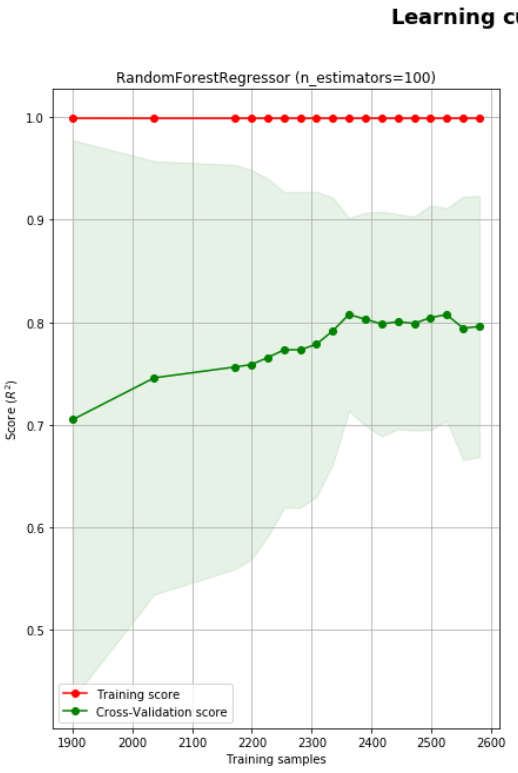
Figure 5.3: Distribution of NO_x concentration

5.2 Data selection

There is a variety of approaches when it comes to the selection of data when creating PEMS. Thanks to the availability of timestamps the data can be treated as time series data. However, it also happens often that these timestamps are ignored and that the data is randomly split into train, test, and validation data. This subchapter clarifies the choices made for this research. It first determines the optimal train-test split ratio, it explains why and how the measured time series were split into multiple intervals and it discusses how the train-test split was applied to the generated intervals to create the final data selection.

Train-test ratio

A train-test ratio can have a major impact on the chosen model and its performance. A commonly used train-test ratio is a 80-20 split [20]. However, the optimal split ratio is dependent on the data available and thus differs per case. In order to test potentially good splits for out data we utilised the learning curve algorithm of sklearn [56]. This algorithms requires an estimator to calculate the score for every train-test split entered. This score is calculated five times to remove any randomness. We use a random forest with a 100 trees as our estimator. Moreover, we create a list with potential train-test splits to test on. This lists starts with a 70-30 splits. As our dataset roughly contains 4,000 rows, a test set of more than 30 percent seems unrealistic. After all, the model should contain multiple scenarios and requires data for this purpose. Furthermore this list consists of a 75-25 and 80-20 list. From 80-20 on we decided to include all splits up to 95-5. These smaller steps from 80 on are based on the fact that these ratios between training and testing seem more realistic for our small dataset. We went up to 95 since the Pavilion8 manual [63] recommends a test set between the five and ten percent.



No. of rows	Train percentage	Score
1901.0	0.7	0.705
2037.0	0.75	0.746
2172.0	0.8	0.756
2199.0	0.81	0.759
2227.0	0.82	0.766
2254.0	0.83	0.773
2281.0	0.84	0.773
2308.0	0.85	0.779
2335.0	0.86	0.792
2362.0	0.87	0.808
2390.0	0.88	0.803
2417.0	0.89	0.798
2444.0	0.9	0.801
2471.0	0.91	0.799
2498.0	0.92	0.804
2525.0	0.93	0.807
2553.0	0.94	0.794
2580.0	0.95	0.796

Figure 5.4: Learning Curve

Figure 5.4 shows the learning curve for our dataset based on the train-test splits. From 80 to 85 percent of the training data the score increases steadily. From 85 on we see a steep increase with two peaks at 87 and 93 percent. Opinions in literature differ when it comes to deciding the optimal split. On one hand there are researchers that recommend taking the first peak, 87 percent, as taking more data could overfit the model. On the other hand, there are also researchers that claim that the highest score retrieved should be chosen, which would be the second peak. With Pavilion8s' recommendation in mind we have decided to go for a middle way and use a 90-10 split.

Creating intervals

Every row has its own timestamps as the data is gathered in time series. However, for a well-functioning

PEMS it is not required to know any of the previous values measured. In other words, it would not harm the results when the rows are shuffled. However, in practice we often see that related work divides its data into intervals, to make sure the PEMS functions on a scenario rather than on randomly chosen snapshot. Another motivation for us to work with intervals is that we want to research the influence of adding information from prior rows. The difficulty is to find the balance between the number of rows in an interval and the number of prior rows to include.

For this purpose we test a variety of interval lengths. Intervals range from 10 till 60 rows with a step size of ten, we refer to them as IV_10, IV_20, and so on. The intervals are created by dividing the measured time series into pieces of the desired length. As most of the time series cannot be divided by the interval size without a number of residual rows, the last interval of almost every time series is shorter. These shorter intervals will not be filtered out as the amount of data is already limited.

Then we proceed by splitting the acquired intervals of IV_10 till IV_60 into the desired train-test ratio split of 90-10. For this purpose we utilise sklearns' *train_test_split* algorithm [56]. Normally, one would input the data, a preferred train-test ratio, and a random state number. In our case the train-test ratio is the 90-10 as established before, however instead of inputting the data, we input all interval numbers of IV_10. If we would not indicate a random state number our results would differ every time we run the algorithm, because the 90-10 split is made randomly by the algorithm. By defining a random state number we ensure that the split is identical, if conditions remain unchanged. But we go one step further and utilise the random state number in our advantage. As said before not all intervals will be of the set length. By varying the random state number we can generate multiple different train-test splits which we can rank to find the best one available. If we would not test for multiple splits the split would influence the choice of the interval size to a great extent and this is something we want to avoid. In this case that means that we want to rank the splits to find a test set that is representative for our train data. This is achieved by creating all train-test splits for all random number states between zero and 1000. And ranking them according to a set of rules:

> **Rule 1:** The mean and standard deviation of the train and test set should not differ by more than five percent.
>
> **Rule 2:** The NO_x concentration in the test set should not exceed the boundaries as set by the train set.
>
> **Rule 3:** The distribution of the NO_x concentration in the test set should be identical or as close as possible to the distribution as observed in the train set.

The first two rules are according to the recommendations as done by the Pavilion8 manual [63] and they should both be passed. The second rule ensures that the NO_x concentrations present in the test set do not exceed the highest concentration in the train set nor the lowest. The third rule tests how representative the test set is with respect to the train set. Pavilion8 and many others in the field do not test for this as the data obtained covers multiple scenarios whereas their main focus is on testing the validity of the PEMS under normal operational circumstances. However, since this is a scientific research and the distribution impacts the choice of the interval size to a great extent, we decided to add this rule as a selection criterion. Additionally, this third rule gives a numerical value which enables us to rank the options. The distribution is calculated as the distribution over the number of bins as established by the Freedman-Diaconis rule. The differences between the percentages per bin are summed and this score gives us a final ranking. This process is repeated for IV_20, IV_30, IV_40, IV_50, and IV_60. Furthermore, this process is also repeated to split the train intervals in train and validation intervals, so that the validation sets can be saved to only apply to the final models.

History
Every interval is now classified as either train, test or validation. Next we can focus on adding columns with the values of prior rows. In this research we also refer to these as historical rows. The advantage of creating those columns is that we are able to use prior information in our model when the model does not have a memory.

Suppose we take IV_10 and add a single row as history data. The first row in our first interval will not have any prior rows and therefore no history can be added to this row. However, to the second row in this interval we can add all values of the first row. The same goes for the third row and so on till the tenth row. This process is repeated for all individual intervals. In this process we focus on the intervals rather than the time series, which means, for instance, that the first row of interval two does not have prior information although this information is available. However, we decided to do so as taking the prior information would only be an option as the second interval is part of the same group, train, test or validation, as the first interval. This makes this an complex situation and therefore we keep the focus on the intervals. As a consequence all first rows in every interval do not contain any prior information. As we delete the rows without history for the history datasets this means that dependent on the history size the amount of data available shrinks. We applied this principle for up to five prior rows. And we repeated the process for all our different interval sizes. When we refer to these datasets we mention the number of history rows as identifier, thus the dataset without history is H0, the dataset with one row of prior knowledge is referred to as H1, and so on.

Best interval
The next step is to determine which of the different interval sizes and corresponding history sizes performs best. This is established with the use of a random forest with 100 estimators. The performance scores of the different combinations can be found in table 5.3. The interval sizes without any prior information are highlighted in grey and the best scoring history size per interval size is highlighted in yellow. The interval sizes without any prior information clearly perform less than the ones with prior information. However, when no prior information is included, IV_10, IV_20 and IV_50 perform best with MAE scores below four. When also focusing on the number of prior rows it stands out that IV_20 and IV_50 retrieve MAE scores below 0.1, while IV_10 has a slightly higher score. However, it is difficult to establish the best interval based on these MAE scores as the scores are all close to another and relatively low. Instead of only looking at these MAE scores the number of rows left will help to make the decision. The difference between the before mentioned well performing IV_10 and IV_50 can be up to 500 rows which is an large percentage of the roughly 3,400 rows available for this research. As IV_50 obviously has the most number of rows left and performs well, both with and without history, this will be the interval size of our choice. As with regards to the history size, the scores are so close no definitive decision is made for the time-being. The optimal history size will be established while testing the models.

Now we have established that we will use a interval length of 50 with a 90-10 split, we want to give some insight into how these intervals behave and which are classified as either train, test or validation. A complete overview for all 87 intervals can be found in appendix C, while figure 5.5 contains a selection of these figures. The figures show that there is quite some variation in the range in which the NO_x concentration differs. Additionally, it is clearly visible that the test intervals are nicely spread over the intervals which are still in order of measurement. By paying attention to the really short intervals one can still identify the old time series.

5.3 Feature selection

While developing a PEMS it is important to minimise the number of features, because the uncertainty of a PEMS may not be more than 15%. This uncertainty level is not only dependent on the performance of the model itself, but is also influenced by the uncertainty of the measurement tools used to measure the features. The influence of this last group can impact the uncertainty to such an extent that it can

Interval size	History size	No. of datapoints	MAE score
10	0	3395	3.370
10	1	3040	0.114
10	2	2690	0.123
10	3	2341	0.128
10	4	1993	0.172
10	5	1656	0.231
20	0	3395	3.590
20	1	3206	0.086
20	2	3021	0.085
20	3	2837	0.083
20	4	2654	0.113
20	5	2478	0.120
30	0	3395	7.145
30	1	3265	0.346
30	2	3136	0.248
30	3	3007	0.268
30	4	2879	0.307
30	5	2752	0.283
40	0	3395	6.976
40	1	3292	0.133
40	2	3191	0.154
40	3	3090	0.212
40	4	2989	0.409
40	5	2893	0.281
50	0	3395	3.856
50	1	3308	0.080
50	2	3224	0.085
50	3	3141	0.087
50	4	3058	0.095
50	5	2978	0.099
60	0	3395	8.937
60	1	3319	0.226
60	2	3244	0.233
60	3	3169	0.220
60	4	3095	0.256
60	5	3021	0.242
	Interval without history		
	Best performing interval for interval length		

Table 5.3: Interval analysis

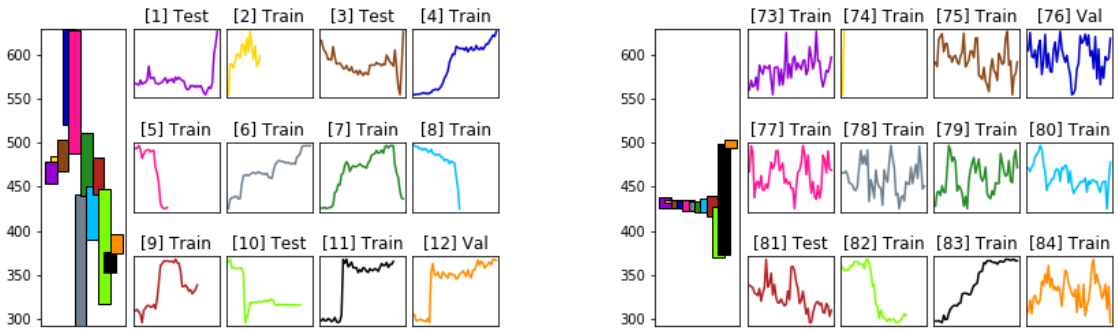
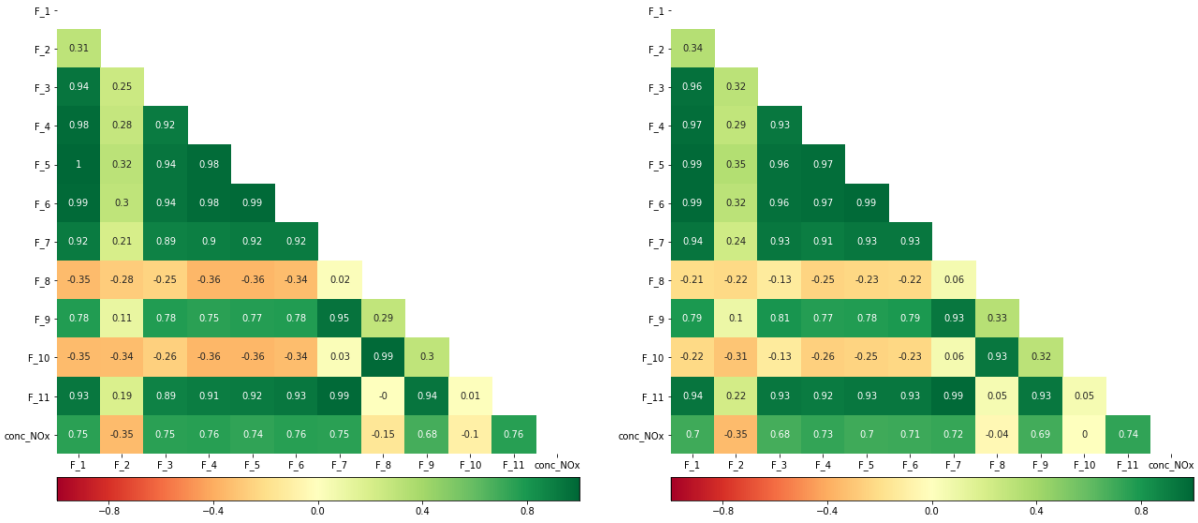


Figure 5.5: Selection of where intervals are located in terms of NO_x concentration

even be that a simple model with a lower performance scores is preferred over a more complex but better performing model. This section applies numerous feature selection methods.

Common practice

We first focus on the feature selection algorithm most commonly used. One of these methods is the creation of a correlation heatmap. It is often made to see whether there are relationships between features. These relationships are undesirable as those can lead to misleading results. In this research we create correlation heatmaps based on pearson and spearman correlation. The first-mentioned gives us insight into the degree of linear relationship between features, while the latter one is focused on the monotonic relationship between features. The correlation heatmap of both correlations can be found in figure 5.6. We speak of a high correlations when the correlation coefficients are higher than 0.5 or lower than -0.5. Both heatmaps clearly contain a lot of highly correlated features. This complicates the feature selection process, as features with correlations above 0.9, are more or less interchangeable. Furthermore, the trends and values in both heatmaps are nearly identical.



(a) Pearson correlation

(b) Spearman correlation

Figure 5.6: Correlation heatmaps based on different correlation types

Although performing the feature selection with highly correlated features is not recommend, we do give it a shot. One of the frequently used algorithms is *SelectKBest* from sklearn [56]. This method uses a scoring function to rank the features, based on feature importance, and it returns the k best features. In this research we performed this method with F-score as well as mutual information as scoring functions. The results can be found in respectively figure 5.7a and 5.7b. The two rankings both have features F_6 and F_11 ranked in their top-3. If we look at the ranking of the other features we see that there is not much consensus. This emphasis the problem of highly correlated features once again, as features with high correlations can be interchanged.

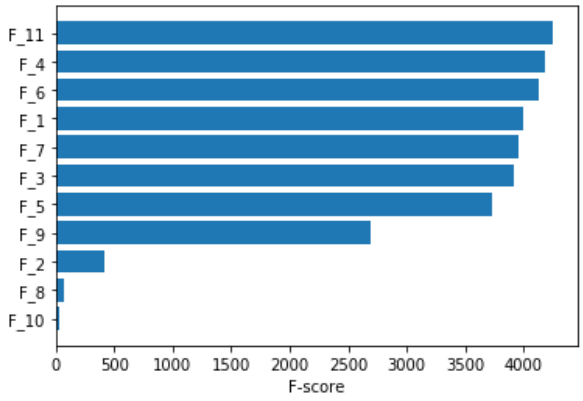
Next we train a random forest with 100 trees to rank the features again. However, this time we permute the column values instead of using the feature importance as we have done before. We run the random forest with the original features and later run it again with one of the features permuted. The difference between the first and the latter is the importance of the permuted feature. This process is repeated for every feature in the data. Furthermore, a random feature with random values is added to our feature selection. Whenever this random feature outperforms one or more real features, those features should not be considered for selection anymore. The idea behind it is that the surpassed features contribute less information than a feature without any knowledge of the process [64]. The outcome of this method can be found in figure 5.7c. Just as we saw before, feature F_6 obtained a top-3 ranking. Also feature F_2 retrieved a high score, whereas this feature was dominantly present in both other rankings.

The next commonly used method is recursive feature elimination (RFE). This method is computationally less expensive than the previous method but is often perceived as less reliable. Just like the permutation method, RFE also requires an estimator. For this purpose a random forest with a 100 trees is used again to simplify the comparison. With RFE one can enter the number of desired features. RFE then recursively considers smaller and smaller sets based on the number of features to select. In the end this results in a feature importance ranking based on the coefficient as assigned to every feature. In order to make ranking of all 11 of our features, we first let the algorithm calculate the most important feature, then the two most important features, then the three and so on. This results in a ranking which can be found in figure 5.7d. Features F_6 and F_11 have both retrieved a top-3 position again. Feature F_2 also contributes a lot according to this ranking which is in line with the results obtained from the permutation method. Next RFE is performed with five-fold cross-validation (RFECV) in order to remove the randomness that normal RFE can still contain. The outcome of the RFECV can be found in figure 5.7e. Unlike the other figures this one does not give a ranking, but rather gives insight into the optimal number of features. Eight would be the optimal number of features according to the figure. However, given that we should minimise the number of features, three features seems more realistic. This idea is also supported by the outcome of the RFE.

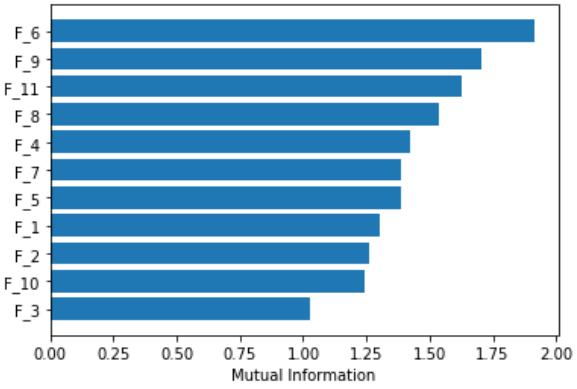
Thus features F_6 and F_11 seem to be good candidates to select for a model. However the various selection methods do not give a conclusive answer when it comes to which third feature to select. Furthermore, it remains difficult to judge whether these results can be trusted as all methods used are not suited for highly correlated features.

Feature elimination

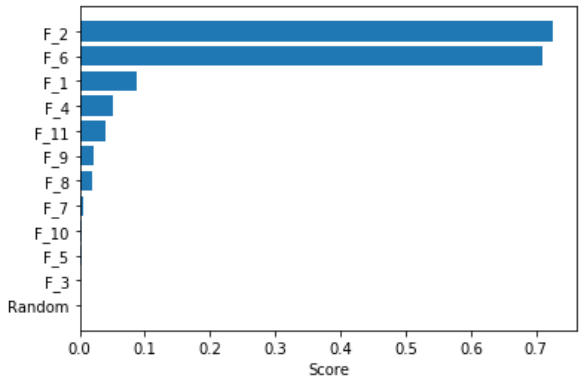
To retrieve more reliable results we have to tackle the high correlations. Therefore, an algorithm is written that eliminates features. Essentially this algorithm does not throw away the features but rather saves them in a list with features that can be replaced by each other. The algorithm starts by calculating the pearson correlation between all features. All feature pairs with a correlation value of one are selected. The algorithms keeps the feature first in the alphabet of every pair and places the other feature on the replacement list. The algorithm proceeds to do the same for a correlation of 0.99 and os on. Whenever two features with replacements are linked together the one first in alphabet remains while the other is added to the replacement list with all of its belonging replacements. The algorithm is designed that one can stop the feature elimination in two ways: (i) maximum correlation acceptable, or (ii) number of



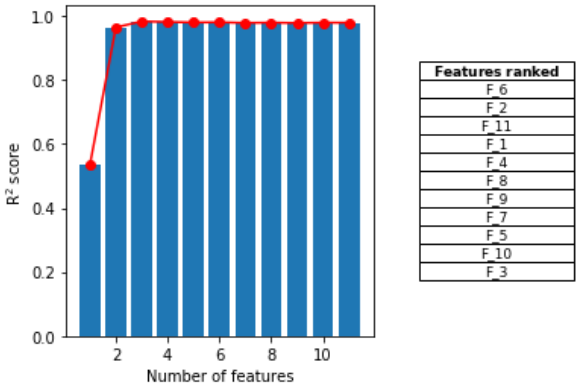
(a) SelectKBest with F-scoring



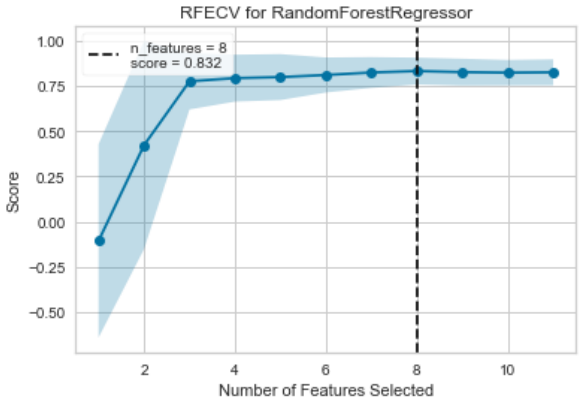
(b) SelectKBest with mutual information scoring



(c) Permutation based on RF



(d) RFE based on RF with ranked features



(e) RFECV based on RF (CV=5)

Figure 5.7: Feature selection algorithms

leader features desired.

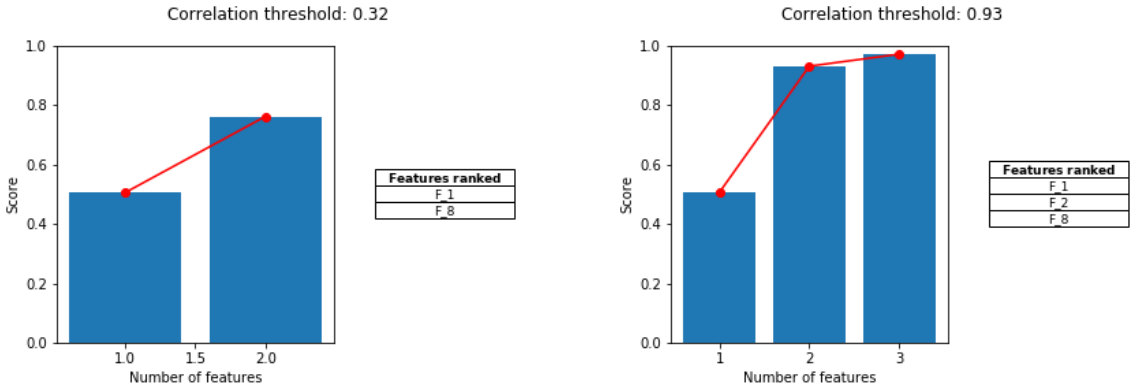


Figure 5.8: RFE with two and three features

RFE is used to visualise this elimination process. As this method is able to visualise up to two features, this is the minimum number of leader features. We first run a RFE, as explained before, with all features and without any restrictions to get a sort of ground truth. Next, we made the replacement list for a threshold of 1 and run the algorithm with only the remaining features. The features on the replacement list thus not taken into account anymore. We perform this process for all thresholds and for both pearson and spearman correlation. The results of the pearson and spearman correlation can be found in respectively appendix D.1 and D.2. These appendices do not contain the figures of all correlation thresholds up to two variables, rather they contain the figures of the correlation thresholds where replacements are made. The results differ in the beginning, but when it becomes crucial for this research there is agreement. For two and three features the results, in terms of remaining features and their ranking, are the same and can be found in figure 5.8. It stands out that especially F_1 and F_2 add to the performance of the model. Furthermore, it strikes that the score obtained with three features is much better than with only two. This confirms the image of the RFECV that three features would be the best option for a model with minimal number of features.

Lead feature	**Replacement features**
F_1	F_3, F_4, F_5, F_6, F_7, F_9, F_11
F_2	*No replacements*
F_8	F_10

Table 5.4: Replacement list

The replacement list as it belongs to the scenario where three features are left can be found in table 5.4. It stands out that feature F_1 has a lot of replacements, while the other two features only have one or no replacements. Looking back at figure 5.1, it now make sense that F_1 and F_7 seem to follow the same trend. Whilst validating the features Emission Care found that it was best not to use F_6 and F_8 for modelling. Hence those two features are only considered in this research when all features are taken into account for a model. Together with the fact that F_6 and F_11, the well-performing feature combination according to traditional methods, both are replaced by F_1 makes that this combination is not considered as an option anymore. The other feature that performed well in part of the traditional methods is F_2, however both feature F_1 and F_2 are already lead features. To gain more insight into the three selected features, we plotted their value against the NO_x concentration. Furthermore, the cumulative distribution over the target range is also plotted as a fourth figure. The resulting figures can be found in figure 5.9. Those figures emphasise the fact that F_1 influences the concentration NO_x the

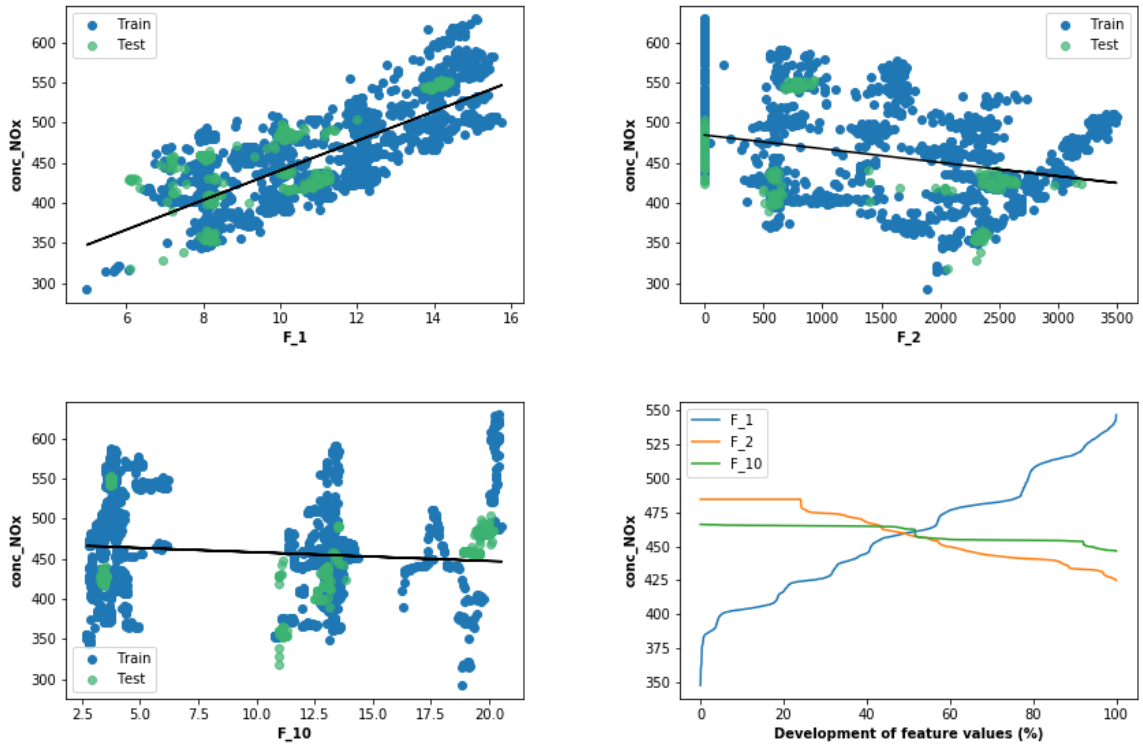


Figure 5.9: Feature(s) versus target

most. Furthermore, we see that F_10 contains a number of gaps when it comes to values. This requires additional attention when testing the final models.

We continue by forming all pairs of two among the features of the different groups. These combinations are plotted against each other and their colour is determined by the value of the NO_x concentration. The resulting figures can be found in appendix E. The outcome of the three combinations consisting of leaders of every group of features can also be found in figure 5.10. As one might expect the combination F_1 against F_2 gives seems to give the clearest image when it comes to the NO_x concentration.

The difficulty with the figures discussed before is that they are based on two features while we concluded before that a model based on three would perform better. For this reason we created a 3D plot with all leader features as shown in figure 5.11. A representation from the view of each of the features can also be found in appendix F. We can identify multiple clusters in the plot and every cluster seems to have one predominant colour. The difficulty, however, is that there are often multiple clusters with the same predominant colour. In practice this probably means that a linear regression will struggle with the complexity of the data, while the more complex regression models are more capable of handling this kind of logic.

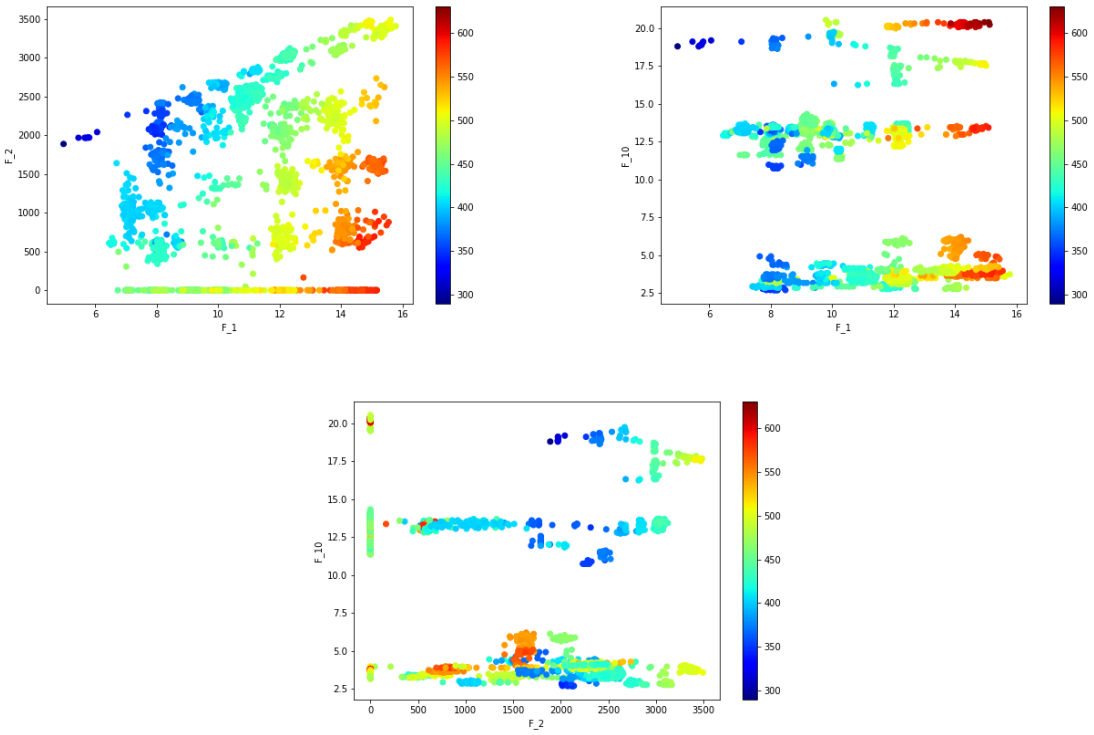


Figure 5.10: Two features plotted against each other with the target as colour

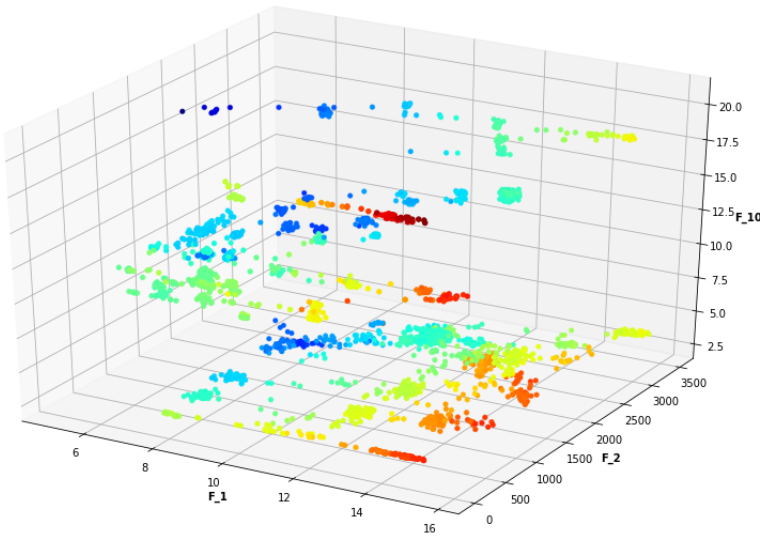


Figure 5.11: Lead features plotted against each other with the target as colour

6 | Methods & Results

For this research a variety of advanced regression models are designed. This chapter starts with the introduction of five different hyperparameter tuning techniques in section 6.1. Subsequently, sections 6.2-6.5 introduce the four types of models, namely linear regression, tree-based, support vector regression and neural networks. For every model the section starts with the method, which describes the process of implementing the model. This is followed by a results section, where the obtained results are discussed. Every type of model is built several times as it enables us to differ among hyperparameter tuning techniques and datsets.

6.1 Hyperparameter selection

Hyperparameters are basically the settings of a model, which need to be established before the model is trained. Hyperparameters can help one to find a balance between under- and overfitting. The related work table in section 3.7, shows us that grid search and genetic algorithms are often used for this purpose. A variety of different genetic algorithms is implemented in related work, which makes it difficult to determine if there is one best practice. In this study we use grid search to select our hyperparameters. This method tests all the hyperparameters independently. For all the possible values of hyperparameter the train and test score is calculated and visualised with the aid of the validation curve as implemented in the sklearn package [56]. The scores of the validation curve are calculated several times, based on cross-validation, so that the uncertainty of the results can also be taken into account. However, there is no consensus on how to interpret the resulting validation curves. One could either select the value of the first peak, before the score decreases again, or one could select the value of the maximum validation score retrieved. The difficulty with the first-mentioned is that this might lead to underfitting, while the latter argument can lead to overfitting. Another downside of this method is that the hyperparameters are determined in parallel, so the effect of the hyperparameters on each other and their ability to work together is unknown. For this reason also an incremental approach of grid search is implemented. In the incremental version we search for the hyperparameters one by one. The ones found will be set to the estimator used. In this way the influence of those hyperparameter decisions are taken into account while finding the next ones. Applying this gives us a total of four hyperparameter methods: (i) first peak in parallel, (ii) first peak incremental, (iii) maximum peak in parallel, and (iv) maximum peak incremental.

The downside of the proposed incremental method is that the order in which hyperparameters are searched influences the values found. For this reason we also use a random search in this research. We use sklearns' *RandomSearchCV* [56] for this purpose. This method selects one possible value for every hyperparameter and calculates the resulting score. This is done for a set number of rounds. In the end it returns the grid which performed best. The advantage of a random search is that it is less time-consuming than a grid search. However, the downside of this method is that potentially a large number of hyperparameter settings is never tested. For this reason it is important not to enter too many values for every hyperparameter, but rather to find reasonable ranges for every hyperparameter. For all cases where hyperparameters are included, the random search will try 100 unique combinations of hyperparameters and cross-validates this five times. Which hyperparameters are selected is based on mean squared error (MSE) scoring.

Normal linear regression does not have any hyperparameters, however the more complex models have. The tree-based models, the support vector regression, and the neural networks all have hyperparameters. As we still have six different datasets and five hyperparameter tuning techniques this leaves us with many variations. As training and testing all of them is time-consuming, this number will be reduced based on results found. As linear regression is based on linearity, results found for this method cannot be generalised to the other non-linear models. Therefore, the tree-based models are used to make decision with regards to which datasets and hyperparameter tuning techniques should be kept. Trees are chosen as these models are the most time-efficient of the non-linear models while still being non-linear which

enables us to generalise the results found. Based on the results the remaining hyperparameter tuning technique and datasets are then applied for support vector regression and neural networks.

6.2 Linear regression

Linear regression is the only linear regression model in this research. However, as it depends on linearity there are many assumptions a model has to meet in order to be deemed valid. In the literature section this assumptions were already discussed, however in the methods the assumptions are mentioned again and are now coupled to a measure or diagnostic to measure whether the assumptions are met. Important is to keep in mind that even a good scoring model, can be invalid when the assumptions are not met.

6.2.1 Methods

As linear regression is a relatively simple machine learning algorithm, it allows us to try many different settings. We start with drafting a regular linear regression based on our data without history, H0. Varying the features enables us to find a proper combination for the linear regression models. As this is determined, the next step is to apply the chosen feature combination to our other datasets, H1 till H5. Based on results found during this process, one dataset is chosen to execute the different types of linear regression. Besides regular linear regression, we then also implement Ridge, Lasso, and ElasticNet regression. Oftentimes a good-performing model is determined based on the performance score retrieved, however for linear regression it is also important that the assumptions, as introduced in section 3.2.1., are satisfied.

Five assumptions

The reliability of the performance score of a linear regression model, depends on how well the five assumptions of linear regression are met. Five models are trained based on H0 with varying feature selections. The feature combinations as made can be found in table 6.1, where CN refers to the combination number. As one can deduct from the table the model based on combination I contains all features, while combination II only contains the selected features. All the other combinations, III till V, consist of two features. Those combinations are the result of combining the selected features into pairs.

CN	Feature combinations
I	F[1,2,3,4,5,6,7,8,9,10,11]
II	F[1,2,10]
III	F[1,2]
IV	F[1,10]
V	F[2,10]

Table 6.1: Feature combinations for linear regression

The features of each combination are scaled to make sure that the range of each feature does not affect the results. The scaling is done with the aid of the *StandardScaler* of sklearn [56]. This scaler subtracts the mean from each feature and divides it by the standard deviation. The standardised features are then used to build five models based on sklearns' implementation of *LinearRegression*. The performance of these models is measured by calculating various diagnostics and by creating visualisations to see whether the five assumptions are met. The first assumption is linearity. Linearity can be demonstrated through a residual plot. In an ideal situation the residuals have the tendency to cluster towards the middle of the plot and are symmetrically distributed. The second assumption is normality which can be measured through the skew and the kurtosis which should be respectively zero and three. The third assumptions states that there may not be multicollinearity. This can be tested with the variance inflation factor (VIF)

which is based on the R^2-score obtained during training of the model. The VIF score is preferably as low as possible. A VIF below one means no correlation, while it is said to be moderate between one and five. Up to ten it is believed to be highly correlated, while a VIF score above 10 is reason for concern. The fourth assumption to test the performance is the homoscedasticity. To meet this assumptions the data in the residual plot should not only be distributed symmetrically, but should also not show any patterns. As this is hard to measure with the eye, also the Breusch-Pagan (B-P) diagnostic is used for this purpose. This diagnostic null hypothesis that there is homoscedasticity. In order to meet the requirement the p-value obtained for this diagnostic should be above 0.05, otherwise the data is set to be heteroscedastic instead. The fifth, and last, assumption is that there may not be any autocorrelation. This assumptions can be tested with the aid of the Durbin-Watson (D-W) diagnostic. The outcome of this diagnostic always ranges between zero and four, with a optimum of two. Except for the first assumption the others are all numerical and can therefore be plotted into a table. To give more insight into the effectiveness of the combinations also a column with insignificant features is added to this table. This is done with the aid of statsmodels' linear regression model [65] which identifies the p-value of every feature included in the model. A p-value below 0.05 is said to be insignificant and hence the features that obtain a p-value below 0.05 are stated in the table.

Execution

The process as described before is repeated for all the datasets. However, as said before, not all feature combinations are tested again. Based on results found for H0, one combination is used to create models for the other datasets. Also here the results are interpreted with the aid of the five assumptions and the residual plots. As a last step the best feature combination with the best-performing dataset is used to compare the different types of linear regression. The alternate forms of linear regression are all forms of penalised linear regression and are known as *Ridge*, *Lasso* and *ElasticNet* regression. In this research sklearns' implementation is used [56].

6.2.2 Results

This section first elaborates on the results found for dataset H0 with the feature combinations as established before. Next, it selects the best feature combination to test the other datasets, H1 till H5. Subsequently the combination with promising results is used to test and compare the different kinds of linear regression.

CN	Features	R^2 train	R^2 test	Skew	Kurtosis	VIF	B-P	D-W	Insign. feat.
I	F[All]	0.95	0.88	0.14	4.66	21.72	0.00	0.27	F_6 & F_10
II	F[1,2,10]	0.94	0.90	0.45	5.33	17.38	0.00	0.24	F_10
III	F[1,2]	0.94	0.90	0.46	5.29	17.37	0.00	0.24	X
IV	F[1,10]	0.59	0.56	-0.03	2.07	2.47	0.00	0.04	X
V	F[2,10]	0.16	0.29	0.46	3.02	1.19	0.18	0.02	X

Table 6.2: [*Linear regression*] diagnostics for H0 retrieved from the different combinations

Selecting promising combinations

The feature combinations, as mentioned in table 6.1, are used to select data from H0 to train and test linear regression models on. The diagnostics obtained from these models are shown in table 6.2. The first assumptions is linearity. Combination I, II, and III perform well according to their R^2-scores. Whereas this R^2-score in the table is based on the training data, the actual versus predicted plot, figure 6.1, displays the performance for the test data. An R^2-score of 0.90 was obtained for the testing data. Considering that training data obtained a MSE of roughly 180 against 280 for the testing data, one might expect some deviation between the actual and predicted emission. We can see that the model with feature combination III seems to follow the trends, but struggles to actually predict the emission correctly.

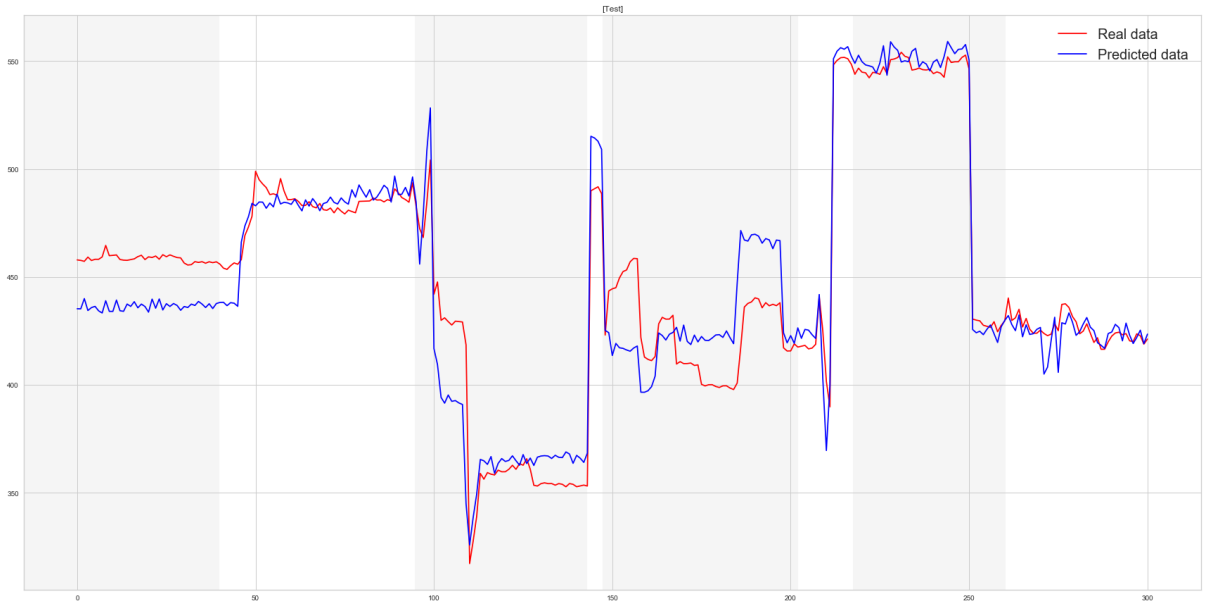


Figure 6.1: [*Linear regression*] actual versus predicted outcome (CNIII, H0)

The residual plots, in figure 6.2, also do emphasise this struggle. The residuals obtained are quite large. Especially for the last two feature combinations there are only a few points that are predicted right or nearly right. With regards to the second assumption, normality, there is no set of features that has both the skew and the kurtosis close to respectively three and zero. Combination V comes closest when it comes to satisfying the both of them. The third assumption is the multicollinearity which is tested with the aid of the VIF. As a VIF score above 10 is reason for concern, the first three methods do not comply this assumptions. For the other two methods there is only a moderate correlation between the features,

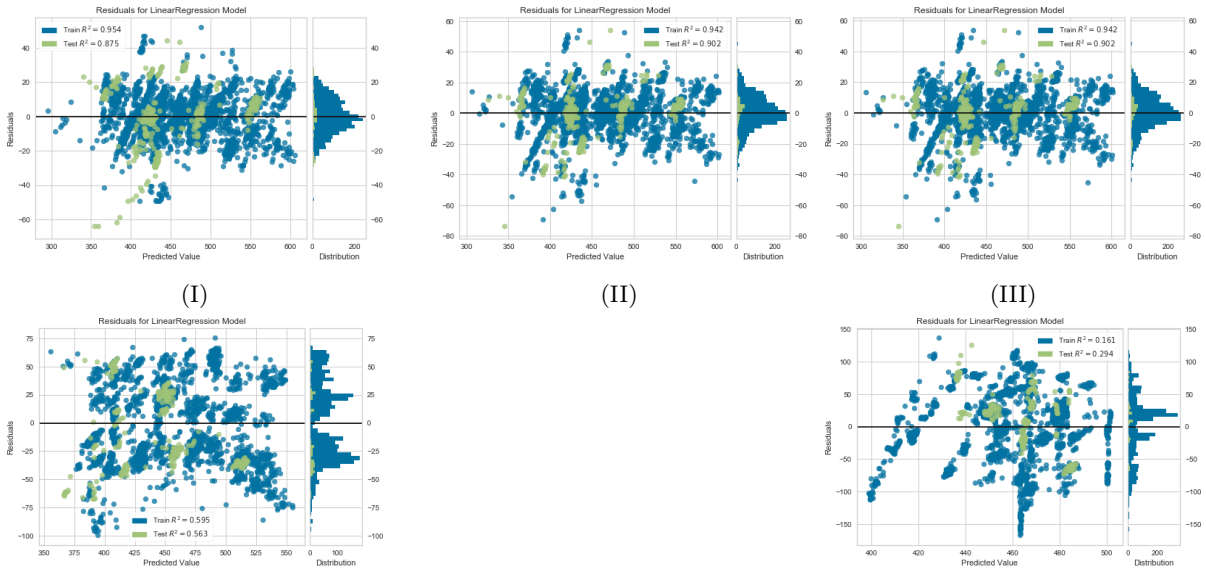


Figure 6.2: [*Linear regression*] residual plots of the H0 models based on the feature combinations

which is acceptable. The fourth assumption, homoscedasticity, requires a Breusch-Pagan score above 0.05. This is only achieved by combination V. In all other cases there is heteroscedasticity rather than homoscedasticity. The fifth assumption is no autocorrelation which is measured by the Durbin-Watson diagnostic. Given that this number ranges between zero and four, all of them are close to lower boundary rather than to two as would be desired. Lastly, the insignificant features are mentioned in the table. The first two methods both contain insignificant features. The insignificance of feature 10 in the second method can be seen in method III, where the deletion of this feature does barely have any effect on the outcome.

Thus, combination V seems to satisfy most of the linear regression assumptions compared to the other combinations. The difficulty, however, is in the performance of this method. The R^2-score as well as the residual plot obtained are not promising. The score as well as the residual plot of combination III are promising, however this model does not meet many of the assumptions. For the other datasets, H1 till H5, combination III is used for building the models. Although this method does not satisfy many of the assumptions, the diagnostics obtained for method V were not convincing enough to go ahead.

Dataset	**R^2-score**	**Skew**	**Kurtosis**	**VIF**	**B-P**	**D-W**	**Insign. feat.**
H0	0.94	0.46	5.29	17.37	0.00	0.24	X
H1	1.00	0.18	2.96	inf	0.00	0.29	X
H2	1.00	0.07	1.56	inf	0.00	0.12	F_1
H3	1.00	-0.05	2.38	inf	0.00	0.05	F_1-1, F_1-2, F_1-3
H4	1.00	0.15	1.78	inf	0.00	0.05	X
H5	1.00	0.20	13.86	inf	0.00	0.89	X

Table 6.3: [*Linear regression*] diagnostics for the different datasets (CNIII)

Results for H1 till H5

The diagnostics of the models based on feature combination III for the different datasets can be found in table 6.3. What immediately strikes is the extremely high R^2-scores obtained for dataset H1 till H5. This trend is also clearly visible in the residual plots where the plot is similar for dataset H1 till H5. In all those cases the residual plot, as can be found in figure 6.3, displays a straight line. However, when checking the diagnostics to see whether the assumptions are met, it strikes that the diagnostic do not seem to differ much among the different datasets. To test whether this behaviour is as expected we also applied the different datasets to our feature combination V, an equal trend was observed.

Therefore, we adapted the datasets to see whether the high scores could be explained. In dataset H1 all previous row values are present including the previous target value. Whenever this target was left out of scope, we see that results do not improve for the other datasets. Thus H1 till H5 then all obtain equal results to dataset H0. In order words, the history target values influence the prediction of the target to a great extent. This can also be seen when looking at the resulting formula, equation 6.1, where the previous target value is almost fully responsible for the outcome. Suppose this linear regression model would be used as actual emission model, then the target would be predicted based on previous predictions. In other words, the resulting model would almost neglect everything that happens inside the plant. This is an undesirable situation. As using history for linear regression is already tricky, previous rows in time series tend to be correlated while autocorrelation must be prevented, we think it is best to neglect the datasets with history when it comes to linear regression.

$$\begin{aligned} \text{conc\_NOx} = 7.95 \cdot 10^{-14} \times \texttt{F\_1} - 7.87 \cdot 10^{-14} \times \texttt{F\_2} + 9.02 \cdot 10^{-14} \times \texttt{F\_1-1} \\ +2.79 \cdot 10^{-14} \times \texttt{F\_2-1} + 55.86 \times \text{conc\_NOx-1} \end{aligned} \tag{6.1}$$

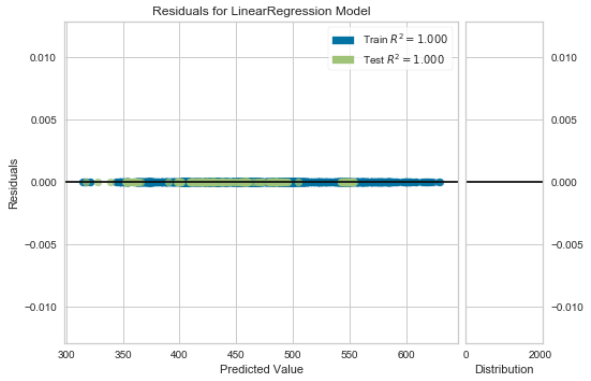


Figure 6.3: [*Linear regression*] residual plot (H1, CNIII)

Regressor	R^2 train	R^2 test	Skew	Kurtosis	VIF	B-P	D-W	Insign. feat.
Linear regression	0.94	0.90	0.46	5.29	17.37	0.00	0.24	X
Ridge	0.94	0.90	0.46	5.29	17.37	0.00	0.24	X
Lasso	0.94	0.90	0.47	5.21	17.10	0.00	0.23	X
ElasticNet	0.78	0.77	0.38	3.36	4.48	0.00	0.05	X

Table 6.4: Diagnostics for the different LR methods (H0, CNIII)

Other linear regression forms

As all combinations seem to follow the same trend, combination III is continued to be used to create the models with. In this section a normal linear regression is tested alongside Ridge, Lasso, and ElasticNet regression. For dataset H0 the scores, table 6.4, and residual plots, figure 6.4, obtained are roughly equal as the ones obtained with normal linear regression. We only see that ElasticNet performs slightly less than the others. The same applies for the diagnostics. In other words, performing alternate forms of linear regression does not improve its predictive power. Out of curiosity we also applied the alternate forms for dataset H1. Also here we see the same trend when it comes to the score and the diagnostics, however in the residual plot we see something interesting. Not only the ElasticNet shows a decreasing trend here, but also does the Lasso model. However, as the score remains 1 and the residual is frankly quite small, therefore we did not look further into it.

Figure 6.4: Residual plots for the different LR methods (H0, CNIII)

It was expected that a linear regression model for prediction of the NOx concentration should be unfeasible. The regression result is better than expected, though it is known that the NOx formation is an exponential function of the temperature level during fuel combustion. The relative high correlation of the linear model could be an indication that the observed NOx levels in the dataset originate at relative low temperature levels, where the exponential effect is not yet escalating. A linear relation is then a

first order estimate of the actual process, having an acceptable regression coefficient. In fact, gasturbines tend to operate at large excess combustion air flows, resulting in high oxygen levels in the exhaust gas. In practice, this large excess air flow has a tempering effect on the temperature levels during combustion. It is expected that linear regression of a component with large (instant) fluctuations, like CO (carbon monoxyde) will be more difficult.
To conclude, although the different models all satisfy part of the assumptions, none of them was able to satisfy all of them. This usually hints that results found are not reliable which is not acceptable for an emission model. Based on physics Emission Care already predicted the unfeasibility of a linear model for the prediction of the NO_x. The regression result is already better than expected, though it is known that the NO_x formation is an exponential function of the temperature level during fuel combustion. The relative high correlation of the linear model could be an indication that the observed NO_x levels in the dataset originate at relative low temperature levels, where the exponential effect is not yet escalating. Together with the fact that the distribution is more based on which data is deemed valid, we conclude that linear regression is a good fit for this research. However, we still included this section as it provided us with insight about our data.

6.3 Tree-based

The tree-based models are the first non-linear models in this research. Singles trees as well as ensembles are used to establish the best possible models. As we still have a large variety of datasets and hyperparameter tuning techniques, this method is also used to reduce the number of options. The conclusion of which datasets and hyperparameter tuning techniques are kept can be found at the end of the tree-based results section.

6.3.1 Methods

From all the non-linear models in this research, the tree-based models are computationally the least expensive. Therefore tree-based models are ideal for testing different methods. During the data selection no decision has been made with regards to the best hyperparameter tuning technique. The same goes for the optimal amount of prior information to take into account. As a result there are still five different hyperparameter tuning techniques and six datasets with different history sizes. This alone results in 30 unique combinations to be tested by the tree-based methods.

An important hyperparameter for each of the tree-based methods is the number of features. During the feature selection the features proved to be highly correlated. For this purpose the number of features is restricted by us rather than that letting every tree-based method handle this itself. The feature selection consists of the three selected features, namely F_1, F_2 and F_10. As the tree-based models are built for both all features and the selected features, this results in 60 sets of hyperparameters. These sets of parameters are then each set to their own estimator in order to calculate the MSE and R^2 scores on the test sets. The models and scores with all features included are meant for referencing, while the other models are used for the prediction of emission values. This process is repeated for all the different tree-based estimators. In this research we test with a single tree and with two kinds of ensembles, stacking and boosting.

max_depth	min=1, max=20, step=1
min_samples_leaf	min=50, max=100, step=2
n_estimators	min=10, max=200, step=10
loss	ls, lad, huber, quantile

Table 6.5: Tree-based hyperparameters

Single tree

The single tree used in this research is sklearn's *DecisionTreeRegressor* [56]. The shortcoming of a single tree is often that is has trouble with generalisation, however the advantage is the ability to visualise a single tree. The most important hyperparameters of a tree include the depth of the tree and the amount of samples each leaf should include at least. The possible values for each of the hyperparameters can be found in table 6.5. Important is to note that these hyperparameter can also restrict each other. If the minimum samples per leaf is reached, for instance, the maximum depth of the tree can still be much deeper but it cannot be reached since there is simply not enough data. For the minimum samples per leaf a commonly used rule-of-thumb is that it should contain at least between 0.5% and 1.0% of the data samples. However, since the dataset in this research is relatively small this would result in very few samples per leaf. Srivastava [66] recommends in his blog to set the threshold to at least 50 samples per leaf. This recommendation is implemented in this research. By implementing this rule we hope to prevent overfitting. A 100 samples per leaf is then set as upper threshold. All values in between, with steps of two, are then possible values for this hyperparameter. A similar rule-of-thumb for the maximum depth was not found. Many researchers range it between five and eight, while others stretch this range up to 32 or even 64. In this research the maximum depth of the tree is set to a 20 as the dataset is relatively small. We recognise that a depth of 20 is large given the size of our dataset, but think a too large depth will be prevented by the minimum samples per leaf setting.

Besides these two hyperparameters the random state of the *DecisionTreeRegressor* is also set to zero to make sure results do not differ per run. The 60 unique combinations then run and the best values found for the hyperparameters are then set to an *DecisionTreeRegressor* and with the corresponding dataset the MSE and R^2 scores are calculated.

Stacking

To test with tree-based stacking an *RandomForestRegressor* is used as implemented by sklearn [56]. We apply the same hyperparameter tuning techniques and values as we did for the single trees. Besides setting the random state, maximum depth and the minimum leaves per leaf, also the optimum number of estimators is a hyperparameter for this method. By default the number of estimators is 100, after years of it being 10. In this research, the number of trees ranges between 10, as it used to be, with a maximum of 200 estimators. All values in between with a step size of ten are considered. As stacking is done in parallel, also the number of jobs can be defined which enables us to run the algorithm in parallel. The number of jobs is maximised in order to minimise the computation time. As with the single tree, we tested all five hyperparameter tuning techniques with all six the datasets again in order to compute their MSE and R^2 scores.

Boosting

To test boosting the *GradientBoostingRegressor* from sklearn [56] is used. The hyperparameter settings are nearly identical to the ones for stacking. However, as boosting cannot be done in parallel, there is no possibility to define the number of jobs for these models. Rather boosting improves the existing model every time. Therefore, a number of different loss functions is tested to find the best fit. The loss functions considered are the least squares regression (ls), least absolute deviation (lad), huber, and quantile loss. Together with all the earlier mentioned hyperparameters these are tested for all five hyperparameter tuning techniques with all six datasets again and once again used in order to compute the MSE and R^2 scores.

6.3.2 Results

This section elaborates on the results found for the different tree-based models. It starts with the results of a single tree and then looks into two ensemble techniques, stacking and boosting, and their results.

	First-Parallel	First-Increment	Maximum-Parallel	Maximum-Increment	Random
H0	191.88	191.88	191.88	191.88	188.56
	0.93	0.93	0.93	0.93	0.93
	426.02	426.02	135.24	135.24	129.77
	0.85	0.85	0.95	0.95	0.96
H1	168.46	168.46	168.46	168.46	168.51
	0.94	0.94	0.94	0.94	0.94
	431.39	431.39	153.60	154.84	196.53
	0.85	0.85	0.95	0.95	0.93
H2	164.19	164.19	164.26	164.26	164.26
	0.94	0.94	0.94	0.94	0.94
	121.65	121.65	121.65	121.65	116.63
	0.96	0.96	0.96	0.96	0.96
H3	299.57	299.57	299.57	299.57	169.10
	0.90	0.90	0.90	0.90	0.94
	122.24	122.24	130.72	130.72	175.29
	0.96	0.96	0.96	0.96	0.94
H4	161.32	161.32	161.34	161.34	158.42
	0.95	0.95	0.95	0.95	0.95
	127.48	127.48	228.46	127.48	120.95
	0.96	0.96	0.92	0.96	0.96
H5	211.39	211.39	211.41	211.41	211.41
	0.93	0.93	0.93	0.93	0.93
	116.95	116.95	116.95	116.95	110.57
	0.96	0.96	0.96	0.96	0.96

All features
Three selected features [F_1, F_2, F_10]

Table 6.6: [*Single tree*] scores (upper MSE, lower R2)

Meanwhile evaluating the obtained scores also the datasets and hyperparameter tuning techniques are evaluated with the purpose of making a selection for the other non-linear models.

Single tree

We use the *DecisionTreeRegressor* to create our single tree models. The models are created based on hyperparameters found while applying the five hyperparameter tuning techniques with the six datasets with different history lengths. This is done for all the data as well as for a selection of features. The scores obtained on the test sets can be found in table 6.6, where the columns represent the different hyperparameter tuning techniques and the rows the different datasets.

The table contains the scores of the datasets without the historical target included for H1 till H5. This is done for the same purpose as when we deleted it from the linear regression, namely that it almost exclusively predicts the target. Also for the single tree, we saw a really big improvement when using another dataset than H0 when the historical target was still included. However, in the current situation we do not see much difference between the different datasets. Especially H0 and H1 seem to perform roughly equal and we see a slightly better performance for the other datasets. The advantage of a single tree is the potential to visualise the it. A visualisation of the two best-performing trees with limited features on H0 and H2 can be found in appendix G.

With regards to the hyperparameter tuning techniques it strikes that especially the first techniques seems to struggle with obtaining good results for the models with a selected amount of features. While the re-

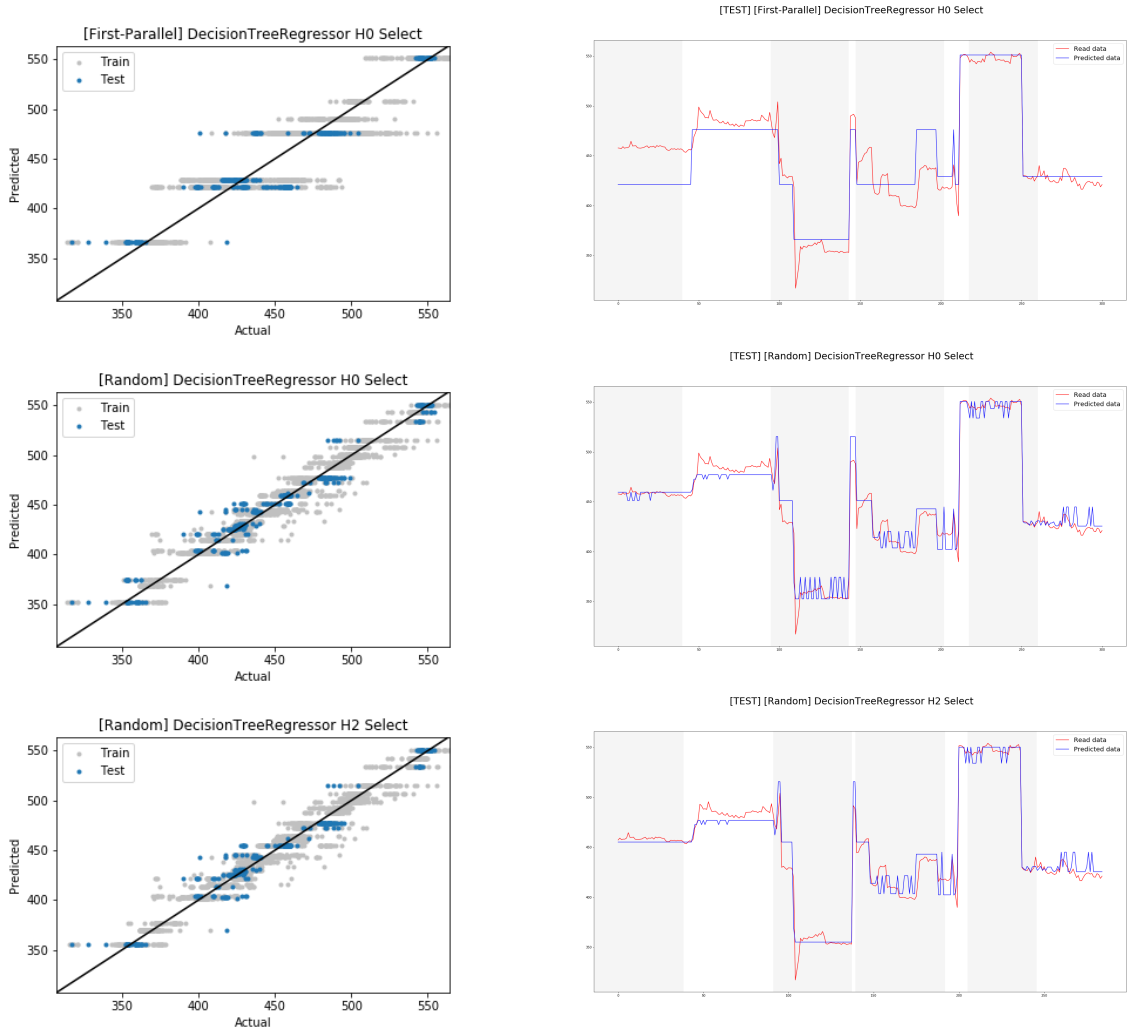


Figure 6.5: [*Single tree*] actual versus predicted outcome

sults found by the maximum and random technique are oftentimes close to one another. In figure 6.5 the actual emission is plotted against the predicted values. The figure displays the first-parallel and random technique for dataset H0 as well as the best-performing decision tree which is build for dataset H2 with the random technique. The figure on the right-hand only displays the performance on the testing data. The plot from the first-parallel technique clearly displays the lack of fit and contains very clear levels in both plots. For both the random techniques the levels are less visible. The scatter plots look very similar, however, we do see a clear difference in the line plots. Although the random techniques perform better, the levels are still quite visible in the line plots.

As the performance scores and the actual versus predicted plots for the maximum and random techniques are close, it is difficult to conclude anything with regards to which technique performs best. While reviewing the hyperparameters of the models with a restricted number of features, it strikes that, although the scores of the different models are close, the hyperparameters differ. All trees have a minimum samples per leaf of 50, however the maximum depth of the trees is more variable. For the first two datsets, H0 and H1, we see a lower maximum depth for the methods with only a limited number of features. However, whenever we focus on H2 and onward it stands out that the maximum depth does not differ between the different hyperparameter tuning techniques. The validation curves as obtained for the maximum depth

	First-Parallel	First-Increment	Maximum-Parallel	Maximum-Increment	Random
H0	159.09 0.94	159.09 0.94	148.12 0.95	148.12 0.95	152.25 0.95
	191.20 0.93	205.30 0.93	158.21 0.94	167.61 0.94	158.23 0.94
H1	160.28 0.94	159.98 0.94	145.20 0.95	144.27 0.95	145.74 0.95
	163.23 0.94	163.23 0.94	152.69 0.95	152.69 0.95	154.30 0.95
H2	163.37 0.94	157.66 0.95	158.66 0.95	157.66 0.95	153.09 0.95
	160.59 0.95	160.59 0.95	154.77 0.95	153.98 0.95	168.41 0.94
H3	157.44 0.95	156.59 0.95	154.09 0.95	154.23 0.95	154.05 0.95
	163.43 0.94	163.41 0.94	160.14 0.95	160.14 0.95	158.60 0.95
H4	164.77 0.94	167.15 0.94	166.46 0.94	165.88 0.94	156.91 0.95
	148.28 0.95	148.28 0.95	147.74 0.95	147.74 0.95	149.74 0.95
H5	162.08 0.95	159.54 0.95	162.08 0.95	160.44 0.95	153.87 0.95
	183.58 0.94	183.58 0.94	155.35 0.95	155.35 0.95	156.43 0.95

All features
Three selected features [F_1, F_2, F_10]

Table 6.7: [*Stacking trees*] scores (upper MSE, lower R2)

and the minimum samples per leaf by H0 can be found in figure 6.6. The validation curves are similar for the different datasets, in terms of shape as well as R^2-score. Whereas we saw the score of the maximum depth rising, the score of the minimum samples per leaf immediately decreases. The latter suggests that a better model, and thus a better score, might be achieved by allowing more samples per leaf, however this increases the chance of overfitting.

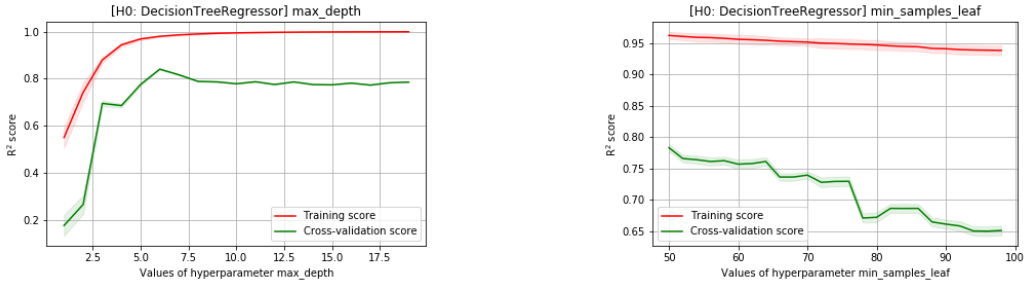


Figure 6.6: [*Single tree*] validation curve for maximum depth of the tree

In conclusion, the random and maximum-techniques give the best performance for the different datasets. The scores seem improve a lot when the historical target is included in the different datasets. However, in the current scenario there is not much variation among the different datasets.

Stacking

The process as described for the single trees is repeated and a similar table is constructed with a stacking tree-based estimator. The results produced by the *RandomForestRegressor* can be found in table 6.7.

The table includes some of the same trends that we observed for the single tree. Also for this estimator there is not much difference between the models based on all data as opposed to the ones that include a selected number of features. Furthermore, this estimator also heavily relies on the historical target and therefore also in this case only historical features are included. We see the same pattern as we saw for the single tree, where the dataset does not matter much for the obtained scores. However, where we saw an improvement from H2 and on for the single tree, that improvement is not visible here. To get insight into the predictions done by the random forests the actual versus predicted target is plotted again. One of the best performing models with not too many history rows, is based on the maximum-parallel technique with dataset H1. The plot is shown in figure 6.7 and it immediately strikes that the levels as we saw for the single tree are not visible anymore. Although the MSE of this model is higher than MSEs we saw for the single trees, the model seems to be a better fit. However, the model still seems to struggle to predict the boundary values correctly.

In conclusion, results found by the different hyperparameter tuning techniques are comparable. Most of the trends we saw for a single tree also apply for the random forests. However, for this estimator we do not see an improvement from a history of two on.

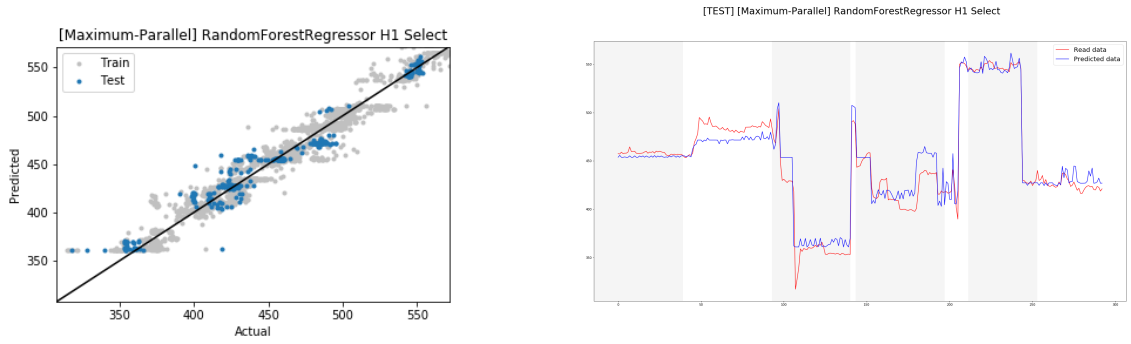


Figure 6.7: [*Stacking trees*] actual versus predicted outcome

Boosting

To perform tree-based boosting a *GradientBoostingRegressor* is used and the process as described for the single tree is repeated. The results of are summarised in table H.1.

The first thing that strikes is the variety of scores for models based on the dataset without history. The same trend as before is visible, where including the historical target boosts the scores from H0 to H1 and on, while no improvement is visible when the historical target is left out of scope. When the historical target is included the scores are much better with even MSE scores below 10. For the scores of data with historical target see appendix H. From previous tree-based models we learned that a good score does not necessarily tell us which predictions are closest. Figure 6.8 displays the actual versus predicted target for the best scoring model, H0 with the random technique. From the scatter plot it is not very clear that the performance is better than from the other trees methods we have seen before. However, the line plot really displays how much better this models fits than the ones we have seen before. There are no levels visible and also does the line not only seems to follow the trends, but also on the desired height. The good thing about the gradient-boosting models is that there maximum depth is smaller than we saw for the random forests and decision trees. This ensures that the models are less complex which is definitely a benefit.

In conclusion, all hyperparameter techniques perform roughly equal again. Furthermore the historical data also does not seem to greatly impact the scores. However, we found that the gradient-boosting model with random technique yield a slightly better score for dataset H1 than it does for dataset H0.

	First-Parallel	**First-Increment**	**Maximum-Parallel**	**Maximum-Increment**	**Random**
H0	116.58	116.58	95.23	120.31	96.40
	0.96	0.96	0.97	0.96	0.97
	167.56	123.88	119.58	110.77	88.52
	0.94	0.96	0.96	0.96	0.97
H1	116.32	143.35	109.77	120.32	101.59
	0.96	0.95	0.96	0.96	0.96
	111.46	111.46	96.20	97.89	90.53
	0.96	0.96	0.97	0.97	0.97
H2	104.25	116.44	134.11	116.65	184.69
	0.96	0.96	0.95	0.96	0.94
	113.67	100.82	103.74	377.46	527.32
	0.96	0.97	0.96	0.87	0.82
H3	121.09	153.50	104.54	108.68	109.81
	0.96	0.95	0.96	0.96	0.96
	118.61	109.70	114.09	109.12	99.37
	0.96	0.96	0.96	0.96	0.97
H4	127.90	134.52	113.07	115.82	189.98
	0.96	0.95	0.96	0.96	0.94
	159.63	122.70	119.34	483.71	111.42
	0.95	0.96	0.96	0.84	0.96
H5	126.77	123.05	127.22	90.62	201.91
	0.96	0.96	0.96	0.97	0.93
	237.84	122.11	129.27	104.61	92.58
	0.92	0.96	0.96	0.97	0.97

All features
Three selected features [F_1, F_2, F_10]

Table 6.8: [*Boosting trees*] scores (upper MSE, lower R2)

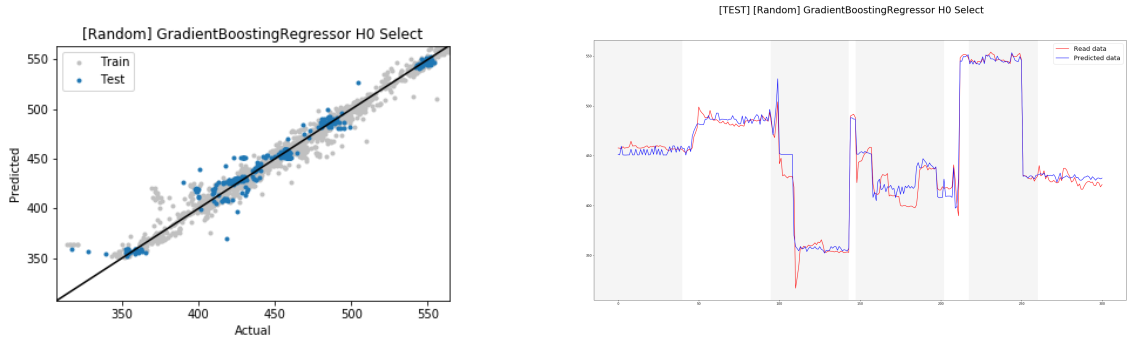


Figure 6.8: [*Boosting trees*] actual versus predicted outcome

Hyperparameter tuning technique and dataset selection

The tree-based results as presented in this section do not show a clear best practice. What mostly stands out is the fact that the minimum samples per leaf of 50 is the limiting factor for all tree-based methods. To ensure that our threshold was not to strict we also tested to a threshold of 0.5% of the samples which is roughly 14 samples per leaf. We found that the results for the random forest slightly improve, but at the cost of only 14 samples per leaf. While for gradient-boosting we a worse score with as many as 41 samples per leaf. In other words, we do not think that the 50 samples per leaf affected our results.

With regards to the hyperparameter tuning technique we oftentimes see that the differences between scores obtained by the parallel and the incremental technique were small. As the first and maximum techniques do gather different insights it makes sense to eliminate either the parallel or the incremental way. We decided to remove the incremental techniques as these cannot be searched in parallel and therefore order of hyperparameters can influence results to a large extent. Furthermore, the incremental techniques cannot be run in parallel which makes them more time consuming. Besides the first-parallel and the maximum-parallel technique the random technique is kept. In this way there still is an element of incremental and the technique leads to refreshing insights which can be useful.

With regards to an optimal history size the image is more clear. The step from no history to a history of one gives the biggest score improvement. Together with the fact that the lower the history size the more data is kept, the optimal history size is one. Besides that, also the dataset without history is kept which will be used as ground-truth.

In conclusion, for the other non-linear models in this research we only apply the first-parallel, maximumparallel and the random technique to the datasets without history and with one row of history.

	H0
Train	47.55 0.98
Test	88.52 0.97
Validation	48.57 0.98

Table 6.9: [*Trees*] best model scores (upper MSE, lower R2)

Best model

The best performing tree-based model obtained was the gradient boosting model with dataset H0 based on a random hyperparameter tuning technique. For this reason we want to gain more insight into this model. First, we start by displaying not only the test scores, as seen before, but also the scores obtained for training and validation. The scores can be found in table 6.9. What strikes is that the test score obtained is quite a bit lower than the scores obtained for training and validation. We also applied the random tuning technique to another dataset that was prepared in the same way. Here we got even more promising results with an MSE for training, testing and validation of respectively 3.64, 11.61, and 7.65. To see whether the model actually has a good fit we also made the actual versus predicted plots for the validation set. These can be found in figure 6.9. We see that the predictions mostly follow the line, however there are also a number of data points that are quite far from the line. But as the line plot nicely shows, mostly the trends and heights are predicted quite good.

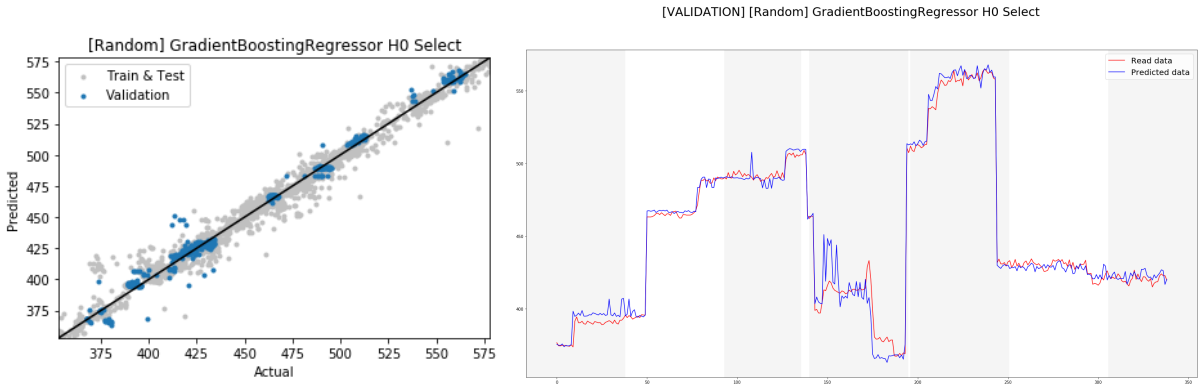


Figure 6.9: [*Trees*] best model: actual versus predicted outcome

To gain more insight into our selected gradient-boosting model a contour plot is created. Normally a contour plot is created for models with only two features, however in this case our model consists of three features. To be able to visualise it anyway, we use a constant value for one of the features. In figure 6.10 feature two is set to a constant value of 0.0, while the other two features vary over their whole range. In the left image the colour of the dots is as they should be with a correct prediction, while the background colour in the right plot displays the colours as the model would predict. The contour plots with variating constant values and variating features can be found in appendix I. However, the image as on the left and the datapoints are not included in those figures as normally the chances are low that datapoints actually have the exact value of the constant.

The tree-based model seems to be able to cope with the upper and lower cluster, however it struggles with the middle one. However, one has to take into account that this image only displays one slice of the whole model and the image only contains a fraction of all data points. In the previous figures we often saw that models have difficulties coping with the boundary values. As 0.0 is a lower boundary value for feature two, it is important to note that a lack of fit on this image does not mean that the model does not perform well. For the purpose of saying anything about the performance of the model it is better to refer back to the MSE and R^2-scores as stated in table 6.9.

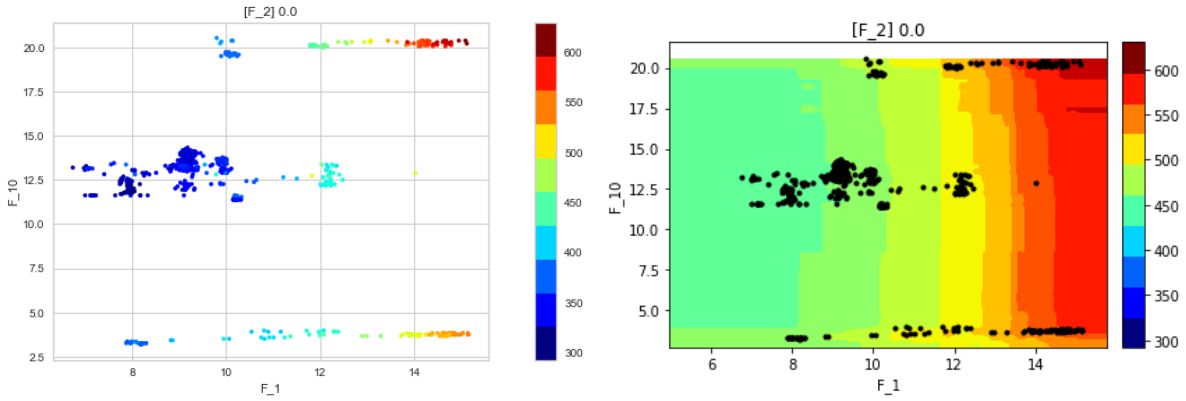


Figure 6.10: [*Boosting trees*] contour plots

6.4 Support vector regression

Support vector regression is, besides neural networks, the model most often used in related work for the creation of an emission model. Whereas other studies often combined support vector regression with least-squares and genetic algorithms, we test support vector regression in this research with a grid search. The implementation of the SVR as used in this research can be found in the method section and results will be discussed thereafter.

6.4.1 Methods

Whether a support vector regressor (SVR) is linear or non-linear depends on the kernel. As established by the tree-based methods, we test with the datasets H0 and H1 and the hyperparameter tuning techniques that are done in parallel and at random. Furthermore, every model is done with all features as well as the selected features. For the implementation of a SVR we utilise the implementation from sklearn [56]. Just as with linear regression, the features need to be scaled. The scaling is done again with a *StandardScaler*.

C	2^x with x: min=-4, max=7, step=1
Epsilon (ε)	min=0.025, max=0.5, step=0.025
Degree	min=1, max=10, step=1

Table 6.10: SVR hyperparameters

In table 6.10 the possible hyperparameter values are stated. However, the kernel, also a hyperparameter of a SVR, is not mentioned in this table. This is due to the fact that we use the kernels to distinguish between the different SVR models. The linear kernel is tested alongside three non-linear kernels, namely the sigmoid, polynomial and RBF kernel. For all these kernels we set two other hyperparameters: C and ε. The first mentioned, C, is a regularisation parameter. A good C value can be found by searching an exponentially growing series [67]. However, the larger the value of C the more likely one is to overfit the model. The standard value for C in the sklearn implementation is one, but oftentimes grids are tested with values up to 1000. In this research the maximum value is restricted at 128, while also testing multiple values below one. The values are established by a power series with two as ground number. The epsilon is known as the error sensitivity parameter and determines the number of support vector in a model. This parameter can have any number bigger than zero. In this case a lower value yields to a higher chance of overfitting. The standard value as given by sklearn is 0.1. In this case the values are 0.025 higher each times and they range till 0.5. Besides C and epsilon there is also a hyperparameter called degree mentioned in the table. This hyperparameter is only taken into account for the SVRs with

a polynomial kernel. It specifies the degree of the polynomial kernel function. The standard value of sklearn is three, but in this research the grid ranches between one and 10.

6.4.2 Results

	First-Parallel	Maximum-Parallel	Random
H0	305.52	305.03	302.36
	0.89	0.89	0.89
	280.03	280.03	304.21
	0.90	0.90	0.89
H1	304.21	304.21	304.46
	0.89	0.89	0.89
	280.83	281.15	280.83
	0.90	0.90	0.90

All features
Three selected features [F_1, F_2, F_10]

(a) Linear

	First-Parallel	Maximum-Parallel	Random
H0	645.92	643.76	644.08
	0.78	0.78	0.78
	439.27	438.64	438.54
	0.85	0.85	0.85
H1	644.38	642.37	642.37
	0.78	0.78	0.78
	433.28	431.01	432.09
	0.85	0.85	0.85

All features
Three selected features [F_1, F_2, F_10]

(b) Sigmoid

Table 6.11: [*SVR*] scores for linear and sigmoid kernel (upper MSE, lower R2)

The results of SVR with a linear kernel can be found in table 6.11a. The results for both H0 and H1 are comparable. Furthermore, it strikes that for all hyperparameter tuning techniques the models with a limited number of features perform better. Although not present in the tables, we see a same trend as we saw for the linear regression. Whenever the historical target is included into dataset H1, the MSE has a perfect MSE score of 0.0. However, when looking into the importance of the historical target, we also see here that the model with H1 relies heavily on this historical target. As we concluded in our linear regression section, this regression method does not seem the best fit for our research. Therefore, SVRs with linear kernels are not discussed any further in this research.
The scores obtained for the SVRs with sigmoid kernel can be found in table 6.11b. The sigmoid kernel retrieves higher MSE scores than the ones we have seen for the linear kernel. However, just like with the linear kernel, we see that the models with a selected number of features performs better than the ones with all features included. The results from this model are quite moderate and also the different plots do not seem to indicate new insight. For this reason the SVR with sigmoid kernels are not discussed any further.

	First-Parallel	Maximum-Parallel	Random
H0	342.46	344.10	302.81
	0.88	0.88	0.90
	331.88	331.59	280.55
	0.89	0.89	0.90
H1	348.03	348.91	304.44
	0.88	0.88	0.89
	339.65	339.69	281.44
	0.88	0.88	0.90

All features
Three selected features [F_1, F_2, F_10]

(a) Polynomial

	First-Parallel	Maximum-Parallel	Random
H0	1625.70	126.00	125.99
	0.43	0.96	0.96
	1588.86	111.48	111.36
	0.45	0.96	0.96
H1	1668.27	130.46	141.02
	0.42	0.95	0.95
	1631.48	112.46	108.03
	0.44	0.96	0.96

All features
Three selected features [F_1, F_2, F_10]

(b) RBF

Table 6.12: [*SVR*] scores for polynomial and RBF kernel (upper MSE, lower R2)

Our next kernel tested is the polynomial one. Scores obtained can be found in table 6.12a. The SVRs

with polynomial kernels based on H0 and H1 retrieve comparable scores for all features as it does for the limited number of features. However, also the results of this kernel do not show something new and the results are quite average. For this reason also results from these models are not highlighted any further.

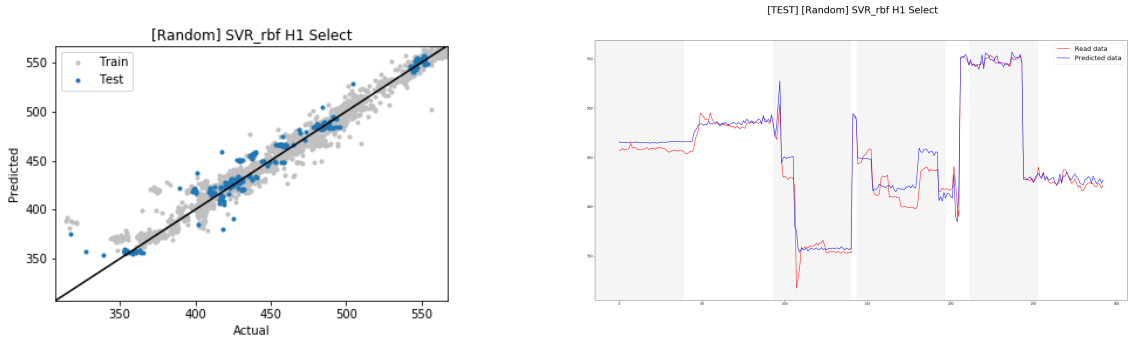


Figure 6.11: [*SVR RBF*] actual versus predicted outcome

The last kernel tested is the RBF kernel. Results of this kernel can be found in table 6.12b. For both the datasets the first-parallel hyperparameter tuning technique seems to struggle with finding proper hyperparameter values. If dataset H1 would include the historical target, this kernel performs really well with MSE score between one and two. All scores for the datasets with historical target can be found in appendix H. However, in the current situation the retrieved scores by the maximum and random hyperparameter tuning technique are still good. The best score is obtained by the model trained on H1 with the random hyperparameter tuning technique. In figure 6.11 we see the actual versus predicted plot from this model. Although this model seems to struggle with the boundary cases, the line plot indicates that the height is less of a problem for this model. Although the SVR with RBF kernel based on H1 achieved the best score, the same SVR based on H0 was not that far off.

In conclusion, the random hyperparameter tuning technique seems to perform best. In almost all scenarios this was this technique resulted in the best performing models. Besides that, the score does not seem to improve from the usage of another dataset. As the results of SVRs with RBF kernel of H0 and H1 with random hypertuning technique were really close, both will be considered as best model. Based on their performance on the validation set one of this models is chosen to give more in-depth insight into.

Best Model

Normally one best model is selected to gain deeper insight knowledge in. However, the results of the SVR-rbf with random technique based on H0 and H1 both get good results. The model based on H1 performs slightly better, however the model based on H0 is less complex as it has less feature inputs. The model based on H0 only consists of three features, while the one based on H1 consists of six. For

	H0	**H1**
Train	96.73 0.97	83.06 0.97
Test	111.36 0.96	108.03 0.96
Validation	35.40 0.99	36.52 0.99

Table 6.13: [*SVR*] best model scores (upper MSE, lower R2)

this reason we calculated the train, test and validation scores here for both of them. The results can be found in table 6.13. As we can see the model based on H1 indeed retrieves a slightly better score for the test set, however the model based on H0 retrieves a slightly higher score on the validation set. As the difference are negligible, we prefer the simpler model. Therefore, the best SVR model is the one with an RBF kernel and hyperparameters based on the random technique and the use of dataset H0. We also applied the random technique to another dataset that was prepared in the same way. Also here we got promising results with an MSE for training, testing and validation of respectively 23.88, 155.68 and 20.17.

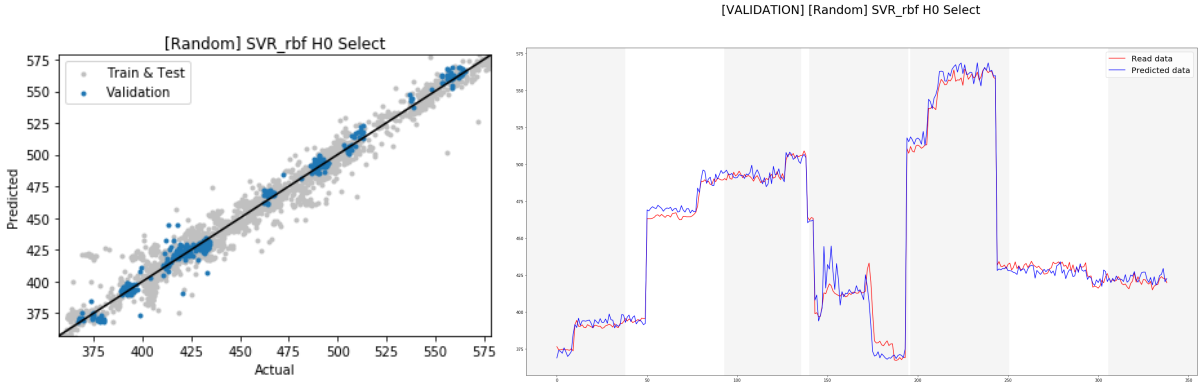


Figure 6.12: [*SVR*] best model: actual versus predicted outcome

The next step is to get the actual versus predicted plot for the validation set. The outcome can be found in figure 6.12. From this figure it becomes clear that the SVR model is also able to cope with the lower and higher values in the range. Also the line plot on the right nicely shows how trends are followed. To say something about the predictive power of this model in a broader sense also the contour plots are printed. With each time one feature as slice, while the other features have values all over the range. All contour plot figures can be found in appendix I. In figure 6.13 we see the slice where feature two is set to 0.00 and the other are variable. On the left the datapoints have the colours as desired, while on the right the background color indicates the identification as the SVR does. The model seems to be able to cope with the high emission values, however its the low emission values where the model struggles. This is especially clearly visible by the lack of dark blue in the contour plot. However with regards to the upper and lower cluster the SVR seems to follow the trends quite all right. However, it is important to keep in mind that the training points visualised here, are only a small part of the dataset. Value 0.0 for feature two is a boundary value and therefore it makes sense that the model would struggle with the identification. However, as relatively many data points had this value it seemed like a good slice to highlight.

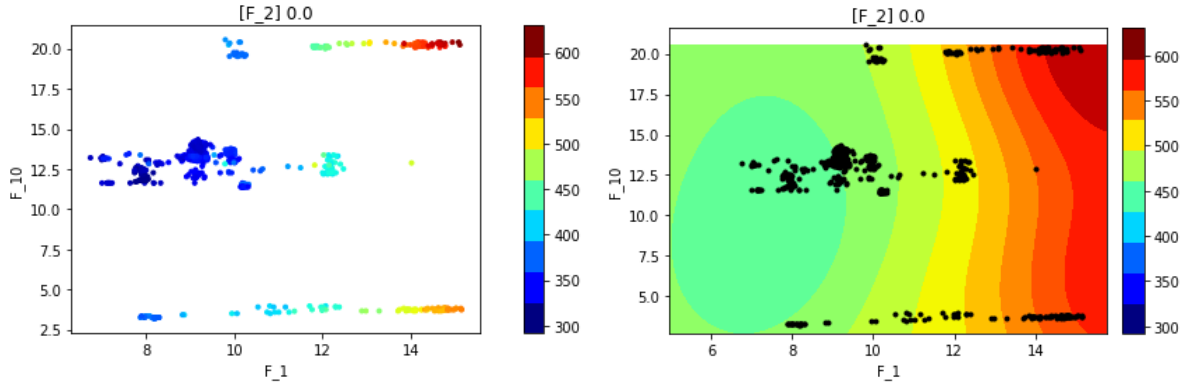


Figure 6.13: [*SVR*] contour plot of best SVR model with F_2 is 0.0

6.5 Neural Networks

Neural networks are used by the current software to create the emission models. The hyperparameter tuning techniques as we have seen before could not be applied to the neural networks, therefore a new way of finding the appropriate hyperparameters is introduced in section 6.5.1. Furthermore, this section describes how we divide the neural networks in different categories. Then in section 6.5.2 the results for the numerous methods will be discussed.

6.5.1 Method

Also the neural networks benefit from the scaling of the features. Therefore, the features are once again scaled with the aid of sklearns' *StandardScaler* [56]. Also the possible hyperparameters have to be defined, before the models can be trained. For the SVR and tree-based methods we used sklearn supported search methods, however as the neural networks are build with keras [68] we will use the kerastuner [69]. The kerastuner also has the ability to apply a random search, so basically the hyperparameters are found by the random technique as we have seen before. The scoring is still done with the aid of MSE. However, also the number of epochs had to be set. The random search has a limit of 200 epochs, however whenever there is no progress in the last ten rounds the algorithm also aborts. The kerastuner is then used to establish the values in their range for each of the in table 6.14 mentioned hyperparameters. The activation function and the number of units both are applied to several layers of the neural networks, whereas the other hyperparameters are to tune the Adam optimiser. Whenever the hyperparameters are set, it is also important to set the number of epochs and the batch size. Based on some initial testing we set the batch size to 50 and the number of epochs to 200. Also in this case the number of epochs actually executed can be smaller as an early stopping mechanism is implemented again. Oftentimes the batch size is set to 32, 64, 128 and so on [70]. Testing with multiple batch sizes between 4 and 128 learned us that between 40 and 60 samples per batch gives us good results. As the interval is set to 50 for this research, we decided to set the batch size to 50.

In this research we test two different types of neural networks. First, we test a normal artificial neural network (ANN) and secondly a long-term short memory (LSTM) network. We haven chosen only to implement an LSTM and not a recurrent neural network (RNN) as RNNs are known for their vanishing gradient problem. For both types of neural networks, ANN and LSTM, distinctions will be made based on the number of hidden layers that they consists of. The settings of the ANNs and LSTMs are kept as similar as possible, so it is easy to compare them. We will test up to three hidden layers as we see that related work usually has neural networks with one or two hidden layers [57, 71] and research shows that four hidden layers are rarely used [72]. First a *Simple* model is build for the both of them. This model does not rely on the keras tuner and simply consists of an input layer, output layer and the Adam

Activation function	linear, sigmoid, tanh, relu
Number of units	min=1, max=50, step=1
Learning rate	min=0.0001, max=0.005, step=0.0001
Beta 1	min=0.80, max=1.00, step=0.01
Beta 2	min=0.990, max=1.000, step=0.001
Epsilon	min=1e-6, max=1e-8, step=1e-8
AMSgrad	True, False

Table 6.14: Neural network hyperparameters

optimiser with standard settings. In case of the ANN this input layer consists of a dense layer with the amount of features as input, while this dense layer is a LSTM laer in the LSTM models. Then the *None* model stands for the fact that there is no hidden layer. Basically the model is similar to the *Simple*, however in this case different hyperparameters are tested. The number of inputs is unchanged for the ANN while the number of units in the LSTM input layer can differ between one and 50. Furthermore, for both types multiple activation functions are tested and the various settings for the optimiser, as written in table 6.14, are tested. Our next model is *One*, the settings for the input layer, output layer and optimiser are similar as we have seen before. However, in this case a hidden layer is added between the input and output layer. For both types of neural nets, this is a Dense layer that has between one and 50 units and one of the optional activation functions. Furthermore, we also have the models *Two* and *Three*. As one might guess these models consists of two and three hidden layers. All these layers are established the same way as the first hidden layer initialised. To make sure the results are not just a lucky strike, all the models are cross-validated five times. Just like we established for the tree-methods, the ANN is executed for H0 and H1. However, as the LSTM is especially drafted for historical data, these models are not only build on H0 and H1, but also on H2.

6.5.2 Results

This section will first discuss the results as obtained for the normal neural networks, ANNs. Next, it also discussed the results as found for the LSTMs. As seen before a best model is also highlighted for the neural networks. Lastly, the results section contains the scores obtained by the CEM software as this is currently used by Emission Care.

ANN

In table 6.15 the scores can be found for the ANNs. It immediately strikes that the scores obtained are lower than the ones we have seen before for the other models. However, how the different datasets and feature selections score in relation to each other is comparable with what we have seen before. The

	Simple	**None**	**One**	**Two**	**Three**
H0	640.26	742.55	416.72	589.47	376.10
	0.78	0.74	0.85	0.79	0.87
	434.03	435.20	461.02	338.75	366.35
	0.85	0.85	0.84	0.88	0.87
H1	673.22	801.61	954.37	462.76	535.24
	0.77	0.72	0.67	0.84	0.82
	429.67	469.98	746.73	470.43	424.94
	0.85	0.84	0.74	0.84	0.85

All features
Three selected features [F_1, F_2, F_10]

Table 6.15: [*ANN*] scores (upper MSE, lower R2)

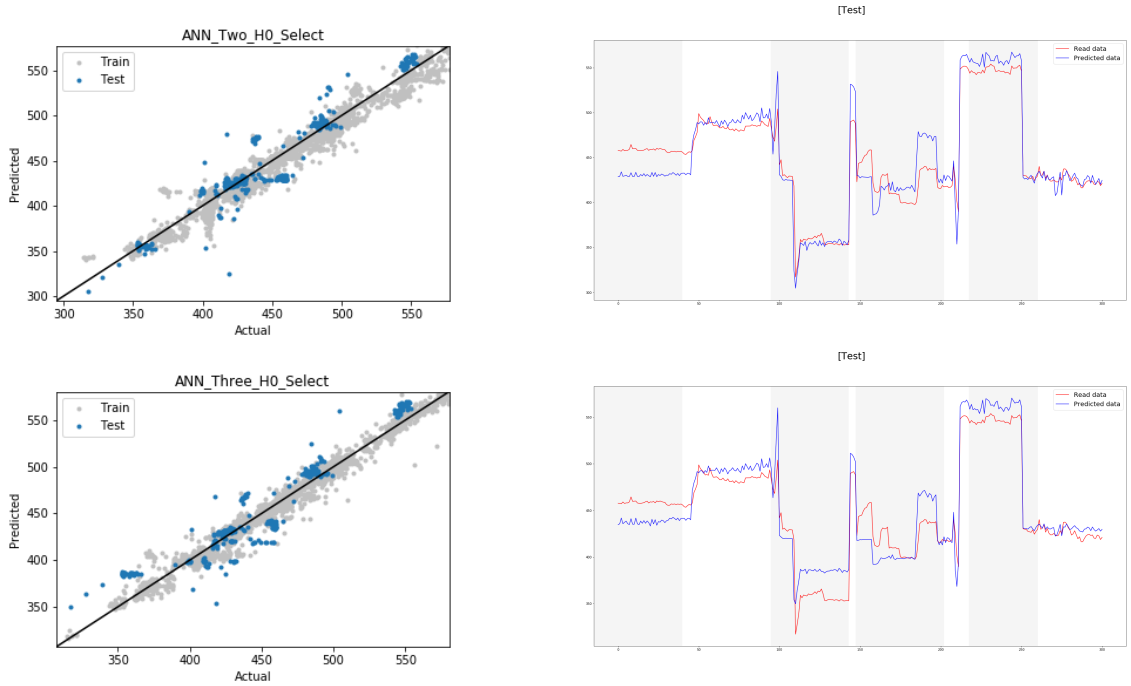


Figure 6.14: [*ANN*] actual versus predicted outcome

models *Two* and *Three* seem to be the only models with a MSE score below 400. To retrieve more insight into their performance their actual versus predicted plots can be found in figure 6.14. Most of the datapoints seem to dance around the line, however compared to the models seen before the range from the line seems a bit bigger. This is also what strikes if we look at the line representation of the actual versus predicted plot.

	Simple	**None**	**One**	**Two**	**Three**
H0	550.91	403.55	521.36	419.71	366.11
	0.81	0.86	0.82	0.85	0.87
	377.71	343.01	369.75	386.63	290.24
	0.87	0.88	0.87	0.87	0.90
H1	5759.36	5805.44	5637.02	5632.62	5891.43
	-0.99	-1.00	-0.95	-0.94	-1.03
	5298.35	5793.55	6261.98	5755.09	5728.40
	-0.83	-1.00	-1.16	-0.99	-0.98

All features
Three selected features [F_1, F_2, F_10]

Table 6.16: [*LSTM*] scores (upper MSE, lower R2)

LSTM

The scores retrieved for the LSTMs can be found in table 6.16. The LSTM models based on H0 are comparable to the ANNs models based on H0, the only difference is in the input layer. However, the LSTMs based on H0 seem to perform slightly better. But when we look at the models based on H1 we see that the LSTMs perform poorly. To gain more insight we have into this, we printed the actual versus predicted plot of model *Three* from both H0 and H1, the figures can be found in figure 6.15. The two

upper figures represent the performance of model *Three* based on H0. We see that in the training set the trends are followed quite good, but we already see quite some variation from the desired situation in the test set. Also model *Three* based on H1 seems to have no problem to handle the training data. However, when we look at the testing part, we see really big difference. It seems like peaks are predicted too late or not even at all. Especially since the historical target is included in these models, one would expect better results.

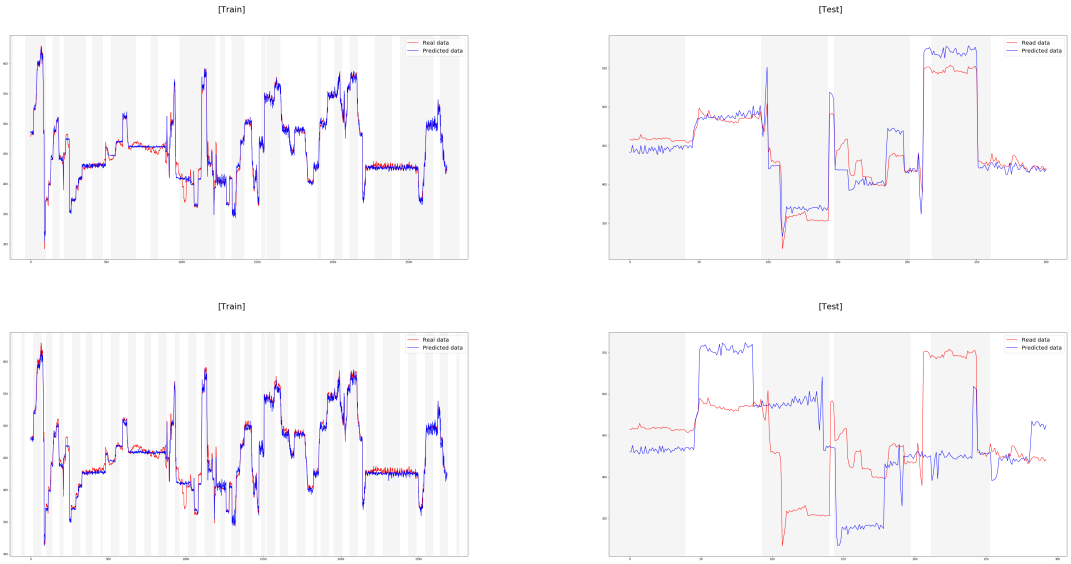


Figure 6.15: [*LSTM*] actual versus predicted outcome

In conclusion, all neural networks tested perform less than we expected beforehand. None of the models is able to predict with a certainty great enough that it could actually be put to use. As both the ANNs and LSTMs obtain much better scores for training than they do for testing, it seems like the models are overfitted. However, this is just a theory for now. As the results of the LSTMs are poorly and we saw from previous models with the historical emission included that they heavily depended on that, we decided to select ANN model *Two* as our best model.

Best model

	H0
Train	132.88 0.96
Test	338.75 0.88
Validation	208.35 0.93

Table 6.17: [*NN*] best model scores (upper MSE, lower R2)

ANN model *Two* is our best model, although the scores are not promising. However, we still found it important to give insight into the neural networks as well. To give more insight the train, test and validation score can be found in table 6.17. Although the scores are not bad, we have definitely seen better before. The same goes for the actual versus predicted plot of the validation set as shown in figure 6.16.

There are still quite a lot of data points that have quite some deviation from the desired line. The same is displayed by the line plot of the actual versus predicted plot. We also applied the kerastuner to model Two for another dataset that was prepared in the same way. Also here we got promising results with an MSE for training, testing and validation of respectively 7.11, 35.75 and 8.01.

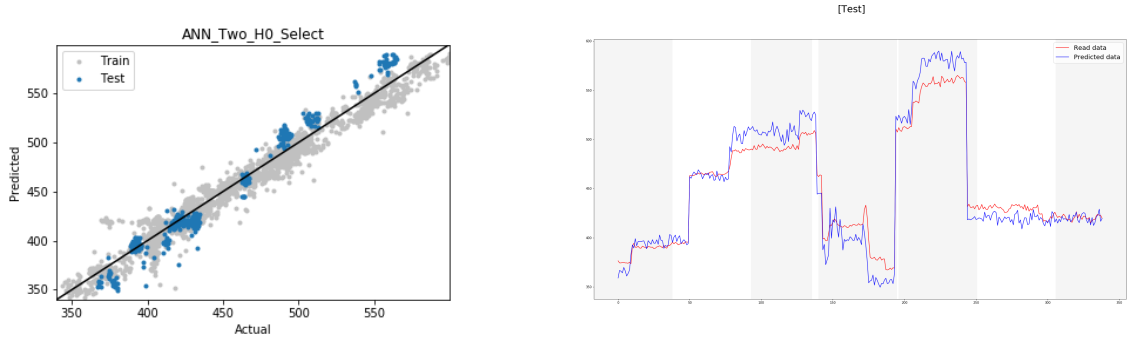


Figure 6.16: [*NN*] best model: actual versus predicted outcome

Next we also made a contour plot for this neural network. The result of this can be found in figure 6.17. Once again we see that this model also fails to predict the lower emission values. Whereas the gradient as we see in the lower and upper cluster seems to be present in the model. However, as said this is just one slice. The fact that the model performs well on this particular slice, does not say anything about its overall performance.

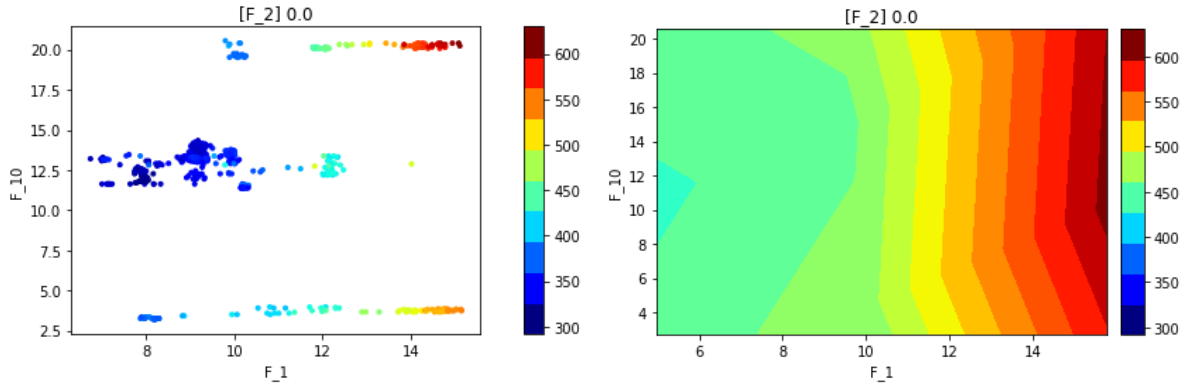


Figure 6.17: [*NN*] contour plot of best neural network with F_2 is 0.0

CEM software

The CEM software currently used by Emission Care to model the emission are also neural network based. To see how this software performs compares to the models implemented by us, the scores are also included in this section. With regards to the original dataset we see that Emission Care developed an emission model with four features from which one combined, whereas our emission model only has three features. If we compare the R^2-scores, respectively 0.970 and 0.971, we see that these are roughly equal. This same trend is observed for the validation dataset. For this dataset the current model has a selection of five features of which one composed versus only two features in our selection. And again the R^2-scores are quite close to another with respectively 0.973 and 0.985. What strikes, however, is the fact that the neural networks in the CEM software achieve higher scores for the first dataset.

7 Discussion

Normally one would discuss the results in this section, but much of the discussion is also already done in the previous chapter as the methodology was adapted based on results found. However, in this chapter we will focus more on the overall picture. We bring the separate parts together and also compare them to the related work. First the data preparation is discussed and thereafter the various machine learning models.

Data preparation

Much of the data pre-processing was already done by Emission Care. Although they also shared the raw data, we used the pre-processed data for this research because we lack the field knowledge to do it properly. However, we still had to cut the data in train, test, and validation sets and add the historical columns. Normally, the validation set consists of the last time series measured rather than testing the whole spectrum. However, in order to keep a scientific approach the train, test and validation were decided to cover the whole spectrum. As one cannot test a model outside the train spectrum, the train set therefore entails the most extreme values. The test set, on its turn, falls inside this range, and then the validation set falls inside this range. This design decision also seems to be visible in our models. The scores obtained for training are usually good as the model trains on it, however the test scores are usually worse, while the test scores are often on the same level as the train scores. However, given the intention to test most of the range, we still feel confident that this technique fits its purpose. This technique also resulted in the fact that the first measured time series is part of the validation set. As the time series component is normally ignored we feel that this should not cause any trouble.

Furthermore, the data still consisted of many features and a selection needed to be made. This selection is to a great extent made based on the correlation among the features. However, it is difficult to say whether this method holds for other datasets then the two tested in this research. In related work we see that PCA is used, which shrinks the amount of features by essentially merging them together. Essentially this keeps one still dependent on the same amount of sensors. However, we hoped to find PCA components where one or two features would be really important, highlighting their usability. Unfortunately, this was not the case. Another approach used by related research is the random forest, but here we found that the random forest struggles favouring features whenever the correlation between them is high. Furthermore, in this research we only focused on finding feature combinations based on linear and monotonic relationships between features, however it might be that better feature combinations could be suggested when also looking into other relationships between features. However, due to time constraints we were not able to test this theory.

Keeping the amount of input features low is really important as the uncertainty of a PEMS not only depends on the performance of the model, but also on the uncertainty of the sensors. In other words, by keeping the amount of features limited, the model uncertainty is kept low. This is why our focus was not only a good performance, but also a low number of features. Comparing our work to related work already proves to be difficult as each research uses data from other gas turbines, but is made even more complex due to number of input features used. In related work we oftentimes see large amount of features that are used as input. This might have retrieved better scores in their situation, although our research does not confirm this view, but makes their models unusable in practice. Comparing our feature selection to Emission Cares' selection also made us realise that too few features is also a pitfall as the model is than too dependent on only a few sensors. No further attention has been paid to this issue in this research, but when the algorithm is put to practice it is important to keep this given in mind.

Machine learning models

Although the linear regression models obtained quite reasonable scores, this models cannot be used by the simple reason that they do not satisfy linear regression most important assumptions, namely that

the data is linear. For this same reason the SVR with linear kernel is also not an option. Furthermore, we tried to imitate the time series for the models by inserting historical columns. However, the resulting models heavily relied on the historical target. This would mean that the emission model would base most of its prediction on another prediction. This would enable the emission model to drift off which is not desirable. Therefore, we decided to take the historical target out of the historical columns. The downside of this decision is that the other sensors barely contribute if their target is unknown and this is also what we see in the results for the tree-based methods, SVRs and neural networks.

The tree-based methods roughly obtained MSE scores between 80 and 190 for the test set. The MSE scores are even as low as 1 when we would take the historical target into account. What strikes is that the scores obtained for the decision trees are comparable to the ones obtained for the random forests. However looking into there actual versus predicted plots has learned us that decision trees lack the finesse for the details. The gradient boosting, on the other hand, is capable quite well of coping with the details as well. This is also shown from the validation set where a MSE score below 50 was obtained and is confirmed by its performance on the second dataset.

The SVRs have a lowest MSE score of 108, which makes the overall test scores of the SVR slightly higher than the ones we have seen for the tree-based methods. However, when we apply validation set to two of our best SVR models we see a MSE score below 40. Which would mean that the SVR actually outperforms our best tree-based model. On the second dataset, however, the gradient boosting model retrieves a better scoring. Also in related work SVR models often yield good results. A quick look at the results learns us that our results seem quite comparable, however not much can be concluded as there are too many differences between the studies.

The neural networks are a bit of a letdown. As many of the related studies use neural networks one expects the results to be, at least, at the same level as the tree-based methods and the SVRs. However, the neural networks clearly perform less with a lowest MSE of roughly 366 for both the LSTM and the ANN. The difference between the train and test score is much bigger for this model than for the others. This might suggests that the neural networks are overfitted which would explain their lack of performance. However, the implementation of all neural networks is also quite basic. So it might also be the lack of steering. When the same feature combinations are used as input in the CEM software, which is also based on neural networks, the scores obtained are comparable to those obtained by the gradient-boosting and SVR model. This would suggest indeed the the lack of performance is (partly) due to our implementation. Also one would expect good results for the LSTMs based on H1 and this models still have the historical target as an input. However, the scores decrease rather than increase, which makes us to believe that something went wrong in the implementation. The odd thing is that the results obtained with the ANN for the second dataset are very promising and they are certainly not inferior to the results of the gradient boosting and SVR model. However, as mentioned before, the second dataset is not examined really close.

The current contour plots display the whole range for the features, however normally certain feature value combinations are not possible as it would be chemically impossible. A mask could be applied to prevent confusion, however due to time-constraints this is not done in this research. The different natures of the tree-based, SVR and NN model are nicely displayed in the contour plots. Although the contour plots represent snapshots, the differences and similarties between the models are striking. The contour plots have a circle like elegance when it comes to the SVR model, while the NN model contains more straight lines. The tree-based model on the other hand, still seems to display some form of layers as we earlier saw in the actual versus predicted plots. However, apart from these characteristics the trends we observe seem to be roughly equal.

8 Conclusions

During this research we aimed to answer our main research question: How can data-driven techniques and machine learning be applied to support the creation of an emission model? However, before we can answer this question first the answers to our research questions need to be answered. While answering these questions we also discuss our achievements and future work.

> **RQ1:** How can feature selection be applied to support the creation of an emission model?

For the feature selection we built a algorithm based on a combination of RFE and correlation among features to come to a feature selection. As the algorithm lacks any knowledge with regards to the physics of the process or the legislation, the algorithm cannot be used to feed recommendations to the emission model on its own. However, as the algorithm provides insight into the data as well as suggestions with regards to feature combinations, this tool can be used as a supporting tool in the current process. The performance of the algorithm is tested on two datasets. In both datasets our algorithm came to a different feature selection then Emission Care did. However, when we train the emission model with the CEM software the scores obtained are roughly equal. This applies for both the datasets. This suggests that algorithm is capable in cutting back time needed to develop an emission model.

> **RQ2:** Based on existing literature, what machine learning techniques are good candidates to be used for an emission model with time series input data?

Linear regression, tree-based algorithms, support vector regression and neural networks were identified as possible emission models. With the exception of LSTMs all of the possible models are unable to deal with time series. For all other models the time element was added through the usage of history columns. Some first tests proved that using the historical target heavily influenced the performance of the model. As this would give the model the opportunity to drift, the historical target was removed from the history columns. The models based on data with the remaining history columns perform as well as the ones without any history.

> **RQ3:** Which machine learning technique is most suitable for an emission model in terms of uncertainty, robustness and maintainability?

The most suitable emission model is not necessarily the one that obtains the best scores, but one that performs well in terms of uncertainty, robustness and maintainability. In practice these three elements are heavily dependent on the number of input features the model incorporates. We found that the linear regression was not a good fit for an emission model. However, the other three techniques all got pretty decent results. The neural networks as implemented in this research slightly under-performs compared to the other methods. Whereas the gradient boosting and the support vector regression retrieve slightly better, comparable scores. In our opinion they could therefore both be interesting for building emission models. If we compare the scores of the SVR and tree-based to the current CEM software, which is based on neural networks, we see that the scores are roughly the same, around a R^2-score of 0.97.

> **M-RQ:** How can data-driven techniques and machine learning be applied to support the creation of an emission model?

The data-driven techniques are represented in the development of the feature selection algorithm. More testing is required to prove the usability of the developed algorithm, but the results from the two datasets applied in this research were very promising. If usability is proven this algorithm could be an ideal supporting tool to cut back the amount of time spent on finding feature combinations and building the corresponding emission models. In order to use the algorithm in practice the coming weeks an interface will be designed. Another activity for future research would be the identification of more complex relationships among the features and the target. The algorithm as-is only identifies monotomic and linear

relationships, while probably many other types of relationships are present among the features. Exposing and utilising these relationships would possibly lead to a models with better predictive power.

With regards to the creation of an emission model we found that introducing a time-sense in the various machine learning methods did not influence the results. However, for the models without time-sense we found that gradient-boosting and SVR were the most suitable candidates. The SVR is already present in a lot of related work, but we barely see the usage of tree-based algorithm. Given the strict regulation it is difficult to judge whether tree-based methods would be acceptable as the outcome is based on a set of questions rather than calculations. However, as the results are very promising the tree-based methods could use more attention in future work.

The feature selection algorithm as developed for this research can be put to use as soon as the interface is developed. With regards to the emission models we proved that adding a time-sense element to the data does not improve the performance. Furthermore, we were not able to beat the scores retrieved by the CEM software. In other words, the neural networks as currently used are still suitable to built PEMSs.

Bibliography

[1] Emission Care, "Internal powerpoint about pems," 2019.

[2] G. Vasconcelos, "How random forests improve simple regression trees?," Sep 2017.

[3] World Health Organization (WHO), "9 out of 10 people worldwide breathe polluted air, but more countries are taking action," 2018.

[4] European Environment Agency, "Nitrogen oxides (nox) emissions," March 2018.

[5] A. Molina, E. Eddings, D. Pershing, and A. Sarofim, "Char nitrogen conversion: implications to emissions from coal-fired utility boilers," *Progress in Energy and Combustion Science*, vol. 26, no. 4-6, pp. 507–531, 2000.

[6] K. Skalska, J. S. Miller, and S. Ledakowicz, "Trends in nox abatement: A review," *Science of the total environment*, vol. 408, no. 19, pp. 3976–3989, 2010.

[7] Rockwell Automation, "Environmental management."

[8] Bioenergy Europe, "Industrial emission directive (ied)."

[9] Emission Care, "Internal document on pems in europe," 2019.

[10] Emission Care, "Internal document on emission legislation in europe," 2019.

[11] Envirotech Online, "En14181 stationary source emissions - quality assurance of automated measuring systems."

[12] Emission Care, "Internal interview about current course of actions," Feb 2019.

[13] O. Antson and T. Pellikka, "Predictive emission monitoring systems, pems and their acceptance and use in europe,"

[14] G. Bonaccorso, *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[15] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*, vol. 821. John Wiley & Sons, 2012.

[16] D. C. Montgomery, C. L. Jennings, and M. Kulahci, *Introduction to time series analysis and forecasting*. John Wiley & Sons, 2015.

[17] P. Prakash and A. S. K. Rao, *R deep learning cookbook*. Packt Publishing Ltd, 2017.

[18] D. D. Gutierrez, *Machine learning and data science: an introduction to statistical learning methods with R*. Technics Publications, 2015.

[19] G. Bonaccorso, *Machine Learning Algorithms: Popular algorithms for data science and machine learning*. Packt Publishing Ltd, 2018.

[20] F. Chollet, *Deep Learning with Python*. Greenwich, CT, USA: Manning Publications Co., 1st ed., 2017.

[21] S. Zhang, C. Zhang, and Q. Yang, "Data preparation for data mining," *Applied artificial intelligence*, vol. 17, no. 5-6, pp. 375–381, 2003.

[22] J. Bai and S. Ng, "Tests for skewness, kurtosis, and normality for time series data," *Journal of Business & Economic Statistics*, vol. 23, no. 1, pp. 49–60, 2005.

[23] S. M. Goldfeld and R. E. Quandt, "Some tests for homoscedasticity," *Journal of the American statistical Association*, vol. 60, no. 310, pp. 539–547, 1965.

[24] *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data.* Wiley Publishing, 1st ed., 2015.

[25] T. Luminous, *Machine Learning For Beginners Guide Algorithms: Supervised & Unsupervsied Learning. Decision Tree & Random Forest Introduction.* William Sullivan, 2017.

[26] D. Steinberg and P. Colla, "Cart: classification and regression trees," *The top ten algorithms in data mining*, vol. 9, p. 179, 2009.

[27] M. Berry and G. Linoff, *Mastering data mining: The art and science of customer relationship management.* John Wiley & Sons, Inc., 1999.

[28] M. Awad and R. Khanna, *Efficient learning machines: theories, concepts, and applications for engineers and system designers.* Apress, 2015.

[29] A. Azadeh, A. Boskabadi, and S. Pashapour, "A unique support vector regression for improved modelling and forecasting of short-term gasoline consumption in railway systems," *International Journal of Services and Operations Management*, vol. 21, no. 2, pp. 217–237, 2015.

[30] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[32] S. K. Kumar, "On weight initialization in deep neural networks," *arXiv preprint arXiv:1704.08863*, 2017.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[34] V. Bushaev, "Adam-latest trends in deep learning optimization.," Oct 2018.

[35] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*, pp. 437–478, Springer, 2012.

[36] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472, IEEE, 2017.

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[38] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks.* Mit Press, 1999.

[39] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, pp. 950–957, 1992.

[40] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent dropout without memory loss," *arXiv preprint arXiv:1603.05118*, 2016.

[41] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. Courville, and C. Pal, "Zoneout: Regularizing rnns by randomly preserving hidden activations," *arXiv preprint arXiv:1606.01305*, 2016.

[42] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[43] H. Zhou, J. P. Zhao, L. G. Zheng, C. L. Wang, and K. F. Cen, "Modeling nox emissions from coal-fired utility boilers using support vector regression with ant colony optimization," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 1, pp. 147–158, 2012.

[44] T. Korpela, P. Kumpulainen, Y. Majanne, A. Häyrinen, and P. Lautala, "Indirect nox emission monitoring in natural gas fired boilers," *Control Engineering Practice*, vol. 65, pp. 11–25, 2017.

[45] Z. Tang, X. Wu, S. Cao, and M. Yang, "Modeling of the boiler nox emission with a data driven algorithm," *Journal of Chemical Engineering of Japan*, vol. 51, no. 8, pp. 695–703, 2018.

[46] G. Cuccu, S. Danafar, P. Cudré-Mauroux, M. Gassner, S. Bernero, and K. Kryszczuk, "A data-driven approach to predict nox-emissions of gas turbines," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1283–1288, IEEE, 2017.

[47] A. Lepore, M. S. dos Reis, B. Palumbo, R. Rendall, and C. Capezza, "A comparison of advanced regression techniques for predicting ship co2 emissions," *Quality and Reliability Engineering International*, vol. 33, no. 6, pp. 1281–1292, 2017.

[48] B. Liu, J. Hu, F. Yan, R. F. Turkson, and F. Lin, "A novel optimal support vector machine ensemble model for nox emissions prediction of a diesel engine," *Measurement*, vol. 92, pp. 183–192, 2016.

[49] G. Li, P. Niu, W. Zhang, and Y. Liu, "Model nox emissions by least squares support vector machine with tuning based on ameliorated teaching–learning-based optimization," *Chemometrics and Intelligent Laboratory Systems*, vol. 126, pp. 11–20, 2013.

[50] P. Tan, J. Xia, C. Zhang, Q. Fang, and G. Chen, "Modeling and optimization of nox emission in a coal-fired power plant using advanced machine learning methods," *Energy Procedia*, vol. 61, pp. 377–380, 2014.

[51] P. Tan, C. Zhang, J. Xia, Q. Fang, and G. Chen, "Nox emission model for coal-fired boilers using principle component analysis and support vector regression," *Journal of Chemical Engineering of Japan*, vol. 49, no. 2, pp. 211–216, 2018.

[52] Y.-l. Wang, Z.-y. Ma, H.-h. You, Y.-j. Tang, Y.-l. Shen, M.-j. Ni, Y. Chi, and J.-h. Yan, "Development of a no x emission model with seven optimized input parameters for a coal-fired boiler," *Journal of Zhejiang University-SCIENCE A*, vol. 19, no. 4, pp. 315–328, 2018.

[53] M. Azzam, M. Awad, and J. Zeaiter, "Application of evolutionary neural networks and support vector machines to model nox emissions from gas turbines," *Journal of environmental chemical engineering*, vol. 6, no. 1, pp. 1044–1052, 2018.

[54] J. A. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle, "Weighted least squares support vector machines: robustness and sparse approximation," *Neurocomputing*, vol. 48, no. 1-4, pp. 85–105, 2002.

[55] H. Wang and D. Hu, "Comparison of svm and ls-svm for regression," in *2005 International Conference on Neural Networks and Brain*, vol. 1, pp. 279–283, IEEE, 2005.

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

[57] G. Kibrya and K. Botros, "A neural network based predictive emission monitoring model for nox emission from a gas turbine combustor," in *ASME 2002 Engineering Technology Conference on Energy*, pp. 137–145, American Society of Mechanical Engineers, 2002.

[58] K. Botros, C. Williams-Gossen, S. Makwana, and L. Siarkowski, "Data from predictive emission modules implemented on ge-lm1600, ge-lm2500 and rr-rb211gas turbines employed in natural gas compressor stations," in *19th Symposium on Industrial Application of Gas Turbines (IAGT), Banff, Alberta, Canada*, Citeseer, 2011.

[59] I. Arsie, D. Marra, C. Pianese, and M. Sorrentino, "Real-time estimation of engine nox emissions via recurrent neural networks," *IFAC Proceedings Volumes*, vol. 43, no. 7, pp. 228–233, 2010.

[60] S. Kanarachos, J. Mathew, and M. E. Fitzpatrick, "Instantaneous vehicle fuel consumption estimation using smartphones and recurrent neural networks," *Expert Systems with Applications*, vol. 120, pp. 436–447, 2019.

[61] Q. Zhang, F. Li, F. Long, and Q. Ling, "Vehicle emission forecasting based on wavelet transform and long short-term memory network," *IEEE Access*, vol. 6, pp. 56984–56994, 2018.

[62] Y. Pan, S. Chen, F. Qiao, S. V. Ukkusuri, and K. Tang, "Estimation of real-driving emissions for buses fueled with liquefied natural gas based on gradient boosted regression trees," *Science of the Total Environment*, vol. 660, pp. 741–750, 2019.

[63] Rockwell Automation, *Model Reference Manual: Pavilion8 Volume I*, p. 180. 2018.

[64] T. Parr, K. Turgutlu, C. Csiszar, and J. Howard, "Beware default random forest importances."

[65] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[66] T. Srivastava, "Tune a random forest model's parameters for machine learning," Sep 2019.

[67] B. Boehmke and B. M. Greenwell, *Hands-On Machine Learning with R*. CRC Press, 2019.

[68] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

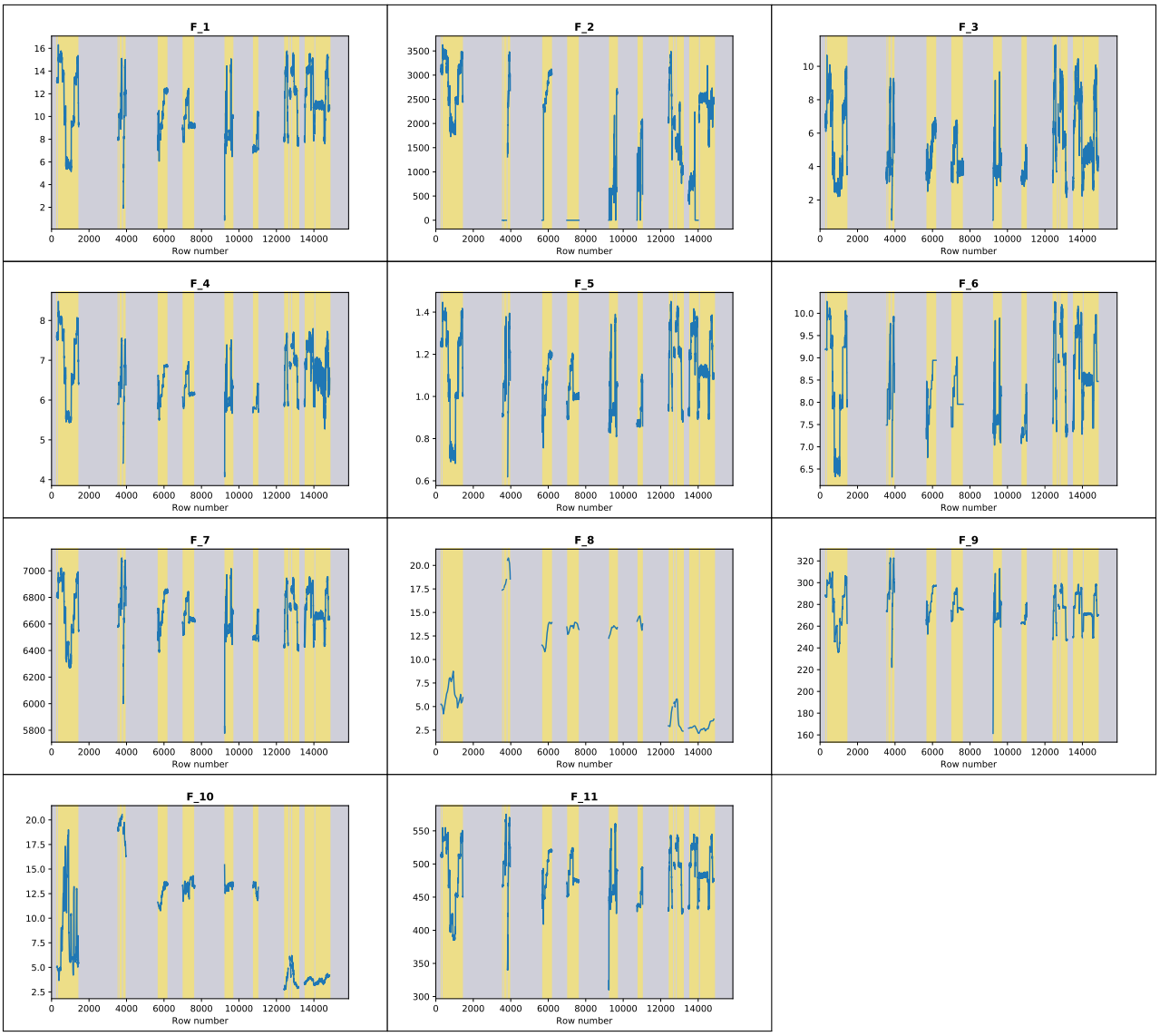
[69] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, "Keras Tuner." `https://github.com/keras-team/keras-tuner`, 2019.

[70] J. Rogel-Salazar, *Advanced Data Science and Analytics with Python*. CRC Press, 2020.

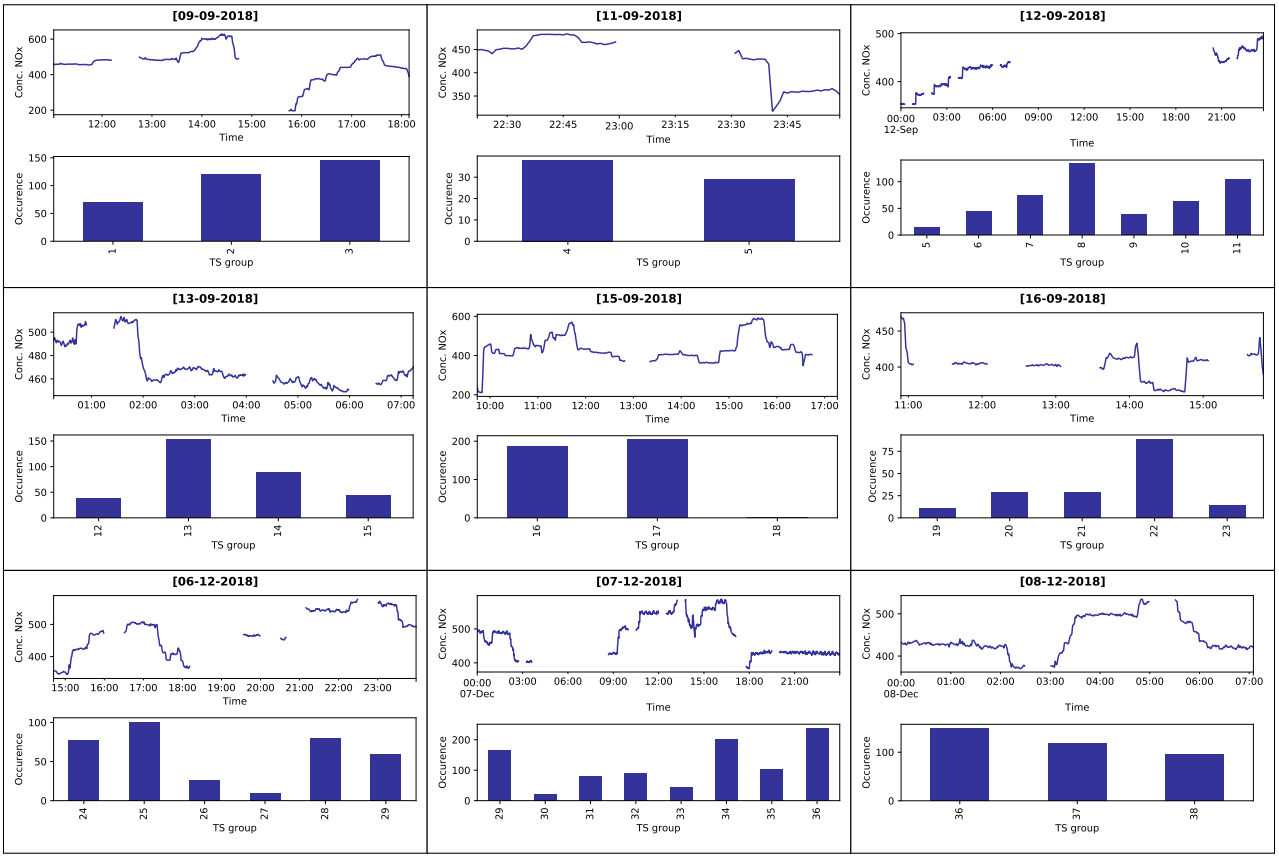
[71] S. M. Zain and K. K. Chua, "Development of a neural network predictive emission monitoring system for flue gas measurement," in *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*, pp. 314–317, IEEE, 2011.

[72] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural network design*. PWS Publishing Co., 1997.

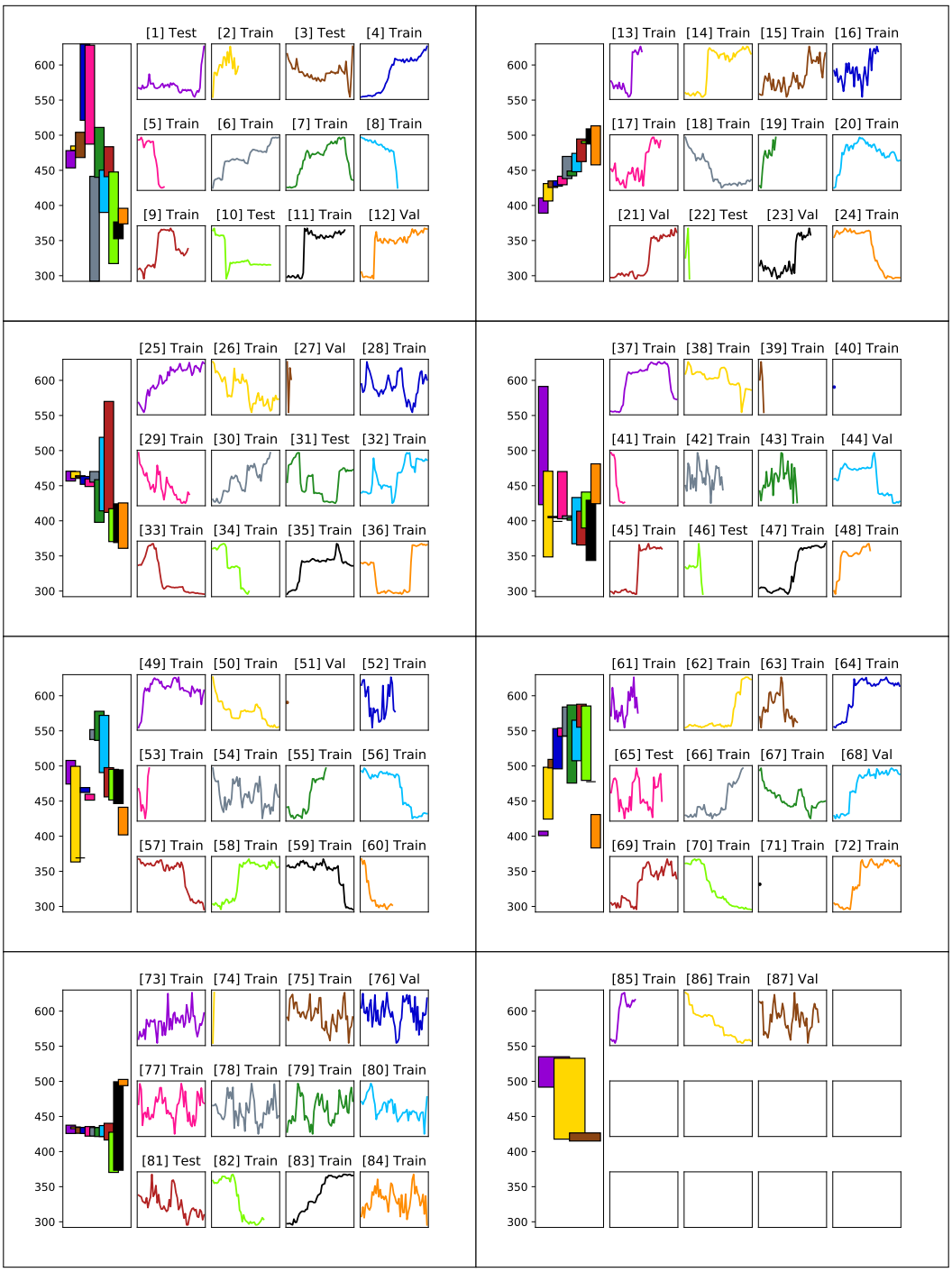
A | Feature Behaviour



B | NO_x concentration over time

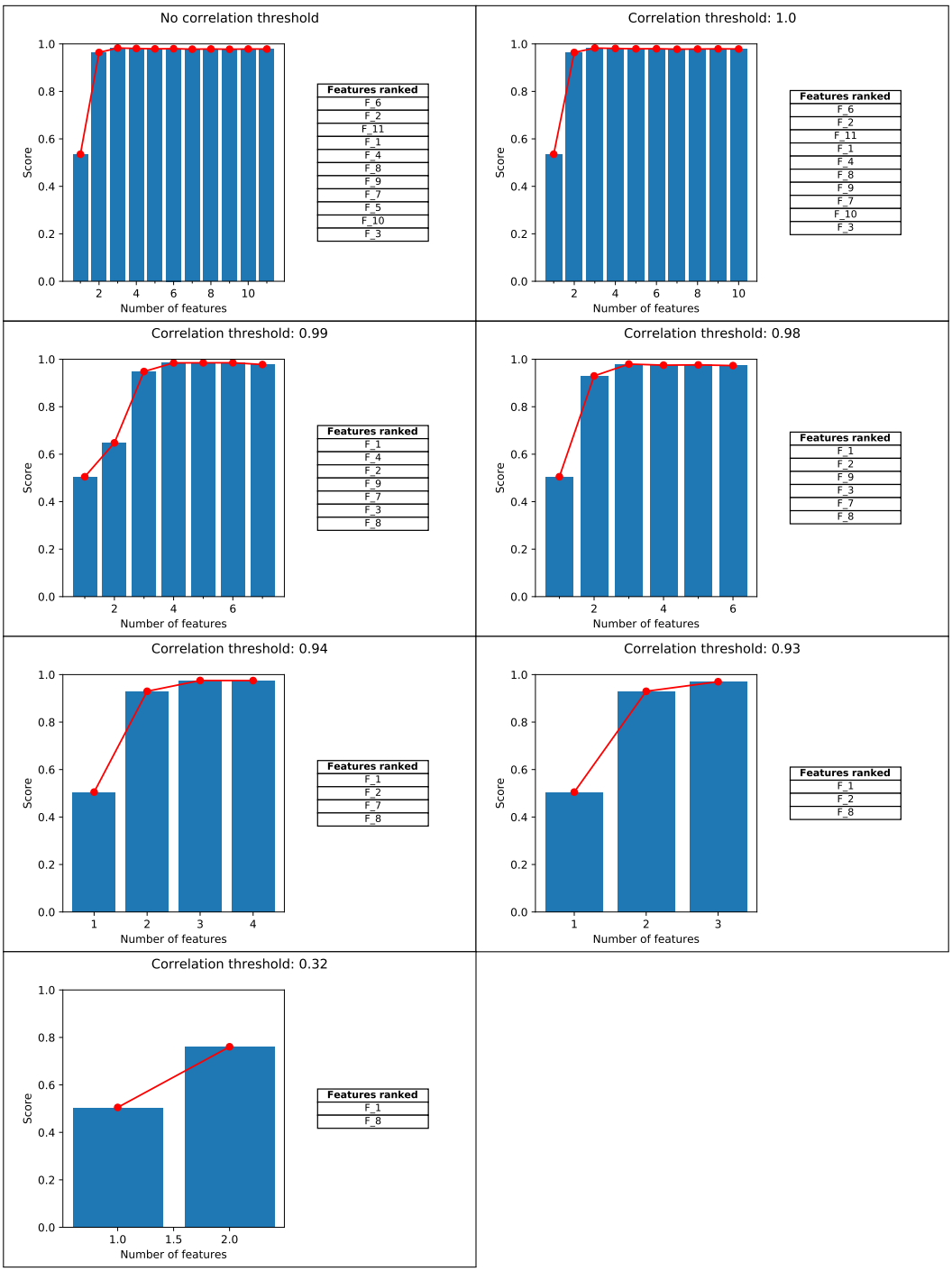


C | Distribution of intervals over the NO_x spectrum

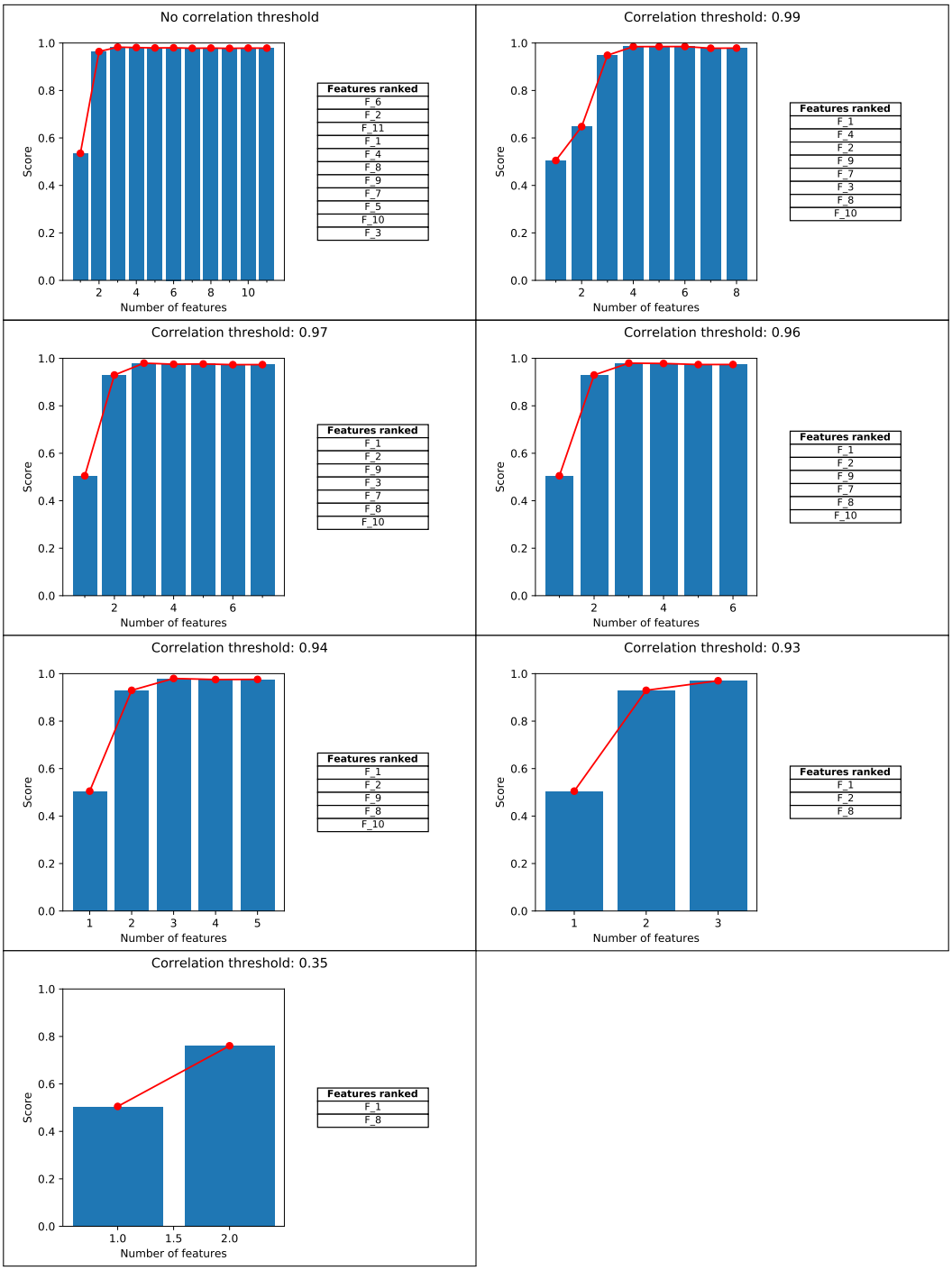


D Correlation based RFE

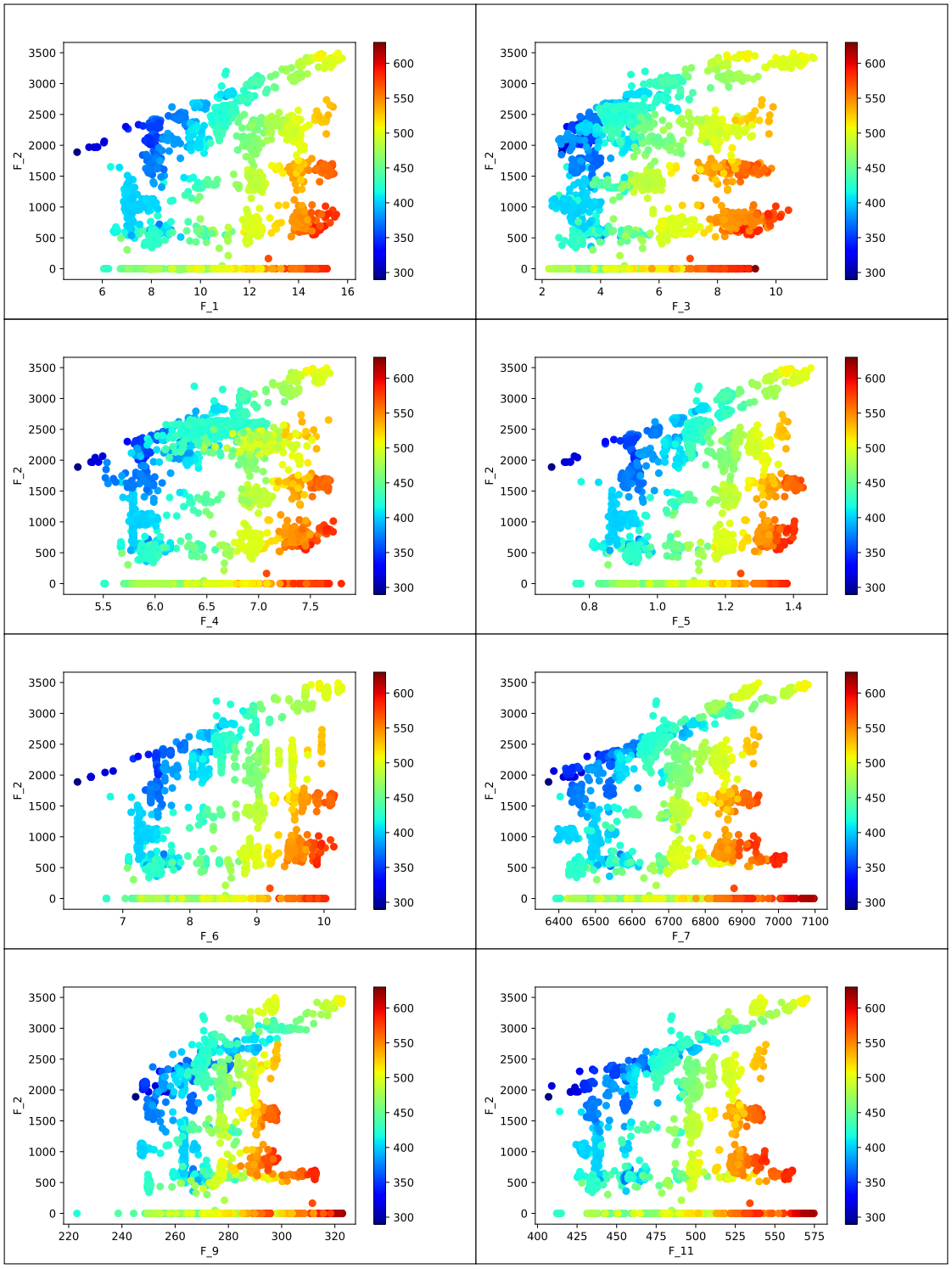
D.1 Pearson

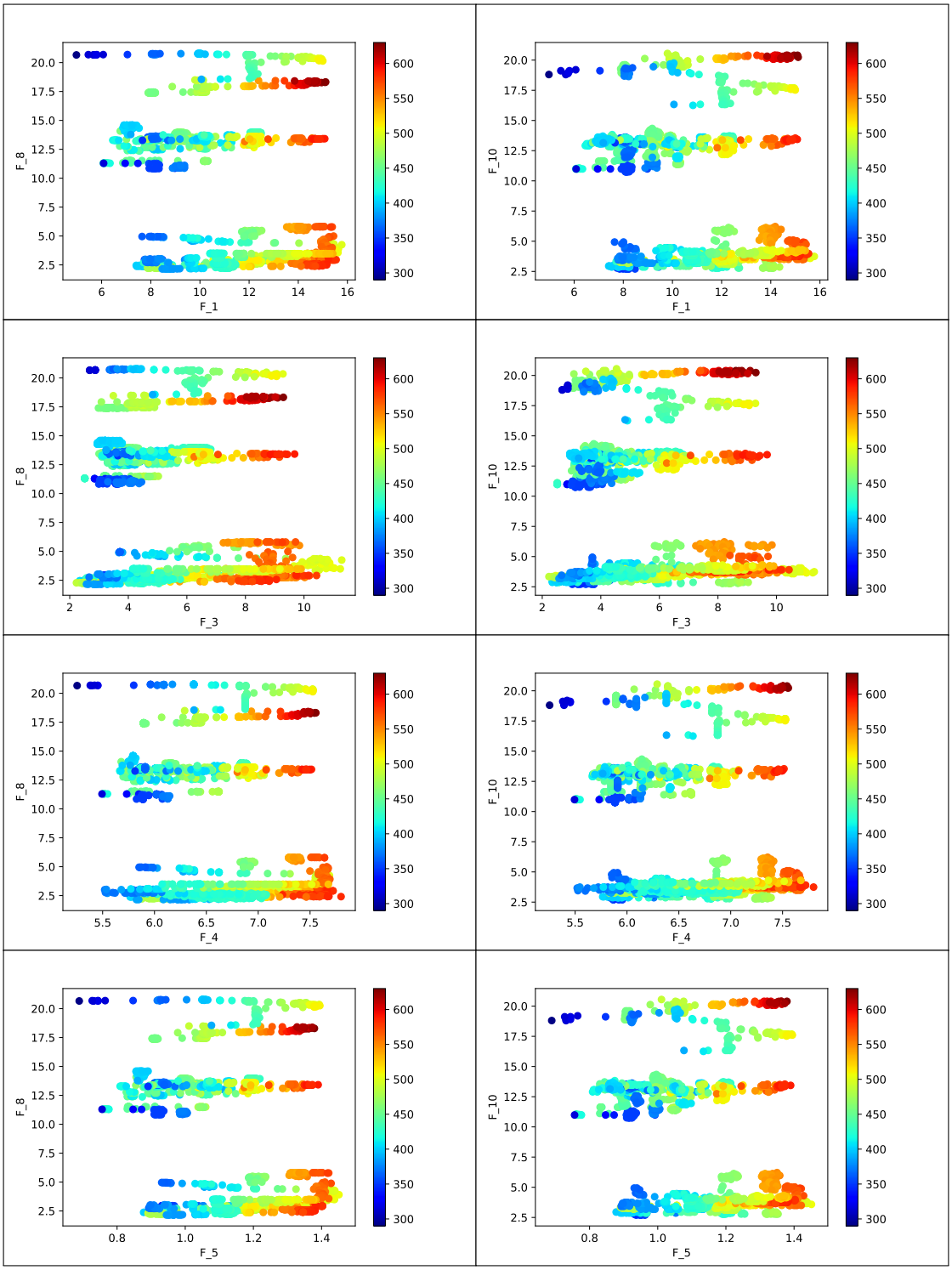


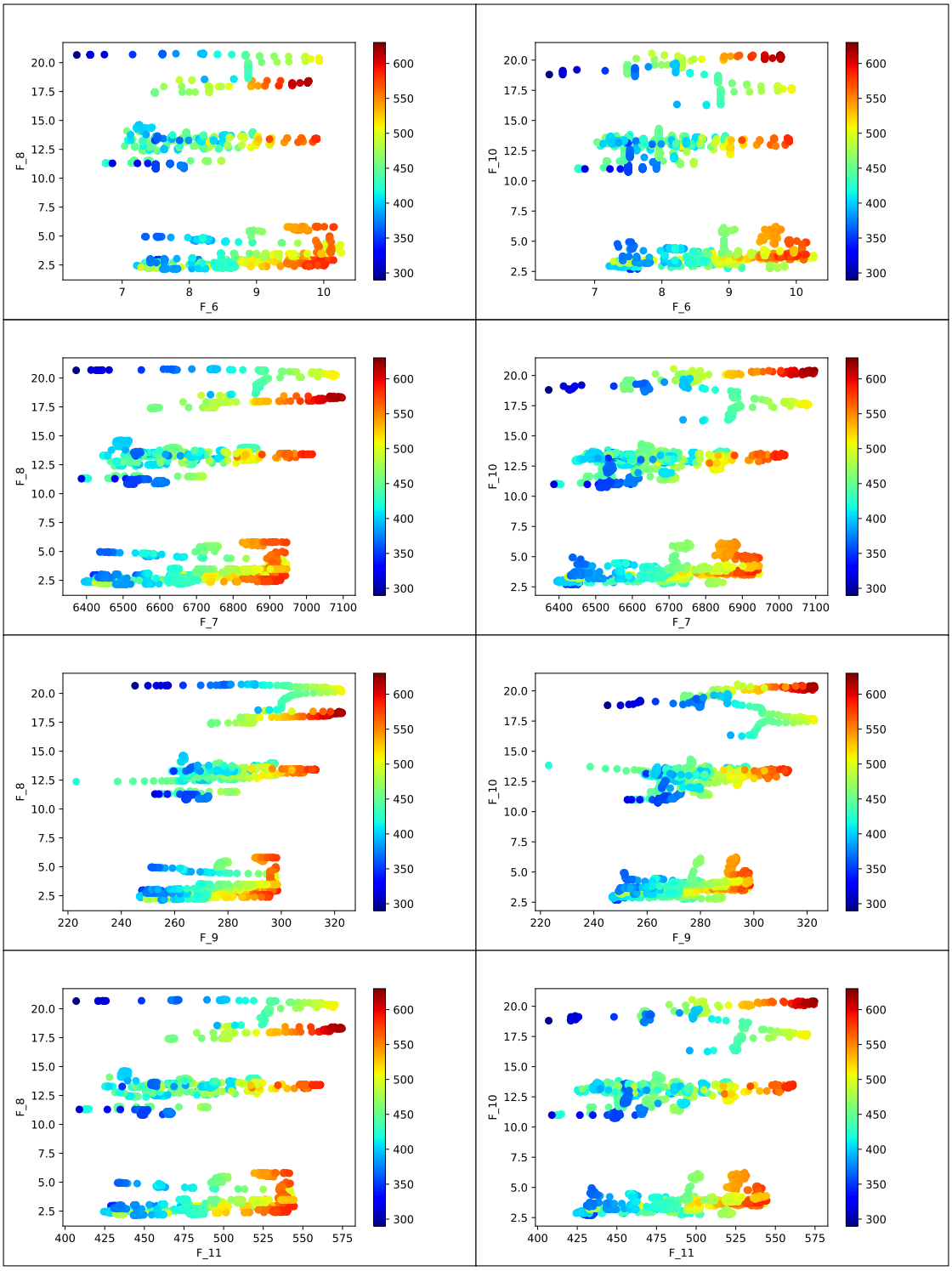
D.2 Spearman

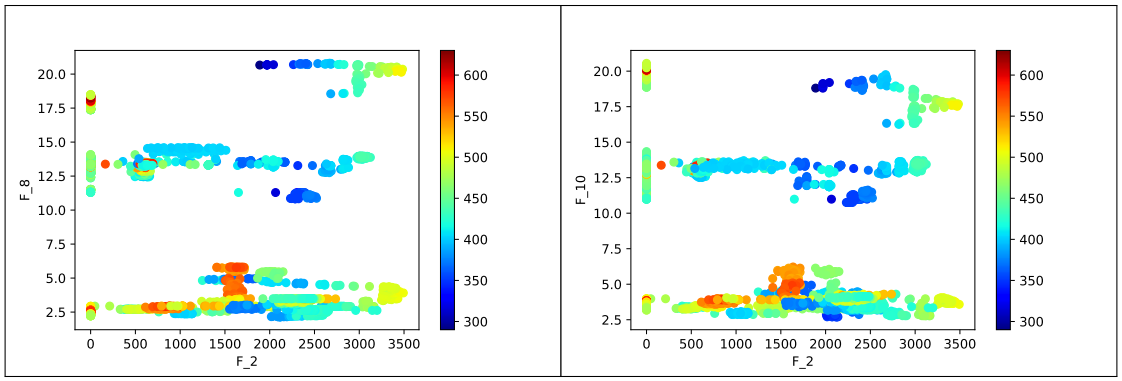

No correlation threshold
Score
Number of features
Features ranked
F_6
F_2
F_11
F_1
F_4
F_8
F_9
F_7
F_5
F_10
F_3
Correlation threshold: 0.99
Score
Number of features
Features ranked
F_1
F_4
F_2
F_9
F_7
F_3
F_8
F_10
Correlation threshold: 0.97
Score
Number of features
Features ranked
F_1
F_2
F_9
F_3
F_7
F_8
F_10
Correlation threshold: 0.96
Score
Number of features
Features ranked
F_1
F_2
F_9
F_7
F_8
F_10
Correlation threshold: 0.94
Score
Number of features
Features ranked
F_1
F_2
F_9
F_8
F_10
Correlation threshold: 0.93
Score
Number of features
Features ranked
F_1
F_2
F_8
Correlation threshold: 0.35
Score
Number of features
Features ranked
F_1
F_8

E | Selected features plotted




F_8
F_1
F_10
F_1
F_8
F_3
F_10
F_3
F_8
F_4
F_10
F_4
F_8
F_5
F_10
F_5


F_8
F_6
F_10
F_7
F_9
F_11


F_8
F_2
F_10
F_2
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0
500
1000
1500
2000
2500
3000
3500
600
550
500
450
400
350
300

F | 3D plot feature view

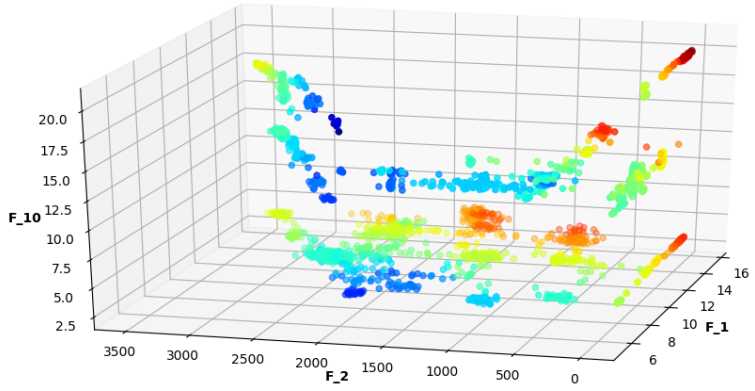


Figure F.1: 3D plot from perspective of feature 1

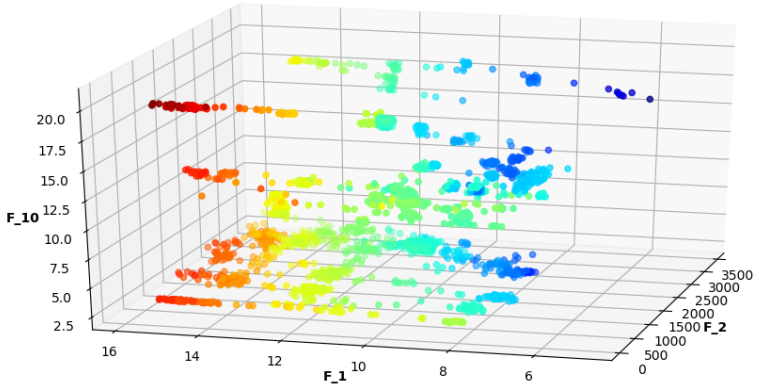


Figure F.2: 3D plot from perspective of feature 2

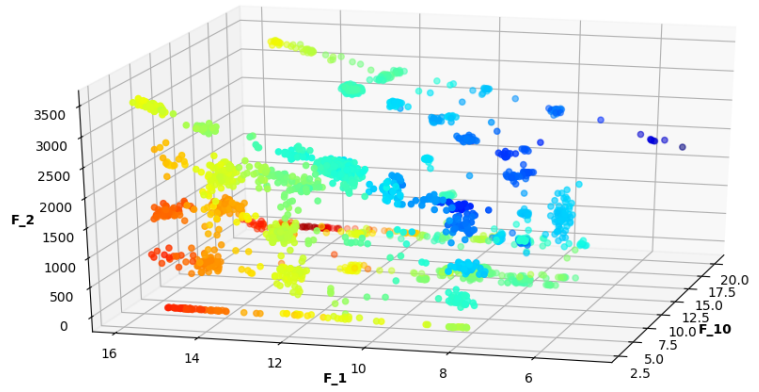


Figure F.3: 3D plot from perspective of feature 10

G | Visualisation of well-performing decision trees

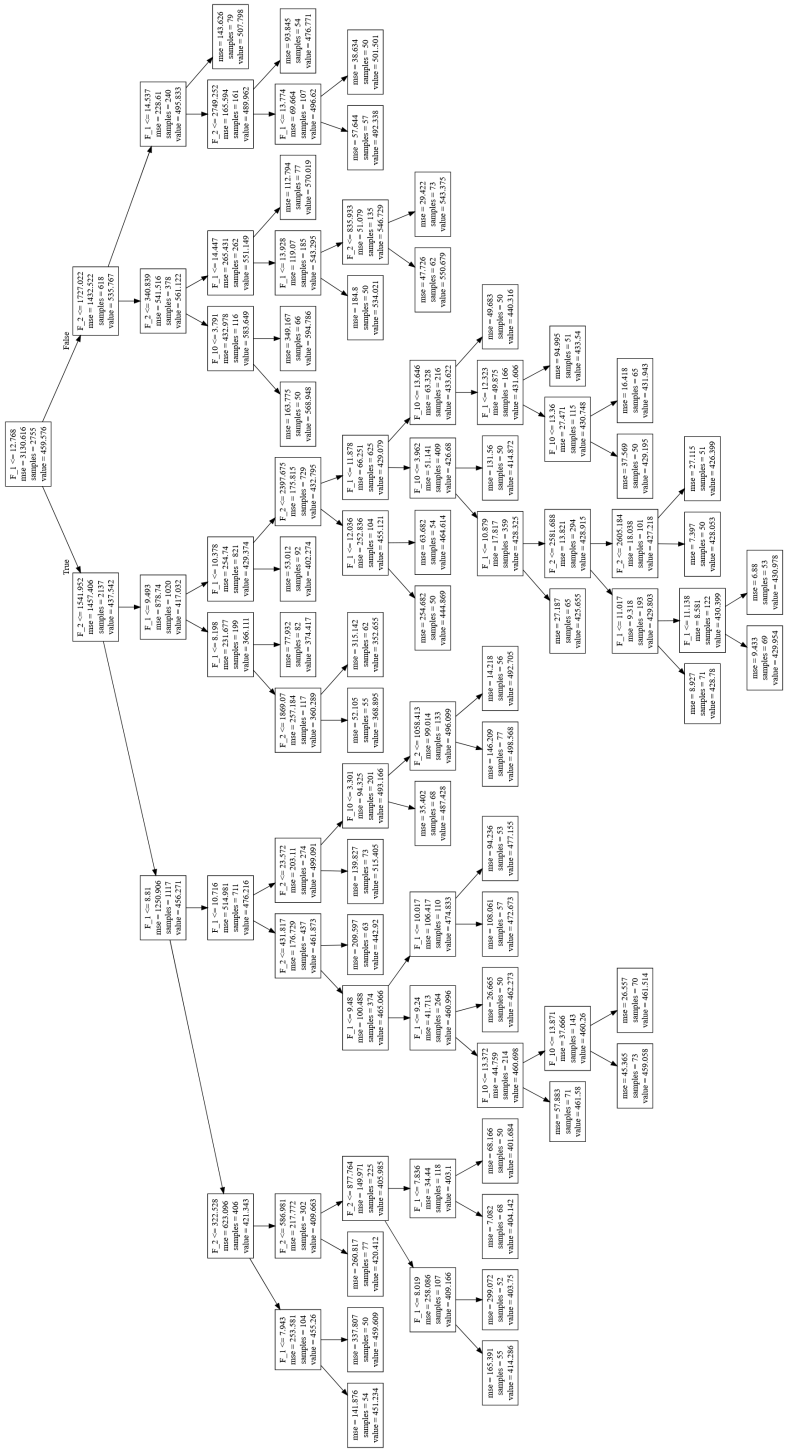


Figure G.1: [*DecisionTreeRegressor*] visualised tree for selected features based on H0

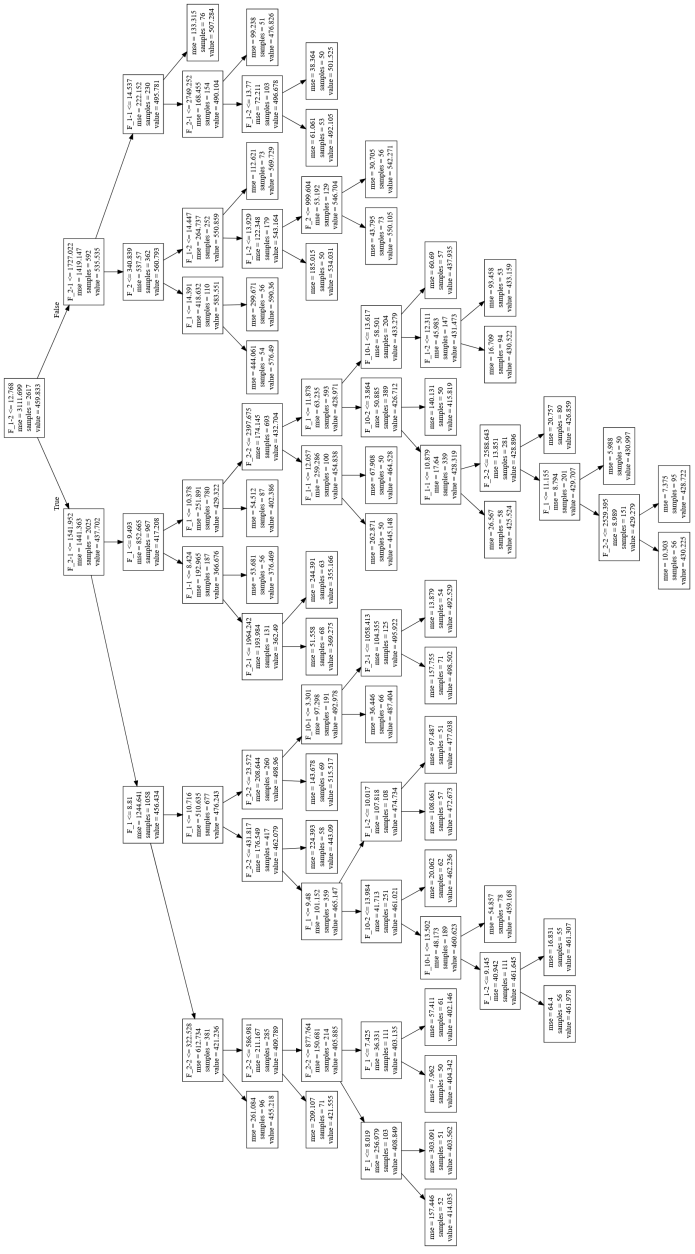


Figure G.2: [*DecisionTreeRegressor*] visualised tree for selected features based on H2

H | Historical target scores

	First-Parallel	First-Increment	Maximum-Parallel	Maximum-Increment	Random
H0	116.58	116.58	95.23	120.31	96.40
	0.96	0.96	0.97	0.96	0.97
	167.56	123.88	119.58	110.77	88.52
	0.94	0.96	0.96	0.96	0.97
H1	6.10	5.76	8.56	8.22	8.07
	1.00	1.00	1.00	1.00	1.00
	4.95	4.95	5.59	4.39	8.51
	1.00	1.00	1.00	1.00	1.00
H2	5.79	6.54	4.76	5.56	8.60
	1.00	1.00	1.00	1.00	1.00
	6.16	6.16	3.98	3.98	8.75
	1.00	1.00	1.00	1.00	1.00
H3	6.33	6.33	5.82	9.87	8.67
	1.00	1.00	1.00	1.00	1.00
	7.47	7.47	5.83	4.88	9.55
	1.00	1.00	1.00	1.00	1.00
H4	7.47	7.47	6.00	6.00	8.92
	1.00	1.00	1.00	1.00	1.00
	7.16	7.16	4.98	4.98	10.32
	1.00	1.00	1.00	1.00	1.00
H5	8.89	8.69	7.26	7.01	9.53
	1.00	1.00	1.00	1.00	1.00
	8.84	8.44	6.59	6.28	10.69
	1.00	1.00	1.00	1.00	1.00

All features
Three selected features [F_1, F_2, F_10]

Table H.1: [*Boosting trees*] scores (upper MSE, lower R2)

	First-Parallel	Maximum-Parallel	Random
H0	1625.70	126.00	126.00
	0.43	0.96	0.96
	1588.86	111.49	111.36
	0.45	0.96	0.96
H1	1601.70	1.17	1.17
	0.45	1.00	1.00
	1.65	1.65	1.65
	1.00	1.00	1.00

All features
Three selected features [F_1, F_2, F_10]

Table H.2: [*SVR*] scores for RBF kernel (upper MSE, lower R2)

I Contour plots of best model

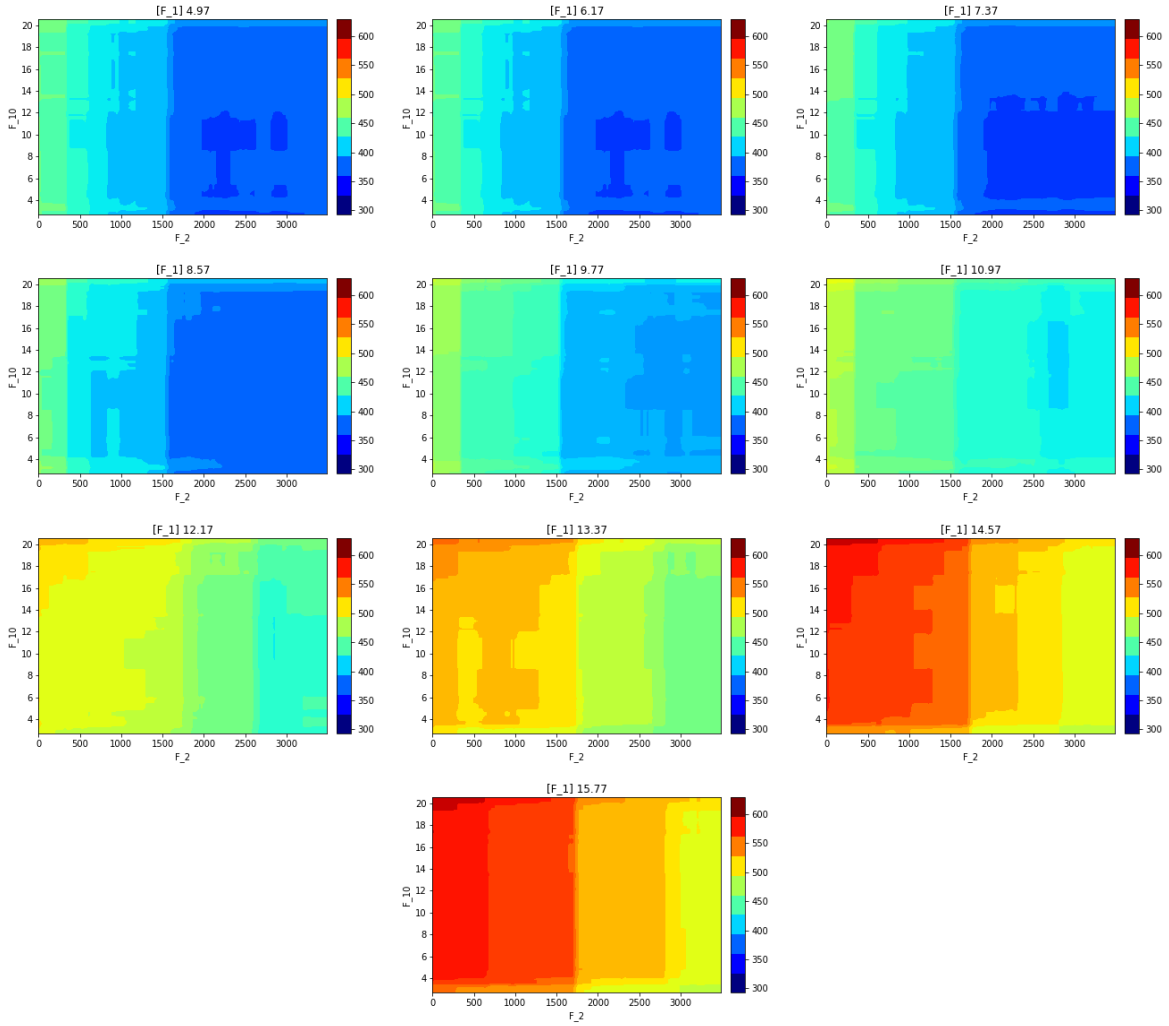


Figure I.1: [*Trees*] best model contour plots F_1

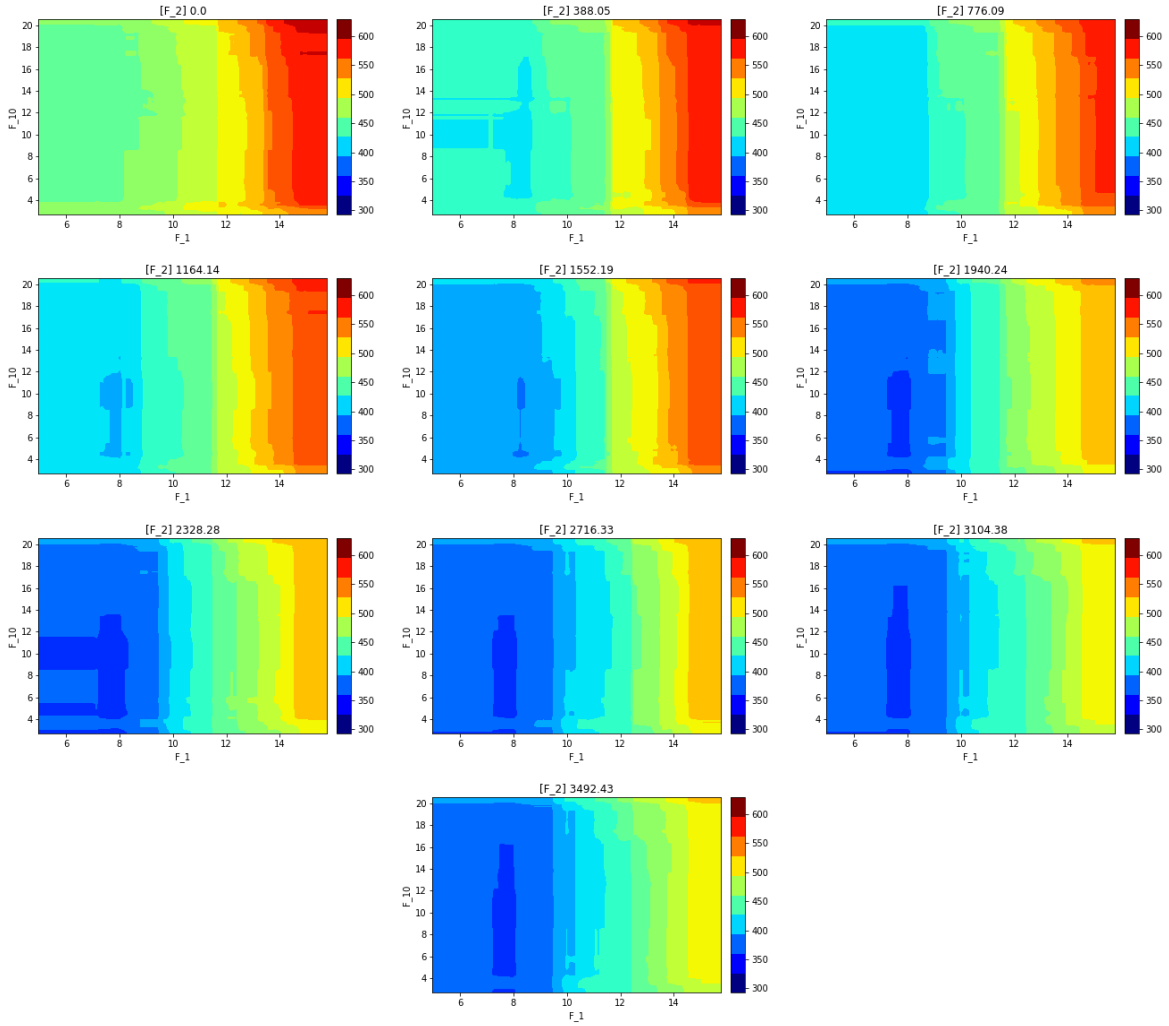


Figure I.2: [*Trees*] best model contour plots F_2

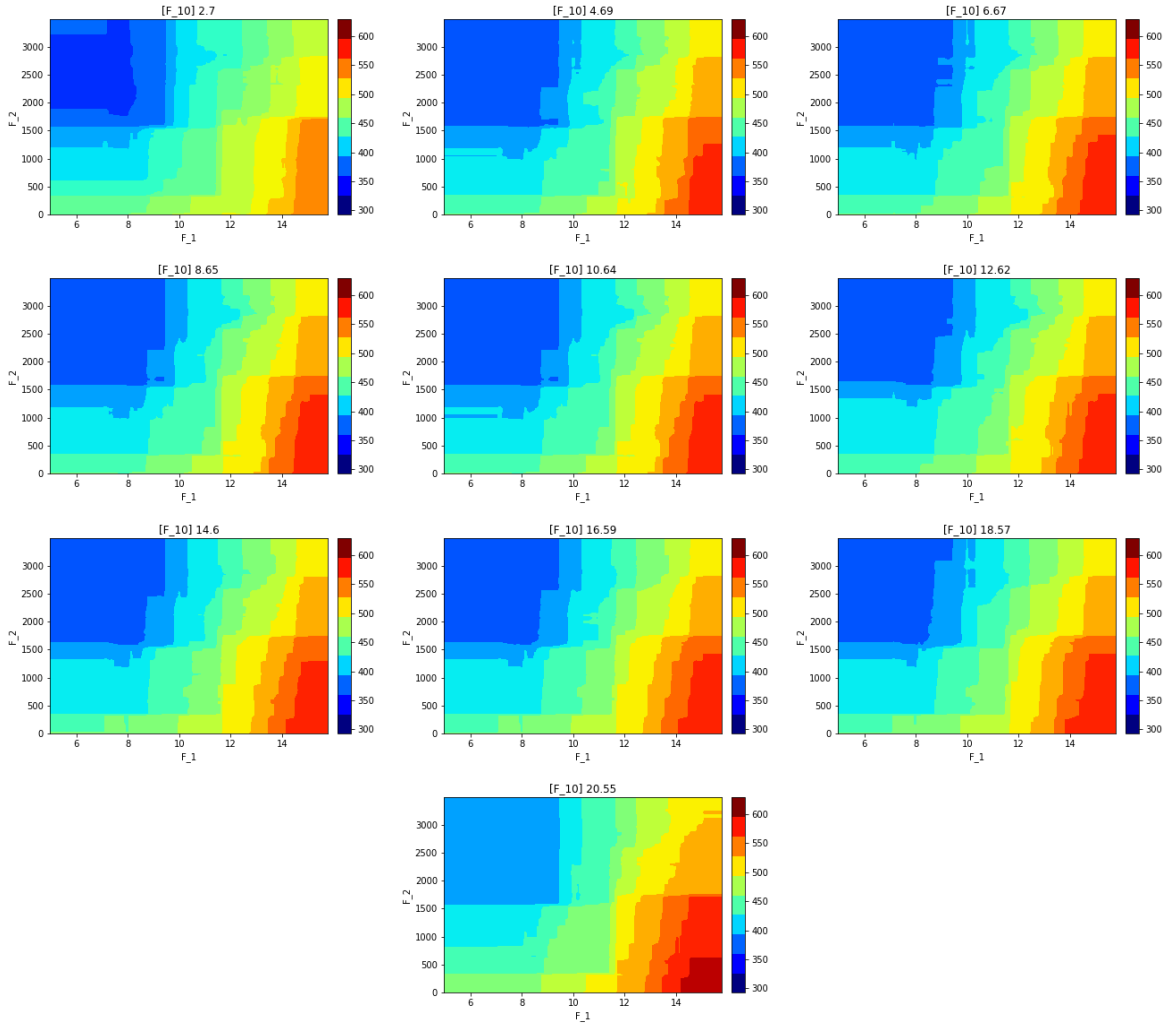


Figure I.3: [*Trees*] best model contour plots F_10

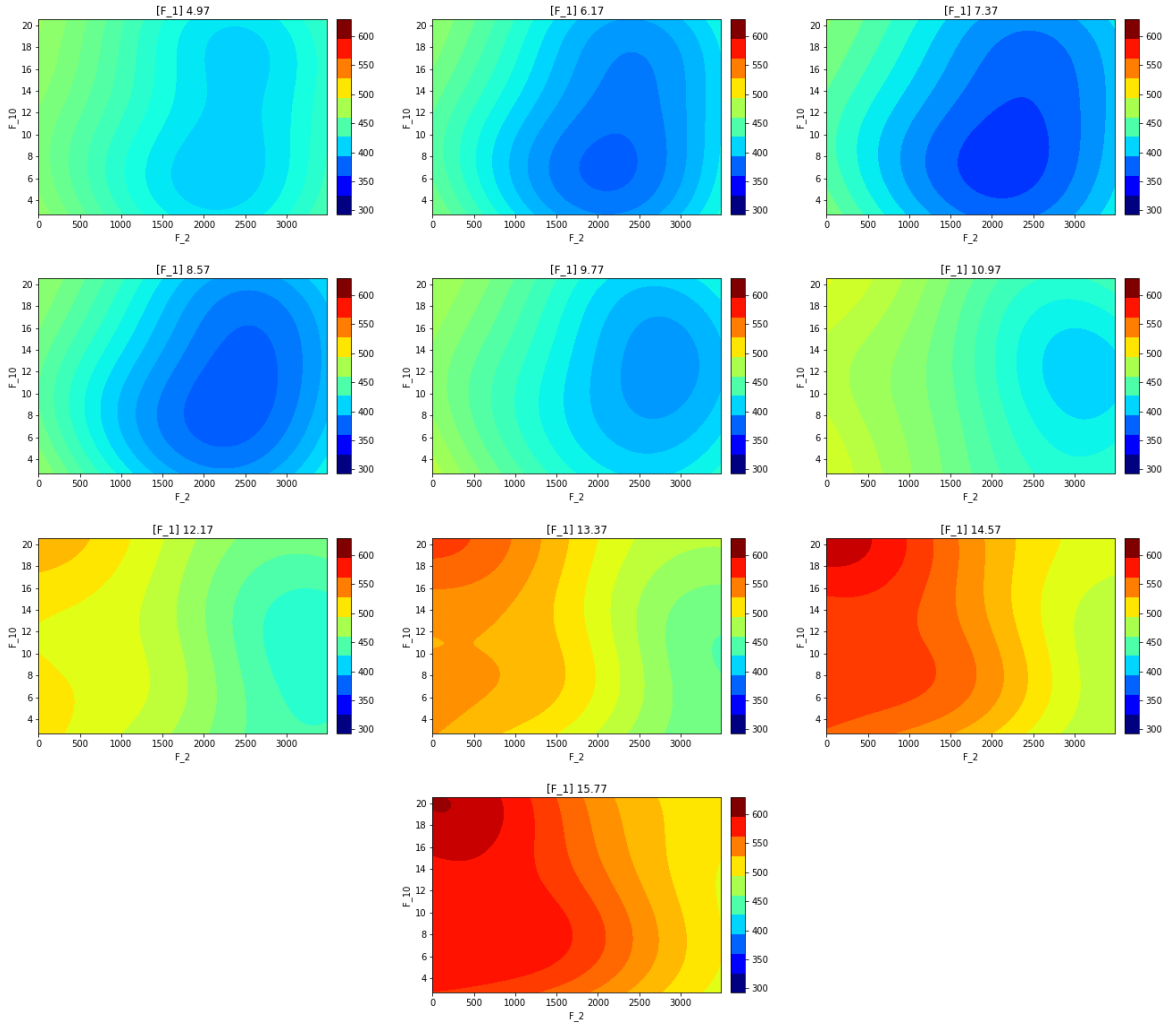


Figure I.4: [*SVR*] best model contour plots F_1

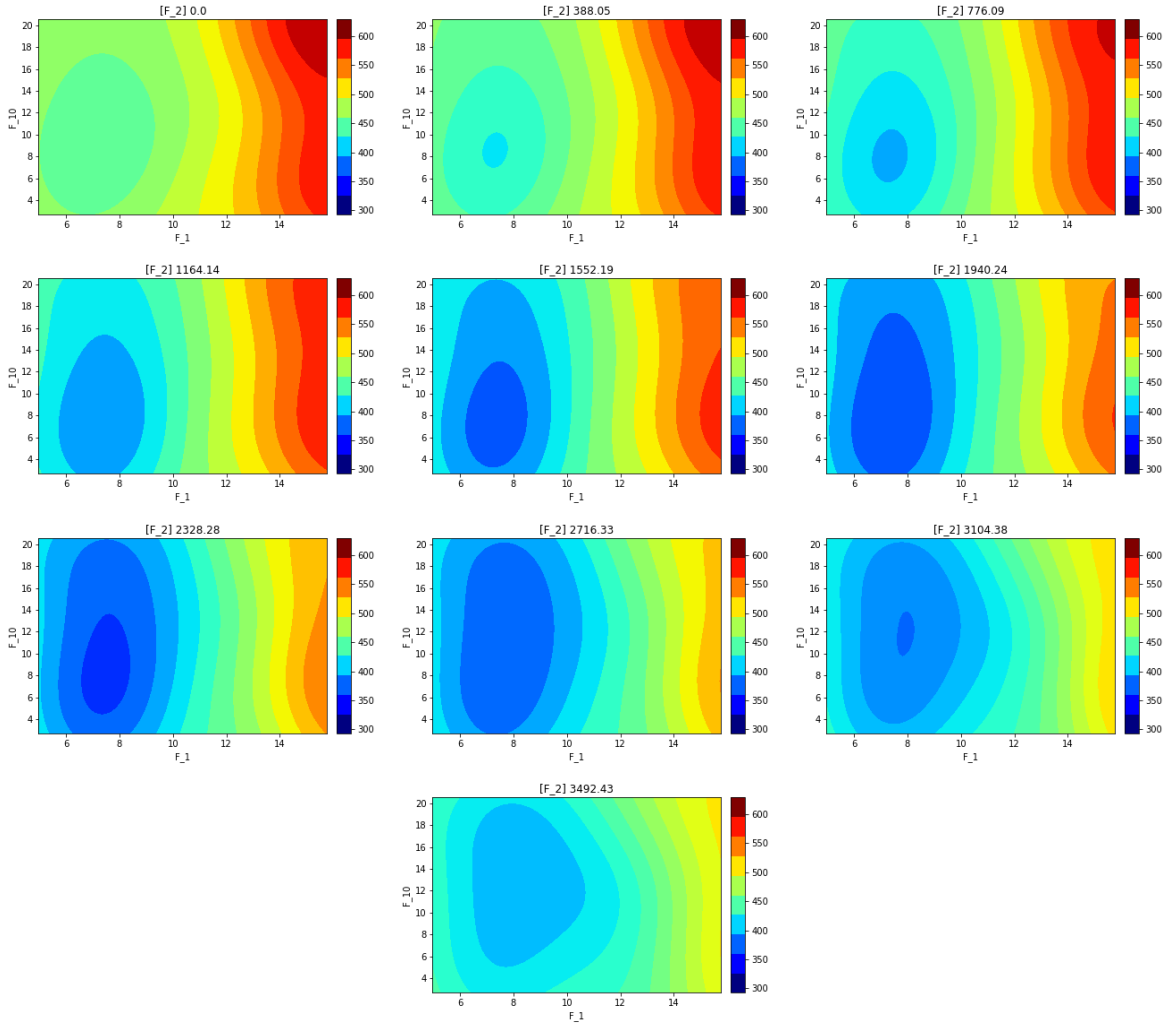


Figure I.5: [*SVR*] best model contour plots F_2

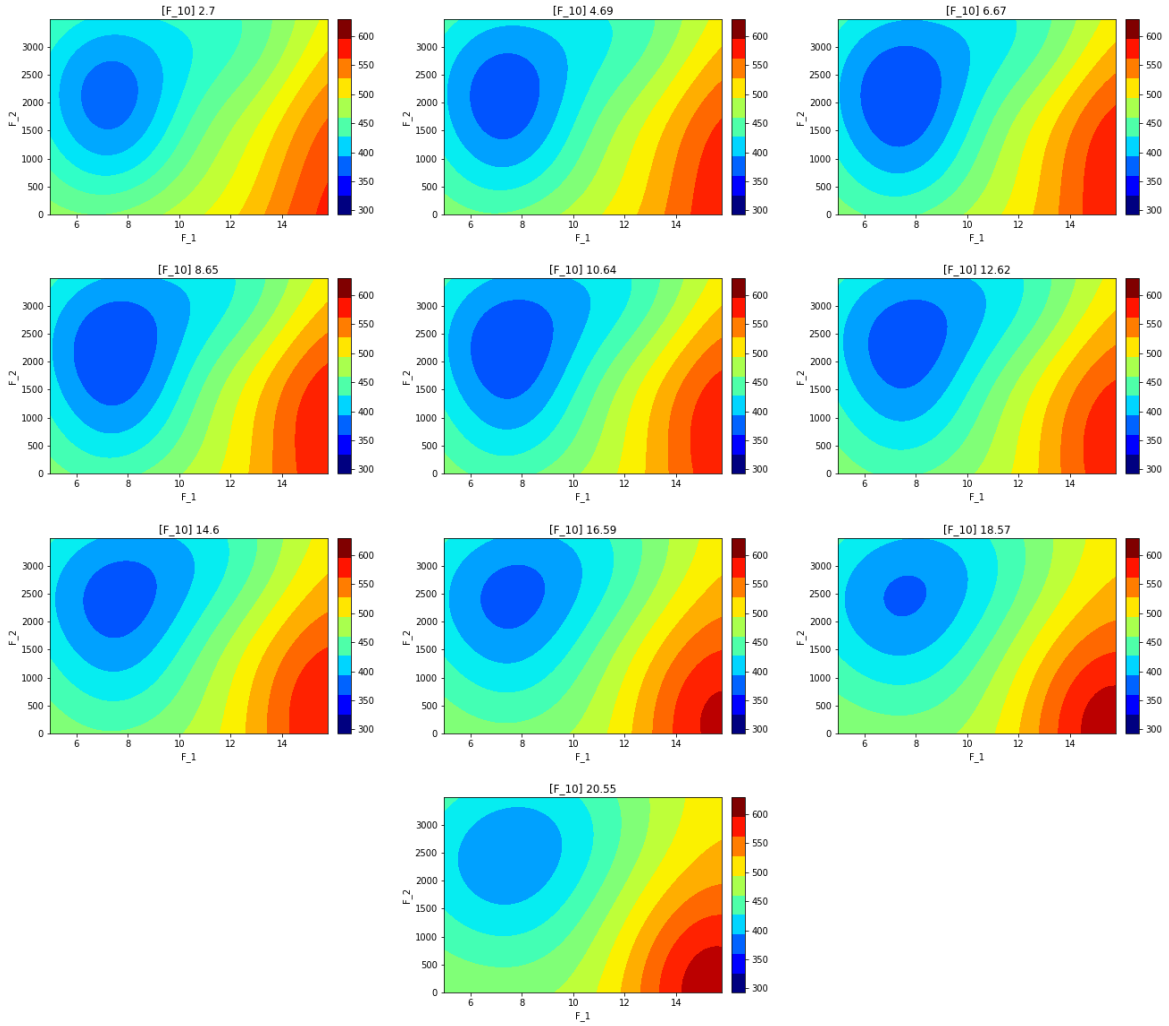


Figure I.6: [*SVR*] best model contour plots F_10

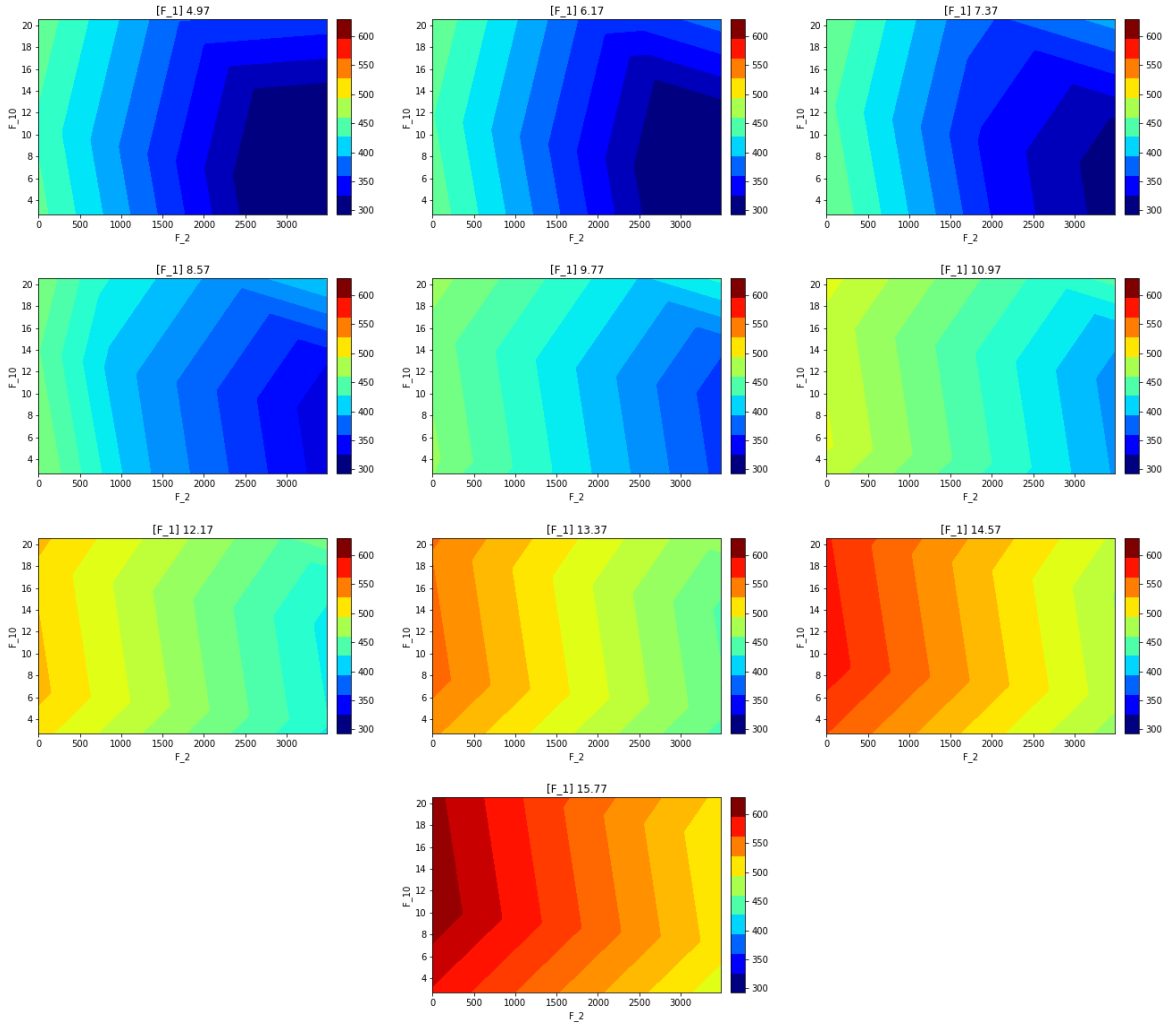


Figure I.7: [*Neural network*] best model contour plots F_1

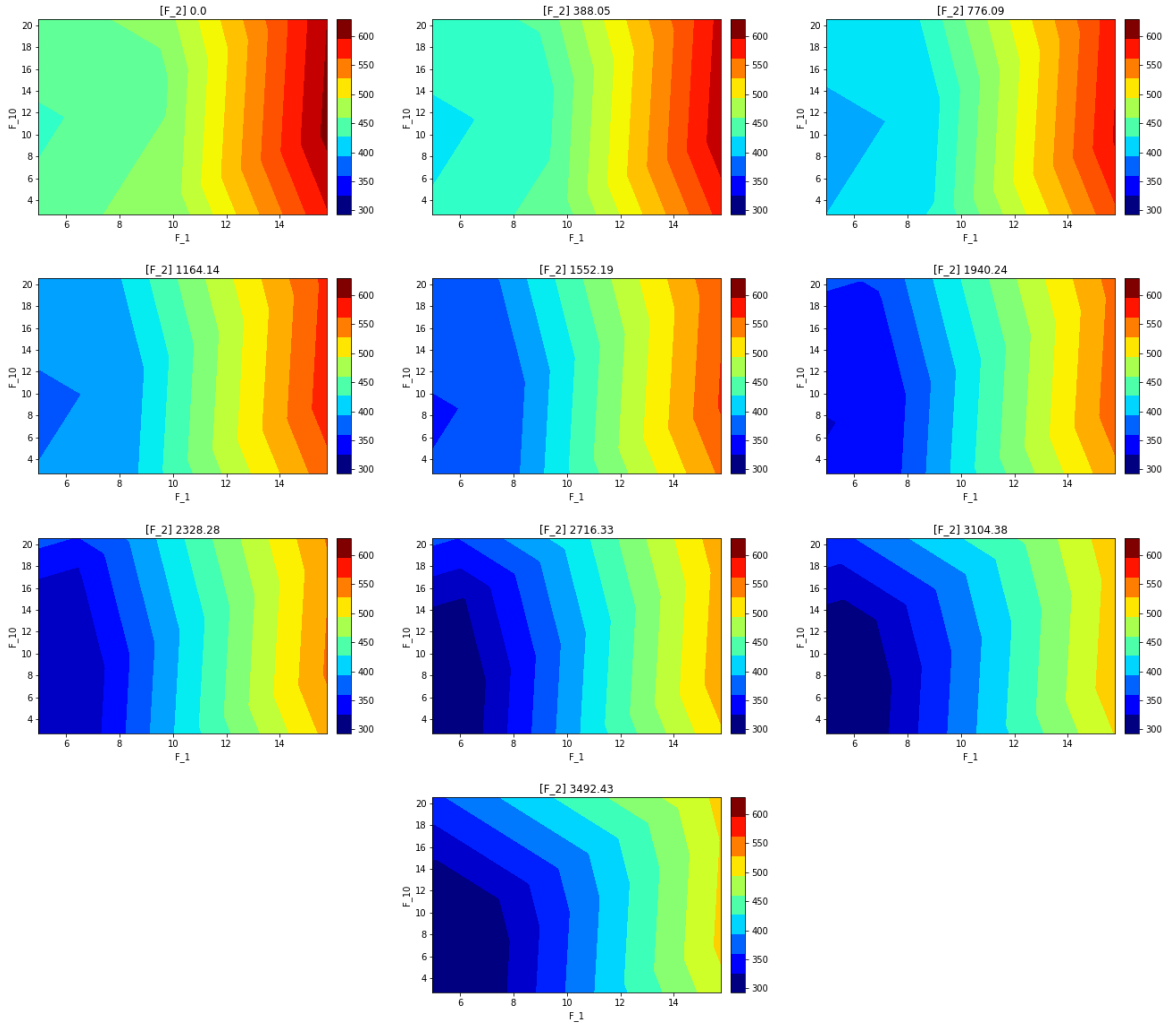


Figure I.8: [*Neural network*] best model contour plots F_2

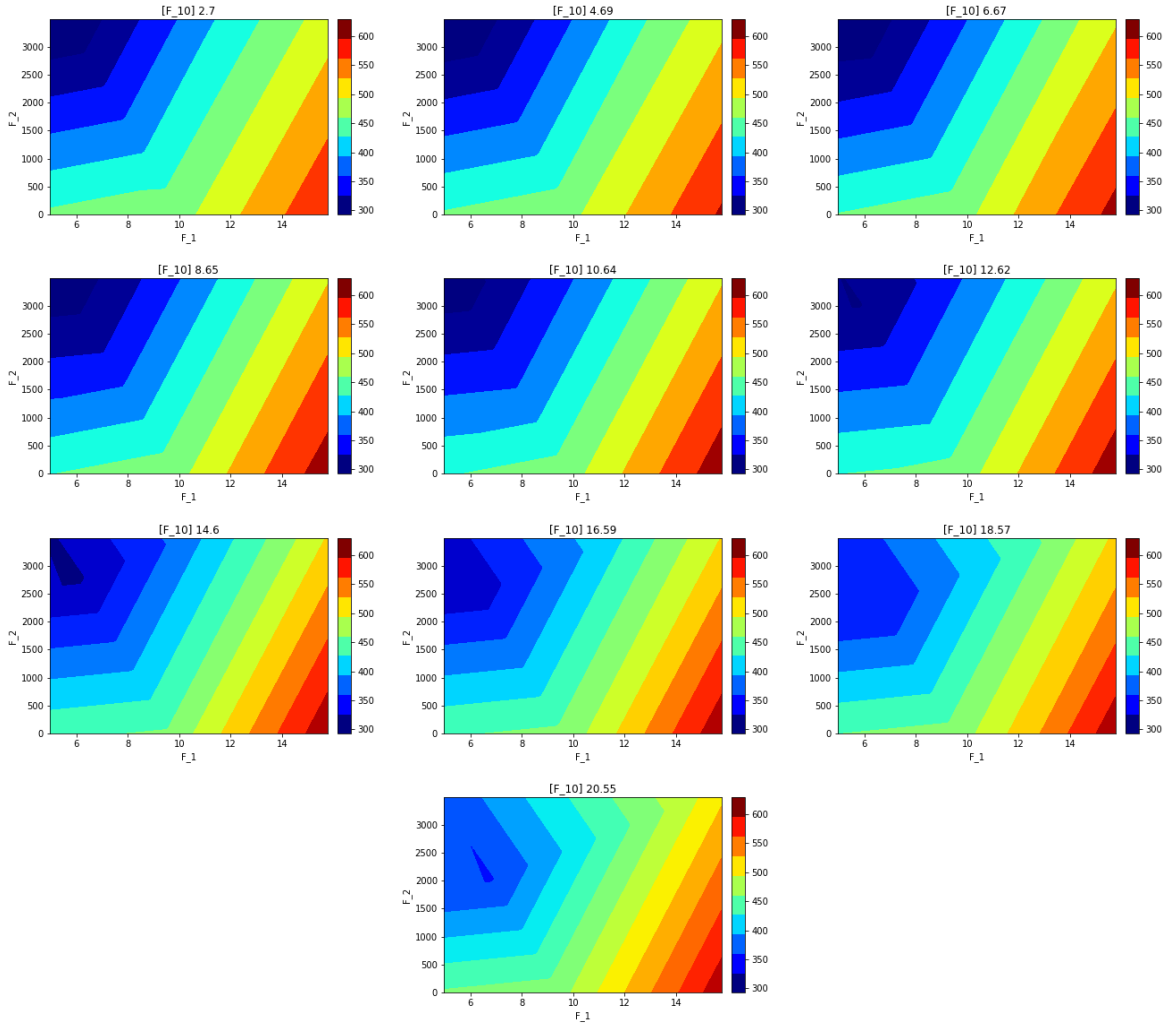


Figure I.9: [*Neural network*] best model contour plots F_10