Building Data Classification and Association

Anna Åberg

Christine Sjölander

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6057
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2018

Abstract

Almost half of the energy consumption in the EU originates from heating and cooling of buildings. The European Commission states that smart control of building systems may reduce the energy consumption. Cloud based smart control or even advanced fault-detection systems are becoming more common and should work for any building in the world. These systems need to receive data from the physical buildings which are commonly managed by a Building Management System, BMS. Today, when connecting an advanced control or analysis system to a buildings' BMS is a manual process, more or less, which is time consuming and error prone. Therefore, it would be beneficial if this process could be automated. This thesis aimed to find machine learning methods that had the potential to be used to fully- or semi-automate the connection process.

By implementing and evaluating models of three machine learning methods, random forest, gradient boosting and neural network, we aimed to find some method able of labelling time series data into a fixed classification system with a precision of 80% or higher. The solutions were tested on three data sets with different complexity and we could show that for a set with low complexity it is possible to achieve perfect classification, i.e. accuracy of 100%. For the more complex sets accuracy decreased to roughly 60% and a fully automated solution from these models would not perform good enough. However, the probability that the correct class was among the top five predictions of the models remained high and therefore they could be used in a semi-automated connection process.

Overfitting was an extensive problem when classifying signals, especially for random forest and gradient boosting models. We believe this is partly due to the data being too homogeneous and the situation could be improved by including data from additional buildings. The problems with overfitting could be seen most clearly when models were trained and tested on data from different buildings. In this case, random forest and gradient boosting models were clearly outperformed by neural network models that still scored about 60% accuracy without any overfitting.

We also attempted to group signals by equipment type. This was done via support vector machines and a string comparison method. The support vector machine solution was only possible to deploy on the least complex data set, but performed well with an accuracy of over 85%. To implement this solution on more complex data sets more knowledge about the system is needed. The string comparison method proved that much information could be gathered from the correlations in the signal names and paths. Nevertheless, it was hard to come to any general conclusions from this since data from only one BMS was used. We believe that the string comparison could give good results in combination with other methods.

Acknowledgements

We have been overwhelmed by the support from our official and unofficial supervisors. Oskar Nilsson and Jonas Bülow at Schneider Electric have patiently answered all our questions about everything from git to explaining how an air-handling unit works. You have pushed us and constantly provided new ideas and angles for us to work with. Bo Bernhardsson and examiner Anders Rantzer have given freely of both time and advice during this project. Thank you for helping us finding a clear thread in the work when we had more ideas than we could handle. Special thanks go out to Emil Johansson who has eagerly provided valuable input on neural networks and KERAS. We wouldn't have been able to understand the rest of our sources without you.

For us this thesis is also the culmination of five amazing years at Lund University. We have been challenged and at times have had so much to do that it seemed impossible to ever finish. However, we've had unwavering support from our families. You have always been there, even when we haven't had the time for you. Thank you. We would also like to thank our close friends and every person we've come to work with along the way. We might have made it without you, but we wouldn't have had half as much fun. And in ten, fifteen years looking back, we will probably not remember mathematical theorems and proofs, but we will remember you. We will remember all the laughter, the warmth and the craziness. So, to all of you that made Lund our home, thank you.

Partition of work and delimitations

For this thesis the ambition was to find methods capable of solving both the classification and association problem. Therefore a handful of machine learning methods were investigated and many tuning aspects were considered for each method. The thesis is therefore a wide study of potential solutions rather than a narrow search of optimal design and optimal parameters for a specific method.

In this thesis the machine learning methods random forest, gradient boosting and neural network was utilised for the classification problem. The first two methods has primarily been investigated by Christine Sjölander while the latter has been Anna Åberg's responsibility. For the association problem a solution with support vector machines was mainly implemented and tested by Anna Åberg. Christine Sjölander was responsible for the string comparison method that was also implemented for the association problem. Further on Christine Sjölander had a greater responsibility for the preprocessing of data while Anna Åberg primarily focused on the design and implementation of feature extraction.

Nomenclature

Association problem The challenge of automatically telling if signals belong to the same equipment or not

Batch A set of windows

BMS Building Management System

Classification problem The challenge of automatically telling which type, i.e. room temperature or air pressure, a signal is

Data set A full set of signals including training, test and evaluation set

Epoch One training cycle where the model has trained on all available training examples

Individual One example in a data set, i.e. a part of an entire measurement of a signal that spans over a fixed time range. See window

Measurement point A sensor or the measurements from this sensor. A measurement point may generate many individuals.

Method General algorithms such as random forest or neural networks, not a specific design for these.

Model One specific implementation of a method with a fixed set of parameters

Signal The time series data associated with one unique measurement point

Signal type The class label of a signal

Signal name The name of the measurement point in the BMS

Signal path The signal path within the BMS. Not including signal name

Structure A part of the options for a model, especially used for neural network models referring to number of layers and neurons

Window Each signal is divided into several time frames of fixed length denoted as windows

Contents

1

Introduction

1.1 Background

Heating and cooling accounts for half of the energy consumption in the EU where 84% of this energy originate in fossil fuels [European Commission, 2016]. The European Commission mentions both improved choice of materials, construction and new heating and cooling equipment in buildings as ways to reduce the energy consumption. They also state that [European Commission, 2016]:

> *Energy use can also be cut by providing better information and control of energy use with intelligent thermostats. They can turn heating off when the set temperature is reached, or even switch off when there is nobody in the building, in particular office buildings.*

This is also explained by industry supervisor Oskar Nilsson who also emphasises that the use of intelligent systems for control and analysis can enable large energy savings. For the energy saving to also become cost efficient quickly these systems rely on ease of deployment. Larger buildings are usually equipped with a BMS that receives its data from components located within the building. An emerging trend is connecting BMSs to services hosted in the cloud such as advanced fault-detection, building energy dashboard or other analytic functions. In Figure 1.1 the coupling between such analysis system and different BMSs is depicted. The cloud services are independent of where in the world the building is located and only depend on a stream of building data. Sensor readings and names for sensors can be remotely accessed from the BMS but naming varies between countries, building owners and system vendors. This presents an issue when it comes to quickly connecting a building to a cloud hosted service.

Today connecting the BMS and the analytic system is a more or less manual labour [Hong et al., 2015b]. A large building with thousands of signals and possibly an unpleasant naming could take many days of work to connect. Obviously, the process

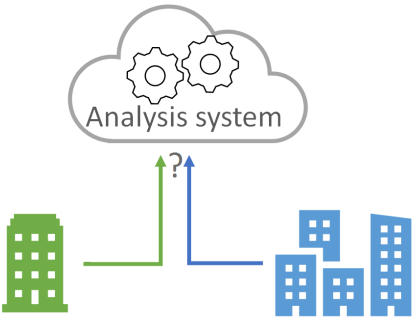


Figure 1.1 Buildings with different BMS connected to the same analysis system is presented in blue and green. The analysis system needs to understand the data gathered from the BMSs.

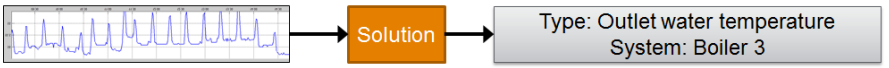


Figure 1.2 From left to right we see data from the BMS, such as a time series measurement, entering some kind of solution implementing for instance machine learning methods. This solution then outputs that the type of the signal is of type "Outlet water temperature" and that it belongs to the system "Boiler 3" in the building.

is cumbersome, time consuming and error prone hence it would be preferable to automate this process. This thesis therefore aims to investigate methods that could simplify and possibly automate the connection process. This process can be divided into two major subproblems that here on will be referred to as the classification problem and the association problem. The optimal solution would decide both the type of signal and to which system, in the building, it belongs to by only processing data from the BMS as depicted in Figure 1.2.

1.2 Problem description

First, assume that there is a set of 100 signals. All these signals belong to one of three types or classes and for every signal there is some data (excluding the class of which it belongs to). The aim of classification is to find a method that can tell which class a specific signal belongs to only by "looking at" the measured data. The data can include time series measurements from a sensor or metadata e.g. the name of the signal. One could be tempted to only use the signal name but since naming conventions vary between countries and even building owners this might not be as straight forward as one would anticipate [Hong et al., 2015b; Hong et al., 2015a]. An

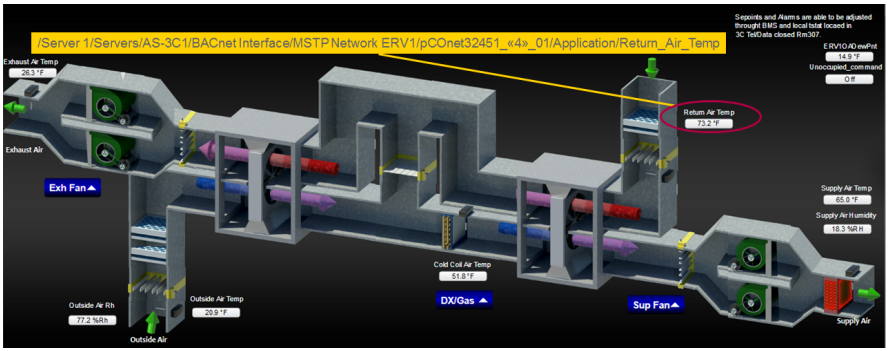


Figure 1.3 An example of an air-handling unit in USA, the signal path and name are presented for the circled measurement point.

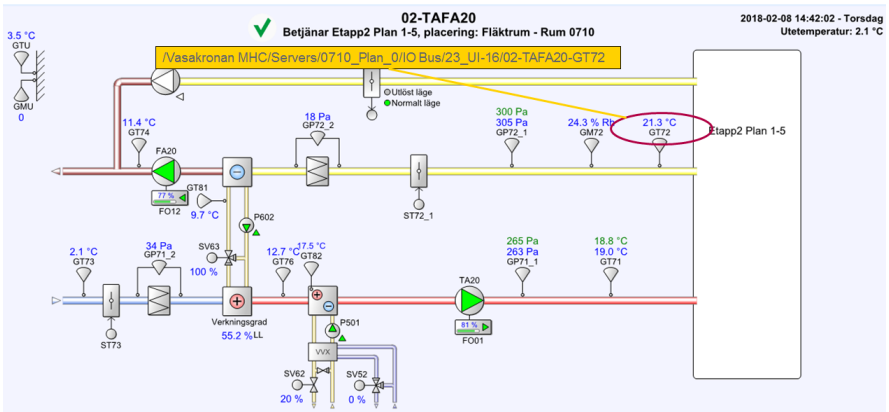


Figure 1.4 An example of an air-handling unit in Sweden, the signal path and name are presented for the circled measurement point. Notice that the same signal would in a US naming convention be very different, see Figure 1.3.

example of different naming conventions is presented in Figure 1.3 and Figure 1.4 where in the American BMS a measurement point would be called Return_ Air_ Temp while in the equivalent in the Swedish system would be 02-TAFA20-GT72. These are obviously different and are just two examples of naming conventions in Sweden and USA. If different naming conventions only within the United States are regarded the issue as well will become even clearer. Here follow three examples of names for the same kind of temperature sensor; Zone Temp, RMT and ART [Hong et al., 2015b]. Without knowledge of all naming conventions it might be hard to automatically from the name of a measurement point decide which class it might belong to.

For the specific problem handled in this project we aimed to work towards the way KGS Buildings[1] classifies signals. KGS Buildings specialises in automated fault-detection and diagnostics on BMS systems and therefore it is of interest to investigate if signals automatically can be classified to their classification system. They have about 1300 classes but we will only work with a data set containing 53 of these. Important to notice is that some of the 1300 classes are overlapping due to development of systems and some of the signals, that rightly should be classified as different, are under normal circumstances impossible to separate into different classes from time series data.

In a BMS there are many signals and some belong to the same subsystem or equipment. An equipment can be an air-handling unit in a ventilation system (such as those seen in Figure 1.3 and Figure 1.4) or simply all sensors found in a physical room. Optimally, one would not only be able to tell different signal types apart from data but also tell which of them belong to the same equipment. This is the definition of the association problem in this thesis.

Also for the association problem it might be tempting to use the information in the path. A path of a signal may hold information of which system the measurement point belongs to and we find two examples in Figure 1.4 and Figure 1.3. The way of naming or creating paths also lack worldwide standardisation. For the American one, found in Figure 1.3 the path is:

> Server 1/Servers/AS-3C1/BACnet Interface/MSTP Network ERV1/ pCOnet32451_«4»_01/Application

while in the Swedish system the path, found in Figure 1.4, would be;

> Vasakronan MHC/Servers/0710_Plan_0/IO Bus/23_UI-16/

The measurement points originate from different buildings and the paths are therefore not describe in the same building. What can be noticed is although that the Swedish path starts off with the name of the building while the American one obviously refers to a server instead. This poses a similar difficulty as for the classification problem.

At hand for this part of the problem there are only five different types of equipment from the data set classified into KGS Buildings' system. Finally, it is assumed that all signals are correctly classified into each class before the association is tackled, i.e. it is assumed that the correct class of the measurement point is known in the association problem.

[1] `http://www.kgsbuildings.com/`

A perfect method would connect all signals correctly for all buildings in the world but 100% accuracy is usually not possible to achieve [Hong et al., 2015b; Hong et al., 2015a; Balaji et al., 2015]. Instead supervisor Oskar Nilsson explains that methods that have an accuracy of at least 80% for 80% of all buildings would be good enough to use for automation of the coupling. A lower accuracy for fewer buildings would probably mean that it is more effective to perform this manually.

1.3 Thesis objective

The primary objective of this thesis is to find methods capable of solving the classification and association problem described in section 1.2. Thus, the variety of signals are meant to be classified into predetermined categories and then grouped regarding the equipment they belong to.

1.4 Data sets

A few things need to be considered when evaluating the quality of a data set. First, the number of measurement points of each type within the set. Each measurement point will be a little different from the others due to the exact physical location of the sensor and the unique noise in the sensor. Many points, with unique characteristics, will therefore prevent the solution from basing its decisions on location- or noise-specific behaviour. Secondly, the number of examples that can be extracted from the measurement points. Every example will differs slightly and many examples will generally provide the solution with more information as it is learning how to make its decisions. The number of possible examples is primarily affected by the time over which the signals are measured as well as how many missing measurements there are. Third, if the data is measured over a long time period. If this is the case, seasonal behaviour in the measurements will be included and the solution will not be seasonally dependent. Finally, it is necessary to study how skew the distribution of examples is between classes. If there are extremely many measurements of one class and few of another, the solution will appear to perform well simply selecting the most common class, whether it is correct or not. It will also be provided with better and more diverse information on the frequently measured class. Below, a brief description of the three data sets available and used in this thesis is found.

Data set *A* consists of measurements from 255 different sensors collected from 51 rooms on 4 different floors in an office Building [Hong et al., 2017]. Each room is equipped with five sensors, one for; humidity, light, carbon dioxide, attendance (a passive infrared sensor) called PIR, and finally one for temperature as can be seen in Table 1.1. The data set spans over 7 days and in general every signal is sampled every 15 minutes during this period. There are

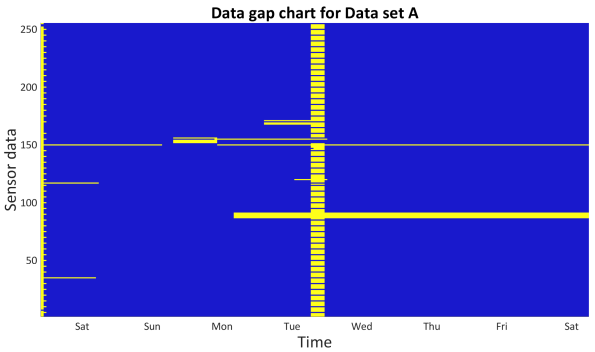


Figure 1.5 The 255 measured signals in data set *A* are each marked in blue when there is data and in yellow when data is missing.

some abnormalities in the measurements: There are missing measurements at sometimes, which can be seen in Figure 1.5. Clear from the figure is that some of the signals only span over 3 days. It should also be noted that the signal path and name was not the original one. This meant that the names and paths used in the BMS, from where the signals originated, was not available and instead a simple naming standard was introduced including the room number as path and the type of signal as the name.

Data set *B* consists of three different BMSs in three different buildings [Hong et al., 2015b]. The buildings are managed by different BMSs. Building *1* uses a system from Trane, Building *2* one from Siemens and Building *3* a system designed by Barrington Controls. All signals are sorted into one of 34 classes and the occurrence of each signal type in the three buildings are presented in Table 1.2, the type names are gathered from [Hong et al., 2015b].

Data set *C* consists of 129 signals measured over 2 to 3 years. The data origins from a Schneider Electric building in Boston that has been part of various research projects. The data has been stored for different purposes and the full data set contains some larger gaps which can be seen in Figure 1.7. All signals belong to one of 53 classes and in Table 1.3 the occurrence of each signal type is presented. The signals also belong to subsystems or equipment in the building. There are five different types of subsystems and in total 33 different equipment. In Table 1.4 all classes that are found in each equipment are presented and in Table 1.5 and all signal classes that might be found in an equipment type are presented. The classification system used for this data set is the one of KGS Buildings mentioned in section 1.1.

Table 1.1 Occurrence of different signal types in data set *A*.

occurrence of signals in data set *A*		
Classification number	**Signal name**	**Number of measured signals**
1	CO2 concentration	51
2	Room Air Humidity	51
3	Luminosity	51
4	PIR Motion Sensor	51
5	Room Temperature	51

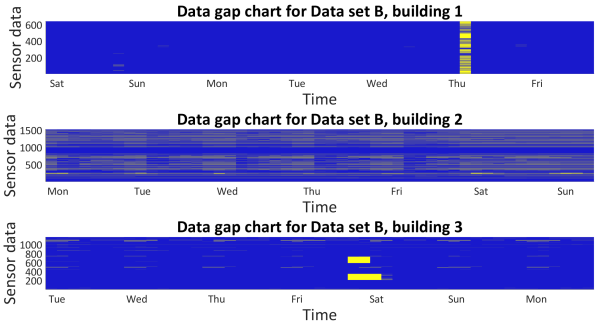


Figure 1.6 The measured signals in data set *B* are each marked in blue when there is data and in yellow when data is missing. Each building is presented separately.

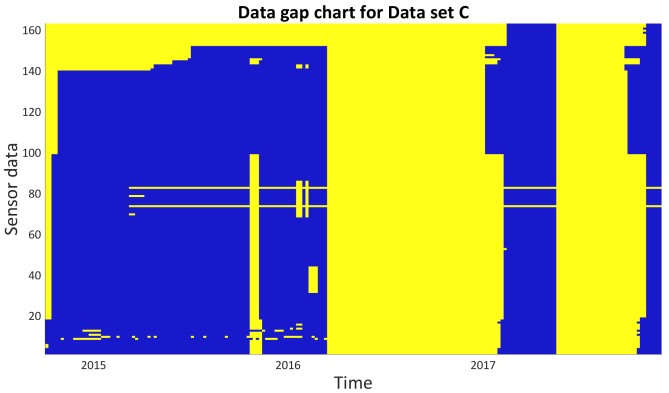


Figure 1.7 The 163 measured signals in data set *C* are each marked in blue when there is data and in yellow when data is missing. The data has over periods been logged for different purposes.

Table 1.2 Occurrence of different signal types in data set *B*.

occurrence of signals in data set *B*				
Classification number	**Signal type**	**Number of measured signals**		
		Building *1*	**Building *2***	**Building *3***
1	co2	16	52	0
2	air flow volume	14	172	9
3	air humidity	54	52	0
4	occupancy	25	25	0
5	room air temperature	159	231	207
6	facility operation status	59	58	41
7	setpoint	14	486	229
8	cold water return temperature	15	4	9
9	cold water supply temperature	18	6	1
10	hot water return temperature	15	1	1
11	hot water supply temperature	27	1	1
12	supply air temperature	2	17	3
13	return air temperature	6	2	4
14	mixed air temperature	5	2	3
15	ice Tank enter temperature	1	2	0
16	ice Tank leave temperature	1	6	0
17	control point	0	0	403
18	valve position	0	0	1
19	air pressure	0	0	215
20	fan speed	0	0	5
21	timer	0	0	16

1.5 Previous work

There is abundant research on general classification problems but for the specific case of building data much less has been published. The most relevant articles are the following. For the classification problem, one approach is to try to minimise the number of correctly labelled examples a solution needs from a system for the solution to be able to classify the others correctly from metadata. In [Balaji et al., 2015] a so-called active-learning method is proposed for classifying and naming signals or measurements points according to conventions. This paper is based on data from four different buildings and an average accuracy of 98% is achieved while using 28% less labelled examples compared to a method based on regular expressions. The solution utilises hierarchical clustering and groups measurement points with similar metadata. It is deemed to perform well for the association problem in buildings that have well labelled measurement points but poorly for those buildings that were lacking data or had metadata that did not contain enough information.

In [Hong et al., 2015a] another approach is used. Here the method instead aimed to classify all signals automatically by implementing a feature based random forest solution. Accuracy within buildings are measured to 92% and 82% across buildings. One of the two buildings used in the paper is the same as in Data set *A*, however they use data from six different sensors where three are the same as in Data set *A*. In the paper, Hong et al. also aimed to find what examples were misclassified. Their solu-

Table 1.3 Occurrence of different signal types in data set *C*.

occurrence of signals in data set *C*		
Classification number	**Signal type**	**Number of measured signals**
1	WaterPressure	4
2	OutdoorAirTemp	4
3	OutdoorAirRH	1
4	CHW1LoopSupplyTemp	1
5	EvaporatorOutletWaterTemp	2
6	ChillerStatus	2
7	EvaporatorInletWaterTemp	2
8	ValvePosition	1
9	CHW1LoopReturnTemp	1
10	CoolingCoilDischargeAirTemp	2
11	CompressorRun	5
12	CompressorStage2Run	3
13	CompressorPartLoadFraction	3
14	ExhaustAirTemp	3
15	ExhaustFanSpeed	3
16	HeatingGasFireValve	3
17	HRExhaustInletAirTemp	3
18	SupplyAirRH	3
19	SupplyAirTemp	5
20	SupplyFanSpeed	5
21	SupplyFanRun	1
22	ExhaustFanRun	3
23	ExhaustFanStatus	3
24	ExhaustAirDamper	2
25	SupplyFanStatus	1
26	RoomAirTemp	9
27	EconomizerStatus	2
28	OutdoorAirFlow	2
29	RoomAirCoolingTempSetpt	2
30	ReturnAirTemp	2
31	RoomAirHeatingTempSetpt	1
32	PumpStatus	2
33	HW1LoopDiffPressure	1
34	HW1LoopDiffPressureSetpt	1
35	PumpRun	2
36	PumpSpeed	2
37	HW1LoopReturnTemp	1
38	HW1LoopFlow	1
39	HW1LoopSupplyTemp	1
40	HeatingPlantEnable	1
41	ZoneSupplyFanRun	5
42	ZoneSupplyFanStatus	5
43	RoomAirCO2	3
44	ZoneSupplyAirFlow	2
45	ZoneSupplyDamperFeedback	2
46	ZoneSupplyAirTemp	2
47	ZoneSupplyAirVelocityPressure	2
48	ZoneCompressorRun	3
49	ZoneHeatingElectricCoil	3
50	CoolingTowerSumpTemp	1
51	CoolingTowerInletWaterTemp	1
52	BuildingCHWCoolingRate	1
53	RoomAirRH	3

Table 1.4 For each equipment the equipment type as well as the signal classes that can be found in the equipment is stated. Data from data set *C*

Signal classes in each equipment in data set *C*		
Equipment	**Equipment type**	**Signal classes in equipment**
BOC General Conditions	5	52
BOC Utility Consumption	4	1, 4, 8, 9
Boiler 1	1	5, 6 ,7
Boiler 2	1	5, 6, 7
Boiler 3	1	22, 23
CAV-1_ 17	3	2, 14, 15, 17, 18, 19, 20
CAV-1_ 1.10	3	10, 11, 12, 13
CAV-1_ 1.11	3	16
CAV-1_ 1.12	3	2, 14, 15, 17, 18, 19, 20
CAV-1_ 25	3	11, 12, 13
CAV-1_ 26	3	16
CAV-1_ 2.10	3	2, 14, 15, 17, 18, 19, 20
CAV-2_ 1.08	3	10, 11, 12, 13
CAV-2_ 1.09	3	16
CAV-2_ 2.09	3	41, 42, 43, 44, 45, 46, 47
CAV-2_ 3.05	3	43
CAV-2_ 3.06	3	41, 42, 43, 44, 45, 46, 47
CAV-3_ 1.10	3	50, 51
CAV-5_ 3.04	3	22, 23, 24
CAV-6_ 3.05	3	22, 23, 24
CB_ 1_ A_ ZN22	2	26, 41, 42, 48, 49
CB_ 1_ A_ ZN23	2	26, 53
CB_ 1_ A_ ZN24	2	26, 53
CB_ 1_ A_ ZN25	2	26, 53
CB_ 1_ A_ ZN26	2	26, 41, 42, 48, 49
CB_ 1_ A_ ZN27	2	26, 41, 42, 48, 49
CB_ 1_ A_ ZN69	2	1, 33, 34, 37, 38, 39, 40
CB_ 1_ A_ ZN70	2	32, 35, 36
CB_ 1_ A_ ZN71	2	32, 35, 36
CB_ 1_ A_ ZN72	2	21, 25, 26
CB_ 1_ A_ ZN73	2	2, 3
CB_ 1_ A_ ZN74	2	11, 19, 20, 26, 27, 28, 29, 30
CB_ 1_ A_ ZN75	2	11, 19, 20, 26, 27, 28, 29, 30, 31

Table 1.5 For each equipment type the signal classes that might occur in an equipment of this type is stated for Data set *C*. This information is derived from Table 1.4.

Signal classes in each equipment type in data set *C*		
Equipment type	**Equipment type name**	**Signal classes in equipment type**
1	Boiler Hot Water	5 ,6 ,7 ,22 ,23
2	Chilled Beam	1, 2, 3, 11, 19, 20, 21, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 48,49, 53
3	VAV/CAV Box	2, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 41, 42, 43 ,44, 45, 46, 47, 50, 51
4	Whole Building Utilities	1, 4, 8, 9
5	Zone	52

tion can find 30% of the misclassified measurement points by using the probability vector outputted from the method. Another paper that also aims to automatically classify measurement points is [Hong et al., 2015b]. The solution is also here based on metadata but also time series data. The solution first trains statistic classifiers on a prelabelled building and then integrates these classifiers into another building regardless if the metadata is similar or not. The method is based on feature design for both metadata and time series data. This data is processed by support vector machines, random forests as well as logistic regression models and the result from each classifier is then weighted together. The accuracy ends up at 63% maximum while only being able to classify 36% of the measurement points with more than 85% likelihood. The authors emphasise that this could be a first step towards automatically solving the classification problem even if buildings have very different metadata. This paper is based on data in data set *B*.

For the association problem it has been much harder to find prior work. Except for the proposal in [Balaji et al., 2015] our effort to find papers dealing with similar problem formulation has been fruitless.

2

Machine learning theory

In [Chollet, 2018] François Chollet states that machine learning arises from the question whether a computer could *"go beyond 'what we know how to order it to perform' and learn on its own how to perform a specified task?"* Therefore, machine learning differs from classical programming. Historically when programming we use rules to tell a computer how to process data and the computer will then give us an output that may be interpreted as an answer. When machine learning is used we provide both input data and/or output data and let the computer find the rules mapping this data. These rules can later be used on similar data to generate new outputs [Chollet, 2018]. The rules are a way for the algorithm to transform the input data into meaningful information and could for instance be linear and nonlinear operations or coordinate changes [Chollet, 2018]. To create these rules a machine learning algorithm needs to be trained with informative examples and have a metric of how well the algorithm is performing. The metric is used as a feedback within the algorithm to update parameters so that the algorithm searches for the best solution[Chollet, 2018].

Providing an algorithm with an answer as well as input data is called a supervised learning. The supervised learning algorithms require answer or labels to train a model [Shukla and Fricklas, 2018]. The opposite of supervised learning is unsupervised learning. By only using input data it may still be possible to find structure in the data. Examples are clustering and dimensionality reduction. A clustering method will divide the different inputs into groups where the individuals are alike in some way, without knowing how the data preferably should be grouped [Shukla and Fricklas, 2018]. Dimensionality reduction on the other hand will allow redundant parts of the input to be discarded. For instance, assume the input is a vector containing the mean value amongst other values [Shukla and Fricklas, 2018]. If the mean value is the same for all individuals, then that value would not contribute with any information and may just as well be removed. More complex examples of dimensionality reduction are principle component analysis and autoencoders. For this thesis supervised learning will solely be considered.

As stated above, the aim of machine learning algorithms is to find rules that can give correct answers to never seen problems. Therefore, it is important that the algorithm finds rules that are specific enough to solve the problem but generalised enough not to fit the rules to the exact pattern found only in the input data on which the algorithm has been trained on, such as noise. When an algorithm has adjusted its rules too specifically to patterns in the data it has been trained on, it is referred to as being overfitted [Chollet, 2018].

A part of machine learning is deep learning algorithms. The deep here is not connected with a deeper or greater understanding but instead refers to the number of layers in a successive model, i.e. a model in which data flows in a single direction from input to output. It is basically a method using many steps to learn the useful representation of the input data. This enables the method to combine input data in many ways and possibly extract more interesting information than algorithms that simply use the input data as is. One commonly used deep learning algorithm is a method called neural networks [Chollet, 2018], which will be presented in greater detail later in this thesis. Algorithms that are not deep are sometimes called shallow and in some cases, e.g. when there is not enough data, these may be a better solution than using deep learning [Chollet, 2018].

Machine learning, and especially deep learning, is in general an area that is built on experimental findings rather than mathematical theory and in [Chollet, 2018] Chollet states:

> *Machine learning isn't mathematics or physics, where major advances can be done with a pen and a piece of paper. It's an engineering science.*

A crucial part of developing new and well performing algorithms hence includes having the right data and hardware [Chollet, 2018].

2.1 Neural network

A neural network consists of so called neurons gathered in layers. A graphical representation of a neural network can be found in Figure 2.1. As the network receives an input every neuron in the input layer, orange in Figure 2.1, performs some kind of calculation. They will then transmit their output to the neurons they are connected to. Every neuron in a layer may be connect to any neurons in the successive layer. The neurons in the next layer will process their input and transmit their output. This will continue until the last layer of neurons has processed the input. The output from the network will then be used to calculate the value of a loss-function, sometimes referred to as cost function. The value of the loss function will be a measure of how well the network generates the correct answer for known input [Karpathy, 2017d].

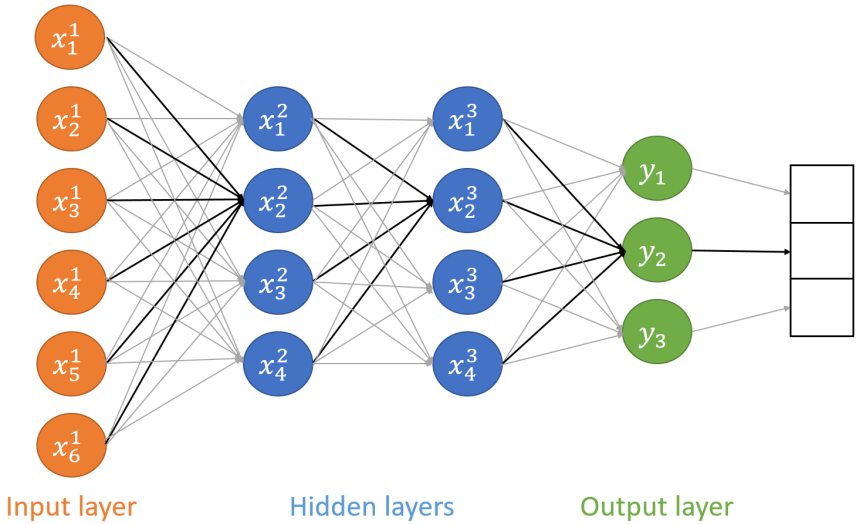


Figure 2.1 Schematic figure over a neural network with an input layer (in orange) consisting six neurons, two hidden layers (in blue) consisting of 4 neurons each and finally an output layer (in green) consisting of three neurons. The output of the network is collected in a vector, here represented by the white boxes.

Every neuron in a network can have a bias and weights [Karpathy, 2017a]. A linear operation in a neuron could for instance be $x_{out} = wx_{in} + b$ where x_{out} is the output, x_{in} the input, w the weight and b the bias. The equation can easily be extended to vector form and thereby include the case where a neuron receives multiple inputs as shown in Figure 2.1 [Karpathy, 2017c]. The bias and weights are updated when the network is trained depending on the loss. The exact nature of the update also depends on a few optimisation options. The update will for instance also be affected by the kind of layers used, how deep the network is (how many layers are used), how many neurons there are in every layer, the activation function of the neurons, the kind of loss function and the optimiser [Karpathy, 2017a].

One common type of layer is the fully-connected layer. In this type all neurons are connected to all neurons in the previous layer, hence fully connected [Karpathy, 2017b]. These layers are commonly used for simple data that can be stored in a two dimensional vector [Chollet, 2018].

An activation function is a function that will process the output from a neuron before it is passed on. It will typically be non-linear [Karpathy, 2017d]. A popular activation function is called rectified linear unit or ReLU, seen in equation (2.1).

$$f(x) = max(0, x) \tag{2.1}$$

Historically, hyperbolic tangent has been used as an activation function in com-

bination with gradient descent optimisers (see below), but models converge faster using ReLU as activation function. An undesirable feature of ReLU is its ability to "kill" neurons, i.e. the output from the unit or neuron will end up always being zero [Karpathy, 2017d]. A solution to this problem can be using the so called Leaky ReLU, defined in equation (2.2).

$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases} \tag{2.2}$$

where α is a small parameter that sets the slope for the function in the left half plane. This function does sometime result in better performance [Karpathy, 2017d].

A couple of other activation functions are ELU and SELU, where SELU is the scaled variant of ELU [Chollet, 2015a]. Exponential Linear Units or ELU has been shown to perform better than ReLU and is defined according to equation 2.3 where α is a user defined parameter and λ a fixed constant, see [Clevert et al., 2015].

$$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases} \tag{2.3}$$

For the output layer there are some recommendations depending on the problem that is to be solved. For binary classification a sigmoid function is usually a good option while for multiple excluding classes softmax, found in equation (2.4) is preferred [Chollet, 2018; Bishop, 2006].

$$\mu(x)_k = \frac{e^{x_k}}{1 + \sum_j e^{x_j}} \tag{2.4}$$

This equation is also known as the normalised exponential [Bishop, 2006]. Here the value for the kth output is normalised by all the exponential value of all j outputs.

While training a network one aims to minimise the loss function, which will be a measurement of how well the network is performing [Bishop, 2006; Chollet, 2018]. Hence the choice of loss function will affect the performance of the network extensively. For problems that are common within machine learning there are some guidelines one may abide when it comes to picking the loss function. For binary classification (classifying a given input as true or false) binary cross entropy is recommended while instead categorical cross entropy is preferd for multiclass classification[Chollet, 2018; Bishop, 2006]. This will only work if the classes are mutually exclusive, i.e. a signal cannot both be a temperature measurement and an air flow setpoint [Shukla and Fricklas, 2018]. The cross entropy is in general a measurement of the difference between two probability distributions and it is defined in equation (2.5)

$$E(\vec{x}) = \hat{y}_n ln(y_n) + (1 - \hat{y}_n) ln(1 - y_n) \tag{2.5}$$

where $\hat{y}_n$ is the prediction for class n and y_n is its true response [Bishop, 2006].

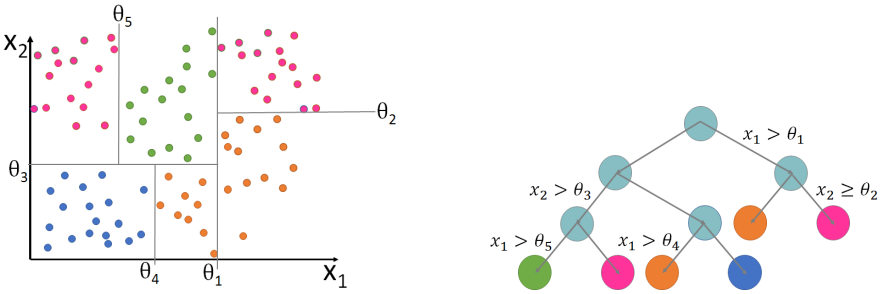
In the case of classification this will be the distribution of the answers or ground truth and the predictions or outputs [Chollet, 2018]. It is important to be aware of that it is not always possible to use the metric which best describes how well the model is performing as loss function. This is due to demands on the loss function to be differentiable and computable using only a small batch of data [Chollet, 2018].

The loss will not be fed back directly to update the weights, it is processed by an optimiser first. Usually the optimiser implements a gradient descent method. If a network has multiple outputs, as in Figure 2.1, there will be as many values of the loss function, which will be averaged over before being used by the optimiser [Chollet, 2018]. A gradient descent method will basically update the weights by taking a small step in the negative direction of the gradient e.g. $\vec{w}_{t+1} = \vec{w}_t - \mu \nabla \varepsilon(\vec{w}_t)$ [Bishop, 2006], where ε is the loss function and $\vec{w}_t$ is the current set of weights. The more advanced method RMSProp is usually a good choice [Chollet, 2018] but the Adam optimiser is also a contender [Shukla and Fricklas, 2018].

Adam is an optimiser utilising first order gradients and is suitable for problems with many parameters and/or large sets of data. It has been found to be both robust and working well for non-convex problems [Powers, 2015]. Although it should be mentioned that it also sometimes fails to converge [Reddi et al., 2018]. Adam is based on RMSProp that utilises the first order gradient and normalises this by using the magnitude of previous gradients [Hinton et al., 2014].

A main objective when using neural networks for classification is for the network to find relationships in data. It is unwanted to model noise into these relationships. This, because the noise will not be the same for all inputs of the same kind but the general behaviour will be. The network has overfitted if characteristics in noise of the training data is a part of the model [Karpathy, 2017d]. There are several options to consider if the model is showing signs of overfitting. Some of them are introducing dropout [Karpathy, 2017e] and reducing the dimensionality of the input [Karpathy, 2017b]. Dropout is set as a rate, e.g. 0.2, for which any given neuron will have that probability to not be active during certain training steps. If the neuron is inactive the output from it will be zero. This reduces the risk of overfitting [Karpathy, 2017e].

Exactly how these different options should be combined strongly depends on the problem. Choosing a well performing neural network design is not straight forward and is usually based on experience rather than theory although some rules of thumb exist [Chollet, 2018].



(a) Classified data in feature domain where different threshold values θ_i divides the feature space.

(b) Decision tree classifying data with the thershold values θ

Figure 2.2 A decision tree classifies data by applying sequential logic test on different feature parameters.

2.2 Ensemble method

An ensemble method attempts to train several predictors, instead of one as in classical machine learning, to solve the same problem. The predictors are also called base learners and the result from individual base learners are in an ensemble method combined to form a better predictor. The base learner can be homogeneous, i.e. of one type, or heterogeneous. The gain of using ensembles is that the combined result is often more generalised than that of a single base learner. Commonly weaker machine learning methods can also come together and form a strong predictor in an ensemble [Zhou, 2012]. Two ensemble methods are random forest and gradient (tree) boosting which are further presented in this section. Both of these methods have decision trees as their base learners.

Decision tree

A decision tree is a feature-space based algorithm. A tree applies a series of logical tests to divide the data into smaller subsets. In Figure 2.2(a) we can see a two-dimensional dataset. The grey lines in the figure partition the datapoints by colour and each colour corresponds to a class. A classification decision tree, seen in Figure 2.2(b), divides the input space $\{x_1, x_2\}$ into the same regions as the grey lines create in Figure 2.2(a). The logical test in each node split corresponds to one of the grey lines in Figure 2.2(a) and a trait of decision trees are that these lines will always be parallel to the original axes [Bishop, 2006]. Each end node in the tree, or more commonly referred to as *leaf node*, denotes a certain class or here colour. If an example ends up in that node it is labelled with that particular class.

Training a decision tree comes down to deciding both what variable, x_1 or x_2, will form the split at each node and the corresponding threshold value θ_i. These splits are evaluated through a split evaluation function [Bishop, 2006]. Two common func-

tions for evaluating splits within a classification tree are the Information Gain IG and the Gini Impurity GI. The Information Gain is maximised with regard to splitting on feature f by minimising equation (2.6)

$$\mathrm{IG}(f) = \sum_{v \in values(f)} p(x_f = v) H(C|x_f = v) \tag{2.6}$$

where f the specified feature, x_f is the value of the feature in example $\vec{x}$ and v is the proposed split value for f. $H(C|x_f = v)$ is the entropy, defined in equation (2.7), of a set were $x_f = v$.

$$H(C|x_f = v) = -\sum_{t=1}^{K} p(C_t|x_f = v) \log_2 p(C_t|x_f = v) \tag{2.7}$$

where $p(C_t|x_f = v)$ is the probability of an example belonging to class C_t if $x_f = v$ [Berzal et al., 2003]. The Gini Impurity attempt to minimise the impurity in the subsets created by the split by minimising equation (2.8)

$$\mathrm{GI}(x_f) = \sum_{v \in values(f)} p(x_f = v) G(C|x_f = v) \tag{2.8}$$

where $G(T)$ is defined as

$$G(C|x_f = v) = 1 - \sum_{t=1}^{K} p^2(C_k|x_f = v). \tag{2.9}$$

A regression tree works in a similar fashion, but here each leaf node represents a value. The splits are evaluated as the split that most decreases an error estimation function $R(T)$, i.e. maximises $R(T) - R(T_L) - R(T_R)$ where T is the exampel set before the split and T_L and T_R are the examples sets in the right and left subnodes respectively after the split. Traditionally, the mean square error, mse, presented in equation (2.10) has been used as an error estimation function [Breiman et al., 1984].

$$R_{mse}(T) = \frac{1}{N} \sum_{t=1}^{2} \sum_{\vec{x}_n \in t} (y_n - \bar{y}(t))^2 \tag{2.10}$$

where, N is the total number of examples in the node, t spans over the subnodes, which are the nodes created when a node is split, and $\bar{y}(t)$ corresponds to the mean output of that subnode. However, [Jerome H. Friedman, 2001] has proposed another criterion, the improved least squares criterion found in equation (2.11)

$$R_f(T) = \frac{K_1 K_2}{K_1 + K_2} (\bar{y}(1) - \bar{y}(2))^2 \tag{2.11}$$

where $K_{1,2}$ are the number of examples in each subnode. This will later be referred to as the *Friedman mse*.

A decision tree is trained through two main phases; the growing phase and the pruning phase. Some pre-pruning also takes place during the growing of the tree. Growing a tree means to recursively partition the samples in such a way that each leaf node, or end node, of the tree is associated with only one class [Kotsiantis, 2013]. The pre-pruning during this phase prevents splits that do not meet certain criteria, for instance the minimum number of samples in a leaf node or minimum number of samples in each partition. The pruning phase attempts to generate subtrees from the grown decision tree. This is to generalise the decision tree and to not overfit to training data[Kotsiantis, 2013].

Random forest

The random forest algorithm consists of multiple decision trees which makes this approach more powerful than ordinary decision trees. The increased number of trees mean a significant increased classification accuracy [Suthaharan, 2016]. Random forest is a so-called bagging method. The bagging algorithm attempts to train base learners to be as independent of each other as possible. In the optimal case, each base learner would get its own training set, but as training data tends to be limited this would result in every unique training set being very small and is therefore not feasible. Instead the bagging algorithm adopts bootstrap sampling. Bootstrap sampling is a method that creates new sample sets of equal size from the original training set by the following approach. Let the training set contain m examples. The subsets are then generated by drawing m samples with replacement from the training set [Zhou, 2012], i.e. every tree gets an training set of equal size to the original training set. Having multiple cases of identical examples occurring in the training sets decreases the computations required for each tree to find the optimal split [Suthaharan, 2016]. Hence every base learner will train on the same amount of data, but different sets. The final choice of the ensemble is commonly done by popular vote, but it is also possible to look at percentage of how the votes were spread over all classes [Zhou, 2012].

Gradient boosting

A boosting method, such as gradient boosting, has as its primary objective to improve weak learners. This is performed iteratively during training by training additional weak learners based on problem areas for the existing ones. Say for instance that the first learner of the ensemble is good at classifying class 1 and 2, however the errors for class 3 are large. In the next step a base learner will be trained on a set where class 3 is heavily weighted. The base learners are finally combined in a weighted fashion to ensure a good overall result, after which an error is calculated and used to decide on what data to train the next iteration of learners [Zhou, 2012].

For a set of responses y and inputs $\vec{x} = \{x_1, \ldots, x_n\}$ the boosting algorithm aims to find the mapping $F(\vec{x})$ between the labels y and the data $\vec{x}$. The algorithm achieves this by minimising the expected value of the loss function $L(y, f(\vec{x}))$ where $f(\vec{x})$

is the estimation of $F(\vec{x})$. This is normally the negative log-likelihood function. In equation (2.12) we find this loss function for the multi-class case [Friedman, 2002].

$$L(\{y_k, f_k(\vec{x})\}_1^K) = -\sum_{k=1}^{K} y_k \log p_k(\vec{x}) \tag{2.12}$$

where $y_k = 1$ if the sample is labelled to class k, $p_k(\vec{x})$ the learners prediction that y_k equals to one knowing $\vec{x}$ and $f_k(\vec{x})$ can be though of as the predictor for class k. $p_k(\vec{x})$ can be rewritten as

$$p_k(\vec{x}) = \frac{e^{f_k(\vec{x}))}}{\sum_{t=1}^{K} e^{f_t(\vec{x})}}.$$

using the symmetric multiple logistic function for $f_k(\vec{x})$. Note that the loss function is a function of a sum over K number of pairs $(y_k, f_k(\vec{x}))$. This results in that one tree per class k is fitted for every time the loss function is calculated. This is done in each iteration step m [Jerome H. Friedman, 2001]. Meaning that the solution can grow very complex with the number of classes.

The gradient boosting algorithm approximates $F(\vec{x})$, the true mapping between labels and data, by a weighted sum, f, of base learners, h. Here the base learners are trees and the weighted sum is expressed in equation (2.13).

$$f(\vec{x}) = \sum_{m=0}^{M} \mu \beta_m h(\vec{x}, \vec{a}_m) \tag{2.13}$$

where M is the total number of boosting steps, or iterations, β_m and $h(\vec{x}, \vec{a}_m)$ are the base learners weights and base learners respectively calculated at the mth training iteration and μ is the shrinkage factor or *learning rate*. $\vec{a} = \{a_1, a_2, \ldots\}$ are the parameters of the learner, i.e. splitting feature, splitting value and tree configuration. $f(\vec{x})$ is calculated in a series of iterations over m, called boosting steps. In each of the iterations a base learner and a weight are estimated and then the weighted learner is added to the learner from the previous step. The step in the direction of the new learner is keept small by the learning rate in order to acheive higher generalisation. The base learner is selected to minimise its squared difference to the negative gradient of the loss function with regard to the output from the base learner of the previous step. The weight minimises the loss between the true response of each individual and the output from the new learner [Friedman, 2002].

2.3 Support vector machine

Support vector machine is an algorithm primarily designed for binary classification problems. The main objective is to divide the data set into two regions or classes

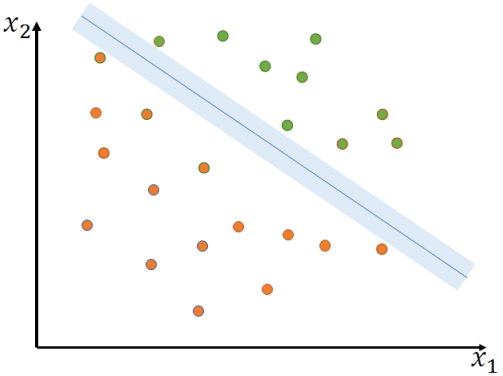


Figure 2.3 Linear support vector machines divides the data domain with a straight line. The algorithm attempts to maximise the distance to the data points, i.e. the light blue area should be as wide as possible.

linearly as seen in Figure 2.3. The algorithm tries to fit a line, see equation (2.14), to maximise the distance to the closest points on each region [Suthaharan, 2016].

$$\mathbf{w}\mathbf{x}^T + \gamma = 0 \tag{2.14}$$

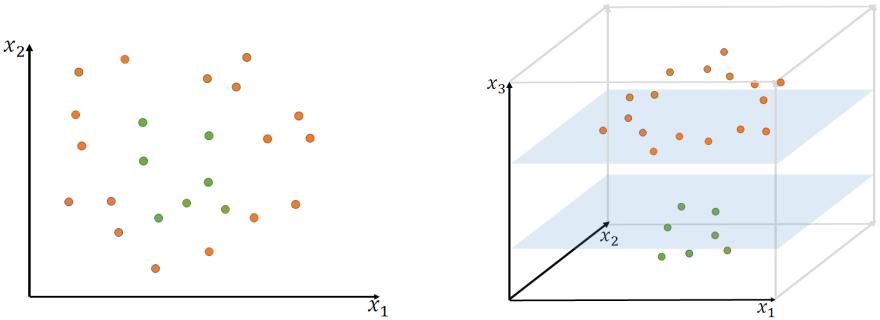
This line separates the data points according to

$$\begin{aligned} D_1 &= \{\mathbf{w}\mathbf{x}^T + \gamma \leq 0\} \\ D_2 &= \{\mathbf{w}\mathbf{x}^T + \gamma > 0\}. \end{aligned} \tag{2.15}$$

The objective is that the region D_1 and D_2 will contain one class each. In practise the algorithm finds two parallel lines, in Figure 2.3 represented by the edges of the light blue area, to locate the boundary of each class and then maximises the distance between these two lines. The algorithm can also handle multiple class problems by combining two-class support vector machines [Suthaharan, 2016].

Support vector machines are easy enough to understand when the data can be separated by a straight line. Figure 2.4(a) however displays a dataset where the data cannot be linearly separable.

A kernel function can be used to transform the data into a higher domain where the classes can be separated linearly [Suthaharan, 2016]. This can be seen in Figure 2.4(b) where the green and orange dots in Figure 2.4(a) have been moved from a 2D space into a 3D space. This new space is chosen with care so that the dots are separated by the light blue planes. Increasing the dimensionality significantly often makes the calculation of new coordinates computationally heavy. To handle this, the so-called kernel trick is often used in support vector machines. This means



(a) Original feature domain. The green and orange smaples can not be divided by a stright line.

(b) Transformed feature domain. The green and orange samples can be divided by a simple plane.

Figure 2.4 A non-linear support vector machine transforms the data using a kernel function to a space of higher dimensionality to separate the data points.

that the kernel function is used to determine the distance between the dots in the new space without calculating the new coordinates. As an example, the mapping; $\Phi(u_1,u_2) = (au_1^2, bu_2^2, cu_1u_2)$, where u_1 and u_2 are coordinates in the original space, can be used to transform data from the 2D space to the 3D space. Further on the kernel function calculates the similarities of two points in the new space as the inner product between them i.e. $\kappa(\vec{u},\vec{v}) = \Phi(u_1,u_2)\cdot\Phi(v_1,v_2)$. The kernel trick, demonstrated below, lies in the fact that only information from the 2D space is required to calculate the inner product in the 3D space.

$$\begin{aligned}
\Phi(u_1,u_2) &= (au_1, bu_2, cu_1u_2)\\
\Phi(v_1,v_2) &= (av_1, bv_2, cv_1v_2)\\
\kappa(\vec{u},\vec{v}) &= \Phi(u_1,u_2)\cdot\Phi(v_1,v_2) =\\
&= a^2u_1^2v_1^2 + b^2u_2^2v_2^2 + c^2u_1v_1u_2v_2
\end{aligned}$$

By selecting $c = 2ab$ this can be simplified to

$$\begin{aligned}
\kappa(\vec{u},\vec{v}) &= (au_1v_1 + bu_2v_2)^2 =\\
&= \left(\begin{bmatrix}u_1 & u_2\end{bmatrix}\cdot\begin{bmatrix}a & 0\\ 0 & b\end{bmatrix}\cdot\begin{bmatrix}v_1\\ v_2\end{bmatrix}\right)^2
\end{aligned}$$

A special type of support vector machine used for classification implements a Radial Basis Function, a kernel function specified in equation (2.16) [Vert et al., 2004].

$$K(\vec{v}_i,\vec{x}_j) = e^{\frac{-\|\vec{x}_i-\vec{x}_j\|^2}{2\sigma^2}} \tag{2.16}$$

where σ is a fixed constant and $\|\vec{x}_i - \vec{x}_j\|^2$ is the squared Euclidean distance between two points $\vec{x}_i$ and $\vec{x}_j$. Such a support vector classifier is implemented in the libraries of SKLEARN which is a free software machine learning library [Pedregosa et al., 2011]. The kernel function is fixed but the classifier has input arguments such as C and γ. C is the parameter that handles the trade-off between misclassification of the training individuals against the decision surface's simplicity, the light blue planes in Figure 2.15. A low value ensures a smooth surface as in Figure 2.15 while a high value ensures a focus on classifying all individuals correctly [scikit learn developers, 2017]. γ is a parameter that decides how far (in the domain of the input) the effect of a single individual will reach. If the value of γ is small individuals that are quite far from each other will be considered similar, a too small value would result in a model unable to separate anything in the whole training set. A large value instead results in that individuals need to be closer to be considered similar and a too large value will end up overfitting the model regardless of the value of C [scikit learn developers, 2017].

2.4 Feature extraction

A feature is a measurable representation of characteristics that can be observed in the data [Shukla and Fricklas, 2018]. Features are used to describe real data in a simplified manner. The real world data might be too complex or too large to handle and features can reduce the complexity by not including every detail but capturing the bigger picture or the most important part of the data [Shukla and Fricklas, 2018].

How well a machine learning algorithm performs depends highly on the extracted features hence this must be done with great thought[Shukla and Fricklas, 2018]. It is worth noticing that more features will not necessarily enhance performance in a machine learning algorithm. Instead performance might even suffer from an excessive number of features.

In classification, features should be designed to ensure that the algorithm has enough information to separate the different classes. The algorithm should optimally be provided with only as many features needed to just tell the classes apart[Shukla and Fricklas, 2018].

In [Hong et al., 2015a] a feature extraction is performed on n time frames of every individual. For each of the time frames the median and variance are calculated. These are then put into vectors. Thereafter the minimum, maximum, median and variance is calculated for the vectors and used as features for the implemented machine learning model [Hong et al., 2015a].

In this thesis we are considering two different kind of problems, *classification* and *association*. The problems are different and it is not unlikely that the machine learning algorithms might need different information and therefore different features.

Some statistical metrics will be used for the extraction of features, information of the utilised metrics is found below. From here on we will denote x as the measured data from a distribution, m_x as the mean value, $P[\,]$ as the probability and $E[\,]$ as the expected value.

Probability Density Function or PDF for a continuous sample space is defined as [Jakobsson, 2013]:

$$f_x(\alpha_1,...,\alpha_P) = \frac{\delta^P P[x_1 \leq \alpha_1,...,x_P \leq \alpha_P]}{\delta\alpha_1,...,\delta\alpha_P}$$

This is the first partial derivative of the probability with regard to all $\alpha_p, p \in [1,P]$. The value of the PDF can for a given sample, α_p, be thought of as a relative likelihood that the value of a random variable would be equal to the value of the sample.

Skewness is the third central moment and also a metric of how asymmetric a PDF is which is illustrated in Figure 2.5, see [Hyvärinen et al., 2018].The normal distribution is seen in black and has skewness equal to zero. Skewness is defined as:

$$\mu_3 = E[(x-m_x)^3]$$

Kurtosis or the normalised kurtosis is the fourth order statistics [Hyvärinen et al., 2018]. The metric strongly indicates if the distribution of x is Gaussian or not since for Gaussian distributions kurtosis will be zero as depicted in Figure 2.6. The definition of normalised kurtosis follows:

$$\text{kurt}(x) = \frac{E[x^4]}{(E[x^2])^2} - 3$$

Discrete Fourier Transform or DFT is defined as:

$$X_{\text{DFT}} = \sum_{n=0}^{N-1} x(n) e^{-i2\pi\frac{k}{N}n} \qquad k = 0,1,2,...,N-1$$

where $x(n)$ is the signal measured in the time domain over N time samples and $i = \sqrt{-1}$ [Swartling et al., 2016a]. Unlike a continuous Fourier transform the DFT is not calculated using an integral but a summation [Swartling et al., 2016b].

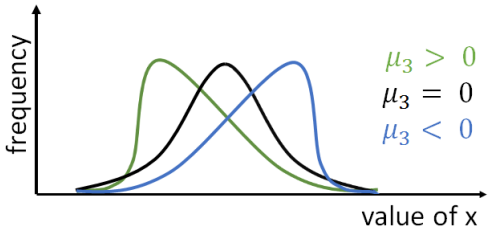


Figure 2.5 Distributions from a PDF with different values of skewness, μ_3. A normal distributed PDF is presented in black.

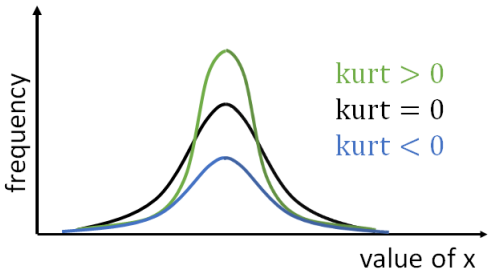


Figure 2.6 Distributions form a PDF with different values of kurtosis, kurt. A Gaussian distributed PDF is presented in black

2.5 Evaluating models

It is a natural assumption that the objective when working on a problem, independent of the work area, is to find a method that solves it optimally. A way of evaluating the results is therefore required. With machine learning the aim is to find a model that finds generalised rules meaning that it performs equally well on training data as for data never presented to the model before [Chollet, 2018]. For this reason, the data set used is normally split into different parts, one for training, one for testing and finally one for evaluation. Only splitting a data set into two parts will not be adequate since the usual procedure when developing a model is:

- Design models and training them using the training set.
- Test models using the test set. From the results changes can be made to models and models can be compared.
- Evaluate the final model.

Clearly the models are tuned with regard to the training set but also, in a way, with regard to the test set. Therefore, there is a risk of overfitting the model to not only the training set but the test set as well. A little bit of information about the test set will be brought into the model when a modelling decision is based on results from the test set. By using a separate evaluation set it is possible to train the model, tune design options and change parameters while still evaluating the model on never before used or presented information [Chollet, 2018]. Usually the data is split so 70% is used for training and 30% for testing and evaluating the model or 60% for traing and 20% for testing and evaluating respectivly [Shukla and Fricklas, 2018].

Evaluation of a model needs at least one metric that provides information about how well the model performs. A common metric used during both training and evaluation is accuracy [Chollet, 2018]. There are a couple of definitions but in thesis it is defined as how many times the output or predication from a model matches the true answer divided by the number of individuals, see equation (2.17).

$$x_{\text{acc}} = \frac{\sum_{n=1}^{N} y_n^{\text{match}}}{N} \tag{2.17}$$

where N is the number of individuals and y_N^{match} is one if the prediction matches the key and zero otherwise. Accuracy can also be defined as

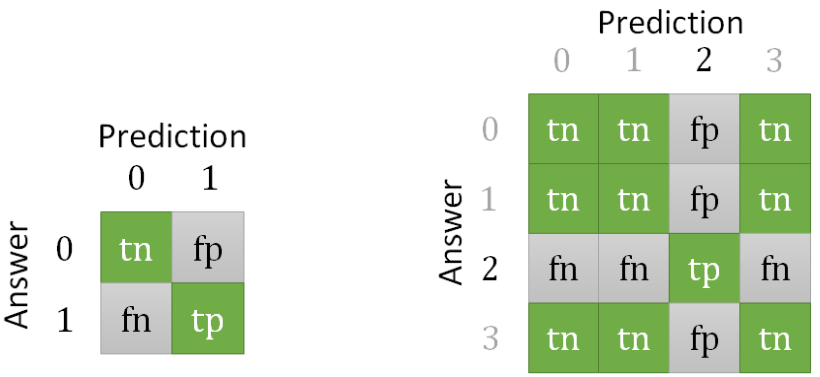
$$x_{\text{acc}} = \frac{tp}{tp + fn} \tag{2.18}$$

and by this definition, accuracy is identical to what is called recall. In the equation *tp* and *fn* refers to *true positives* and *false negatives*. A handful of other metrics are based on these two and *true negatives* and *false positives*, *tn* and *fp* respectively. In the binary case these values are intuitive. As seen in Figure 2.7(a) *tp* is obtained when the prediction or output as well as the answer takes the value 1 while *fp* is obtained when the prediction is 1 and the answer is 0. In a similar manner *tn* is added to when both prediction and answer is 0 while for *fn* the prediction must be 0 and the answer 1 [Powers, 2007]. In Figure 2.7(b) a similar matrix is presented. If each individual class is studied separately the same analogy may be used. Take class 2 as an example; if the prediction is 2 and the answer is 2 there is a *tp* while if the answer is anything but 2 there is a *fp*. If the prediction is anything but 2 while the answer is 2 there is a *fn* and otherwise a *tn* will be obtained.

If the accuracy is calculated at the end of training epoch for both test and training set it is possible to spot if a model is overfitting. The accuracy difference, defined in 2.19, is a metric for overfitting.

$$\text{Accuracy diff} = x_{\text{acc}}^{\text{train}} - x_{\text{acc}}^{\text{test}} \tag{2.19}$$

If the value is large, above a few percentage points, it means that the model has based rules on structure only found in the training set meaning that it has overfitted



(a) Binary case.

(b) Multiple class case. Here the matrix is with regards to class 2.

Figure 2.7 Confusion matrices showing the location of *true positives* and *negatives* as well as *false positives* and *negatives*

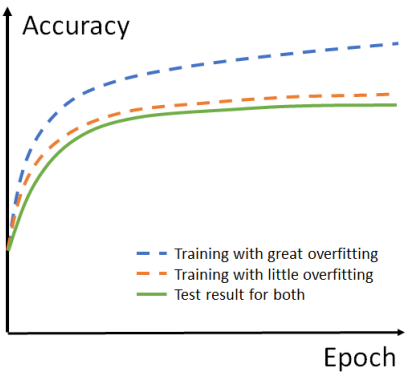


Figure 2.8 Accuracy during training for test and training set on an overfitted model, in blue, compared to that of a non overfitted model, in orange. Note the difference in accuracy for the training set, in blue or orange, and the test set in green.

[Karpathy, 2017f]. In Figure 2.8 accuracy curves for the training set and test set are depicted. Here it is clear that the accuracy difference after training will provide insight into weather a model has overfitted to the training data or not.

As mentioned above, *tp*, *tn*, *fp* and *fn* form several metrics that may be used for analysis. The accuracy or recall could for instance be derived from equation (2.18). A complementary metric to recall is what may be called inverse recall, specificity or true negative rate [Powers, 2007]. While recall or accuracy contains information of

how well the rule/model is able to find the correct positives, specificity contributes with the same information for the negative examples [Powers, 2007]. The definition of specificity is:

$$x_{\text{specificity}} = \frac{tn}{fp + tn}$$

Accuracy and specificity are biased for the case when the data is skewed by the number of positive and negative examples. Picking the best model so that these metrics have the highest values does not necessarily yield the truely best performing model [Powers, 2007]. Therefore it is wise to study other metrics as well.

One interesting way of viewing the performance of different models is to put these in a so called ROC graph. The ROC graph has recall, here equivalent to accuracy, as y-axis and fall out as x-axis. Fall out is defined as:

$$x_{\text{fallOut}} = \frac{fp}{fp + tn}$$

A perfect classifier would then be placed at [0,1] in the graph, equivalent to accuracy equal to 1 and fall out equla to 0, while a poor model would be far from the y-axis and close to the x-axis [Powers, 2007]. In [Hong et al., 2015a] both recall and fall out is used for evaluation as well. It should however be noted that due to the bias in accuracy and specificity, the ROC graph is only feasible for binary classification.

Informedness, $x_{informed}$, is a metric created specifically to be unbiased. It is the probability that the model is informed regarding how well a random method would perform. The metric describes how *informed* the model is while making its predictions. An image recognition classifier on a set with 95 images of cats and five of dogs could theoretically achieve high accuracy simply by classifying everything as cats. The informedness measure for this model would be less than zero, i.e. -0.05. This is a result of the model not performing better than simply guessing. A perfect model, a model that would classify all cats and dogs correctly, would get an informedness score of 1. The mathematical definition follows below:

$$x_{\text{informed}} = \frac{tp}{tp + fp} - \frac{fp}{fp + tn} \tag{2.20}$$

3

Execution

3.1 Preparing data for machine learning

Machine learning algorithms are built and trained on data in a specific format. The datasets in this project had different formats making it necessary to handle the datasets individually. A script was constructed with one function for each dataset. It handled the different data formats and reformatted them. This meant that the following scripts could be guaranteed data on a specific format and not depend on which BMS the data was originally logged in.

Another issue has been the quality of the data. Data sampled in a BMS can have all sorts of errors. It can be wrongly measured, meaning too high or low values or NaN values, the sensor can be turned off for long periods of time, each measurement point can be recorded non-chronologically, the times between the measurement points can be irregular due to delays in the computer systems. All of this makes guaranteeing data sampled and measured correctly and in the same way impossible. Therefore the datasets are pre-processed through a script. The script takes files generated in the formatting data script as input together with required sampling time and window length. In order, the steps are as follows:

Sorting The signals are individually sorted in chronological order.

Removing NaN-values All NaN-values are removed from the data.

Dividing into windows The signals are divided into windows of a length determined by the user, in this thesis 48 hours. This is done for two reasons: to shorten the data necessary for the algorithms to work on a real case and to create more examples for the algorithms to train on. The original sample time of the signal is estimated and if there are more than four samples missing the starting point of the window is reset.

Resampling The windows are resampled to the desired sampling time, in this thesis 5 minutes. The script uses linear interpolation but does not interpolate

between points of a certain time difference called gap time. Instead the function will start over trying to sample a new window at the end of the gap. The gap time depends on an estimation of the original sampling time of the signal.

Storing The windows' time stamps, measurements, class labels, signal names and potentially equipment labels are stored in lists which are converted to csv files. There are three sets of these data files one for training, one for testing and one for evaluation. About 70% of all examples are used for training while 20% and 10% are used for testing and evaluation respectively.

For data used in the classification problem these steps are taken individually per signal. For association data we hypothesised that time difference between event, defined among the features below, would be important. This made it necessary to ensure that all windows within the same equipment spanned over the same time frame. This was taken into consideration during the *Dividing into window*-step. In the *Resampling*-step the allowed gap time were up to four times the original sampling time for the classification problem and up to ten times for the association problem.

Finally, a script calculating the features, which would be used as input to the models, was created. The script was divided into two methods calculating features, one for classification and one for association. In turn these were divided into the different feature sets to enable examination of how different features affected the models. The function takes parameters that regulate the feature sets that are to be calculated, if the features should be normalised and whether or not individuals with zero mean and zero variance should be eliminated from the set. The features are normalised feature-wise. This is done by calculating the mean and standard deviation for each individual feature over all windows, e.g. the mean of all maximum values. Then every separate feature in every window is normalised with regard to the calculated mean and standard deviation for each feature. It is also possible to give a mean and standard deviation as input to use for normalisation. This is used to ensure that a training set and test set are normalised using the same parameters. The specific feature sets used for the classification problem are presented in Table 3.1.

For association the focus was to find features that could help determine if two signals were in sync. This meant a focus on when things happen rather than what happened. The following features were defined

Peaks Time stamps for the top five peaks in the data are calculated along with time stamps for the top five minimums.

Gradients Time stamps for maximum and minimum gradient are calculated.

Table 3.1 The feature sets for the classification problem shortly described and defined.

Description of feature sets for the classification problem	
Feature set	**Description**
stat	Simpler statistical measurements are included in this set. For each signal the minimum and maximum value is calculated along with the mean value and variance.
stat+	This set contains more advanced statistical measurements. As defined in section 2.4 skewness and kurtosis are calculated. The set also includes a test statistic and a p-value for the null hypothesis that the data is normally distributed.
grad	All features in this set are based on gradient measurements. The maximum positive and negative gradient are calculated together with the mean gradient which contains information about the drifting within the signals over time.
freq	This set contains frequency measurements. The data is transformed to the frequency domain using Discrete Fourier Transform and frequencies for the five highest energy peaks are selected. For the windows with data distributions that do not contain five energy peaks the remaining features, or in this case frequency values, are set to zero.
acf	The autocorrelation function is the basis of this set. The time lags were calculated for the three highest peaks in the autocorrelation.
mse	The mean square error between the data and a fitted polynomial of degree four is the only feature in this set.

Frequencies The data is transformed using Fast Fourier Transformation. Thereafter frequencies for the top five and minimum five energy peaks are extracted. Further on, the frequency for the median energy is calculated.

At last, it is also clear, both from the examples of signal names in section 1.2 and in [Balaji et al., 2015], that the signal names contain information on the class of each signal and the signal paths on which equipment the signals might belong to. A feature that in a useful way could represent this information would be valuable but one must remember the problem different naming conventions, mentioned in section 1.2, poses.

3.2 The classification problem

For all data sets used in this thesis we implemented three different machine learning methods; *random forest, gradient boosting* and *neural networks*. The aim was to examine different attributes within the data sets as well as conclude what machine learning method might be most valid to continue working with. Early, it became conspicuously clear that we for some of the data sets training sets had a skew distribution in the classes which we thought we needed to address. Therefore, we used weights to balance the impact of the feedback from examples in the models. The output for examples belonging to rarer classes were weighted higher than for those belonging to more common ones. Hence the output from examples belonging to rarer classes had a greater effect on the update of weights in models. For random forest and gradient boosting models, we balanced the feedback using the built-in option in the methods implementing the models in the package SKLEARN[Pedregosa

et al., 2011] in PYTHON. To balance the feedback for the neural networks we used methods in SKLEARN to calculate weights which could be fed to the models. They were implemented in KERAS, a PYTHON written high-level API [Chollet, 2015b], running on top of TENSORFLOW which is an open source software library for high performance numerical computation [Abadi et al., 2015].

After running the model, we evaluated the result by examining accuracy for the top one, three and five guesses. For top five this meant that if the correct class was among the five highest outputs in the output vector for an example, the algorithm was said to be correct for that individual. We also calculated the informedness of each model and the difference in accuracy between the training set and test set after training to measure overfitting. Finally, for all random forest models we calculated, using a built-in function within the classifier, the feature importance. This metric is a percentage weight every individual feature has for all the decisions in a model i.e. how many times a certain feature is used to split a node divided by the total number of splits. We looked at these by looking at a) the feature importance over each feature set varied over the individual models and b) the average feature importance for every feature over all models for each data set.

All implemented gradient boosting and neural network models had a few things in common. As for gradient boosting the tree parameters determining how many samples that are required for a node to split or to be a leaf node were set to the default values 2 and 1 respectively. As for neural network models they all consisted of an input layer with weights, different number of hidden layers and finally an output layer implementing the softmax activation function. All layers were of the type fully connected and the input layer always had the same number of neurons as the number of inputs provided i.e. if five features were fed to the model there would have been five neurons in the input layer. The output layer always had the same number of neurons as the number of classes in the data the model was trained on.

Data set *A*

To even have a chance at solving the real-world problem where one aimed to correctly classify signals into over 1000 classes one must first be able to solve a smaller problem well. Therefore we look at Data set *A* which only contains 5 classes. Here we aim to see how some basic changes in the parameters affected the outcome of the methods.

For random forest we examined how requirements for the minimum number of samples for a node to be split or to be a leaf node affected the solution. Requiring two samples to split and a minimum of one sample for it to be a leaf node is default for the methods in SKLEARN.

For gradient boosting we evaluated the effect of different maximum number of features considered for each split had as well as the impact the function used to evaluate

each split had. Friedman mean square error (Friedman mse) is default and recommended in SKLEARN for the split function.

For the neural networks we especially aimed to determine the effect of different feature sets and dropout. The number of neurons was varied with regard to the number or features used, i.e. in the larger feature set there are 20 features and the models had 20 neurons in the first hidden layer while the smaller feature set only had 11 features hence the first hidden layer only had 11 neurons in the models using this feature set. The number of neurons in succeeding layers are in a similar way dependent on the number of features except from the output layer.

Data set *B*

This data set is mainly interesting since it is the only one containing data from different buildings classified into the same classification system, however it also contains more classes, in total 21, compared to Data set *A*. Data set *B* contains three buildings, but as seen in Table 1.2 the signal types from all buildings do not overlap. We therefore chose to test the difference between testing and training on signals from the same buildings as well as training on data from one building and testing on data from another.

We created two sets of data for this data set. For Data set ***B1+2*** we mixed signals from buildings *1* and *2* for both training and testing and for Data set ***B2*** we used signals from Building *1* for training and signals from Building *2* for testing. Worth noting is that these two buildings only contain 16 of the 21 classes. By comparing these two sub-data sets we were provided with the opportunity to investigate how well the methods perform on completely new data, not just data from the same measurement points measured over a different time frame. To ensure this comparison was as fair as possible, the same models were used on both subsets.

Also for this data set we choose not to investigate the effect of all options in the methods. For random forest we fixated the required number for a node to split or to be a leaf node and instead focused on the other two parameters, the maximum number of features considered for each split and the split evaluation function. The default value for maximum number of considered features is the square root of the maximum number of features and Gini Impurity for the split evaluation function. Just as for random forest we choose to fixate some of the parameters altered for Data set *A* in the gradient boosting models, namely the maximum number of features considered for each split and the split evaluation function. Instead we altered between four different learning rates where the default value is 0.1. Finally, for the neural network models the same parameters were investigated as for Data set *A*, described in the previous section, i.e. dropout and feature sets.

Data set *C*

This data set only contains 53 classes but is classified according to the classification system holding about 1300 and the data set is still the most complex regarding the number of classes. This brings us as close to reality as we will get in this thesis. Hence it is of interest to consider how many combinations of parameters in the models affect the performance.

Random forest and gradient boosting models are quick to train and therefore we tested as many different combinations of parameters possible only by changing one at the time. For random forest models we primarily tried to alter the number of trees, maximum numbers of features considered for each split and the split evaluation function. After that we also tried to vary the number of samples required to split a node or for it to be a leaf node together with a variation in the maximum number of features considered for each split. For the gradient boosting models, we instead varied the number of trees, the learning rate, maximum numbers of features considered for each split and the split evaluation function.

For the Neural networks there are many parameters that could be changed and optimised for the problem. Although it would not be feasible to test all possible parameter combinations since the number would be too great for the training of them to be performed in a reasonable time. Therefore, the approach for choosing interesting models to test has been iterative. First some initial models are trained and tested and the results from these models became the basis of the choice of how to design the next set of models. For the neural networks the primary focus has been to begin investigation of the effect of dropout, the depth of the network, the number of neurons, the optimiser as well as some activation functions.

3.3 The association problem

Only Data set *A* and *C* were divided into subsystems or equipment. For Data set *C* these can be seen in Table 1.5. Data set *A*, as explained in section 1.4,had signals sorted depending on the rooms they were measured in. In this section these rooms are the subsystems for Data set *A*.

Data set *A*

We choose to start implementing a solution for the association problem with Data set *A* as it only included one type of equipment, i.e. rooms. All equipment also included five signals belonging to the same five classes. This was not the case for Data set *C*, which can be seen in Table 1.5 and Table 1.4, and therefore Data set *A* was easier to work with. The benefit of using the class information for this data set and also test how important some features could be for the models were obvious to us.

We divided the problem into four subproblems. Instead of asking the question which of all of these signals belong together we instead asked e.g. "*Which Room Air Humidity signals belong with which CO_2 signals?*", "*Which Luminosity signals belong with which CO_2 signals?*" and so on. On every subproblem we then trained support vector machine models with the kernel function defined in equation 2.16. Different models were created for the subproblems which allowed us to examine the effect the parameters C and γ, defined in section 2.3, had on the performance.

We wanted to provide the model with informative input data. As already stated in section 3.1, we aimed to find features which could provide information on whether or not signals were in sync but also an informative feature which could tell us something about how the path and signal names of signals correlated. A way of measuring if two signals have an event, such as a peak, at the same time is to take the two time stamps for this event and calculate the difference of these. The difference in the frequency peaks which a signal contains could also say something about the systems since there might be frequencies in common for an equipment. Therefore, we decided to not only calculate a feature vector for every individual but to use the difference between two feature vectors as input to the models instead. The signal path and name was incorporated into the feature vector. The path and name, of signals that were to be compared, was used and a measure of similarity was calculated by using DIFFLIB.SEQUENCEMATHER in PYTHON. The comparison generated a similarity metric between 0 and 1, where 1 meant identical strings. A value of e.g. 0.3 meant that the two paths had a common substring which was 30% of the total length of the path. It should be remembered that we did not have access to the original signal paths for this data set as mentioned in section 1.4. Instead we created our own path names containing the room number, class type and window number for that signal. Obviously, this made-up naming convention would come to affect the results from any model when these names would be used as input.

Data set *C*

As mentioned above, Data set *A* had the great advantage of having the same type of five signals in every subsystem. This was not the case for Data set *C* which becomes tremendously clear when one studies Table 1.5 and Table 1.4. One can also notice that there is not one signal class that is in common for all equipment of the the same type. This would make the kind of solution used for data set *A* much larger and much more time consuming to create. We therefore decided to implement a solution that would only use the information from the path to group signals by equipment. This solution did not contain any machine learning method and was instead purely based on a basic numeric comparison of the strings.

The string comparison method implemented here are based on the same principals as for the path feature in Data set *A*. However, first signal paths and names within each class were compared and all identical sub strings were removed. This was an attempt to remove class dependencies within the signal paths and names and thereby

increasing the relative equipment dependency. Then all reduced signal paths and names were compared to each other and a percentage measurement for the similarity was calculated using DIFFLIB.SEQUENCEMATCHER. Finally, a prediction was made based on a threshold value for that similarity. Different threshold values were tested.

4

Classification: results and discussion

In this section the results for the classification problem are presented and discussed for different data sets and models. The classification problem, described in section 1.2, is when we seek the class of which a measured signal belongs to. The results for the three data sets *A*, *B* and *C* are presented separately. For each data set models of the three methods random forest, gradient boosting and neural network were implemented and trained according to section 3.2.

4.1 Data sets

In the preparation of the data sets, presented in section 1.4, signals were divided into two-day-windows and resampled to have a sample time of five minutes. The windows were subsequently split into a training, test and evaluation set. For each data set, the number of individuals in each set can be found in Table 4.1. We can clearly see that Data set *A* is the smallest of the three, but considering that there are only 5 classes in that data set its size is not a big issue. For Data set *B1+2* and *B2* there were 16 classes but obviously not 16 times more examples in either of them. Data set *C*, being the largest set from the beginning, had 163 measurement points before being prepared and ended up being divided into about 37000 individuals.

As we divide the data into training, test and evaluation set we tried to create sets which were independent. This was because we wanted to be able to present exam-

Table 4.1 Number of individuals in each data set for training set, test set and evaluation set.

Number of individuals in each data set				
Data set	**Training set**	**Test set**	**Evaluation set**	**Total**
A	396	113	56	565
B1+2	2319	662	330	3311
B2	1185	324	161	1670
C	25741	7354	3677	36772

ples to the models that it had never seen before while evaluating it. We also tried to present the models with as varying examples as possible during training for them to be generalised to the overall behaviour within a type and not model behaviour specific to a sensor.

It is very likely that many of the individuals within the data sets were similar to each other. This also means that the training, test and evaluation sets were similar. This was partly due to signals being split into windows meaning that the windows are not independent from the beginning. Another reason for the sets being uniform was the low number of buildings in each set. Only Data set *B* contained more than one building and even in that case the number of buildings were not high. It could be that our training, test and evaluation sets were too similar to actually be good for testing and evaluating the models. One solution to this could be finding more data from other buildings classified into the same system and another to make sure that individuals from certain signals are only used for evaluating the model. Although for our data sets this approach could result in too few signals in the training set meaning too uniform data during training instead.

4.2 Data set *A*

In this section models trained on Data set *A* are presented. In Table 4.2, Table 4.3 and Table 4.4 model parameters and accuracy difference for random forest models, gradient boosting models and neural network models are presented respectively. For the random forest models, we choose to vary the number of samples required for nodes in the trees to be able to split and be a leaf node. The gradient boosting models differ in how many maximum features are considered for each split as well as what function is to be used for evaluating the split. For the neural network models we change the number of neurons and the dropout rate. The activation function used is LeakyReLU for all models. We came to only use it since the other common activation function, ReLU, exhibited the characteristic problem of killing neurons which resulted in awful performance for the neural network models. Therefore only LekayReLU is used for Data set *A* but also for Data set *B*, see the results in section 4.3 for the latter data set. Neither did we choose to perform a deep study of the effect of different activation functions since we in this thesis aimed to find models that had the potential of solving the classification problem rather than being optimally tuned for it.

Let us now consider the far-right column containing the accuracy difference, as defined in equation (2.19). For random forest models in Table 4.2 we notice that all models have an accuracy difference well below one percentage point. In Table 4.3, showing gradient boosting models, the result is slightly better as the corresponding value is constantly zero. This means that the algorithm performs perfectly on both training and test set. Finally, in Table 4.4, we see how the neural network models

Table 4.2 Random forest models for Data set *A*.

Overview of random forest models on Data set *A*						
Model	**Trees**	**Split samples**	**Leaf samples**	**Max features**	**Split function**	**Accuracy diff /percentage points**
1	300	5	2	sqrt	Gini Impurity	0.4
2	300	15	2	sqrt	Gini Impurity	0.4
3	300	5	10	sqrt	Gini Impurity	0.2
4	300	15	10	sqrt	Gini Impurity	0.2
5	50	2	1	sqrt	Gini Impurity	0.0

Table 4.3 Gradient boosting models for Data set *A*.

Overview of gradient boosting models on Data set *A*					
Model	**Boosting steps**	**Learning rate**	**Max features**	**Split function**	**Accuracy diff /percentage points**
1	100	0.1	sqrt	friedman mse	0.0
2	100	0.1	log2	friedman mse	0.0
3	100	0.1	sqrt	mse	0.0
4	100	0.1	log2	mse	0.0

Table 4.4 Neural network models for Data set *A*.

Overview of neural network models on Dataset *A*							
Model	**Feature set**	**Layers**	**Neurons**	**Activation func**	**Optimiser**	**Dropout rate**	**Accuracy diff /percentage points**
1	*stat*, *stat+*, *grad*, *freq*, *acf* and *mse*	3	20, 40, 40	LeakyReLU, $\alpha = 0.3$	Adam	0.2	1.8
2	*stat*, *stat+*, *grad*, *freq*, *acf* and *mse*	3	20, 40, 40	LeakyReLU, $\alpha = 0.3$	Adam	0	2.0
3	*stat*, *stat+* and *grad*	3	11, 22, 22	LeakyReLU, $\alpha = 0.3$	Adam	0.2	2.5
4	*stat*, *stat+* and *grad*	3	11, 22, 22	LeakyReLU, $\alpha = 0.3$	Adam	0.1	-0.8
5	*stat*, *stat+* and *grad*	3	11, 22, 22	LeakyReLU, $\alpha = 0.3$	Adam	0	0.4

perform with regard to the accuracy difference. Here the values vary from around zero to over 2 percentage points. The neural network models have little higher values in general, but for none of the three methods are the accuracy differences high enough to warrant worry about overfitting.

In Figure 4.1 the accuracy for the models' top guess is displayed. We can clearly see that all models perform well on this data set as all scores over 95%. The gradient boosting models, in the figure seen as blue circles, are very consistent in their performance of 100% and neither model appears to be affected by the differences in model parameters. For random forest we see an improvement for model 5. In this model the minimum number of samples required for a node to split was set to 2 and the minimum number of samples required for a node to be a leaf set to 1. Both values are the lowest possible for these parameters and give the trees the most possible freedom to divide the data. The increase is however only one percentage point and lowering these parameters also opens up for the trees to overfit to the data more as the feature space in theory can be divided into regions with only one data point in each.

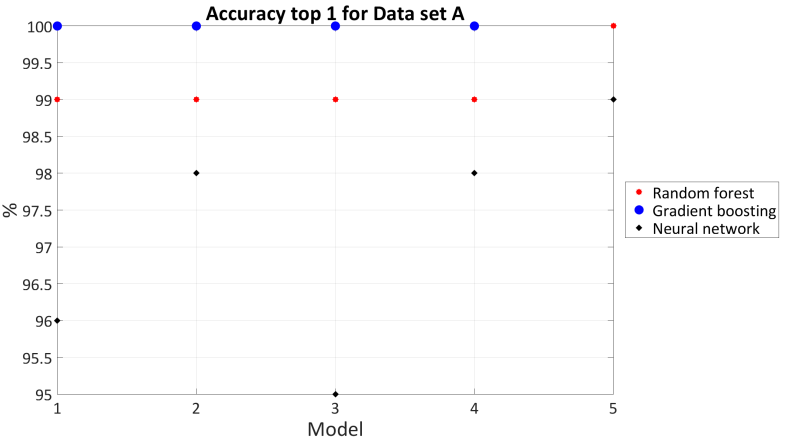


Figure 4.1 Accuracy for the top guess for random forest (stars, red), gradient boosting (circles, blue) and neural network (crosses, black). The models are trained and tested on Data set *A* with five-minute sample time and two days window length. Note that all models have an accuracy well above 80%.

The neural networks appear to be strongly affected by the dropout rate. Model 1 and 3 with the highest rate perform the worst and if we also take the accuracy difference, presented in Table 4.4, into account it is clear that the models with low or no dropout do not overfit more than those with a higher dropout rate. This could be because we are dealing with a simple problem with few classes and the network does not have to model noise into the model during training to perform well. Between model 1-2 and 3-5, which differ on the features set on which they were trained, there does not appear to be any visible difference.

In Figure 4.2 the accuracy for any of the models top three guesses being correct is seen. The random forest models, red stars, are plotted first and therefore not visible as gradient boosting models, blue circles, are plotted on top. Here we can see that all models, except one, perform perfectly with accuracy of 100%. The neural network model that does not achieve a perfect score is model 3 where a smaller network is used with the smaller feature set and a high dropout rate. It could be that such a high dropout rate prevents a small network from finding the interesting patterns since too much information is dropped out.

Accuracy is however not everything, we also wish for our models to make smart, informed choices. In Figure 4.3 informedness, see equation (2.20), for the models are plotted. This picture very well reflects Figure 4.1 and tells us that not only are the models accurate in their predictions, the predictions are also not random and lucky, but based on an informed pick which is what we want.

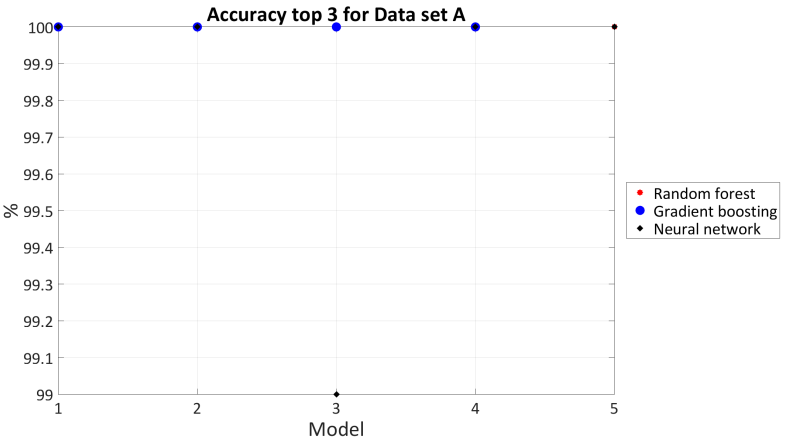


Figure 4.2 Accuracy for added probability for top three guesses for for the same models and data as in Figure 4.1. The random forest markers are plotted first and are therefore hidden behind the gradient boosting markers. All models have an accuracy above 99%.

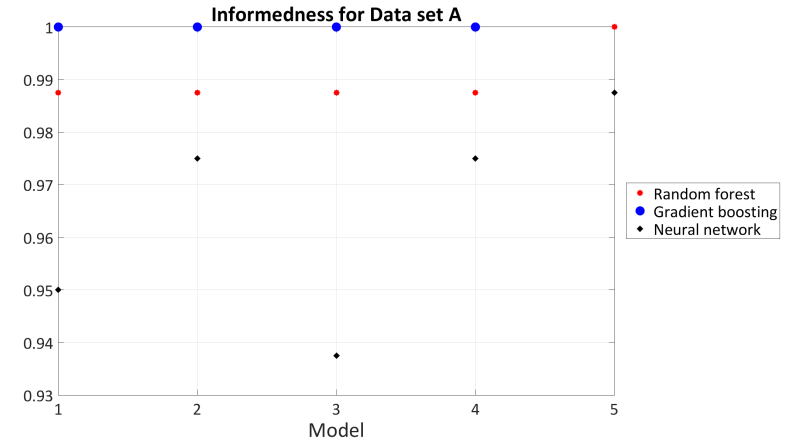


Figure 4.3 Informedness for top guess for the same models as before. Gradient boosting models perform best.

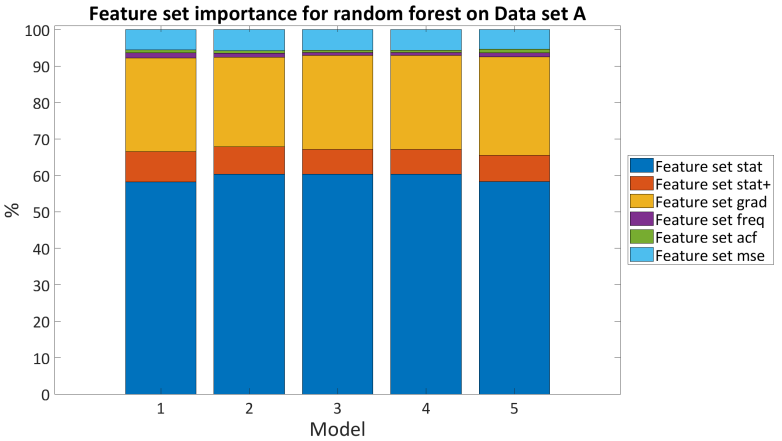


Figure 4.4 Feature set importance for each random forest model. The models are trained and tested the same data as before. There is no obvious difference between the models but feature sets *stat* and *grad* are cleary the most importance ones.

For Data set *A* all models perform extremely well. This is probably due to the simplicity of the data. There are few classes and they are different enough that signal types are easy to separate, even by simply looking at them. In Figure 4.4 the total features importance within each feature set for the random forest models is seen. Though we can see small differences for each model it is clear that Feature set *stat* carries most information for this data set. Observe that the feature sets contain different number of features and the feature importance does not compensate for this. In Figure 4.5 we see the average mean feature importance for each feature over all random forest models. It is clear that Feature set *stat*, *stat+*, *grad* and *mse* contain most of the information. As for the neural network models, 3-5, not trained on all six feature sets neither do we see any decrease in performance that cannot better be explained in another way. Therefore, it seems probable that using all feature sets for this data set is redundant.

4.3 Data set *B*

In this section we evaluate and discuss models trained on Data set *B*. Observe that we trained on two different subsets of data: *B1+2*, where data from two buildings was used for training and evaluation, and *B2*, where data from one building was used for training and data from another building for evaluation. The same models were implemented and trained on the subsets in order to compare how the data affected the results.

In Table 4.5 we see model parameters and accuracy difference for random forest

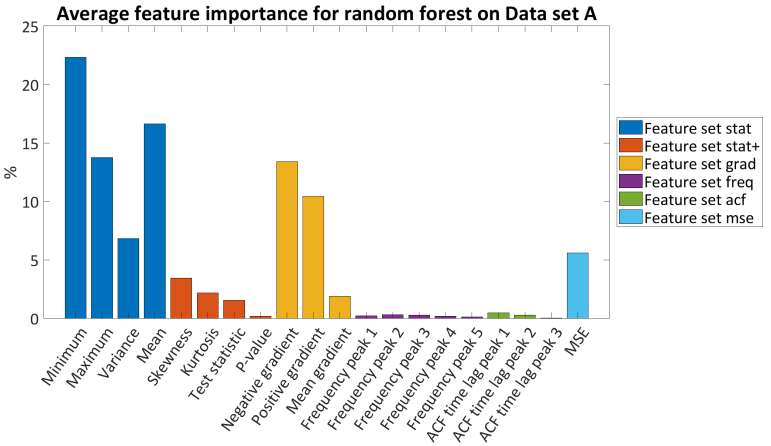


Figure 4.5 Average importance over all random forest models for each feature. Features in feature sets *stat*, *grad* and *mse* hold the most information.

models trained on Data set *B*. For this data set we chose to vary the maximum number of features considered for finding an optimal node split and what function evaluates the quality of the split. Table 4.6 displays model parameters and accuracy difference for the gradient boosting models. The parameter that is varied is the learning rate of the model. Parameters and accuracy difference for the neural network models can be seen in Table 4.7. It is the number of layers and the dropout rate that varies between the models, the variations are similar to those made for Data set *A*, see Table 4.4.

Considering the accuracy difference seen in Table 4.5 and Table 4.6 we see a big issue with this data set. For all random forest and gradient boosting models the accuracy difference is close to or well over 10 percentage points. It is around 10 percentage points for Data set *B1+2*, but around 30 percentage points for Data set *B2*. This clearly means that all the models have overfitted. One reason for this might be that the tree sizes have not been limited in any way, due to the number of samples required for a node to split is 2 and for it to be a node 1 for both random forest models and gradient boosting. This makes it possible for the trees to perfectly fit to the data during training. The overfitting should be kept in mind as we study the other metrics of performance for these models.

For neural network models in Table 4.7 the accuracy difference, and therefore overfitting, is less than for random forest and gradient boosting models, however it is still present for some models. We also see some cases where the accuracy difference is negative. This means that the accuracy was higher for the test set than the training set and is *not* a sign of overfitting no matter its absolute value. A large neg-

Table 4.5 Random forest models for Data set *B*.

Overview of random forest models on Data set *B*							
Model	**Trees**	**Split samples**	**Leaf samples**	**Max features**	**Split function**	**Accuracy diff, *B1+2* /percentage points**	**Accuracy diff, *B2* /percentage points**
1	300	2	1	sqrt	Gini Impurity	10.4	29.6
2	300	2	1	log2	Gini Impurity	10.4	29.6
3	300	2	1	sqrt	Information Gain	9.5	28.9
4	300	2	1	log2	Information Gain	9.5	28.9

Table 4.6 Gradient boosting models for Data set *B*.

Overview of gradient boosting models on Data set *B*						
Model	**Boosting steps**	**Learning rate**	**Max features**	**Split function**	**Accuracy diff, *B1+2* /percentage points**	**Accuracy diff, *B2* /percentage points**
1	100	0.05	sqrt	friedman mse	9.3	32.7
2	100	0.1	sqrt	friedman mse	10.9	33.2
3	100	0.15	sqrt	friedman mse	10.3	34.7
4	100	0.2	sqrt	friedman mse	10.9	33.2

Table 4.7 Neural network models for Data set *B*.

Overview of neural network models on Dataset *B*								
Model	**Feature set**	**Layers**	**Neurons**	**Activation func**	**Optimiser**	**Dropout rate**	**Accuracy diff, *B1+2* /percentage points**	**Accuracy diff, *B2* /percentage points**
1	*stat*, *stat+*, *grad*, *freq*, *acf* and *mse*	3	20, 40, 40	LeakyReLU, $\alpha = 0.3$	Adam	0.2	-2.7	-0.1
2	*stat*, *stat+*, *grad*, *freq*, *acf* and *mse*	3	20, 40, 40	LeakyReLU, $\alpha = 0.3$	Adam	0.1	-1.7	0.2
3	*stat*, *stat+*, *grad*, *freq*, *acf* and *mse*	3	20, 40, 40	LeakyReLU, $\alpha = 0.3$	Adam	0	2.6	14.5
4	*stat*, *stat+* and *grad*	3	11, 22, 22	LeakyReLU, $\alpha = 0.3$	Adam	0.2	-0.8	-10.1
5	*stat*, *stat+* and *grad*	3	11, 22, 22	LeakyReLU, $\alpha = 0.3$	Adam	0.1	1.0	-11.6
6	*stat*, *stat+* and *grad*	3	11, 22, 22	LeakyReLU, $\alpha = 0.3$	Adam	0	-1.9	-1.5

ative value rather points to a too large dropout rate. The sign can easily be explained by how dropout works. It is only active during training and means that 20% or 10%, depending on dropout rate, of nodes in the network are not active. Therefore, the information in the network will be lower during training than testing and might have been too small for the network to make a good decision on. When the model is then evaluated on the test set there is no dropout of information hence the model will have access to more information which in these cases increased performance. As we can see in Table 4.7 the large negative values only occur for the smaller networks indicating that the dropout rate is a bit too high hence too little information is available for the small models to work with. Dropout is used to decrease the risk of overfitting to training data and if we consider the result of model 6 we notice that the accuracy difference is only 1.5 percentage points which is not an extremely high value. This supports the theory of the dropout rate being a bit too high for models 4 and 5. For Data set *B1+2* we see very little signs of overfitting at all. Model 3 has the highest accuracy difference with a value of 2.6 percentage points. This is not as low as desirable, but not high enough to be a real problem. We see that it is also model 3 that have overfitted for Data set *B2*. This is explained by the fact that this model has a large number of neurons in combination with a dropout rate of zero.

Data set *B1+2*

In this section we study the results from models trained and tested on both Building *1* and Building *2* from Data set *B*. In Figure 4.6 we see the accuracy for the top guess. The accuracy is clearly lower for this data set, in comparison to Data set *A*. Both random forest and gradient boosting do still perform over 80%, which is the desired minimum limit set by Schneider Electric, while the neural networks do not quite reach this limit.

All gradient boosting and random forest models appear to perform roughly speaking equally well in Figure 4.6, differing at maximum two percentage points. The red markers for random forest model 2 and 3 are in the figure hidden behind the markers for gradient boosting model 2 and 3. For random forest we can detect a small increase in performance from models 1 and 2 to models 3 and 4. It would suggest that the Information Gain as split evaluation function performs better than Gini Impurity. We also see no improvement between using square root or second logarithm of the total number of features to set maximum number of features considered for each split. As for the gradient boosting models we see an increase in accuracy for model 1 to 3 and then a decrease with model 4. This would suggest the optimal learning rate for this data set lies between 0.1 and 0.2. With 0.15 being the best choice we have tried. The overfitting discussed in the previous section is however significant for all models.

The neural network models have a larger variation in performance and we can especially see that models 4 and 5 perform worse than the rest. These two models have fewer neurons in every layer and a non-zero dropout rate. The other neural network models have an accuracy of about 75%. For the smaller networks in models 4, 5 and 6 a clear difference in accuracy can be noticed in Figure 4.12. Model 6 clearly outperforms the other two and model 4 performs just better than model 5. The only difference between the models is the dropout rate where rate 0, for model 6, gives the best result but interestingly enough model 4 with rate 0.2 performed better than model 5 with rate 0.1. It is hard to come to any conclusions but it could be, as stated before, that a dropout rate of 0.1 and above is too high and that it is a mere coincidence that model 4 performs better. For the first three models, where once again only the dropout rate is varied, we cannot see any great difference in accuracy from Figure 4.6.

In Figure 4.7 we find the accuracy for the correct class being among the top three guesses of the models. We can see that all models perform above the desired 80% limit. Compared to Figure 4.6 we see an increase of about 10-15 percentage points for all models. It also appears as if random forest models perform better than gradient boosting models and we can see the same correlation with the split evaluation function as for top one accuracy. However, conclusions that we came to for learning rate in gradient boosting models for top one accuracy does not apply any more. Differences within the neural network models have become much smaller and they

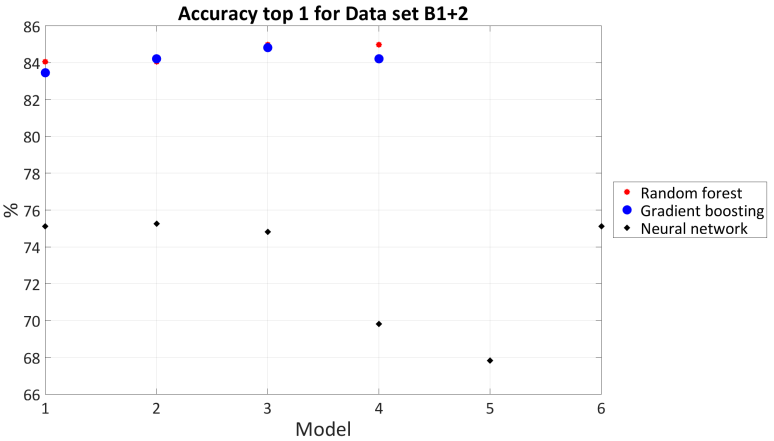


Figure 4.6 Accuracy for top guess for random forest (stars, red), gradient boosting (circles, blue) and neural network (crosses, black). The models are trained and tested on Data set *B*, buildings *1* and *2* with five-minute sample time and two days window length. Gradient boosting and random forest models do over all outperform the neural network models.

now only differ with about 1 percentage point.

As for the accuracy of the top five guesses, seen in Figure 4.8 the increase in performance is still noticeable for neural network models, but for random forest and gradient boosting models it is only around one percentage point. As can be calculated from Table 4.1 one percent of the test set roughly equals to one individual meaning that gradient boosting and random forest models only had one additional correct answer in the top five compared to the top three guesses. This is a too small difference to base any conclusions on. Noticeable for the neural network models is that it is still model 5 that performs the worst.

Informedness for models trained on Data set *B1+2* can be seen in Figure 4.9. Just as in the case with Data set *A* the figures confirm what could be seen in the accuracy plot for the top guess. Once again this indicates that the models making the most correct choices are also making the most informed ones.

When studying the feature set importance, Figure 4.10, and average feature importance, Figure 4.11, it is clear that the data is still primarily separated by Feature set *stat*. The importance is however more evenly distributed over the other features and feature sets than for Data set *A*. Especially, it appears as if Feature set *freq*, containing frequency measurements, has gained importance. It is reasonable that with a more complex data set, such as Data set *B*, more complex information is required. Frequency measurements usually contain a lot of data information that remains hidden in measurements made on non-transformed data.

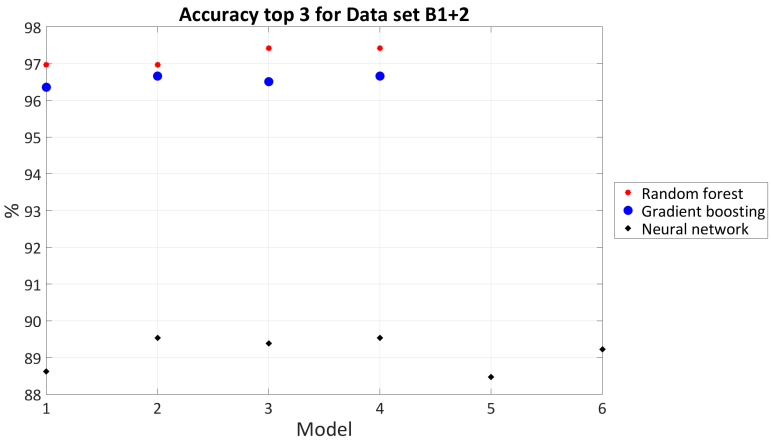


Figure 4.7 Accuracy for added probability for top three guesses for the same models and data as in Figure 4.6. Also here the gradient boosting and random forest models outperform the neural network models. All models have an top three accuracy well above 80%.

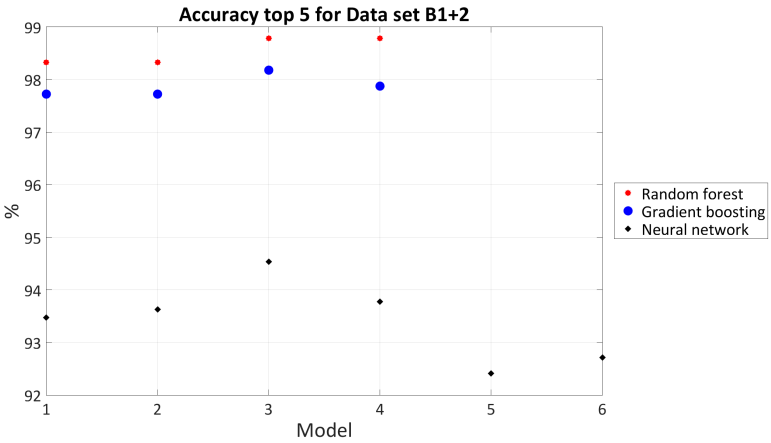


Figure 4.8 Accuracy for added probability for top five guesses for the same models and data as before. The pattern from top accuracy and top three accuracy is visible here again.

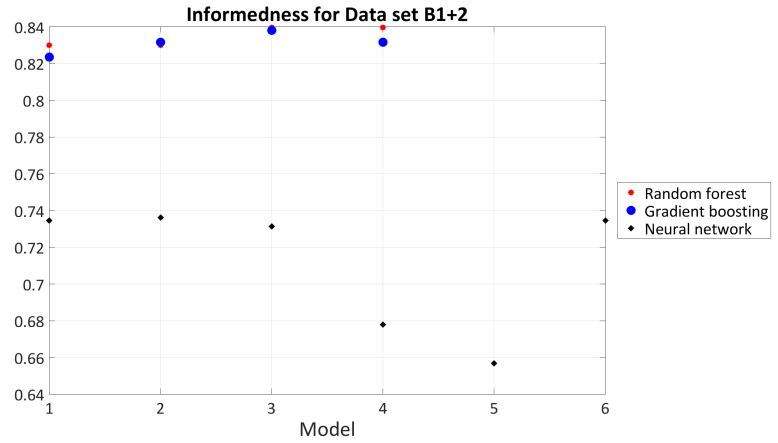


Figure 4.9 Informedness for top guess for the same models as in Figure 4.6. The trend from the accuracy graphs is visible here as well.

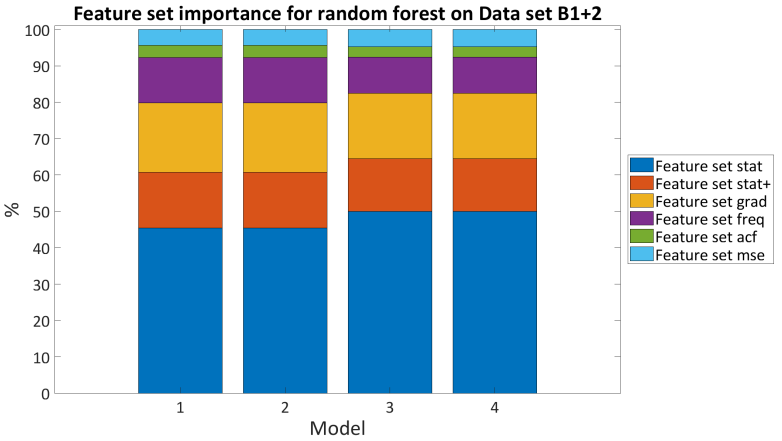


Figure 4.10 Percentage feature set importance in random forest for each model. A small difference between models 1 and 2 can be seen if compared to models 3 and 4. The first to models have Gini Impurity as a split function while the last to have Information Gain.

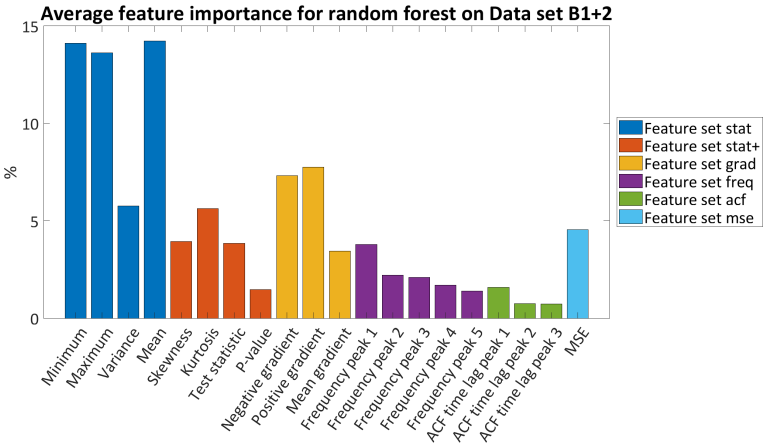


Figure 4.11 Average importance over all random forest models for each feature. Now many more of the features plays an important role to the models compared to what could be seen for Data set *A* seen in Figure 4.5.

Overall, we may, for random forest, conclude that Information Gain is slightly preferable to Gini Impurity as split evaluation function since it grants slightly better results. For this data set it also appear as if the choice of the maximum number of features available at a split did not affect the outcome of the models. For gradient boosting models, we could instead find that an appropriate learning rate would be around 0.15. Finally, we noticed, for the neural networks that 0.1 was too high as a dropout rate for the smaller networks and that the dropout rate did not affect the result of the models much if the networks were bigger. Neural network models overall performed worse than the other methods when regarding accuracy. Nonetheless, all random forest and gradient boosting models were overfitted. In this case it might not matter that the performance of these models were better as they would probably not be able to be used for data that differs much more from the training set than the test set did. In the next section we study the effect of training on data from one building and testing on data from another building to further investigate this.

Data set *B2*

In this section results from models trained on Building *1* in Data set *B* and tested on Building *2* in Data set *B* are presented. In Figure 4.12 the accuracy for the top guess is presented. It is clear that some of the models struggle with classifying a building they have not trained on. The decrease in performance is greatest for random forest and gradient boosting, both of which are shallow learning methods. This is a sign of the neural network models finding more general correlations within the data. It also strengthens our hypothesis that gradient boosting and random forest models

overfitted to the training data, which we suspected based on the accuracy diff in Table 4.5 and Table 4.6. Just as for Data set *B1+2* random forest models 3 and 4, with Information Gain as split evaluation function perform slightly better than models 1 and 2. For gradient boosting models we see no clear connections with the variation of the learning rate.

Let us now focus on the difference in performance for the neural network models in Figure 4.12 compared to Figure 4.6. Worst is obviously model 3, which were one of the better ones for Data set *B2*, and we can also notice that this model has the largest decrease in accuracy for all the neural network models. This indicates that model 3 has overfitted to the training data, which is confirmed by the accuracy difference as discussed earlier and due to a dropout rate of zero. Models 1 and 2, that have dropout rate 0.2 and 0.1 respectively, on the other hand do not show as much decrease in accuracy as model 3 and neither have they overfitted with regard to the accuracy difference in Table 4.7.

Further investigation for the neural network models can be made. If one studies model 6, that just as model 3 has a dropout rate of zero, in Figure 4.12 and compare it to model 4 and 5, that has the same structure but a non-zero rate, there are some things one may notice. First, the behaviour seen for models 1-3, where accuracy is dropping with a lower dropout rate is not seen. We can also notice that neural network model 5 appears to be the best overall method, however the accuracy difference of this model must be taken into consideration. From Table 4.7 we find that the accuracy difference is -11.6 for model 5, indicating a too high dropout rate.

Models 1-3 and models 4-6 do not only differ in the complexity of the layers, but the second set of models also use fewer feature sets. As they perform as good or better than models 1-3 it could mean that for this data set Feature set *stat*, *stat+* and *grad* contain better information for separating the signal types while the other feature sets might contain at least some features that confuse the models instead of helping them make informed decisions. Confusing features include features that in no way separate the classes from each other or features containing high noise. If these kind of features are used as input and the model try to base its rules on them the model could end up performing worse than if these features were excluded. The worse performance may derive from the model making decisions on information that in reality does not matter, i.e. information that in no way separates classes. In Figure 4.16 we see the feature set importance of the random forest models. This figure confirms Feature set *stat*, *stat+* and *grad* carrying over 80% of the feature importance. Unlike random forest, neural network can recombine features. It is possible that this ability enhances the information within the features further.

As for accuracy for top three and five guesses, seen in Figure 4.13 and Figure 4.14 respectively, performance increases for all models by 6-20 percentage points. Some neural network models perform above the 80% limit for top three guesses already and for top five guesses all models come out over 80%. Some gradient boosting

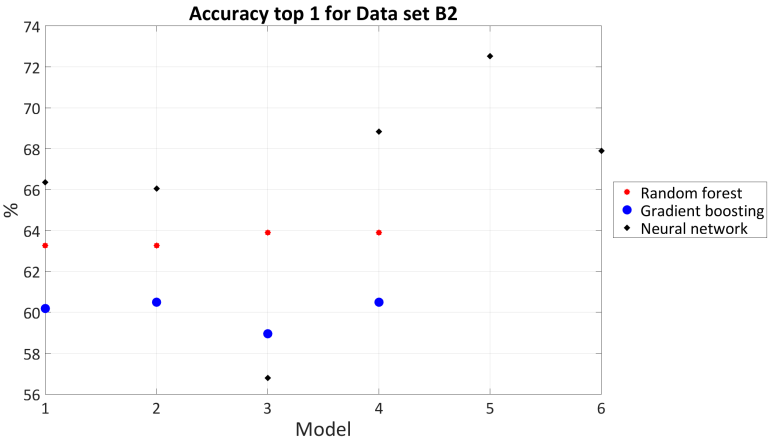


Figure 4.12 Accuracy for top guess for random forest (stars, red), gradient boosting (circles, blue) and neural network (crosses, black). The models are trained on Data set *B*, Building *1* and tested on Data set *B*, Building *2* with five-minute sample time and two days window length. The greatest change in accuracy from Data set *B1+2* in Figure 4.6 is that the neural network models has the highest accuracy here. Overall the random forest and gradient boosting models have a 20-25% lower accuracy here.

and neural network models perform equally well with regard to accuracy for top three and five. Nevertheless, top performance for neural network models are still higher than for gradient boosting models. It is also clear that random forest models continue to struggle to deliver in comparison to the other methods. Accuracy for top three and five, in Figure 4.13 and Figure 4.14, also shows the increased performance of neural network models 4-6 in comparison to models 1-3.

As could be expected by now informedness, in Figure 4.15, mirrors the accuracy for top one and clearly indicates that the neural network models perform the best. The metric is based on the top one guesses for each model and tells us that the neural network models make the most informed guesses, i.e. have captured most of the important information in the data.

Concerning feature set importance, we can conclude from Figure 4.16 that Feature set *acf*, containing autocorrelation measurements has lost out in favour of Feature set *mse* if we compare it to feature set importance for Data set *B1+2* in Figure 4.10. From comparison of average feature importance in Figure 4.17 and Figure 4.11 however we see a tendency that important features from Data set *B1+2* have become even more important in this experiment. For this data set these measurements should possible be treated with caution as random forest models are heavily overfitted with an accuracy difference of close to 30%.

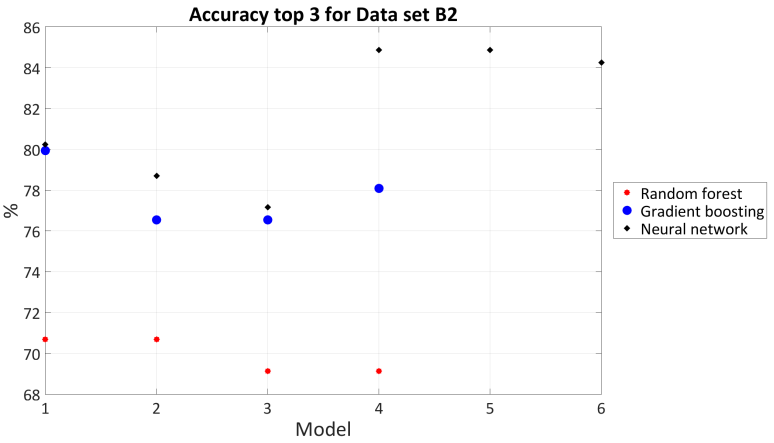


Figure 4.13 Accuracy for added probability for top three guesses. The neural network models are still overall best but the gradient boosting models have almost as high values here.

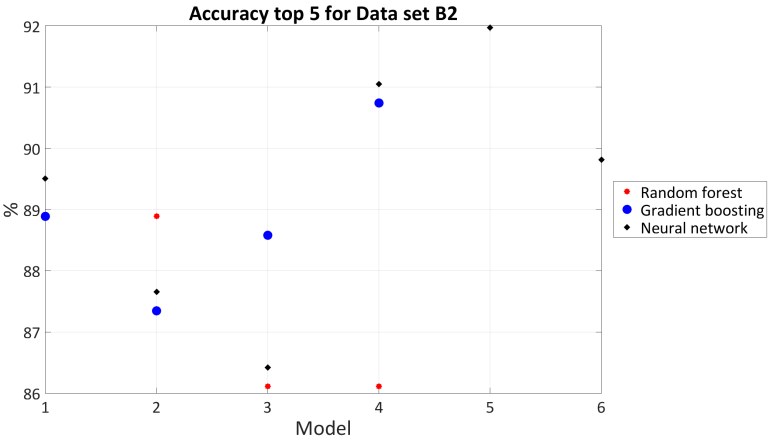


Figure 4.14 Accuracy for added probability for top five guesses for the same models and data as before. Now there is just a small difference between the methods and all models have an top five accuracy above 80%.

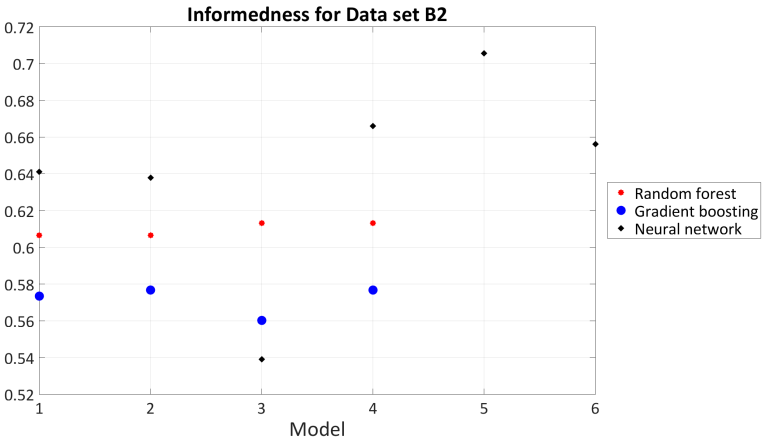


Figure 4.15 Informedness for top guess for the same models and data as before. A similar trend as for top one accuracy in Figure 4.12 can be noticed.

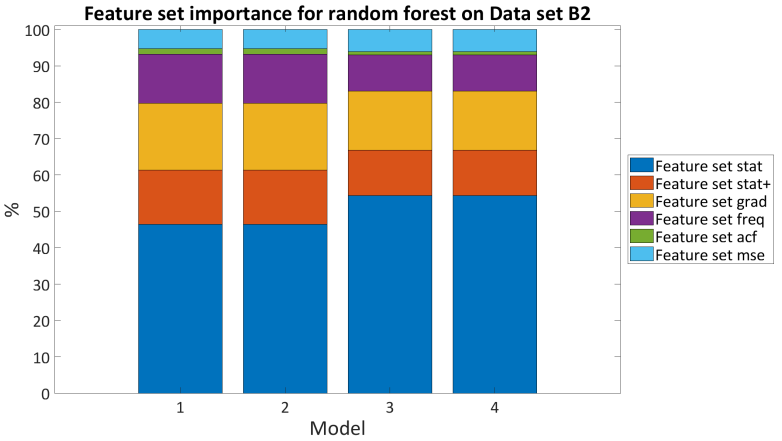


Figure 4.16 Percentage feature set importance in random forest for each model. The models are trained and tested on the same data as before. Just as for Data set *B1+2* there is a small difference when using the two split functions.

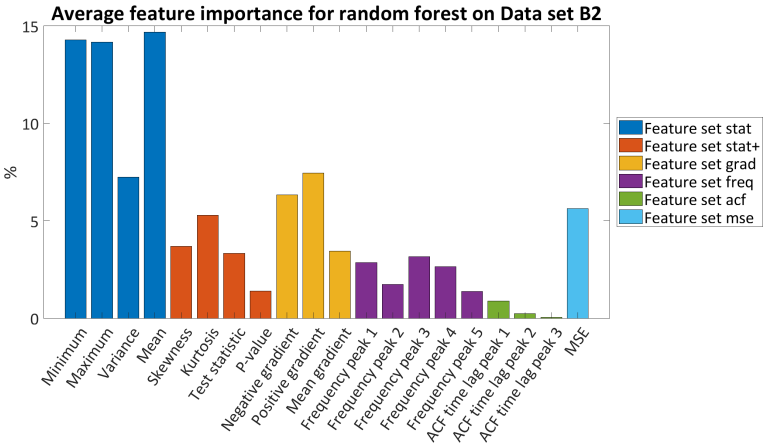


Figure 4.17 Average importance over all random forest models for each feature trained on the same data set as before. If compared to the equivalent graph for Data set *B1+2*in Figure 4.11 we find that Feature set *acf* has lost importance.

The difference in results from Data set *B1+2* and Data set *B2* are quite grand. When training on Data set *B1+2* one would by the graphs presented assume that gradient boosting and random forest were the best methods to use. The accuracy difference, which is much higher for these methods than for neural network models, do however contradict this. Results presented in this section for Data set *B2* does also demonstrate a clear decrease in performance for all models implementing either random forest or gradient boosting which, in combination with the huge increase of accuracy difference, proves that the models are overfitting. Instead the neural network models have, at least for some of the models, a performance that is pretty much the same as for Data set *B1+2*. Especially model 5 has no greater change in performance between the data sets. Another thing that could be noted for the feature sets are that the random forest based 80% of its decisions on Feature sets *stat*, *stat+* and *grad*. These are also the feature sets used for the top performing neural network models.

4.4 Data set *C*

In this section results for models trained on Data set *C* are presented. In Table 4.8 we find the random forest models trained for this data set. We tested different combinations of number of trees, maximum number of features considered for each split and what function to use to evaluate the quality of the split. Further on we also tested different combinations of how many samples were required for each node to split or be a leaf node and the maximum number of features considered for a split. In Table 4.9 we find the evaluated gradient boosting models. Here we varied the number

of boosting steps, the learning rate, the maximum number of features considered for each split and the split evaluation function. The neural network models are found in Table 4.10. For this data set the number of layers, the number of neurons, the activation function, the optimiser and the dropout rate were altered. Input for all models were all feature sets, i.e. feature sets *stat*, *stat+*, *grad*, *freq*, *acf* and *mse*.

The accuracy difference is presented to the far right in Table 4.8, Table 4.9 and Table 4.10 for random forest, gradient boosting and neural network models respectively. Clearly random forest models overfit to the training data. The accuracy difference varies between 26 and 10 percentage points. The best models are those with a higher value for the number of samples required for a node to split or to be a leaf node. Higher values appear to decrease the overfit to the training data. These parameters control how the trees in the model can divide data by determining how many individuals there have to be left in a node for it to divide further or be an end node. Lower numbers therefore result in trees being able to divide data into smaller regions, possible one for each individual in the training set. By increasing these values, we are clearly forcing the models not to make decisions on single individuals but always larger groups. This suggests that a higher value for the split samples and the leaf samples generalise the models. It could be that even higher values could be worth examining. From Table 4.8 we can find no clear impact on the accuracy difference from the different split functions.

Let us now consider the accuracy difference for gradient boosting models instead in Table 4.9. First of all, we notice that the lowest value is achieved for models with the highest learning rate which might tempt us to come to the hasty conclusion that this is a preferable learning rate and that the models are promising. If we jump ahead a bit and consider the actual accuracy presented in Figure 4.18 it becomes obvious that all those models perform absolutely terrible. A model with accuracy below 10% is useless and this can never be compensated for by a really low accuracy difference since the output of the model will not be trustworthy any way.

Consider all gradient boosting models in Table 4.9. Those with learning rate 0.2 have obviously the lowest accuracy difference but if we consider the top one accuracy in Figure 4.18 have an accuracy below 10% which makes them uninteresting. Other models, with a lower learning rate has an accuracy difference 7-20 percentage points. Even the lowest value of 7 percentage points indicates that some overfitting to the training data has occurred. This value would preferably be a bit lower, especially since the training and test set are similar as mentioned in section 4.1. However, other than the learning rate, the accuracy difference is seemingly strongly affected by the numbers of boosting steps. A lower number appears to decrease the tendency of overfitting. The number of boosting steps determines how many times trees are added in the gradient boosting model. A large value allows the method to fit many trees to decrease the output error and the noise in the input will then play a part in the models' decision making. By decreasing the number of boosting steps even

Table 4.8 Random forest models for Dataset *C*.

Overview of random forest models on Data set *C*						
Model	**Trees**	**Split samples**	**Leaf samples**	**Max features**	**Split function**	**Accuracy diff /percentage points**
1	50	2	1	sqrt	Gini Impurity	26.5
2	50	2	1	log2	Gini Impurity	26.5
3	50	2	1	All	Gini Impurity	27.0
4	50	2	1	sqrt	Information Gain	26.3
5	50	2	1	log2	Information Gain	26.3
6	50	2	1	All	Information Gain	26.4
7	100	2	1	sqrt	Gini Impurity	26.5
8	100	2	1	log2	Gini Impurity	26.5
9	100	2	1	All	Gini Impurity	27.0
10	100	2	1	sqrt	Information Gain	26.6
11	100	2	1	log2	Information Gain	26.6
12	100	2	1	All	Information Gain	26.5
13	200	2	1	sqrt	Gini Impurity	26.4
14	200	2	1	log2	Gini Impurity	26.4
15	200	2	1	All	Gini Impurity	26.9
16	200	2	1	sqrt	Information Gain	26.5
17	200	2	1	log2	Information Gain	26.5
18	200	2	1	All	Information Gain	26.4
19	250	2	1	sqrt	Gini Impurity	26.5
20	250	2	1	log2	Gini Impurity	26.5
21	250	2	1	All	Gini Impurity	26.7
22	250	2	1	sqrt	Information Gain	26.4
23	250	2	1	log2	Information Gain	26.4
24	250	2	1	All	Information Gain	26.7
25	300	2	1	sqrt	Gini Impurity	26.4
26	300	2	1	log2	Gini Impurity	26.4
27	300	2	1	All	Gini Impurity	26.6
28	300	2	1	sqrt	Information Gain	26.5
29	300	2	1	log2	Information Gain	26.5
30	300	2	1	All	Information Gain	26.5
31	300	5	2	sqrt	Gini Impurity	23.4
32	300	5	2	log2	Gini Impurity	23.4
33	300	10	2	sqrt	Gini Impurity	19.7
34	300	10	2	log2	Gini Impurity	19.7
35	300	15	2	sqrt	Gini Impurity	16.5
36	300	15	2	log2	Gini Impurity	16.5
37	300	5	6	sqrt	Gini Impurity	14.6
38	300	5	6	log2	Gini Impurity	14.3
39	300	10	6	sqrt	Gini Impurity	14.3
40	300	10	6	log2	Gini Impurity	14.3
41	300	15	6	sqrt	Gini Impurity	13.2
42	300	15	6	log2	Gini Impurity	13.2
43	300	5	10	sqrt	Gini Impurity	10.3
44	300	5	10	log2	Gini Impurity	10.3
45	300	10	10	sqrt	Gini Impurity	10.3
46	300	10	10	log2	Gini Impurity	10.3
47	300	15	10	sqrt	Gini Impurity	10.3
48	300	15	10	log2	Gini Impurity	10.3

Table 4.9 Gradient boosting models for Dataset *C*.

Overview of gradient boosting models on Data set *C*					
Model	**Boosting steps**	**Learning rate**	**Max features**	**Split function**	**Accuracy diff /percentage points**
1	100	0.1	sqrt	friedman mse	12.6
2	100	0.2	sqrt	friedman mse	0.1
3	100	0.1	log2	friedman mse	12.6
4	100	0.2	log2	friedman mse	0.1
5	100	0.1	sqrt	mse	12.6
6	100	0.2	sqrt	mse	0.1
7	100	0.1	log2	mse	12.6
8	100	0.2	log2	mse	0.1
9	200	0.1	sqrt	friedman mse	17.9
10	200	0.2	sqrt	friedman mse	0.1
11	200	0.1	log2	friedman mse	17.9
12	200	0.2	log2	friedman mse	0.1
13	200	0.1	sqrt	mse	18.1
14	200	0.2	sqrt	mse	0.1
15	200	0.1	log2	mse	18.1
16	200	0.2	log2	mse	0.1
17	250	0.1	sqrt	friedman mse	19.4
18	250	0.2	sqrt	friedman mse	0.1
19	250	0.1	log2	friedman mse	19.4
20	250	0.2	log2	friedman mse	0.1
21	250	0.1	sqrt	mse	19.1
22	250	0.2	sqrt	mse	0.1
23	250	0.1	log2	mse	19.1
24	250	0.2	log2	mse	0.1
25	300	0.1	sqrt	friedman mse	20.4
26	300	0.2	sqrt	friedman mse	0.1
27	300	0.1	log2	friedman mse	20.4
28	300	0.2	log2	friedman mse	0.1
29	300	0.1	sqrt	mse	20.4
30	300	0.2	sqrt	mse	0.1
31	300	0.1	log2	mse	20.4
32	300	0.2	log2	mse	0.1
33	100	0.05	sqrt	friedman mse	7.0
34	100	0.05	log2	friedman mse	7.0
35	100	0.05	sqrt	mse	7.0
36	100	0.05	log2	mse	7.0
37	200	0.05	sqrt	friedman mse	12.0
38	200	0.05	log2	friedman mse	12.0
39	200	0.05	sqrt	mse	12.0
40	200	0.05	log2	mse	12.0
41	250	0.05	sqrt	friedman mse	13.7
42	250	0.05	log2	friedman mse	13.7
43	250	0.05	sqrt	mse	13.7
44	250	0.05	log2	mse	13.7
45	300	0.05	sqrt	friedman mse	14.9
46	300	0.05	log2	friedman mse	14.9
47	300	0.05	sqrt	mse	14.9
48	300	0.05	log2	mse	14.9

further it could be possible to reduce overfitting. Finally we find, just as for random forest, that the split function has no apparent effect on accuracy difference.

Finally, we scrutinise the performance of the neural network models with regard to the accuracy difference, as seen in Table 4.10. Most obvious is that these models have an overall lower accuracy difference, which has also been true for Data set *B*. The worst models have an accuracy difference of just above 11 percentage points

Table 4.10 Neural network models for Dataset *C*.

Overview of neural network models on Data set *C*						
Model	**Layers**	**Neurons**	**Activation func**	**Optimiser**	**Dropout rate**	**Accuracy diff /percentage points**
1	4	20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0	5.3
2	6	20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0	5.8
3	8	20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0	7.2
4	10	20, 20, 20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0	8.3
5	10	40, 40, 60, 60, 60, 80, 80, 60, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0	8.9
6	6	40, 60, 80, 80, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0	11.0
7	10	25, 30, 40, 45, 50, 55, 55, 50, 45, 40	LeakyReLU, $\alpha = 0.3$	Adam	0	10.2
8	6	25, 30, 35, 35, 30, 30	LeakyReLU, $\alpha = 0.3$	Adam	0	7.3
9	10	40, 40, 60, 60, 60, 80, 80, 60, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0.1	3.9
10	6	40, 60, 80, 80, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0.1	2.8
11	10	25, 30, 40, 45, 50, 55, 55, 50, 45, 40	LeakyReLU, $\alpha = 0.3$	Adam	0.1	2.5
12	6	25, 30, 35, 35, 30, 30	LeakyReLU, $\alpha = 0.3$	Adam	0.1	1.4
13	10	40, 40, 60, 60, 60, 80, 80, 60, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0.2	0.5
14	6	40, 60, 80, 80, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0.2	1.7
15	10	25, 30, 40, 45, 50, 55, 55, 50, 45, 40	LeakyReLU, $\alpha = 0.3$	Adam	0.2	0.2
16	6	25, 30, 35, 35, 30, 30	LeakyReLU, $\alpha = 0.3$	Adam	0.2	0.0
17	10	40, 40, 60, 60, 60, 80, 80, 60, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0.3	-1.0
18	6	40, 60, 80, 80, 60, 60	LeakyReLU, $\alpha = 0.3$	Adam	0.3	-0.7
19	10	25, 30, 40, 45, 50, 55, 55, 50, 45, 40	LeakyReLU, $\alpha = 0.3$	Adam	0.3	-2.1
20	6	25, 30, 35, 35, 30, 30	LeakyReLU, $\alpha = 0.3$	Adam	0.3	-3.1
21	6	20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0.1	0.2
22	8	20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0.1	-0.7
23	10	20, 20, 20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0.1	0.7
24	6	20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0.2	-2.3
25	8	20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0.2	-2.4
26	10	20, 20, 20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	Adam	0.2	-2.1
27	6	20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	rmsprop	0.1	-0.7
28	8	20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	rmsprop	0.1	-1.0
29	6	25, 30, 35, 35, 30, 30	LeakyReLU, $\alpha = 0.3$	rmsprop	0.1	0.1
30	6	40, 60, 80, 80, 60, 60	LeakyReLU, $\alpha = 0.3$	rmsprop	0.1	3.7
31	6	20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	rmsprop	0.2	-2.4
32	8	20, 20, 20, 20, 20, 20, 20, 20	LeakyReLU, $\alpha = 0.3$	rmsprop	0.2	-3.6
33	6	25, 30, 35, 35, 30, 30	LeakyReLU, $\alpha = 0.3$	rmsprop	0.2	-1.1
34	6	40, 60, 80, 80, 60, 60	LeakyReLU, $\alpha = 0.3$	rmsprop	0.2	1.3
35	6	25, 30, 35, 35, 30, 30	ELU, $\alpha = 1.0$	Adam	0.2	-1.3
36	6	40, 60, 80, 80, 60, 60	ELU, $\alpha = 1.0$	Adam	0.2	1.6
37	6	25, 30, 35, 35, 30, 30	SELU	Adam	0.2	-0.8
38	6	40, 60, 80, 80, 60, 60	SELU	Adam	0.2	0.1

while the best ones are just below one percentage point. Here the dropout rate plays a major part for the accuracy difference. A too low value, such as zero, obviously correlates to higher values of accuracy difference. A too high value for dropout, on the other hand results in models that underfit and just as has been reasoned before it could be that too much information is lost during training of the model to find the overall important patterns in the data. What is an appropriate dropout rate is strongly dependent on the complexity of the structure of the neural network model as noted for models in both Data set *A* and Data set *B*. Looking only at accuracy difference it appears as if a dropout rate of about 0.1-0.2 is the best.

Accuracy for the top guess can be found in Figure 4.18. As we can see gradient boosting models (blue circles) with learning rate 0.2 perform significantly worse than other models. In Figure 4.19 the same data is presented, but with these models removed. This is to ease comparison between models and methods. As the same behaviour can be seen throughout all metrics in this section we have chosen to

present all other figures without gradient boosting models with learning rate 0.2.

In Figure 4.19, showing top guess accuracy for all models except gradient boosting with learning rate 0.2, it is clear that random forest and gradient boosting models perform better than neural network. But none of the models achieve the desired 80% accuracy. By looking at the last twelve gradient boosting models we see the effect the number of boosting steps can have. These models have a learning rate of 0.05 and either 100, 200, 250 or 300 boosting steps. We can see that the accuracy is increasing with the number of boosting steps. It appears as if these models level out on the same level as models with learning rate 0.1. The relation to the number of boosting steps is not as clear for the models with a learning rate of 0.1, but there is no contradiction.

Furthermore, we can in Figure 4.19 for the random forest models see a relation between the number of trees and top one accuracy. Noticeable is also that the combination of all features considered for the best split and Gini Impurity as split evaluation function appears to be a bad combination as models 3, 9, 15, and 21 show a drop in accuracy. We can also see that increasing the number of samples required for a node to split and especially for being a leaf node decreases performance.

By focusing on the result for the neural network models found in Figure 4.19 it is clear that the performance is more scattered than those for the other two methods. For models 1-8 the performance appears to be rather good for the neural network models. However, considering the accuracy difference for these models, found in Table 4.10, the models seem to overfit to the training data as the difference varies between 5-10 percentage points which is quite much. Therefore, it may be concluded that not using dropout is a bad idea even though it actually grants pretty good results for the test data with regards to accuracy. That the model still performs well may be a result of too similar training and test set.

Now let us focus on the second batch of neural network models, all with a dropout rate above zero. These models, models 9-20, are models with 6 and 10 layers and many more neurons compared to models 1-8. The first models perform just as well as those with fewer neurons and then we see a clear decrease in Figure 4.19. Models 9-12 has a dropout rate of 0.1 while model 13-16 has a dropout of 0.2. From Figure 4.19 we may just see that the accuracy is a little bit worse for the higher dropout rate although it, as expected, becomes clear in Table 4.10 that dropout reduces the overfitting, i.e. reduces the accuracy difference. For dropout rate 0.2 the accuracy difference is even better and it could be that the models that appear to perform slightly worse in top one accuracy capture more of the interesting behaviour in the data rather than noise. For model 17-20 the dropout rate is increased to 0.3 and the accuracy becomes worse while the accuracy difference stays close to what was measured for rate 0.2 indicating that 0.3 might be slightly too high for the dropout rate.

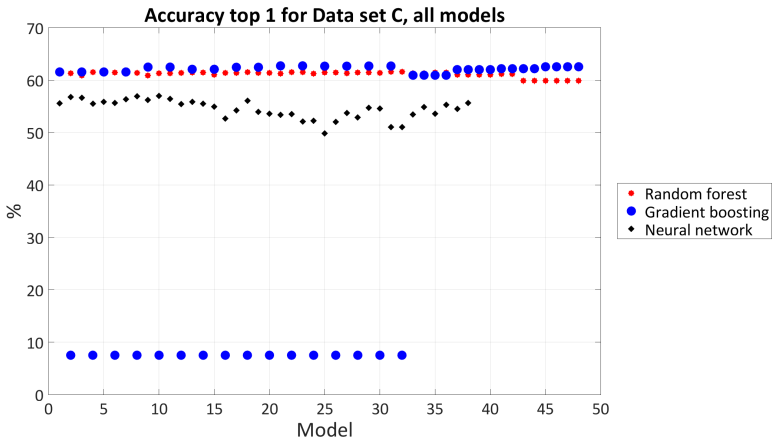


Figure 4.18 Accuracy for top guess for random forest (stars, red), gradient boosting (circles, blue) and neural network (crosses, black). The models are train and tested on Dataset *C* with five-minute sample time and two days window length. For the first partition of gradient boosting models every second model stands out. These models have a learning rate of 0.2 and an extremely low accuracy of less than 10%.

Models 21-26 are designed with fewer neurons and has a dropout rate of 0.1 for models 21-23 and 0.2 for models 24-26. The accuracy is steadily decreasing for these models and the accuracy difference in Table 4.10 stays low. This indicates that the models are too simple to capture enough of the behaviour in the data. Within these two sets we alter the number of layers between 6, 8 and 10 as seen in Table 4.10. In Figure 4.19 however we see no clear trend with regards to the number of layers.

As can be seen in Figure 4.19 models 27-34 perform better than the worst models but still quite poorly in comparison to the other neural network models. These models have the same structure, number of layers and neurons, as either model 2, 3, 8 or 10 but are implemented with dropout 0.1 or 0.2 and a different optimiser. The best performing structure is clearly the one used in models 10, 30 and 34 but the optimiser rmsprop, used for model 30 and 34, is clearly outperformed by Adam, which is used in model 10.

The final models are implemented with different activation functions. Model 35 and 36 has ELU while models 37 and 38 have SELU. Model 35 and 37 have fewer neurons than the other two while all of them have the same number of layers, i.e. 6. Model 36 and 38 have the same structure as model 10. The structure of model 10 still outperforms the other, SELU clearly results in better performance than ELU but leakyReLU is by far the best when it comes to accuracy for the top 1 guess.

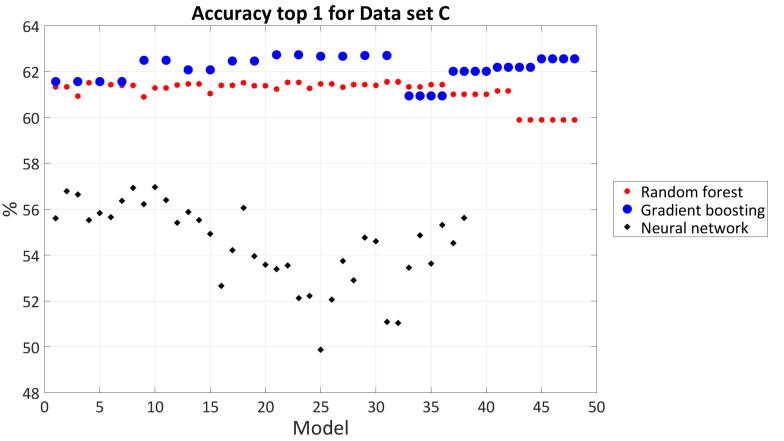


Figure 4.19 The same results as in Figure reffig:BoC:accTop1OutZoom but the models with learning rate 0.2 are here removed. It is clear that random forest and gradient boosting models are outperforming the neural network models, the difference is overall about 8 percentage points

In Figure 4.20 we find the accuracy for any of the top three guesses being correct. We would like to remind the reader of gradient boosting models with learning rate 0.2 being removed in the figure. As for the rest of the models we see that almost all models perform better than 80%. We can also see that gradient boosting models perform better than either of the other methods. As for accuracy of top one, it is clear by the last twelve models that the number of boosting steps increases performance. It is also more clear here that models 33-48 level out on the same level as models 21-31 (odd numbers) that could be seen in Figure 4.19. This would indicate that the same minimum has been found in the loss function for both learning rates.

As for random forest models, red stars in Figure 4.20, it is possible to distinguish a curvature in the performance of the first 35 or so models that were not evident for top one accuracy. For models 1-25 the parameter difference lies primarily in the number of trees. It would suggest that the more trees the better. Models 25 and 26 seem to be the top performers. As seen in Table 4.8, these models have the highest number of trees, 300 and the Gini Impurity as split evaluation function. These traits are shared with model 27 which demonstrates a decrease in performance. This model differs in the maximum number of features considered for each split.

For the neural network models the pattern is similar in Figure 4.19 and Figure 4.20. Most obvious when comparing these two graphs are that the best neural network models now perform almost as well as the random forest models, around 84-86%. Overall the top 3 accuracies is less scattered than the top 1. What also becomes clear after further comparison is that the best performance is now seen for model

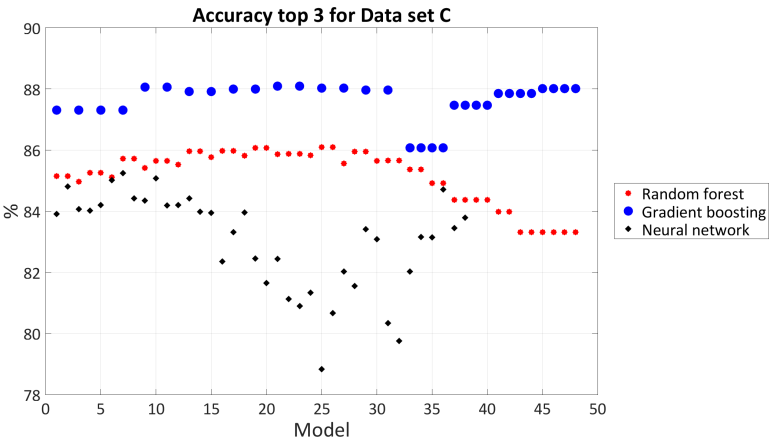


Figure 4.20 Accuracy for the top three guesses for the same models as in Figure 4.19 on the same data. The difference in accuracy between neural network and the other two methods are lower than for the top accuracy. It is also clear that the gradient boosting models have a little higher accuracy than random forest models now. Most of the models have an accuracy above 80%.

7. Keeping in mind that the accuracy difference, found in Table 4.10, is higher than 10 percentage points for model 7 it tells us that the model has strongly overfitted. This indicates that model 7 has been better at finding correlations in the noise in the training data rather than finding interesting patterns in the data. Model 10 on the other hand, that is not far behind model 7 on top three accuracy in Figure 4.20, has an accuracy difference of just below 3 percentage points which is a much more acceptable value and therefore probably the better model. Finally, we can also notice that model 36 stands out in comparison to accuracy top one in Figure 4.19 and is, with regard to top three accuracy, one of the top five performing neural network models.

In Figure 4.21 we find the accuracy for the top five guesses. For gradient boosting we can here see a slight ringing in the performance of model 10-30 with the apparent period of four models. This corresponds to different combinations of the maximum number of features considered for a split and what function is used to evaluate the split. From the top 5 accuracy we can deduce that evaluating the split with mean square error together with considering the square root of the maximum number of features for each split is optimal. We have a harder time detecting this ringing in Figure 4.19 and Figure 4.20 but might be explained by scaling in the figures. For random forest it is very clear that performance for every third model has decreased. Some of this can be contributed to the increased resolution of the y-axis, but the increase exceeds this. The models correspond to considering all features for the

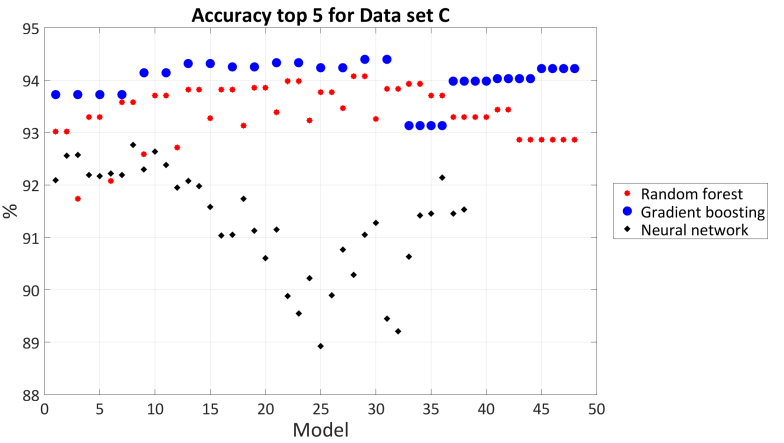


Figure 4.21 Accuracy for added probability for top five guesses for the same models as in Figure 4.19. the difference between the methods is even lower although the gradient boosting models still has the highest accuracy.

optimal split which seems to be a bad option.

Within the neural network models, the difference in performance for the top five accuracy depicted in Figure 4.21 is much more similar to top one accuracy than top three accuracy was. Once again, the first 11 models perform better and while model 36 still has an accuracy in the same range and it is no longer among the top 5 models.

A similar behaviour as seen in previous figures depicting models trained on Data set *C* can also be found for the informedness metric in Figure 4.22. Mostly the informedness metric depicted resembles accuracy top 1 in Figure 4.19.

Feature set importance and feature importance for random forest can be found in Figure 4.23 and Figure 4.24 respectively. We can see that parameter differences affect the feature set importance, however Feature set *stat* continues to hold a lot of sway. This is the first data set where we can see a significant difference of feature set importance over the models. Feature set *stat* increases for some models as Feature set *acf* and *mse* decreases significantly. In models 1-6, 7-12, 13-18, 19-24 and 25-30 we see the same apparent behaviour. The difference between these models are the number of trees and we can therefore conclude that this does not affect the feature selection. For the three first models in each set we use Gini Impurity for split evaluation and for the last three Information Gain. In both these cases we see no difference between using the square root and second logarithm of the total number of features for selecting the optimal features for each split which correspond to the first two bars. The third bar for each split evaluation function corresponds to

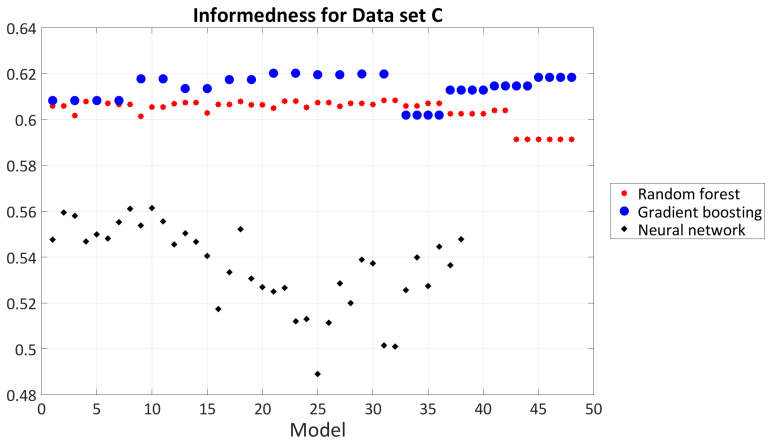


Figure 4.22 Informedness for top guess for for the same models and data as in Figure 4.19. The results resembles those of top one accuracy seen in Figure 4.19.

considering all features when determining the optimal split and every second of these had Information Gain as the split function. These were the same models in which we could see a decrease in accuracy in Figure 4.21. It might be that using the same feature set in over 60% of the splits simply doesn't make it possible for the models to separate signal types properly in a general way.

For this data set we found that completely different models performed the best compared to Data set *A* and Data set *B*. Still it was obvious that all random forest and gradient boosting models struggled with overfitting to training data also for this data set. The accuracy difference was high even for the best models and since the test and training set is so similar this might be a reason to worry about how these models will perform on data from another building. In all, the gradient boosting performed better than random forest models and apart from the higher accuracy difference outperformed the neural network models. Clear for gradient boosting models was that a learning rate of 0.2 was much too high and resulted in poor performance. Instead a learning rate of 0.05-0.1 improved accuracy. For random forest models we could see that the more trees used the better the model became. The splitting function that granted the best performance was Gini Impurity. The choice of splitting function also affected which features decisions were based on. One thing that did not affect feature importance nor performance noticeably was if the maximum number of features considered for a split was set to the square root or second logarithm of the total number of features, but if all features were used performance decreased. For the neural network models, the most obvious result was that not having dropout resulted in models that strongly overfitted to the training data. The best dropout rate among those tested, with regard to the accuracy difference and overall performance,

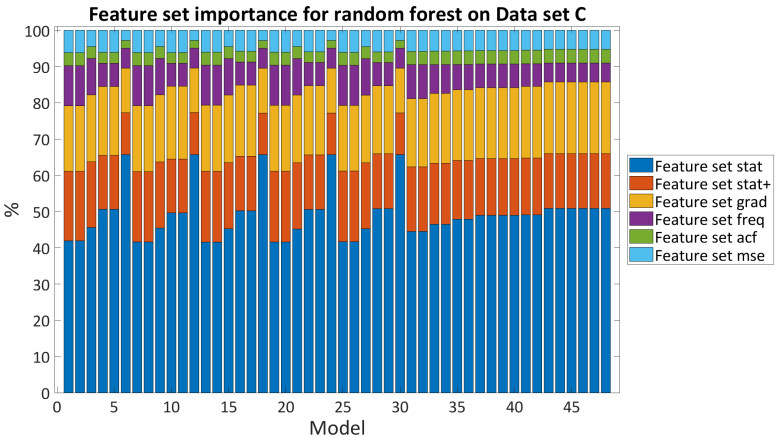


Figure 4.23 Percentage importance for feature set in random forest for each model. The models are trained and tested on the same data as before. For this data set a clear difference is noticed for models that considered all features for a split and had Information Gain as the split function. Here over 60% of the decissions are based on Feature set *stat* compared to about 45% for the other models.

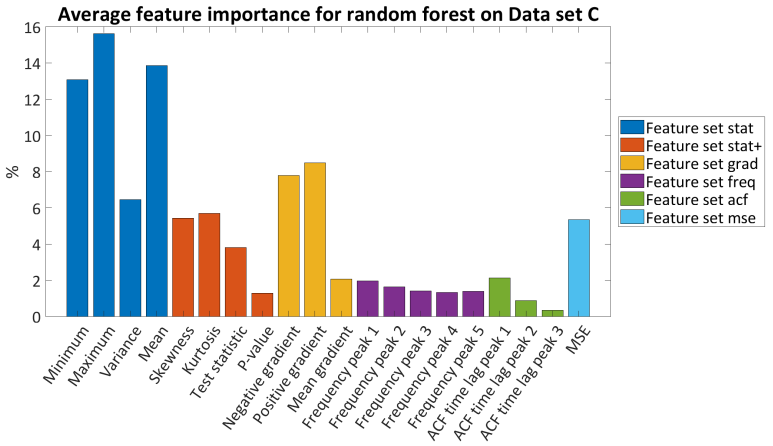


Figure 4.24 Average importance over all random forest models for each feature. The most important features are found in feature sets *stat*, *stat+*, *grad* and *mse*.

appeared to be 0.2. It also became quite clear that Adam was to prefer over rmsprop and that leakyReLU was the best activation function to use. Finally, we could also see that just making a network deeper and not adding neurons to the layers did not enhance performance. Overall, for neural network models we deem model 10 to be the best. The structure of model 10 was also the most favourable throughout the tests of different parameter settings.

4.5 Summary

In all, great differences in the feature importance graphs have clearly been visible for the different data sets. For Data set *A* there was no intermediate difference between the models, see Figure 4.4, and Feature set *stat* and *grad* provided the base for about 80% of the decisions in the random forest models. For Data set *B* a small difference between models with different split evaluation functions appeared, see Figure 4.16. These differences can also be seen for Data set *C* in Figure 4.23. Especially Feature set *stat* increased in importance while Feature set *freq* decreased when using Information Gain. A worsened performance could be seen for the models implementing Information Gain. It could be that there is some information in the frequency domain, which Feature set *freq* contains, that the models with this split evaluation cannot exploit.

Overall, there are more features used for Data set *B* and Data set *C* than for Data set *A* as can be seen by comparing Figure 4.17, Figure 4.11 and Figure 4.24 with Figure 4.5. It is expected that when trying to classify data into bigger, more complex systems with more classes the models will need to utilise more information to perform well. However, it is clear that Feature set *stat*, *stat+*, *grad* and *mse* play an important role for all data sets. It could be that the less important features in Feature set *freq* and *acf* confuse the models rather than helps them. Especially if the same features are important to the neural networks as for random forest models. This is not unlikely since we for Data set *A* evaluated neural network models only using feature set *stat*, *stat+* and *grad* which resulted in better performance.

To summarise, we have achieved very high accuracy for the simplest data set but as the complexity grew with more classes in the other data sets the accuracy dropped. It has also become clear that overfitting is a greater issue for random forest and gradient boosting models than for the neural network models. However the highest accuracy scores has overall been seen for gradient boosting models. A more thorough summary and conclusion can be found in section 6.2.

5

Association: results and discussion

In this section the results for the implemented solutions for the association problem is presented and discussed. The association problem, described in section 1.2, is when we seek to find which signals belong to the same equipment. Only two of the three data sets could be used, data sets *A* and *C*. For Data set *A*, support vector machines were implemented while for Data set *C*, a string comparison was instead utilised, as described in section 3.3.

5.1 Data set *A*

While dividing signals into windows for the association problem there was one major difference from the partition made for the classification problem. We required that signals belonging to the same equipment needed to be measured over the same time frame. The added condition resulted in less data which could be used in the machine learning work process. For Data set *A* there was at first less than one set of examples from each equipment and therefore we chose to increase the allowed gap time. The maximum allowed gap time was chosen to leave us with approximately the same number of individuals as for the classification problem. For Data set *A* we may compare the following numbers with those found in Table 4.1: for the training set there were 395 individuals, for test there were 130 and finally for evaluation there were 60 individuals. Remember that there were five individuals belonging to the same time frame in the same equipment. Hence there were many more negative examples than positives, for e.g. in the evaluation set there were 12 examples of every class (60 examples through 5 classes) and for one support vector machine model trained there was 144 examples where only 12 were positive. Clearly the input data was skewed.

For the association problem we decided to do a pairwise comparison for this data set as explained in section 3.3. This resulted in four subproblems, one for each signal

Table 5.1 Support vector machine models for Dataset *A*.

Overview of support vector machine models on Dataset *A*		
Model	**C**	γ
1	50	0.015
2	50	0.01
3	50	0.0075
4	50	0.005
5	75	0.015
6	75	0.01
7	75	0.0075
8	75	0.005
9	100	0.015
10	100	0.01
11	100	0.0075
12	100	0.005

type (Room Air Humidity, Luminosity, PIR Motion Sensor and Room Temperature) that is compared to the CO_2 signals. Our choice of comparing all other signals with CO_2 signals was an arbitrary choice and not built on any theory. As we will show below the results are satisfactory anyway. No tests on how models would perform if another signal than CO_2 was chosen for comparison were performed, but it is reasonable to believe that other signals have other correlation and that it would affect the result in some way.

One major assumption made in section 1.2 for the association problem was that the known class labels were all correct. This is a good assumption to make when trying to formulate a feasible solution but one that would be hard to live up to under real conditions. If deployed, any solution would first classify signals and then attempt to associate them. This means that the fulfilment of this criterion is based on how good the classification solution would be and as we have seen, it is far from perfect in most cases. It is likely that the proposed solution used here would not work if the class labels were more unreliable as correlations between class types are presumably different. We believe that one way of examining this could have been to test the methods on data in which some of the signals were mislabelled, this would provide information of the robustness of the solution. The output from one of the models for classification that did not have an accuracy of 100% could have been used to generate such labels.

In Table 5.1 the parameters of the support vector machines trained for this data set are presented. The same models are trained for each subproblem and the results will be presented subproblem-wise in the subsections below.

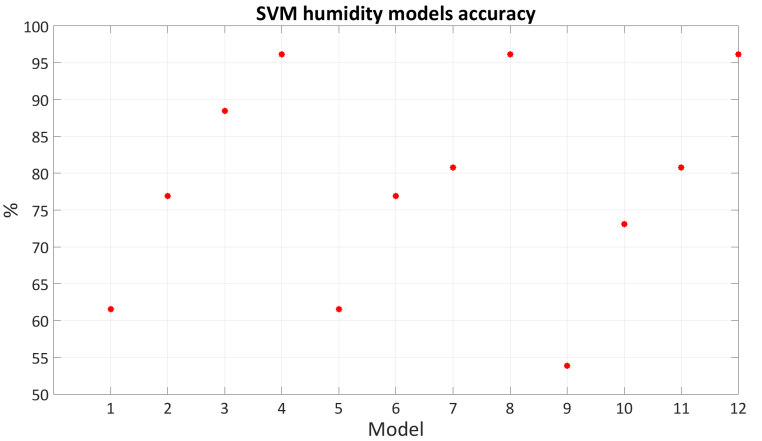


Figure 5.1 Accuracy for support vector machines trained on Data set *A* to compare signals of type CO_2 and humidity. Within the three model groups, models 1-4, 5-8 and 9-12, γ were lowered which enhanced the performance. Across the groups the values of C were changed which did not affect the performance.

CO_2 versus Humidity

In Figure 5.1 the accuracy of the different models is presented. For the association problem we will only study the top 1 accuracy since we have chosen to formulate the problem as a binary classification problem. We clearly see a trend repeating for models 1-4, 5-8 and 9-12. Within these groups the parameter γ is altered as seen in Table 5.1 and it is clear that the lowest value for γ results in the best accuracy. This indicates that the input vectors from couples of signals that belong to the same equipment are quite far away from each other in the feature set domain. For the smallest value of γ, models 4, 8 and 12, there are not any greater difference between the choice of parameter C. A smaller C does although appear to be preferable if one compare model 3 with models 7 and 11, all with the same γ. Models 1 and 5 also perform better than model 9 which indicates that a lower C value could be better.

Accuracy does not tell the whole story and in Figure 5.2 the specificity demonstrates something completely different than what the accuracy, depicted in Figure 5.1, did. First of all, all models differ little in specificity, at most less than 0.8 percentage points. Here it is although important to remember that there are extremely many answers that should be negative in the data set since we ask the question "*Does this CO_2 signal belong with any of the hundreds of humidity signals?*". The answer will only be yes for one of the humidity signals for all the CO_2 signals and therefore a small difference in specificity is still an indicator of a difference in performance between models. The difference seen appears to be coupled to the parameter C where a lower value, models 1-4 has the lowest value, appear to result in lower

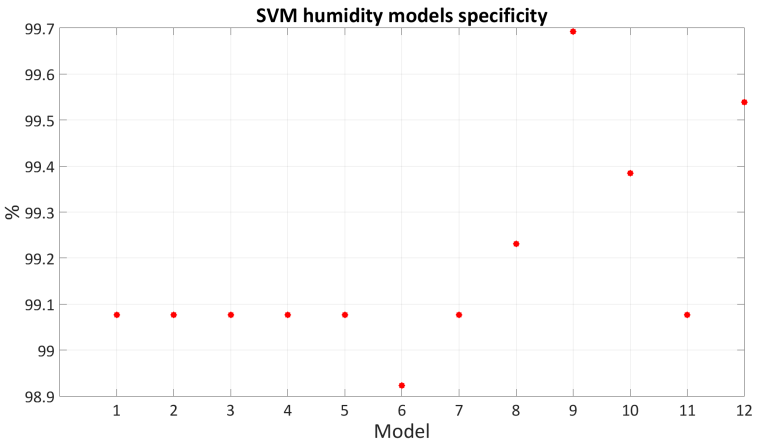


Figure 5.2 Specificity for the same machines as in Figure 5.1. The result were overall good, but small enhancements could be seen with higher values of C in models 9-12.

specificity. The best models with regard to specificity are models 9, 12 and 10 all with $C = 100$.

In Figure 5.3 the wrong per right score for positive answers is presented, i.e. if we only consider the times the model says "*Yes, these signals belong together!*" and divide the number of incorrect answers by the correct ones. For the worst performing models, models 1 and 5, we find that for every 5 correctly coupled signals the model will also say that 2 signal couples which do not belong to the same equipment do belong together. The best model, model 12 will instead for every 25 correct couplings of signals make 3 incorrect couplings which is a much more attractive value. Worth to notice is that model 9 performs almost as well as model 12.

Finally, we study the models in a ROC graph seen in Figure 5.4. Best performing is model 12. One should notice the small range of which the false positive rate spans. On each metric presented either model 9 or model 12 are the best performing model. Since the difference in accuracy between the models is more than 40 percentage points, see Figure 5.1, we deem model 12 to be the best among the 12 models. Model 12 has the highest value for C, meaning that the model seeks to classify every individual correctly instead of keeping a simple decision surface. It also has the lowest value of γ which generalises the model more than a higher value would have. Hence this might be a good combination of parameters that is keeping the model general enough but still enables it to perform well for the problem.

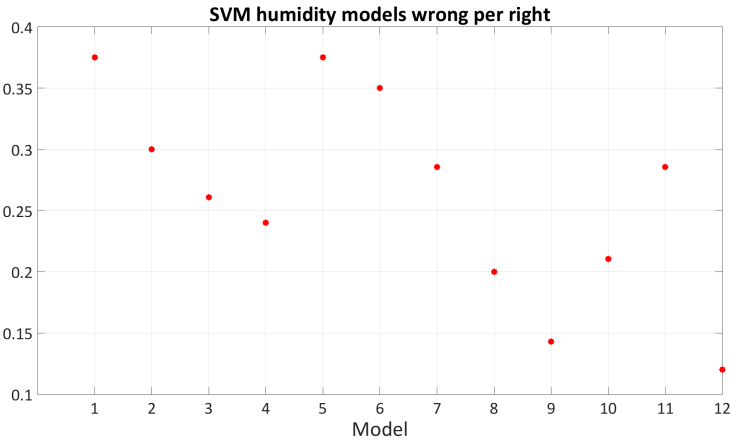


Figure 5.3 The ratio between the number of wrong guesses and the number of correct guesses for the machines presented in this section. γ and C, which are varied over the models, control the complexity of the decision surface. The best result was achieved with a high C and low γ in model 12.

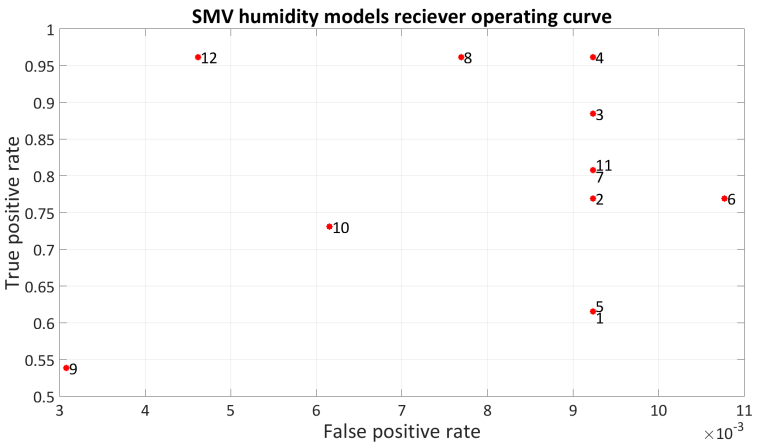


Figure 5.4 The ROC curve for machines trained to compare signals of type CO_2 and humidity. Note the small span of the x-axis. Model 12 performed best as it is closest to (1,0).

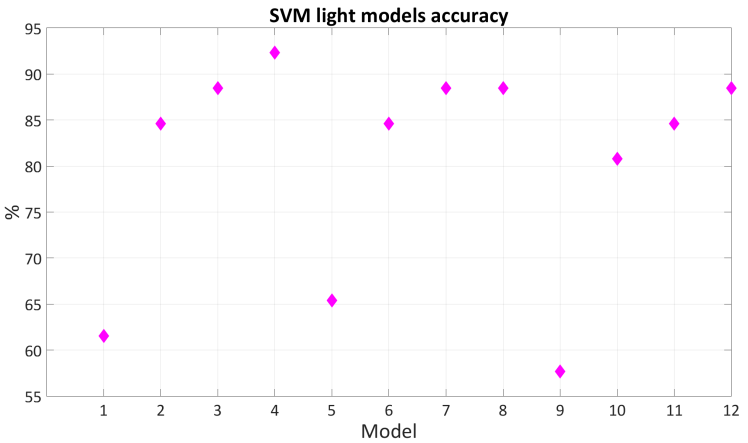


Figure 5.5 Sam as for Figure 5.1, but for comparison between CO_2 and light. The same pattern is found with higher accuracy for lower γ.

CO_2 versus Light

In Figure 5.5 the accuracy is presented for the subproblem where CO_2 signals are compared to luminosity signals. Here the parameter C appears to influence the result much more in comparison to the corresponding figure for the humidity subproblem, see Figure 5.1. The highest value is found for model 4 and for the lowest value of C we can see an increase of accuracy for a smaller value of γ. This trend for γ can also be seen for the highest value for C but not strictly for intermediate $C = 75$. Models 3, 7, 8 and 12 cannot be separated by accuracy viewed in Figure 5.5 but are all second best from the look of it.

The specificity depicted in Figure 5.6 provides us with little new information. The only clear thing we can see is that model 10 performs worse than all other models which are hard to tell apart only regarding specificity.

In Figure 5.7 we find that model 9 has the highest value for the wrong per right metric, which can be interpreted in the same way as for the subproblem presented above. Model 4 has the lowest value with model 3, 7, 8, and 12 very close behind. This conforms to what we found for accuracy in Figure 5.8.

Just like for specificity we can in the ROC graph in Figure 5.8 see that model 10 stands out as being the worst model with an acceptable true positive rate. We can also conclude that model 4 once again appears to perform best with models 3, 8, 7 and 12 following close behind and once again being impossible to separate.

Overall model 4 has performed the best for this subproblem. Interesting are the four models following closely behind. Model 3, with a little higher value for γ, i.e.

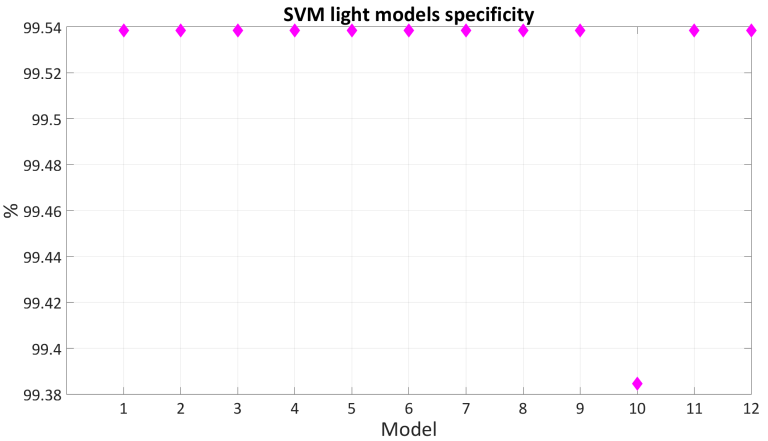


Figure 5.6 Specificity for the same machines as shown in Figure 5.5. All models perform equally except for model 10 which performs worse.

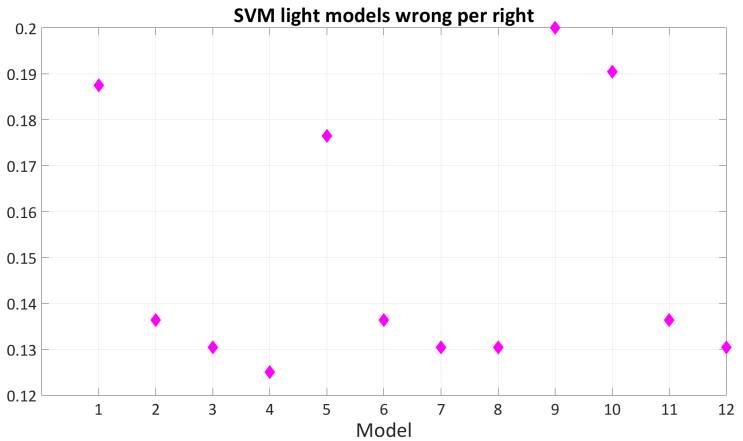


Figure 5.7 The ratio between the number of wrong guesses and the number of correct guesses for machines presented in this section. This result confirms result found for the accuracy in Figure 5.5.

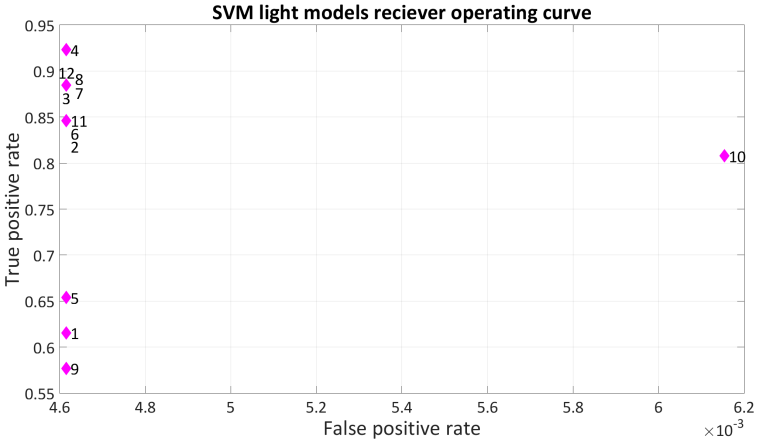


Figure 5.8 The ROC curve for machines presented in previous figures. Being closest to (1,0), model 4 performs best. Model 10 has an acceptable true positive rate, but a high false positive rate.

0.0075 instead of 0.005, and the same value for C. Models 7 and 8 with the same γ as for model 3 and 4 respectively but a higher value for C, i.e. 75 instead of 50. Finally, model 12 with $C = 100$ and $\gamma = 0.005$ and from this it appears as if a higher value of γ is connected to a better performance. This means that individuals can be far away and still be considered similar. That the lowest value for C granted the best model indicates that even if the model is made more complex with a more complex decision surface it is not possible to separate the individuals better in this feature domain.

CO_2 versus PIR

The accuracy for the subproblem where the PIR signals are compared to the CO_2 signals is found in Figure 5.9. Here we find a pattern, similar to the pattern for accuracy for the subproblem including the humidity signals. The highest value is here found for model 4 followed by model 8 and 12 indicating that the lowest value for γ once again provides the best performance, at least if one only regard accuracy.

Specificity is presented in Figure 5.10. The best value is here found for models 1, 5 and 6 while model 3, 4, 7, and 11 has the lowest values. This means that model 4 is good at matching signals that do belong together but unfortunately is also good at matching signals that do not belong together.

Just as expected from the specificity graph model 4 has not the best rate for the wrong per right metric which can be seen in Figure 5.11. Instead we find that model 6 performs the best followed by model 1, 5, 8 and finally 12.

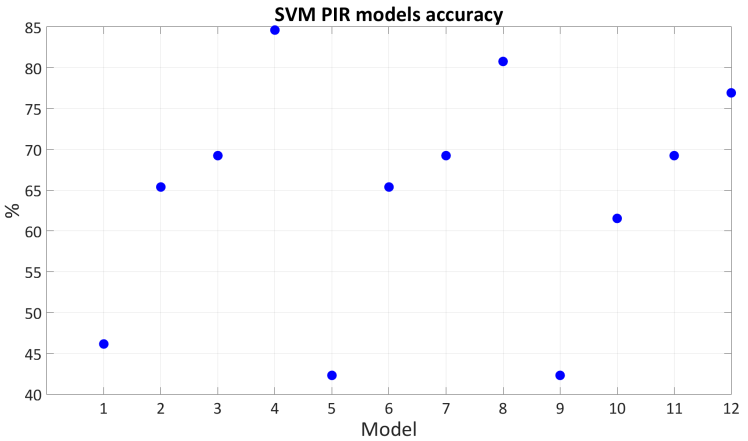


Figure 5.9 Same as for Figure 5.1 and Figure 5.5, but for comparing type CO_2 and PIR. As for previous subproblems the accuracy is enhanced with lower γ and no effect can be seen from C

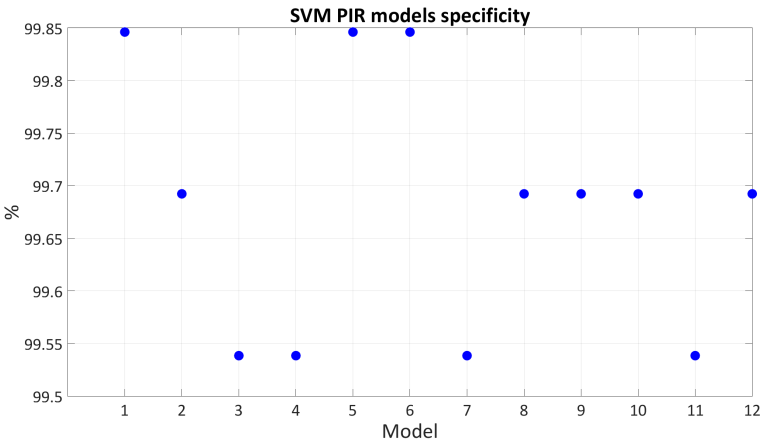


Figure 5.10 Specificity for support vector machines seen in Figure 5.9. Models 1,5 and 6 are the best performing ones. Model 4, with a high accuracy, seem to be good at matching both signals that belong and those that do not.

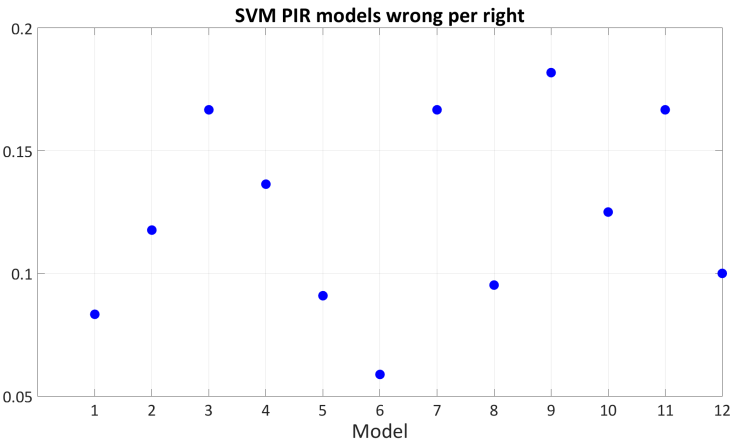


Figure 5.11 The ratio between the number of wrong guesses and the number of correct guesses for machines presented in this section. Best performing was model 6, closely followed by model 1, 5, 8 and 12.

At last we study the graph showing ROC in Figure 5.12. Here models 8, 12 and then 6 would be deemed best. Based on all presented metrics, model 8 or 6 is performing best. Model 8 has an accuracy score of 15 percentage points higher than model 6 and since the difference in wrong per right and specificity is so small model 8 should probably be deemed better, especially since the placement in the ROC graph is much better. Once again, do notice the small range over which the false positive rate spans, in comparison to the true positive rate, in Figure 5.12.

CO_2 versus Temperature

Now let us direct our attention to the final subproblem, connecting CO_2 signals with temperature signals. In Figure 5.13 we find the accuracy presented for the models. Once again the over all best accuracy can be seen for the models which have the lowest value for γ, i.e. models 4, 8 and 12.

When instead taking the specificity metric, seen in Figure 5.14, into account the value of C appears to have a greater impact on the result. Here the best models appear to be models 8, 10, 11 and 12 while model 4 actually performs the worst.

The wrong per right, depicted in Figure 5.15, is not showing the best values among all models trained for all different subproblems. However, the best performing models are clearly models 8 and 12.

Finally, we consider the ROC graph in Figure 5.16. First of all, we notice that the false positive rate spans over the greatest difference of values compared to the ROC graphs for the other subproblems. We can clearly see that model 4 performs among

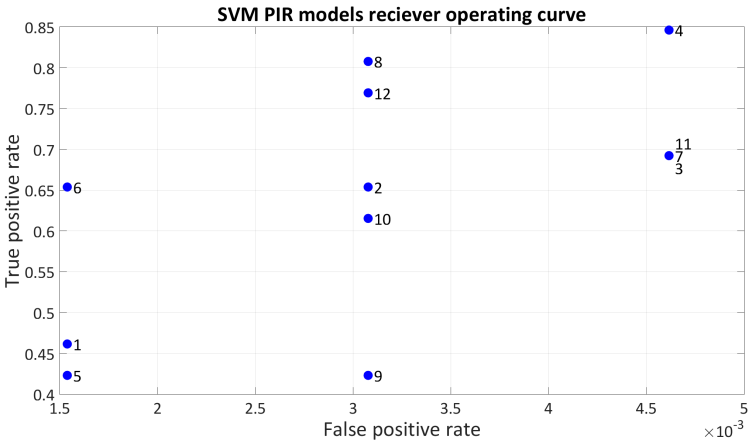


Figure 5.12 The ROC curve for machines trained comparing type CO_2 and PIR. Models 8 and 12 performs best, followed by model 6.

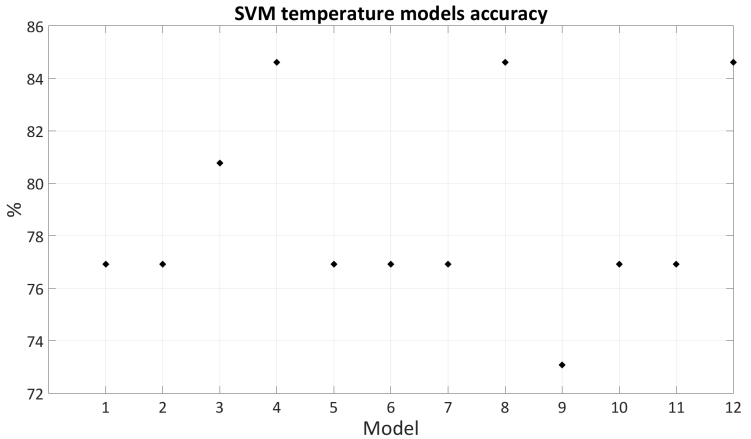


Figure 5.13 Same as for Figure 5.1, Figure 5.5 and Figure 5.9, but for comparison of type CO_2 and temperature. Also for this subproblem the best accuracy is achieved for models with lower γ-value.

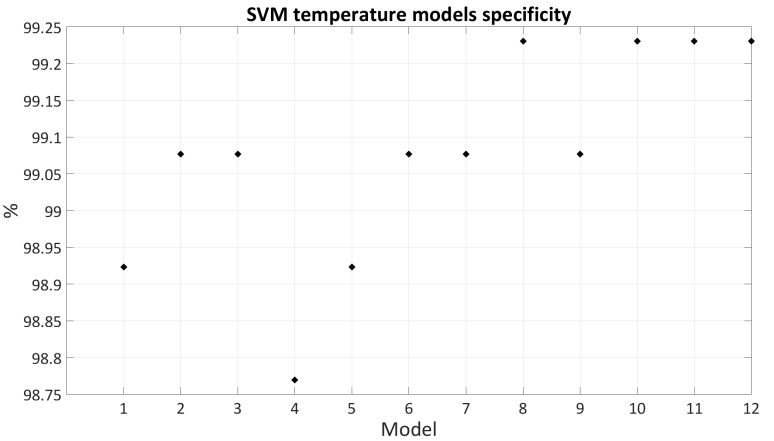


Figure 5.14 Specificity for machines seen in Figure 5.13. Models 8, 10, 11 and 12 performed best, but model 4, which had among the highest accuracy, performed worst.

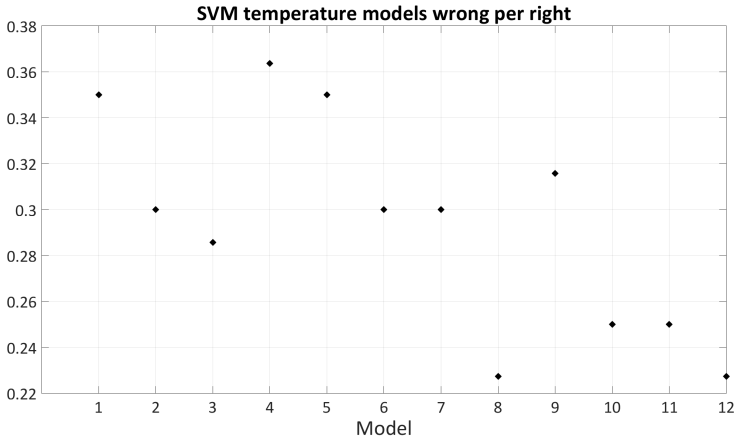


Figure 5.15 The ratio between the number of wrong guesses and the number of correct guesses for the same models as presented in the two previous figures. The result were worst than for the other subproblems with model 8 and 12 being top performers.

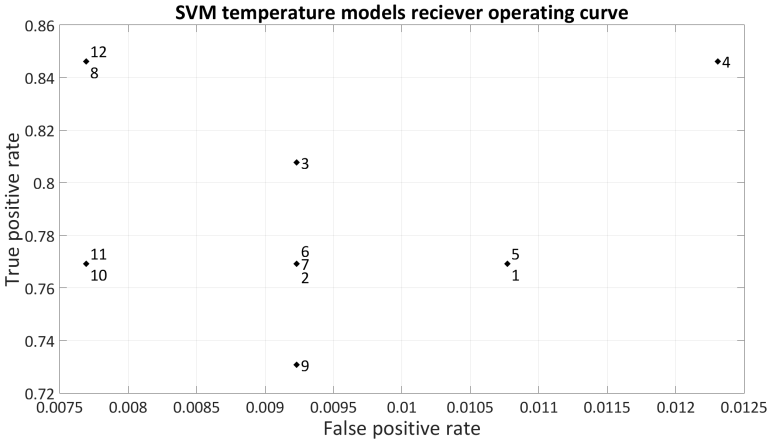


Figure 5.16 The ROC curve for machines presented i this section. In comparison to Figure 5.4, Figure 5.8 and Figure 5.12, showing the ROC for the other subproblems, the x-axis here spans over the greatest difference. Models 8 and 12 were the best and model 4 the worst.

the worst and models 12 and 8 are the absolute best. It is not clear which of models 8 and 12 is the best performing model for this subproblem. They cannot by any metric presented be separated hence it is probably wise to pick model 8 with a lower value for C granting a little less complex model.

Best support vector machines

The models deemed to perform best for each subproblem are model 12 for the one including humidity, model 4 for luminosity, model 8 for PIR and for temperature, model parameter settings can be found in Table 5.1. From the output of these models we may calculate an overall accuracy which is 88% which would be acceptable for use on a real problem where an automated connection needs to be made. In this section we present the result for these support vector machine models for the subproblem they performed the best for. We also compare the methods with and without the string feature.

Let us first study Figure 5.17 depicting the accuracy for each model trained on each subset. The difference between using the string feature (in blue) and not using it (in red) is striking. For the first three subproblems the difference in accuracy is about 15 percentage points while for the fourth subproblem, where CO_2 signals are compared to temperature signals, the difference is over 30 percentage points. This is a strong indicator of the importance the path and name hold. To not use this data for the association problem in some way would probably be a waste. What may be noticed if we compare the result for the support vector machine models that utilises the string feature is that all of them perform better than for the test data. From

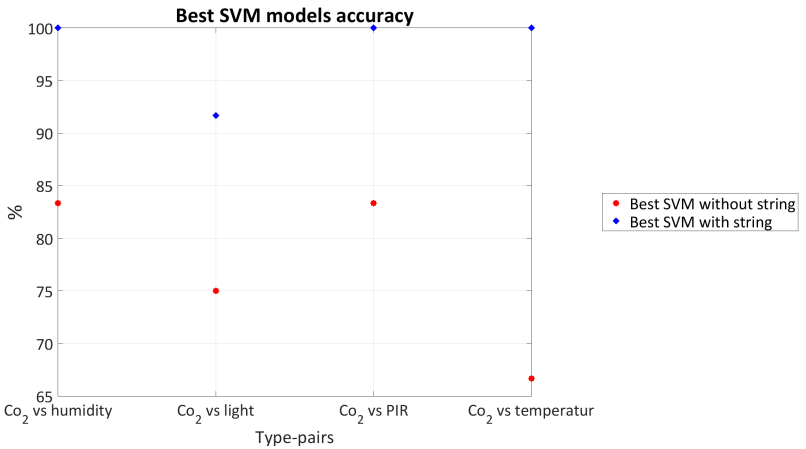


Figure 5.17 Accuracy for the best support vector machines found for each subproblem evaluated on the evaluation set. The blue dots represent models with a name and path comparisons as an input feature and the red without. There was a large drop in accuracy for models not using the strings. Comparing with previous figures depicting accuracy, there also were an increase in accuracy for the evaluation set compared to the test set.

Figure 5.1, Figure 5.5, Figure 5.9 and Figure 5.13 we find that the accuracy for the subproblem with humidity signals is about 96%, for the one including luminosity it is 92%, for PIR it is about 80% and for temperature it is 85%. Clearly, we see better results for all but for the luminosity subproblem. This could indicate that the evaluation set has nice examples for most of the models which they have found good patterns for and it also hints that the evaluation set might be much too similar to both the training and test set as mentioned before.

For specificity, seen in Figure 5.18 we notice the greatest difference for any models so far. The models that do not include the path as a feature are clearly having trouble with the signals that do not belong together. The low value tells us that these models have a tendency to couple signals that do not belong to the same equipment. This can also be concluded from Figure 5.19 where the wrong per right metric may be studied. The difference is also here striking. Overall the models that do not use the path as a feature tends to connect 8 incorrect signal couples for every 10 signal couples. The corresponding number for the models that do utilise the path is less than one incorrect guess per 10 examples.

Finally, we come to examine how the models perform in the ROC graph seen in Figure 5.20. Once again, it is obvious that the path feature plays a key role.

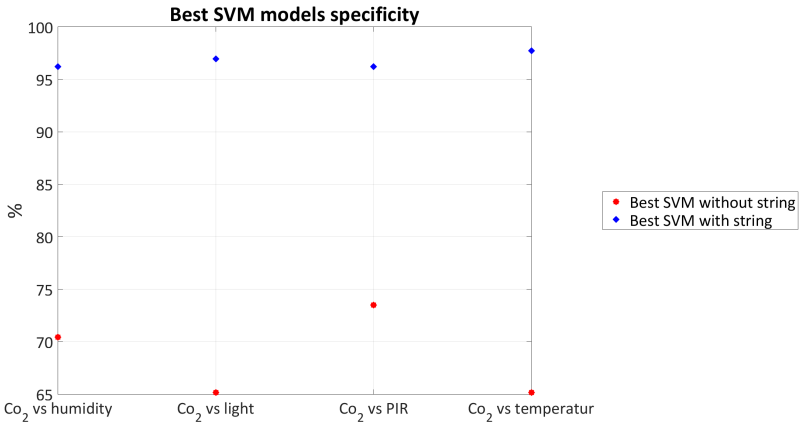


Figure 5.18 Specificity for models presented in Figure 5.17. The models not taking strings into consideration had significantly lower scores, indicating a tendency of matching signals not belonging together.

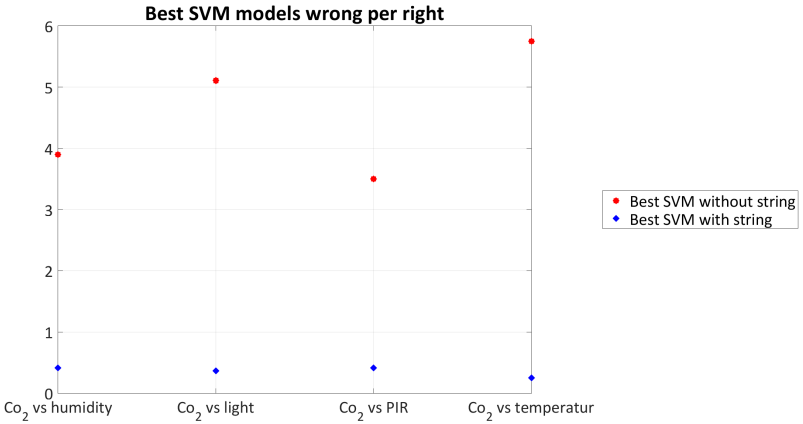


Figure 5.19 The ratio between the number of wrong guesses and the number of correct guesses for the best support vector machines. Just as in Figure 5.18 it was clear that the models with strings tended to couple signals which did not belong together.

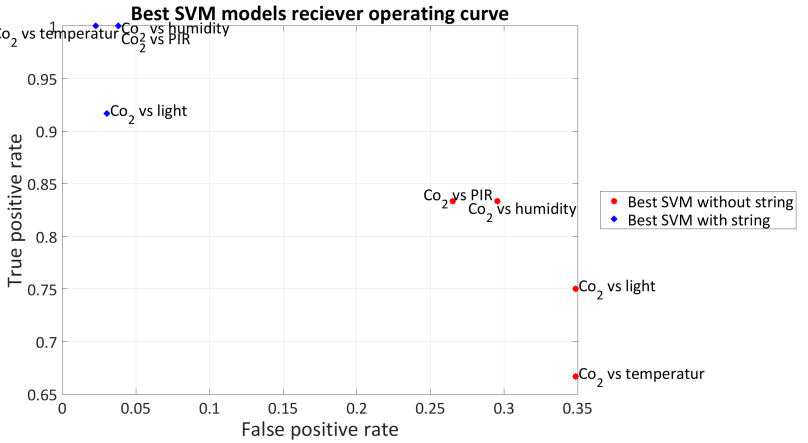


Figure 5.20 The ROC curve for the best support vector machines trained presented in this section. The string comparison makes a clear difference for the better.

Table 5.2 String models for Dataset *C*.

Overview of string comparison models on Dataset *C*	
Model	**Threshold**
1	0.80
2	0.85
3	0.86
4	0.87
5	0.88
6	0.89
7	0.90
8	0.91
9	0.92
10	0.93
11	0.94
12	0.95

5.2 Data set *C*

In this section we present the results of the string comparison method implemented for the association problem on Data set *C*. In Table 5.2 we see the different threshold values for the different models. The corresponding recall values can be found in Figure 5.21. As expected the accuracy decreases as the threshold value is increased. This is reasonable as the threshold sets the value for when two strings are considered equal enough to belong in the same equipment. As this threshold is increased the requirement on how alike the strings must be to be classified as "*Belongs together*" rises. Pairs that do belong together but have not got similar strings enough are missed hence the lower accuracy.

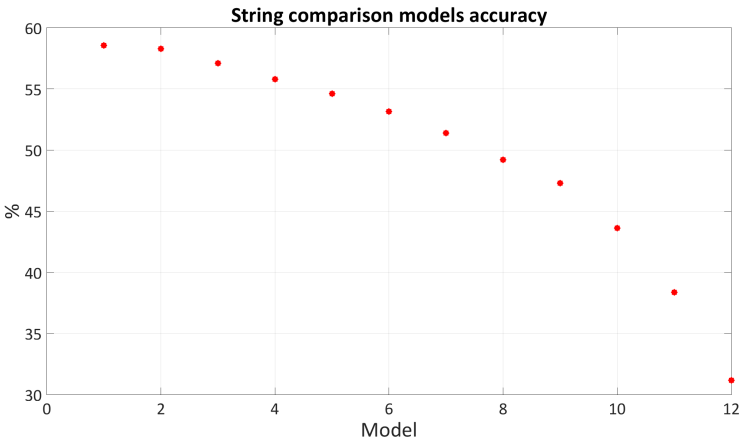


Figure 5.21 Accuracy for the string comparison models trained on Data set *C*. As expected, the accuracy decreased with increasing threshold value.

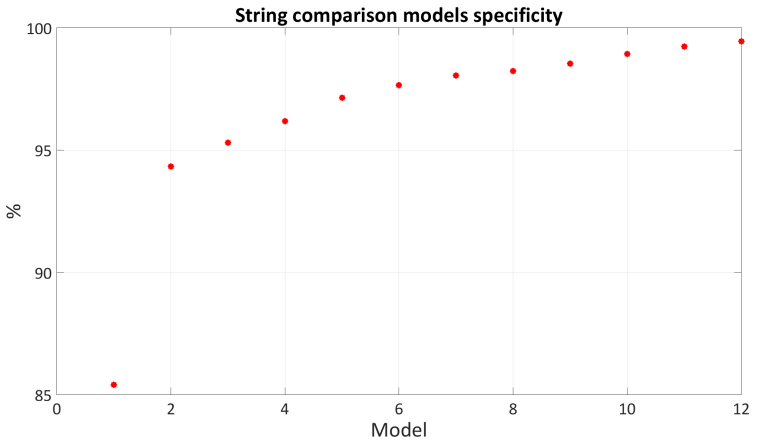


Figure 5.22 Specificity for the models presented in Figure 5.21. As less signals were wrongly included int he same group with an increased threshold the specificity increased with the threshold.

In Figure 5.22 we can see the specificity of the models. This metric increases as the threshold increases which is due to less signals being wrongly included into the same group. The desired outcome of recall and specificity is that both measurements should be as high as possible. For this method it is a hard trade off. Some of the stronger contenders would be models 2-4.

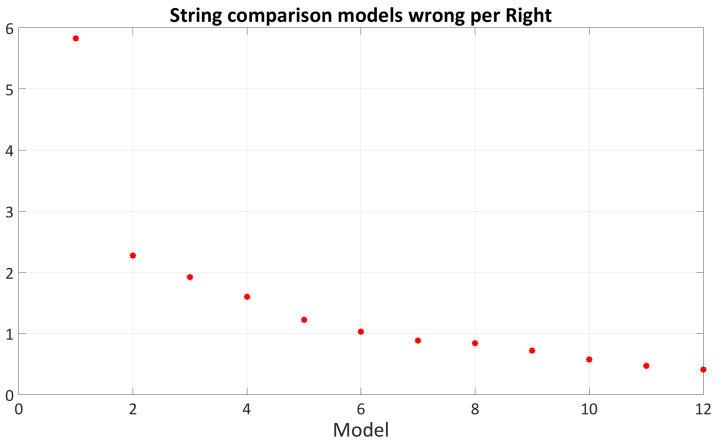


Figure 5.23 The ratio between the number of wrong guesses and the number of correct guesses for the best string comparison models. As for specificity in Figure 5.18 the result is getting better with higher threshold.

In Figure 5.23 we find how many times the model guesses wrong divided by how many time the model guesses right on an equipment pairing. This also decreases with an increasing threshold value which, for the same reasons as listed above, can be expected. Here model 2 receives a score of around 2.3 meaning that for every correct match 2.3 wrongly once were made by the algorithm. This is a bit higher than what we would like and to end up at only having a 1-1 ratio we need to regard model 6. The accuracy for model 6 was although much lower than what we would like, only about 53%.

In Figure 5.24 we find the ROC curve for the string comparison method. These results should be above the line $y = x$. And so it is for most cases. An optimal model, guessing everything right, would be seated in $(x, y) = (1, 0)$. It is clear that model 2 ends up far from this point however it is definitely in the right quarter. From the ROC curve models 3 and 4 perform approximately equally well.

Clearly it has been hard to decide which model is the best for this method. The answer will depend on the main objective and the requirements one has for the method. To decide what is an acceptable rate of the wrong per right metric could for instance be a good start but without this kind of decision made it is impossible to deduce which model was the best. What might be wise to remember is that one of the threshold values used here could give very different results for signals with names and paths named after another naming convention than the one used for Data set *C*. If one would tune this parameter on one building and then use this method with tuned threshold on another building, with another BMS, the performance could, in the worst case, end up pretty bad.

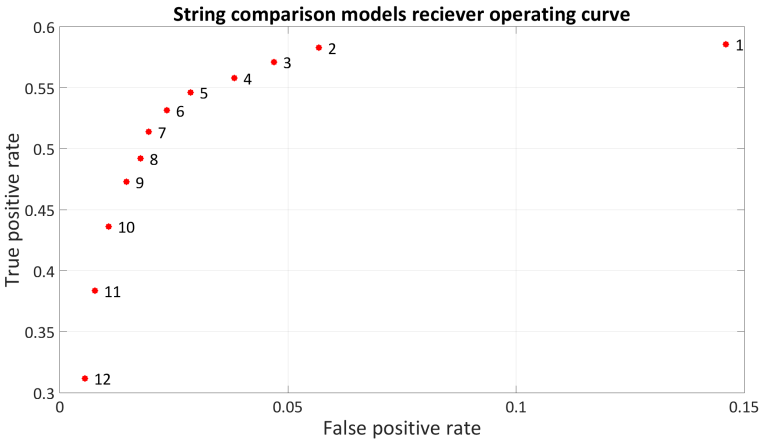


Figure 5.24 The ROC curve for the string comparison models. No model performed brilliantly, but all models were in the top right quarter of the unit square with model 3 and 4 being the best ones.

6

Conclusions

6.1 Data sets

The data available for this project has caused some issues. Due to the similarities in training, test and evaluation sets it is hard to create generalised models. This is caused by a few things but mostly due to only having access to one building in each classification system for most data sets. In these cases, it is possible to train models on the buildings, but hard to corroborate on the classification system. It is perhaps even truer for Data set *C* where the signals were split into many windows, making the individuals even more dependent on each other.

For Data set *B* and *C* we have also had unbalance between the classes and equipment which can be seen in Table 1.2, 1.3 and 1.5. Without countermeasures the algorithms could improve their performance by simply guessing on the most common type. We have dealt with this by weighting the types differently depending on how common they are. It is unclear from this thesis how a fully balanced set of signals would impact the solutions.

6.2 The classification problem

In Table 6.1 we see a summary of the results for the different models on the data sets. We see clearly that we have been unable to solve the classification problem to full satisfaction for all data sets, i.e. we have not for all data sets achieved a top one accuracy above 80%. This is the required accuracy for a fully automated classification process. To have a semi-automated process where the solution generates class suggestions could however still speed up the work with connecting the BMSs and analysis systems significantly. What we have been able to show is that machine learning methods are a feasible approach towards solving the classification problem in, at least, a semi-automated manner. We believe that the proposed solutions hold enough promise to merit further investigations.

Table 6.1 Overview of the overall best performing models for each method on the classification problem.

Best results for the classification problem				
Data set	**Method**	**Accuracy top one /%**	**Accuracy top five /%**	**Accuracy diff /percentage points**
A	Random forest	100	-	0.0
	Gradient boosting	100	-	0.0
	Neural network	99	-	0.4
B1+2	Random forest	85.0	98.8	9.5
	Gradient boosting	85.0	98.1	10.3
	Neural network	75.0	94.5	-11.6
B2	Random forest	63.2	88.9	29.6
	Gradient boosting	60.2	90.7	33.2
	Neural network	72.8	92.0	-1.5
C	Random forest	59.8	92.9	10.3
	Gradient boosting	61.0	93.1	7.0
	Neural network	57.0	92.7	2.8

For Data set *A* no signs of overfitting could be seen for any model and the accuracy for top 1 guess spanned 99-100%. Even if the data set is simple, only consisting of five classes, this still indicates that the methods used can solve the classification problem extremely well if fed with the right information.

For the little more complex system in Data set *B* we could notice signs of overfitting for training and test data from the same buildings. We also saw how this became a bigger issue as data from a building not in the training set was introduced for in the test set. By training and testing on different buildings in Data set *B*, we were able to show that it is possible to introduce the algorithms to completely new data and still receive useful results, especially for the neural network models that did not overfit. Unfortunately, with random forest and gradient boosting models there are problems with significant decrease in performance in this instance. We believe this to be caused by overfitting and not having enough generalised data when only training on one building.

Within buildings, in Data set *B1+2*, some of the gradient boosting and random forest models still had a top one accuracy of about 85% but these models were overfitted. Models that did not overfit, here only some of the neural networks, had an accuracy of about 75%. Corresponding numbers for accuracy top five was about 98% for the gradient boosting models and 84% for the neural network models. Top one accuracy is for all methods close to the requirement for a fully automated classification and the top three accuracies are high enough to imply a well performing semi-automated classification process. Across buildings, i.e. for Data set *B2*, the highest accuracy was of almost 73% for top one and 92% for top five belonging to a non-overfitted neural network model. The top one accuracy may be compared to the numbers presented in section 1.5 where Hong et al. in [Hong et al., 2015a] achieved an accuracy of 92% within a building an 82% across two buildings. Their accuracy scores are higher but since they only base their methods on six classes and we are using 16 classes an accuracy of 75% within a building and 73% across

two buildings is still a promising result. We may also compare the accuracy across buildings for our model, 73%, to 63% accuracy which was achieved in [Hong et al., 2015b]. Here the same data set is used and that our seemingly non overfitted model performs almost 10 percentage points better points to the auspicious prospects the methods used in this thesis holds.

For the target classification system in Data set *C* the highest top one accuracy was of only 61% and was seen for both random forest and gradient boosting models. It should however be noted that these models were overfitted and neural network models, that did not show signs of overfit, only achieved a top one accuracy of about 57%. The accuracy is overall lower than what is required for a fully automated classification process. However, as the best models had a top five accuracies of 93%, it is likely that these models could be used in a semi-automated solution. To pick among five suggestions instead of over 50 class options is obviously less cumbersome.

Going from Data set *A* to Data set *B* and finally to Data set *C* it is also clear that the performance of the models is decreasing. The increase in data set complexity is obviously a challenge for all methods. It can also be seen that for neural network models the parameter choice is increasingly important.

We have proved that the feature importance varies over the data sets. It seems clear that minimum and maximum value along with the mean of the data plays an important part over all three data sets. Gradient measures and the mean square error between the data and a fitted polynomial also carry weight quite consistently over the sets. It is however noticeable that the more complex the set, the more distributed is the feature importance. It would be preferable to further investigate how different combinations of feature sets would affect the result. As seen for neural network model 6 on Data set *A* fewer features can improve the result.

For random forest and gradient boosting models it is harder to come to general conclusions. We have seen little impact on either of the methods for either choosing the square root or second logarithm of the total number of features for the number of features considered for each split. We did however for Data set *C* see a decrease in performance for random forest models if all features were considered. Neither can we come to conclusions about which function to use to evaluate the split in the trees as this varies over the data set and has very little effect anyway. As for overfitting however, we can see for Data set *C* that it decreases when increasing the number of samples required for a node to split or be a leaf node. For gradient boosting overfitting seems to increase with the number of boosting steps, but also be strongly affected by the learning rate.

For the neural network a structure of six hidden layers consisting of 40, 60, 80, 80, 60 and finally 60 neurons were especially favourable for getting an overall good performance. With this structure the accuracy difference was best for dropout rate

Table 6.2 Overview of the overall best performing models for each method on the association problem. For SVM the average results for all best model is presented.

Best results for the association problem			
Method	**Data set**	**Accuracy /%**	**Wrong per right**
SVM with path and name	*A*	98.0	0.4
SVM without path and name	*A*	77.0	4.5
String comparison	*C*	56.0	1.6

0.3 but considering the other performance metrics we would deem that a dropout rate of 0.1 would be preferable. This might still need further investigation since test and training sets are much too alike. From the tests performed it also appears as if Adam is the optimiser to prefer and leakyReLU is the best activation function among the tested alternatives.

We can conclude that neural network models generally overfit less or roughly speaking as much as random forest and gradient boosting models. For less complex data sets, random forest and gradient boosting do better, but neural network clearly generalises better.

6.3 The association problem

Results on the association problem are varying which can be seen in the presentation of overall results in Table 6.2. On Data set *A* we have found that the implemented solution with support vector machines performs quite well, in fact on all comparisons it is possible to find a model with an accuracy over 80% on the test set. However this solution quickly becomes very complex and hard to implement on classification systems with less structure or more flexibility as the assumption is that it is always possible to find a certain signal type within an equipment, i.e. that a *Boiler* equipment would always contain a measurement point of type *Outlet water temperature*. We believe that more knowledge about the BMSs and classification systems is required to create a smart signal division.

As the best performing models were also trained without the path name as a feature we could clearly see the importance of that feature. Not using the path and name in any way would probably be a waste of the valuable information that, for us, increased accuracy more than 15 percentage points. One remark needs to be made however, the naming convention created for Data set *A* was a very nice one. To see exactly how much information can be gathered from the path and name for this method further investigation and more data with a more realistic naming convention would be required.

For the string comparison method implemented for Data set *C* we conclude that these strings hold a significant amount of information. It is however impossible to

come to general conclusion as the strings can differ significantly between buildings. String comparison is probably not a valid method to exploit in a real-life situation as it is, but could possibly be used in combination with other methods.

7

Future work

7.1 Data sets

One limiting factor during this thesis has been the data sets. For machine learning to be general the algorithms need general data which means several examples from all signal and equipment types and data from several buildings. We believe that, to continue with the work we have done, it will be imperative to increase the amount of data. Ideally, enough data would exist to perform tests on the following premises:

- Training on one building and testing on the same. This is a basic test where models may overfit without clear signs being revealed by the testing them on the test set. However, if good results are not attained here, they cannot be expected for more complicated test cases.
- Training on one building and testing on one other building. This test would show whether the model overfits or not, but with training on only one building it would be hard to create a generalised model.
- Training on a set of several buildings and testing on individuals from one building within the set. Training on several buildings should give a more generalised model and testing on a known building should still give good results. However, signs of possible overfitting might remain hidden.
- Training on a set of buildings and testing on one building not in the training set. This is the ideal test case and the one that most fully replicates a real-life situation. A good result on this test case indicates a very generalised model without overfitting.

As briefly explained in section 3.2 we have used functions within the method to handle unbalanced data set, i.e. a different number of individuals in each class, with weighting errors from classes differently. We think it would be interesting to investigate what difference a perfectly balanced training set would result in, i.e. a set

where there are the same number of individuals belonging to each class. A perfectly balanced set is the ideal for machine learning, but the problem with an unbalanced set should in theory be compensated for by weighting the feedback from the loss function.

Finally, we think that it could be promising to investigate other window lengths. For the results presented in chapters 4 and 5 we have used data windows spanning over two days and sampled with five minutes interval. One would hypothesise that longer windows could contain more information and therefore be beneficial to the algorithms. Nevertheless, this must be weighed against the practical aspect. The aim is that the final project of this work should be deployed fairly quickly in any real-life situation and costumers should not have to collect data for months in order to use it.

7.2 Classification

As demonstrated in Table 1.3, Data set *C* contains a number of classes of status types, e.g. ExhaustFanStatus and SupplyFanStatus. These types are a feedback signal from an actuator to a controller on what status it has. In a correctly functioning system these types have the same value as the corresponding run type, e.g. ExhaustFanRun and SupplyFanRun, which is the signal from the controller to the actuator. This means that the types are impossible to statistically separate from each other. We therefore propose to remove the status types, or to combine status and run types, and see if the result is improved. There could be other signal classes which are to similar to be statistically separable. This theory is supported by the low top one accuracy and high top five accuracy for Data set *C*especially, which indicates that all models trained for this data set confuses some classes with other classes. A study of which classes the models fail to separate would be of interest as this could be used for designing features holding better information separating these classes. A first step in this work could be analysing the confusion matrix for each model.

For neural networks we have clearly demonstrated that the parameters and the combinations thereof greatly affect the results. If one wishes to continue study neural networks it is therefore important to continue testing this. One should also note that the neural networks parameters we have tested are in no way all possibilities for neural networks in KERAS. We would especially recommend further looking into different optimisers and activation functions and try different structures of the network.

From this thesis it has become clear that different feature sets and different features had varying impact on the decision making for the random forest models. Hence, we believe that much can be gained from further investigation of the impact different constellations of features have on the methods. It might also be wise to further study the features themselves and see if they could be improved by adding new ones. For instance, we believe that a feature providing information on whether an individual

only takes integer values could be of interest. Another possibly good feature could be the unit of the measurement. We also came to think of how seasonal variations affected signals especially those connected to indoor and outdoor temperature and therefore a feature containing information of the season could be of use. Finally, it could prove useful to couple certain metrics, such as mean value and variance, to specific hours such as in or outside office hours. We believe this could be significant since we for some signal types have seen a clear difference during these times while for others the difference has almost been unnoticeable. If one wishes to utilise the office hours one does need to remember that data might need to be adapted to the time zone in which the building is located, otherwise the feature will have no reasonable meaning. We also noticed that the feature holding information on the mean square error for a fourth order polynomial mapped to the data was important. It might therefore be interesting to see if other orders of polynomials or maybe even other functions mapped to the data could be just as useful.

Together with feature exploration it would also be interesting to examine whether or not the models and methods find different connections within the data. This could for instance be seen if the classes that the models are most sure of differs between the models. One way to do this is to combine the predictions from different models, perhaps in a weighted fashion, and thereafter see whether the overall result increases, similar to what was suggested in [Hong et al., 2015b].

As we could see for the association problem and in [Balaji et al., 2015] and [Hong et al., 2015b] the signal paths and names seemed to contain much information. We believe that it would merit investigation to combine a machine learning algorithm with a string comparison for the classification problem as well. One way of doing this would be to let a machine learning algorithm predict a class for all individuals in the set. Depending on how sure the algorithm was on the class the individuals can then be divided into those considered labelled with certainty and those requiring further examination. The string comparison could then be done through finding similarities within signal paths and names among individuals deemed correctly classified and then comparing these to the signals not labelled with enough certainty.

One other feasible approach to combine algorithms is to divide the types into bigger groups, for instance sort all different types of temperature signals into one group and all status/run signals into another, and then let one algorithm classify signals into these groups. This could then be combined with a more specialised algorithm which could train on classifying classes within each subgroup. If this is to be tested we believe that a random forest or gradient boosting model would be good for the first partition and neural network for the more specialised subgroup classification. A random forest or gradient boosting model could be preferable for the coarse partition since they had such high accuracy for the data sets with fewer classes where the differences across classes were great. Another advantage for these methods was also the shorter training time compared to neural network. Neural network can find

more complicated correlations and would therefore probably perform better on the subgroups which might be harder to distinguish for each other.

Finally, it could prove successful to test out other classification methods than those mention in this thesis. We do however believe that the best solution will come from somehow combining techniques that find varying information in the data.

7.3 Association

As we could see in chapter 5, the signal paths and names seem to contain information useful in the association problem. We believe that it is possible to further investigate this and either combine a string comparison method with some other method or possibly find some other method capable of retrieving the interesting information in the strings. It is however important to remember that any such method needs to be independent of how signals are named, as this differs greatly between BMSs, and instead focus on correlation between strings.

With more knowledge about the BMSs and classification system it should be possible to further develop the features for this problem. We have focused on statistical events and when they occur, but it is likely that there are other things worth considering. One issue for us with this problem is that we could find no other way to find correlations between signals than a direct comparison. This is computationally heavy, and grows fast with the data as all individuals must be compared with each other. One big improvement would therefore be to exclude this from the process, either by smarter features or finding a way to limit the amount of comparisons necessary.

It could be feasible to investigate other methods for this problem. One method considered, but which had to discarded, was clustering the signals. In a clustering method, which is unsupervised, individuals are grouped by proximity in some feature space. We found it hard to come up with features that worked well for this type of method. However, with features that place signals belonging to the same equipment close to each other this type of method could work well and eliminate the need for calculating the difference between all signals. It could also be reasonable to explore clustering on the signal paths and names.

One final option we have briefly discussed, is removing as much of the class behaviour as possible from the signals by creating time series models for each class type and then perform association on the residuals between each individual and the time series model. This requires more computations, but could possibly be an important step when preparing the data for the association problem. Such preprocessing could become significant if clustering methods are to be used, this since the behaviour in signals are more similar for signals belonging to the same class rather than equipment.

Bibliography

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). *TensorFlow: large-scale machine learning on heterogeneous systems*. Software available from tensorflow.org. URL: `https://www.tensorflow.org/`.

Balaji, B., C. Verma, B. Narayanaswamy, and Y. Agarwal (2015). "Zodiac: organizing large deployment of sensors to create reusable applications for buildings". In: *Proceedings of the 2Nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. BuildSys '15. ACM, Seoul, South Korea, pp. 13–22. ISBN: 978-1-4503-3981-0. DOI: `10.1145/2821650.2821674`. URL: `http://doi.acm.org/10.1145/2821650.2821674`.

Berzal, F., J.-C. Cubero, F. Cuenca, and M. J. Martın-Bautista (2003). "On the quest for easy-to-understand splitting rules." *Data & Knowledge Engineering* **44**:1, p. 31. ISSN: 0169023X. URL: `http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true\&db=lih\&AN=7911504\&site=eds-live\&scope=site`.

Bishop, C. (2006). *Pattern recognition and Machine Learning*. Springer.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and regression trees.* The Wadsworth statistics/probability series. Belmont, Calif. : Wadsworth, cop. 1984. ISBN: 0534980538. URL: `http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=cat01310a&AN=lovisa.000243294&site=eds-live&scope=site`.

Chollet, F. (2018). *Deep Learning with Python*. Manning Publications Co.

Chollet, F. et al. (2015a). *Activations*. URL: `https://keras.io/activations/` (visited on 2018-05-23).

Chollet, F. et al. (2015b). *Keras*. https://github.com/fchollet/keras.

Clevert, D.-A., T. Unterthiner, and S. Hochreiter (2015). "Fast and accurate deep network learning by exponential linear units (elus)". *ARXIV* **2015arXiv151107289C**.

European Commission (2016). *Commission Staff Working Document Review of available information*. Tech. rep.

European Commission (2016). *Heating and cooling*. URL: https://ec.europa.eu/energy/en/topics/energy-efficiency/heating-and-cooling (visited on 2018-04-22).

Friedman, J. (2002). "Stochastic gradient boosting." *Computational Statistics & Data Analysis* **38**:4, pp. 367 –378. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=inh&AN=7222041&site=eds-live&scope=site.

Hinton, G., N. Srivastava, and K. Swersky (2014). *Neural networks for machine learning: overview of mini-batch gradient descent*. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (visited on 2018-05-20).

Hong, D., J. Ortiz, A. Aloke Bhattacharya, and K. Whitehouse (2015a). "Sensor-type classification in buildings". *CoRR* **abs/1509.00498**.

Hong, D., H. Wang, J. Ortiz, and K. Whitehouse (2015b). *The Building Adapter: Towards Quickly Applying Building Analytics at Scale*. Department of Computer Science, University of Virginia and 2IBM T.J. Watson Research Center, Yorktown Heights.

Hong, D., Q. Gu, and K. Whitehouse (2017). *High-dimensional Time Series Clustering via Cross-Predictability*. University of Virginia.

Hyvärinen, A., J. Karhunen, and E. Oja (2018). *Independent Component Analysis*. John Wiley and Sons, Inc.

Jakobsson, A. (2013). *An Introduction to Time Series Modeling*. Studentlitteratur AB, Lund.

Jerome H. Friedman, a. (2001). "Greedy function approximation: a gradient boosting machine." *The Annals of Statistics* 5, p. 1189. ISSN: 00905364. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true&db=edsjsr&AN=edsjsr.2699986&site=eds-live&scope=site.

Karpathy, A. (2017a). *Backpropagation, intuitions*. URL: http://cs231n.github.io/optimization-2/ (visited on 2018-02-28).

Karpathy, A. (2017b). *Convolutional neural networks: architectures, convolution / pooling layers*. URL: http://cs231n.github.io/convolutional-networks/ (visited on 2018-02-28).

Karpathy, A. (2017c). *Linear classification: support vector machine, softmax*. URL: http://cs231n.github.io/linear-classify/ (visited on 2018-02-28).

Karpathy, A. (2017d). *Neural networks part 1: setting up architecture*. URL: http://cs231n.github.io/neural-networks-1/ (visited on 2018-02-28).

Karpathy, A. (2017e). *Neural networks part 1: setting up the data and the loss*. URL: http://cs231n.github.io/neural-networks-2/ (visited on 2018-02-28).

Karpathy, A. (2017f). *Neural networks part 3: learning and evaluation*. URL: http://cs231n.github.io/neural-networks-3/ (visited on 2018-05-15).

Kotsiantis, S. B. (2013). "Decision trees: a recent overview". *Artificial Intelligence Review* **39**:4, pp. 261–283. ISSN: 1573-7462. DOI: 10.1007/s10462-011-9272-4. URL: https://doi.org/10.1007/s10462-011-9272-4.

Pedregosa, F., 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 (2011). "Scikit-learn: machine learning in Python". *Journal of Machine Learning Research* **12**, pp. 2825–2830.

Powers, D. (2007). *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Flinders University of South Australia.

Powers, D. M. W. (2015). "What the f-measure doesn't measure: features, flaws, fallacies and fixes". *CoRR* **abs/1503.06410**. arXiv: 1503.06410. URL: http://arxiv.org/abs/1503.06410.

Reddi, S. J., S. Kale, and S. Kumar (2018). "On the convergence of adam and beyond". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryQu7f-RZ.

scikit learn developers (2017). *Rbf svm parameters*. URL: http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html (visited on 2018-05-23).

Shukla, N. and K. Fricklas (2018). *Machine Learning with TensorFlow*. Manning Publications Co.

Suthaharan, S. (2016). *Machine Learning Models and Algorithms for Big Data Classification*. Springer.

Swartling, M., N. Grbic, and B. Mandersson (2016a). *Lecture 10*. URL: https://www.eit.lth.se/fileadmin/eit/courses/eti265/lectures/notes-lecture10.pdf (visited on 2018-05-28).

Swartling, M., N. Grbic, and B. Mandersson (2016b). *Lecture 5*. URL: https://www.eit.lth.se/fileadmin/eit/courses/eti265/lectures/notes-lecture5.pdf (visited on 2018-05-28).

Vert, J.-P., K. Tsuda, and B. Schölkopf (2004). *Kernel Methods in Computational Biology*. MIT Press.

Zhou Zhi-Hua, P. D. (2012). *Ensemble methods. [Elektronisk resurs] : foundations and algorithms.* Chapman & Hall/CRC machine learning & pattern recognition series. Boca Raton, Fla. : CRC Press, 2012. ISBN: 9781439830055. URL: http://ludwig.lub.lu.se/login?url=http://search.ebscohost.com.ludwig.lub.lu.se/login.aspx?direct=true\&db=cat01310a\&AN=lovisa.005043590\&site=eds-live\&scope=site.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
July 2018

Document Number
TFRT-6057

Author(s)
Anna Åberg
Christine Sjölander

Supervisor
Oskar Nilsson, Schneider Electric
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden
Anders Rantzer, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Building Data Classification and Association

Abstract

Almost half of the energy consumption in the EU originates from heating and cooling of buildings. The European Commission states that smart control of building systems may reduce the energy consumption. Cloud based smart control or even advanced fault-detection systems are becoming more common and should work for any building in the world. These systems need to receive data from the physical buildings which are commonly managed by a Building Management System, BMS.

Today, when connecting an advanced control or analysis system to a buildings' BMS is a manual process, more or less, which is time consuming and error prone. Therefore, it would be beneficial if this process could be automated. This thesis aimed to find machine learning methods that had the potential to be used to fullyor semi-automate the connection process.

By implementing and evaluating models of three machine learning methods, random forest, gradient boosting and neural network, we aimed to find some method able of labelling time series data into a fixed classification system with a precision of 80% or higher. The solutions were tested on three data sets with different complexity and we could show that for a set with low complexity it is possible to achieve perfect classification, i.e. accuracy of 100%. For the more complex sets accuracy decreased to roughly 60% and a fully automated solution from these models would not perform good enough. However, the probability that the correct class was among the top five predictions of the models remained high and therefore they could be used in a semi-automated connection process.

Overfitting was an extensive problem when classifying signals, especially for random Forest and gradient boosting models. We believe this is partly due to the data being too homogeneous and the situation could be improved by including data from additional buildings. The problems with overfitting could be seen most clearly when models were trained and tested on data from different buildings. In this case, random forest and gradient boosting models were clearly outperformed by neural network models that still scored about 60% accuracy without any overfitting.

We also attempted to group signals by equipment type. This was done via support vector machines and a string comparison method. The support vector machine solution was only possible to deploy on the least complex data set, but performed well with an accuracy of over 85%. To implement this solution on more complex data sets more knowledge about the system is needed. The string comparison method proved that much information could be gathered from the correlations in the signal names and paths. Nevertheless, it was hard to come to any general conclusions from this since data from only one BMS was used.We believe that the string comparison could give good results in combination with other methods.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-110

Recipient's notes

Security classification