UNIVERSITY OF TWENTE.

Faculty of Science & Technology

A Machine Learning Approach to the Detection of Malfunctions in HVAC Units

T.C. Veldman

B.Sc. Thesis
Advanced Technology
January 2022

Formal Methods and Tools Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Contents

Glossary

AI Artificial Intelligence

API Application Programming Interface, the way a particular program should be interfaced with

AUC Area under curve

CSV Comma Separated Values, a way of saving tabular data

DET curve Detection error tradeoff curve, way of evaluating the error rates of a classifier

FNR False Negative Rate, the ratio between *hot* datapoints labeled as *normal* and the total datapoints labeled as *hot*. Complements the False Positive Rate.

FPR False Positive Rate, the ratio between *normal* datapoints labeled as *hot* and the total datapoints labeled as *normal*. Complements the False Negative Rate.

HVAC Heating, Ventilation and Air Conditioning

ML Machine Learning

NS Nederlandse Spoorwegen (Dutch railways)

ROC curve Receiver operating characteristic, way of evaluating the diagnostic ability of a classifier

rs_nr Rolling Stock Number, the identification number of a particular train set

scikit-learn A machine learning package for Python

WE WagenEinde (Coach end)

1 Introduction

Physical systems tend to malfunction or break after an extended period of use. Furthermore, a failure of a single component may cascade to cause other problems. It may also be indicative of the necessity for broader maintenance. This general problem has existed for centuries [1], but retrieving and storing information about a system was a labor-intensive job. With the rise of computers, the gathering of extensive performance data for specific systems took off [2]. Since around the 1970s, the practice of vast data collection has evolved through two major waves, firstly becoming more prevalent in academia and later broadening into the domains of industry and business.

The collected data may be used for a multitude of applications, depending on the source, type and quality. One such application is the classification of new datapoints based on existing data. Being able to do this in an automated manner allows for quick and accurate problem detection, if done correctly. It may be used to find periods of interest in the functioning of some particular system, detect malfunctioning or aid in the decision-making for future developments.

1.1 Motive

This thesis details a case study on outlier detection in multivariate time-series data of HVAC (Heating, Ventilation and Air Conditioning) units installed in a particular type of NS (Dutch railways) train. NS collects data about the temperatures inside and outside the train coaches, which may be used to detect the functioning or malfunctioning of the cooling circuit of a nearby HVAC. The current method of fault detection during normal operation is fully human-driven, insofar that a service request has to be submitted by an NS employee or a traveller must file a complaint in order for a cooling malfunction to be registered. Faults may also be found during regular maintenance, which happens about four times per year.

The value of the temperature readings is dependent on many factors. The main factors are the outside temperature and whether the train is operational, but also the number of people in a coach, the time of day, whether the sun is shining or whether the doors are opened are influential factors. Furthermore, the definition of being broken is vague in itself: cases exist where an HVAC malfunctioned on one day, but seems to be functional again in the days after without any maintenance intervention having occurred. This multivariate nature of the measurement data voids the possibility for a simple threshold-based approach: the operational rules are not clear-cut enough to accurately describe each possible situation with high enough accuracy. The coach temperature may be allowed to surpass the setpoint temperature for some amount of time in some particular situations, without that being marked as a malfunction. In other situations a loss of functionality has occurred while the coach temperature is around or far below the setpoint temperature.

In light of the need for a more advanced detection solution and the nature of the data, the choice for a ML (Machine Learning) approach was made. On one hand because, when applied correctly, ML is able to take into account all the major rules and minor details that come with this particular system. On the other hand because ML can be more future-proof than e.g. numerical methods. More on this in section 2.3 on page 6.

1.2 Aims and Approach

The data in question is a noisy time-series, which moreover has occasional missing datapoints and seemingly unpredictable spurious behavior. A ML algorithm must be implemented to handle the different measurements and detect time-ranges where the temperature readings in a coach can be safely classified as an HVAC cooling malfunction. This algorithm must minimize false positives and maximize true positives. The algorithm will be limited to the detection of cooling malfunctions, indicated by excessively high temperatures.

To come to this end, a program will be created which takes as input the datafiles which are provided by NS. It firstly filters these files for completeness, incomplete datafiles are dropped. After this, the dataset is transformed so that it may be readily inserted into a ML model, the details of which may be found in section 4.4.1 on page 13.

The features which will be extracted are the temperature difference to the setpoint temperature, as well as the temperature difference to the median of the temperatures of all HVACs. A rolling window will be applied to the dataset to decrease the number of datapoints and thus increase the speed of the program, while not losing large amounts of information. A Logistic Regression model, Neirest Neighbors Classification model and Linear Support Vector Classification model will be separately used to determine which class a particular datapoint belongs to. The hyperparameters of the aformentioned models will be tuned using a 5-fold GridSearch Cross Validation method. The results of the optimized models are then analysed for correctness and performance, after which the best performing model is selected.

2 Background

2.1 Machine Learning

To properly explain ML is to dive deep into statistics, AI (Artificial Intelligence) and the study of generalization, trying to explain the many techniques that were developed since the inception of AI as a study in 1956 [3].

The idea of AI had already been around since 1943, when Warren McCulloch and Walter Pitts published their article on “A logical calculus of the ideas immanent in nervous activity” [4], inspired among others by Alan Turing. Its infancy ended with the Dartmouth workshop, where John McCarthy assembled a team of scientists to work out the specifics of McCulloch and Pitts’ work. AI went on to evolve through two periods of lay-low and revival, we currently find ourselves at the prosperous end in this second phase of revival. The many techniques of AI have now proven themselves time and again, as may be inferred from the current widespread use by the Big Tech companies, the breakthroughs in high-level competitions of strategy games and the mimicking of human abilities. Google alone has employed AI in thousands of projects, machines are better at chess than the current world champion and machine-based speech-, face- and handwriting recognition are considered routine technology.

ML is a part of AI, together with problem-solving, knowledge, reasoning, planning, communicating, perceiving and acting [3]. Some of these parts may interlink, ML is a technique that can be taken up within a chain of methods, though it can also work well on its own.

The exact definition of ML is quite uncertain and changes depending on the person asked. IBM defines it as ”a branch of artificial intelligence and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.” Tom M. Mitchell explains it as ”the study of computer algorithms that improve automatically through experience” [5].

Machine learning is certainly not magic. A good rule of thumb is ”If I can see a pattern, then some ML model will too.” The creation of a properly trained and tuned ML model requires considerable work and knowledge from the creator. Furthermore, machine learning is mostly limited to simple numeric data, and not every problem can be expressed in mere columns and rows of numbers.

2.2 The Machine Learning Workflow

The ML process knows several steps, some of which are more important than others. Chapter 2 of *The Machine Learning Landscape* [6] lays out a path for a full-fledged ML project.

In short, the steps are as follows:

1. Data is cleansed and transformed, so that it can be used as input for a ML model.
2. Part of the data is set apart, this is the test set that is used for final validation.
3. One or more models are chosen, which are trained and tuned based on the remaining data.

4. The model is validated using part of the training data. If necessary, step 3 is done again.
5. The trained and tuned model goes through a final validation using the test data.
6. If step 5 is satisfactory, the final model can be deployed.
7. Any problems that arise in the real-world application can be addressed by starting again at step 1.

2.3 Why Machine Learning?

The data used in this case study are simple numerical values, which may be readily transformed into a format suitable for machine learning. The question might arise why a numerical method is not used. Several reasons exist for this choice.

Firstly, because of other requirements that are put on this case. The resulting product needs to be easily extendable to other train types, as well as being adaptive to mistakes encountered when applied in the real world. When using a numerical method, this would all have to be done by hand. A machine learning model requires perhaps as much effort to be taught these things, but once a working model is up and running, it is much easier to retrain it than to have to alter a hand-tailored program. A major step is therefore saved in the future extension of the created program by taking the time to train a ML model now.

Furthermore, a threshold-based approach does not appropriately label all cases of malfunction. Data may be gathered regardless of a train being in operation or not, possibly causing temperature readings which are out of bounds but which do not say anything about the HVACs' functioning. On extremely hot days the temperature may exceed an otherwise appropriate boundary, which means an HVAC may not be able to compensate for the temperature but is still fully functional.

3 Context

3.1 The train

The scope of this report is limited to a particular train type which NS operates. These trains consist of several coaches. Each of these coaches has an HVAC unit build into the roof at either end. Some minor differences exist between HVACs in different coach types, but functionally they are all equivalent.

Each train set is built up out of different types of coaches. The different coaches have a predefined order to which all sets adhere. Although the coaches are not permanently linked, coach swaps rarely occur. However, an HVAC may be swapped out for maintenance, meaning that it is not safe to assume that the data for a particular coach in some known train set will always be of the same HVACs. Train sets may differ in length, but a train sets never change in length.

3.2 The HVACs

Each coach is equipped with two HVAC units, both of which follow the same programming, irregardless of its type. The primary purpose of the entire HVAC system is to keep the internal coach temperature at a pleasant level for the travellers by providing temperature-treated and filtered air.

The software which runs an HVAC unit is proprietary and must therefore be viewed as a black box. Its functional inputs are given by three different thermometers: two for measurements within the confines of the coach and another for the external temperature. Additional thermometers are installed inside the air ducts and the HVAC itself, but their measurement data is not stored. These thermometers function to aid the fault- and safety system, and are not used in this case study.

The HVAC aims to maintain a setpoint temperature for each deck, which depends on the external temperature readings. It achieves this through the use of an electric heater and a vapor compression cooler, which operate in a mutually exclusive manner. These heat or cool the air to the desired temperature. Several fans pull a mixture of external and recirculated air through these two components and a particle filter, and subsequently push it into the passenger compartment. A functional failure in any of these components is reflected in the data obtained by the thermometers.

The thermometers for each HVAC unit are placed near the entrances to the passenger compartment. An additional thermometer on the exterior hull of the train is shared between the HVACs. Each HVAC unit is responsible for temperature control on one side of a coach, abbreviated to WE (Coach end). Its components are marked with either WE1 or WE2, depending on the side at which it is installed. However, since the two sides of a coach compartment are directly connected to each other, air can freely flow between both sides of a coach. This means that each HVAC has some amount of influence on the temperature measured on the other side of its coach.

3.3 The Data

Temperature data for each thermometer is provided as a time-series with a period of ten minutes. The time-series are grouped per train set per day in a single CSV (Comma Separated Values) dataset. These files have not been filtered, there are many empty and incomplete datasets. The actual measurements take place about once every five seconds, but while getting the data in the aforementioned format, NS has averaged the temperature over an interval of ten minutes.

The data is presented with the following fields:

- rs_nr (Rolling Stock Number)
- Several date and time fields with different formats, each representing the timestamp of a measurement
- The number of measurements used to get the 10-minute average
- Averaged temperatures for each thermometer

The exact coach and side at which any particular HVAC is located is stored in the name of each temperature column. This information, along with the timestamp and rs_nr can be used to uniquely identify a datapoint. Some example data is provided in table 1.

Table 1: Some partially obfuscated temperature data as provided by NS.

rs_nr	date	hour_local	datetime_binned	count	coach1_T_WE1	coach1_T_WE2	coach2_T_WE1	coach2_T_WE2	coach3_T_outside	coach3_T_WE1	coach3_T_WE2	...
[redacted]	2020-08-18	14	2020-08-18T14:10+0200	120	26.00000	27.00000	22.00000	23.11250	23.88655	25.63866	25.00000	...
[redacted]	2020-08-18	11	2020-08-18T11:40+0200	119	23.50840	24.50000	24.50000	24.50420	22.55042	24.07983	24.00000	...
[redacted]	2020-08-18	12	2020-08-18T12:40+0200	120	24.48333	25.17500	21.75000	23.00000	22.01250	25.03750	23.63750	...
...												

4 Method

4.1 Overview

With future extensibility in mind, an Object Oriented approach was taken to create a watchdog for HVAC malfunctions. This method not only simplifies the procedure of adding more functionality to the project, but it also the aids in the application on other train types. NS dictates the program be written in Python.

This program may broadly be divided into three parts: Several sets of transformers to alter the structure of the datasets and extract features, a model class to define which transformers are run in which order, and a runner script to set up a basic working environment and start one or more models.

4.2 Conventions

The end program is not strictly structured, as the author was free to implement it the way he pleased. However, for readability's sake and ease of use, some conventions were adhered to. These are outlined in this section.

4.2.1 Transformers

Let us start with the transformer, the main building block of a model. Each transformer is a subclass of both `BaseEstimator` and `TransformerMixin`, as provided by the package scikit-learn, hence implying each transformer follows the scikit-learn API (Application Programming Interface). This means a transformer class implements the methods `fit()` and `transform()`. Since the `fit()` method has no actual functionality in a transformer, it was implemented in a transformer base class from which all other transformers inherit. The `transform()` method always takes a single input, the type of which may be freely chosen by the programmer. Following the conventions of `BaseEstimator`, any settings are specified before `transform()` is called, be it during instance initialization or thereafter via scikit-learn's `set_params()`.

Transformers may be divided into several different categories, depending on the function they provide. The author followed the custom of using a different file for each category, as this clearly shows the boundary between the different transformers. Transformers were given a name that reflects the operation they perform.

4.2.2 Model

The model ties together multiple transformers and a final scikit-learn model. Model instances are created in the runner script. As with the transformers, each model's settings are set during initialization. A model has a `run()` method, to execute the model. Since the transformers follow the scikit-learn API, a scikit-learn Pipeline can be used to set up the model execution.

4.2.3 Labels

The application of supervised ML requires the availability of labels. To ease the linking of labels and data, the label file contains the following columns:

- rs_nr
- date
- start_time
- end_time
- coach
- side
- label

All the information listed is required to uniquely identify a temperature measurement.

4.3 Model Training

4.3.1 Features

Any ML model needs features which provide adequate information about whatever data it needs to process. These features should ideally be as few as possible to keep the computational cost low, while still containing all relevant information. The features that were used in this case were the difference between the current temperature and the setpoint temperature, and the difference between the current temperature and the median of temperatures of all HVACs in this train set.

The difference between the current temperature and the setpoint temperature is used to include information about how much the temperature deviates from what is desired. Neither the coach temperature nor the setpoint temperature can provide this data on their own. Including both the setpoint temperature and the current temperature is also not wanted, since then the model may attempt to find correlations between one of these features and any other features that are provided. Such a correlation likely does not exist. Hence the difference between the two temperatures is used as a feature, to provide information about the temperature and setpoint and the correlation between them.

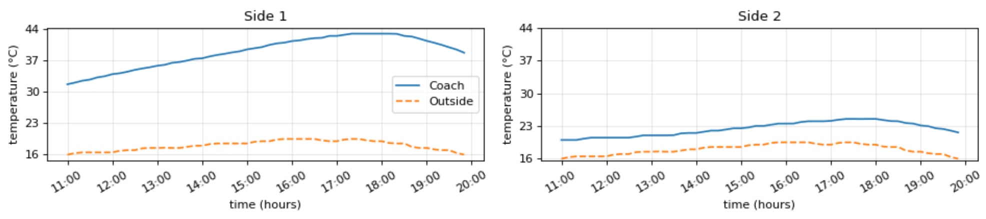
The choice behind including the difference between the current temperature and the median of temperatures of all HVACs follows much the same reasoning. Here the median is chosen to include information about the other HVACs, which has influence on the classification of an HVAC: If all HVACs follow a similar path over a certain time period, it is likely they are all functioning correctly. It is the outliers, the HVACs which deviate from this shared norm, that are more likely to be malfunctioning. The median -not the mean- is chosen to improve computational speed. If one were to use the mean, some effort needs to be taken to exclude the current HVAC from the calculation, which costs extra calculations. Using the median circumvents this necessity, allowing for a quicker program.

No other features were extracted from the temperature data due to time limitations. Additional feature candidates are discussed in section 6.1 on page 24.

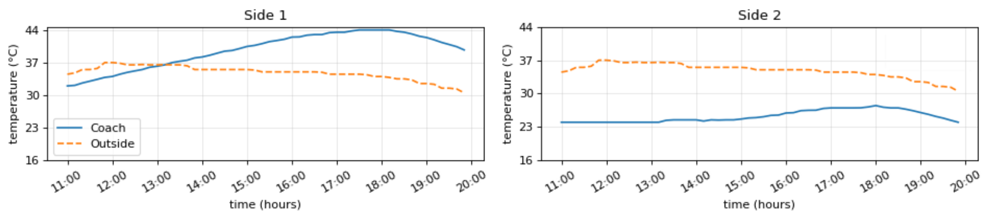
4.3.2 Labels

Supervised Machine Learning models need some information on the classification of their training data, which is provided in the form of labels. Datapoints must be labeled by hand, which in this case allows the programmer to define which temperature readings classify as normal and which as being too hot.

The classification of a particular temperature reading was based on graphs created from the data provided by NS. See figure 1 for some example data, depicting two coaches of the same train on the same day. In this example, both coach A side 1 and coach B side 1 were labeled as *hot* for the entire day, whereas coach A side 2 and coach B side 2 were both labeled as *normal*. When comparing their temperatures to the other coaches (which are not graphed), the conclusion could be made that they follow a similar path. Also notice that both temperatures at side 2 are clearly influenced by the temperatures at side 1, as is evident in the slight rise around 18:00. Furthermore, the outside temperature readings of coach A appear to be wrong, as all other HVACs have readings more similar to those of coach B. This also needs to be compensated for by the model when calculating a setpoint temperature.



(a) Coach A



(b) Coach B

Figure 1: Temperature graphs of both working and failing HVAC on a hot day. Solid blue lines indicate internal temperature readings, whereas the dashed orange lines represent the measured external temperature.

Such extreme cases as portrayed in figure 1 are easy to label, but oftentimes the difference in temperatures is not as clear-cut, see figure 2 on the following page. The temperature on both sides are stable and appear maintained, but the temperature of side 2 is several degrees above that of side 1. It is unknown whether this means that the HVAC on side 2 is starting to fail, or whether there is some other factor that influences this temperature outside of HVAC operation. This HVAC was not labeled.

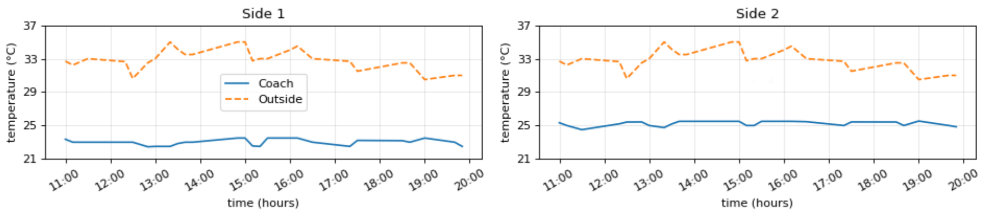


Figure 2: Temperature graphs of HVACs where it is unclear whether it is functioning properly. Solid blue lines indicate internal temperature readings, whereas the dashed orange lines represent the measured external temperature.

Another case can be seen in figure 3, where some excessively high temperatures are shown. However, these HVACs were labeled as *normal*, because when comparing these temperatures to those of the other HVACs, it was found they all follow a similar path. This train was likely not being used and had therefore been turned off, while the temperature data was still sent to NS.

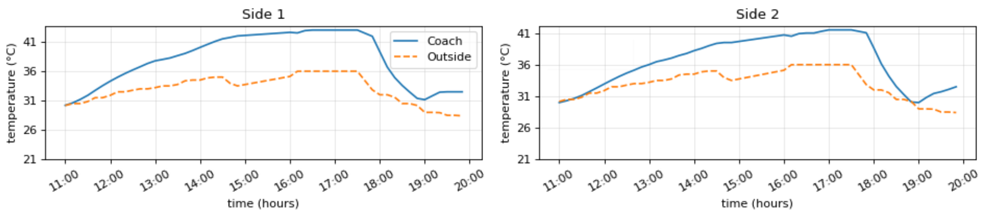


Figure 3: Temperature graphs for a train that is likely not in operation. Solid blue lines indicate internal temperature readings, whereas the dashed orange lines represent the measured external temperature.

As can be seen, the classification of temperature readings must take into account a multitude of factors. These factors should ideally also be represented in the features presented to the ML model.

Some amount of inbalance exists within the set of labels used to train the models for this case: There are a total of 60 *hot* labels and 264 *normal* labels. This has been accounted for in the Logistic Regression and Linear SVC models by balancing the weights given to each labeled datapoint, but not in the Nearest Neighbors model, as the latter does not allow supplying weights based on class.

4.4 Classes

The model, transformers and runner script interact with each other in a particular manner, as can be seen in figure 4 on the following page. The runner script creates a Model instance with the necessary parameters and calls its `run()` method. This model then guides the input data through several different transformers in a predefined order. Each Transformer instance must define a `transform()` and a `fit()` method. As explained in section 4.2.1 on page 9, each Transformer class implements a BasicTransformer class. After the transformation is done, the data can be used as training input for a predefined ML model. The `run()` method returns the trained model, the training set and a validation set that was taken apart from the input data. This split of training and test data is further elaborated on in section 4.4.2 on page 15.

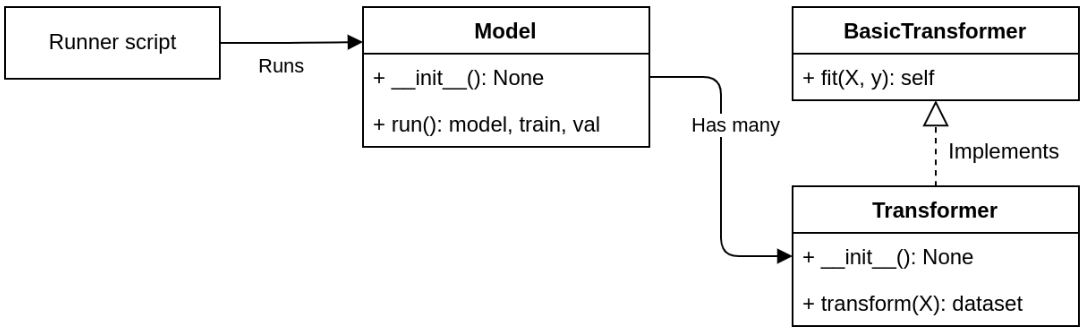


Figure 4: The generalized classes and their interaction.

4.4.1 Transformers

The author has chosen three categories: Filters, Extractors en Restructors. Each set of transformers plays a different role in getting from a raw datafile to a data structure usable by a ML model. A Filter class filters a dataset according to some criterion, dropping data that does not meet these criteria. An Extractor class processes the data, extracts desirable features and appends them to the dataset, so the Restructors may access them later on. Each desired feature has its own Extractor, so that features may be easily skipped or reordered. A Restructor somehow changes the structure of a dataset. That is, it does so without extracting any new features like what the Extractors do. In short, the Restructors transform a dataset such that it may be readily given as input to a ML model.

Several transformers were created to handle the data for the trains in question. These are listed and explained below, ordered in the order that they're applied to the dataset.

FilterIncompleteDataframes
Datasets are dropped according to the amount of data they contain, based on the ratio between datapoints present and total datapoints expected.

Since no method of filling missing data has been implemented yet, only complete datafiles are used. This equates to about $10\,000$ out of a total of $23\,000$ files.

ExtractSetpointTemperature
Extract a setpoint temperature from the outside temperature values.

The calculation of this value is based on information provided by this train's reference book, and follows a path similar to the one portrayed in figure figure 5.

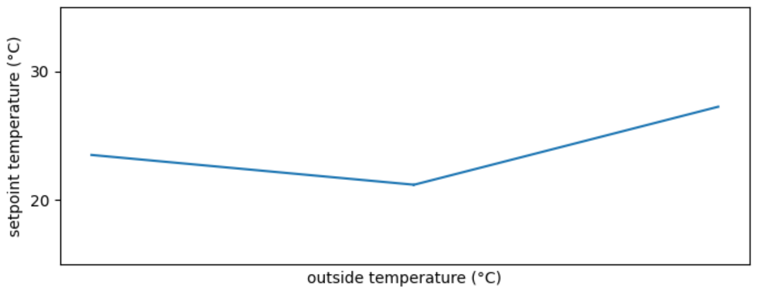


Figure 5: The approximate path of the setpoint temperature plotted against the outside temperature. Outside temperature values were deliberately left out.

ExtractDatetime
A Python `datetime` object is created from the date information stored in the datasets. This makes for easier datetime handling later on in the program.

TransformStructure
The structure of the datasets are transformed to match that of the label file. This makes label lookup easier. Furthermore, and more importantly, the new structure may be readily inputted into a scikit-learn model. The new structure has the following columns:

date	The date of this measurement.
time_start	The start time of this measurement.
time_end	The end time of this measurement, always 10 minutes after time_start.
rs_nr	The unique ID of each trainset.
coach	The coach this HVAC is placed in.
side	The coach side where the HVAC is placed.
T_hvac	The temperature for this HVAC.
T_setpoint_diff	The temperature difference to the setpoint temperature.
T_median_diff	The temperature difference to the median temperature of the other HVACs at this time.

The columns `date`, `time_start`, `time_end`, `rs_nr` and `HVAC` are used to uniquely identify a datapoint. The other columns contain the data that may be used as input for a scikit-learn model. Some example data is provided in table 2 on the following page.

Table 2: Some partially obfuscated transformed temperature data ready for model training.

rs_nr	date	time_start	time_end	coach	side	T_hvac	T_setpoint_diff	T_median_diff	label
[redacted]	2020-08-09	18:00:00	20:00:00	3	2	25.66959	-1.47369	0.39786	normal
[redacted]	2020-08-09	11:00:00	13:00:00	5	1	30.49532	-5.30963	-4.70728	hot
[redacted]	2020-08-09	12:00:00	14:00:00	5	1	32.38027	-6.99673	-6.59119	hot
...									

ApplyRollingWindow
Averages out the columns with data using a rolling window.

This is done for several reasons: On one hand because the temperature fluctuations under examination happen over the course of hours, not minutes. A sampling period of ten minutes is too small. On the other hand to reduce the number of measurements, hence improving speed.

A window with a size of $2\,\mathrm{hours}$ and a step size of $1\,\mathrm{hour}$ was chosen. This significantly reduces the number of measurements per HVAC per day from 54 to 8, while still keeping a good amount of accuracy. The step size of $1\,\mathrm{hour}$ ensures a degree of redundancy within the data, allowing one hour of data to be checked twice: once against the previous hour and once against the next. Using a larger window size may degrade model performance, as temperature fluctuations indicating HVAC malfunction exist which span a period of only two to three hours. A larger window would cancel out these fluctuations.

ConcatenateDataframes
Merges all datasets from the different datafiles into one large set.

AddLabels
Labels are retrieved from a Microsoft Excel spreadsheet and appended to the dataset. This label file follows the convention laid out in section 4.2.3 on page 10. Labels are appended to the dataframe, as can be seen in table 2.

DropMissingLabels
Any data that is not labelled cannot be used in training a model. It is therefore dropped from the training set.

4.4.2 Model

The model class determines the workflow of the ML process. Since each train type will have different data available and require different ways of processing, a separate model class is needed for each type.

The class created for this case implements the transformers defined in section 4.4.1 on page 13, as well as some caching functionality for transformed data. This class takes as input some information on the data files and labels, then splits the provided data into a training set and a validation set on a 4:1 ratio. This ratio is chosen because only about 1500

labeled datapoints exist. Splitting 4:1 allows enough datapoints to be in the validation set for a sensible conclusion to be reached from it, while not impeding the training of the ML model. It then trains a ML model with the training set and finally returns the trained model, training set and validation set. This implies that any validation and further processing is up to the runner script.

The model is created such that it labels the temperature readings on an hourly basis. Some additional processing needed to be done once those labels have been created to determine whether an HVAC should be taken in for maintenance. This includes filtering singleton events: The step size of $1\,\mathrm{hour}$ on a $2\,\mathrm{hour}$ window means that an actual short-time malfunction should label at least two readings as *hot*. Furhermore, if an HVAC is labeled as broken for only one single day, it is likely still functioning, but an HVAC that is labeled as broken for several days in a row is very probably malfunctioning. Further research on this matter is not included in this case report.

Splitting the data into a training and validation set is done per train, that is per rs_nr, meaning all the data of a particular train is in only one of the two sets. This is done because there is a great correlation between different days for the same train: An HVAC being broken on one day very likely infers that same HVAC being broken the days after. To stop this phenomenon from messing up the validation scores, the separation per rs_nr is done.

Several different ML models were applied on the processed dataset. These are the Logistic Regression model, the Nearest Neighbors Classification model and the Linear Support Vector Classification model, all supervised learning models. Each model's hyperparameters were optimized through cross validation and evaluated by the use of ROC curves (Receiver operating characteristic curves) and DET curves (Detection error tradeoff curves). Since there is a label inbalance in the training data, the class weights were balanced before being entered into a model.

ROC curves allow for condensing the vital performance information of a ML model into a single value, by calculating its AUC (Area under curve). This area was then compared against the AUC of the other models. The best model is usually the one that has the highest AUC. Additionally, the DET curve gives some insight in the prevalent error type (false positive or false negative) for particular classification thresholds. This case wants to minimize the number of false positives, that is, the number of working HVACs being classified as broken. A properly working HVAC being called in for maintenance costs NS much more money than a broken HVAC that flies under the radar for a short while.

Each ML model was trained with the same dataset, which had 1183 datapoints. Each datapoint contains the information described in the TransformStructure class on page 14. Hyperparameter tuning was performed with this dataset, using a 5-fold GridSearch algorithm as supplied by scikit-learn. The ROC curves and DET curves were create from a separate testing dataset of 342 datapoints. The two mentioned datasets did not have any overlap, i.e. the testing data was completely new for the models.

Logistic Regression
Logistic Regression is a fairly standard ML model that can be used for a wide variety of applications. Logistic Regression in a binary classification case such as this first calculates a probability value between 0 and 1, by solving the logistic regression hypothesis. It then solves the optimization problem that comes with finding the best decision boundary. [7] The Logistic Regression model performs well in binary classification problems such as this case, regardless of the number of dimensions.

Scikit-learn's implementation has several hyperparameters that have been optimized: The regularization strength (C-value), maximum number of iterations (max_iter) and the type of solver used.

Nearest Neighbors Classification
Nearest Neighbors is a type of model stemming from before the birth of ML. It is a very simple model that derives the class of a certain datapoint based on a particular number of neighboring datapoints. The Nearest Neighbors classifier works well on low-dimensional data, but is computationally expensive, as it stores all the training data for prediction.

Scikit-learn's implementation has several hyperparameters that have been optimized: The number of neighbors taken into account (n_neighbors), the leaf size for the tree algorithm used to find nearest neighbors (leaf_size) and the way of giving weight to a particular neighbor (weights).

Linear Support Vector Classification
The Linear SVC model is based on the line of Support Vector Machine models. This model is more advanced than the previous two, using hyperplanes to find the best margin boundaries for a particular dataset. [8] Support Vector Models often prove more useful in the context of high-dimensional datasets, but may still perform well in a low-dimensional case such as this.

Scikit-learn's implementation has several hyperparameters that have been optimized: The regularization strength (C-value), maximum number of iterations (max_iter) and the tolerance for the stopping criteria (tol).

4.4.3 Runner

A basic runner script has been created. Being a program entrypoint, this is the only part of the source code that is not contained within a class. The script runs the model described above, saves the trained model for later use and plots a confusion matrix, ROC curve and DET curve of the trained model and the validation set.

In essence it is only necessary to run the provided model once. The trained ML model can then be saved and loaded again at a later point in time. The data transformation pipeline can be gotten from the model without the necessity to train again. Classification can then be done by calling `predict()` on the loaded ML model with the transformed data.

4.5 Package

The program described in previous parts of this section has been packaged for easy distribution and continued development. The structure is as follows:

```
hvac_watchdog/
├─ model/
│  └─ Model.py
├─ transformers/
│  ├─ Extractors.py
│  ├─ Filters.py
│  └─ Restructors.py
└─ main.py
```

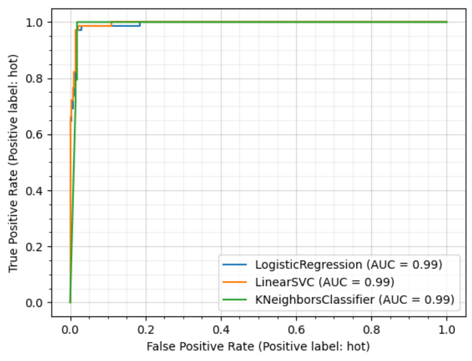
5 Results

This section shortly describes the results that were gotten by doing the model training and hyperparameter optimization described in section 4.4.2 on page 15. It presents a small window into both the general and some specific test results.

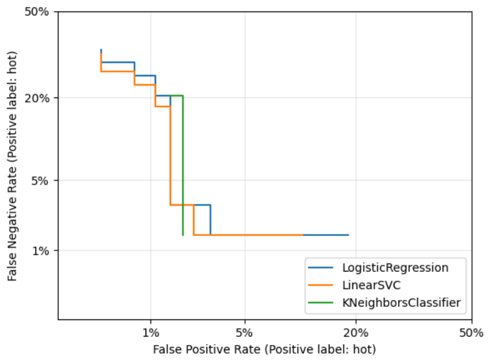
5.1 Unoptimized Performance Curves

Figures 6a and 6b show the ROC curves and DET curves before hyperparameter optimization has taken place.

Notice that according to the DET curves, all three models have a greater tendency to classify False Negatives than False Positives.



(a) ROC curves



(b) DET curves

Figure 6: Unoptimized performance curves for the Logistic Regression (blue), Linear SVC (orange) and Nearest Neighbors Classification (green) models.

5.2 Optimizing Values

Below are listed, per ML model, the optimal hyperparameters used to obtain the figures in the succeeding sections.

Logistic Regression
The best mean score of 0.964 was found for these hyperparameters:

C	Tested for values from 1×10^{-6} until 1×10^{8} on a logarithmic scale, then narrowed down to 1×10^{-5} to 1×10^{-3}. The optimal value ranges from $3.764\,935\,806\,792\,467\,5 \times 10^{-4}$ to $5.462\,277\,217\,684\,342 \times 10^{-4}$, a mean value of approx. 4.6×10^{-4} was used.
max_iter	Tested for values 10, 100 and 1000. The number of iterations seemed not to matter, so the default of 100 was chosen.

solver	`lbfgs` performed better than `liblinear`.

Nearest Neighbors Classification
The best mean score of 0.965 was found for these hyperparameters:

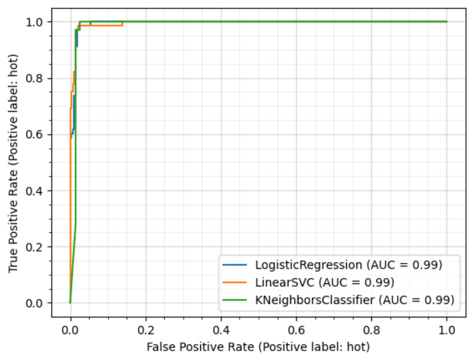
n_neighbors	Values ranging from 2 until 30 were used. 20 worked the best.
leaf_size	Tested for values between 10 and 100. The leaf size did not have any impact in the test score, so the default of 30 was used.
weights	`distance` performed better than `uniform`.

Linear Support Vector Classification
The best mean score of 0.973 was found for these hyperparameters:

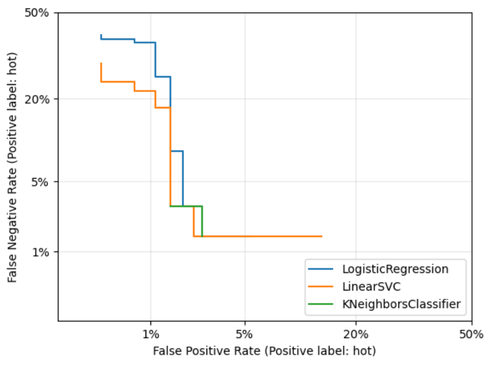
C	Tested for 1×10^{-5} to 1×10^{-3}, like for Logistic Regression. The value 4.6×10^{-4} happened to be a good candidate again.
max_iter	Tested for 100, 1000 and $10\,000$. The number of iterations seemed not to matter, so the default of 1000 was chosen.
tol	Tested for values from 1×10^{-6} until 1×10^{-2}. The value 2.89×10^{-6} performed the best.

5.3 Optimized Performance Curves

Figures 7a and 7b show the ROC curves and DET curves after training the model with the hyperparameters listed in section 5.2 on the preceding page. Notice that the FPRs (False Positive Rates) have improved a bit. The AUCs of the ROC curves have not increased.



(a) ROC curves



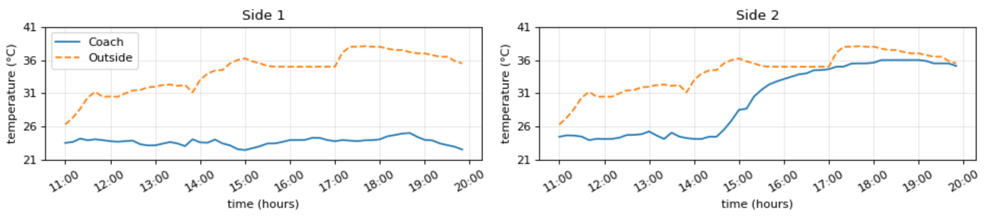
(b) DET curves

Figure 7: Performance curves for the Logistic Regression (blue), Linear SVC (orange) and Nearest Neighbors Classification (green) models.

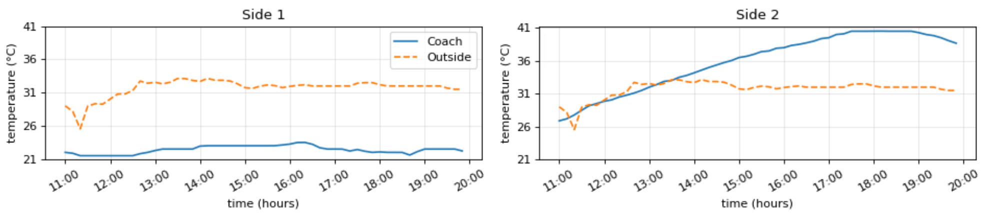
5.4 Data Labeling

Some of the testing data given as input to the model which contains fairly evident HVAC failures is shown in figure 8. This data was labeled with *hot* by the three models as follows:

Logistic Regression	Coach A side 2, between 15:00 and 20:00 and coach B side 2, between 11:00 and 20:00
Linear SVC	Coach A side 2, between 14:00 and 20:00 and coach B side 2, between 11:00 and 20:00
Nearest Neighbors	Coach A side 2, between 15:00 and 20:00 and coach B side 2, between 11:00 and 20:00



(a) Coach A



(b) Coach B

Figure 8: Temperature graphs for some failing HVACs within a particular train used as model input. Solid blue lines indicate internal temperature readings, whereas the dashed orange lines represent the measured external temperature.

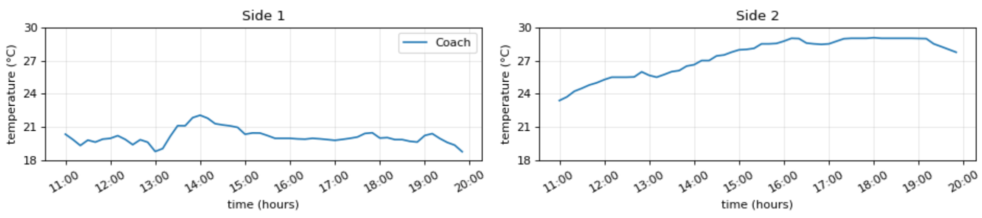
Some of the testing data that was incorrectly labeled is shown in figure 9 on the following page. In figure 9a, the period from 12:00 and further was labeled as *hot*, the models predicted that period as follows:

Logistic Regression	12:00 to 14:00 was labeled as *normal*, 14:00 onward was classified as *hot*.
Linear SVC	12:00 to 14:00 was labeled as *normal*, 14:00 onward was classified as *hot*.
Nearest Neighbors	12:00 to 15:00 was labeled as *normal*, 15:00 onward was classified as *hot*.

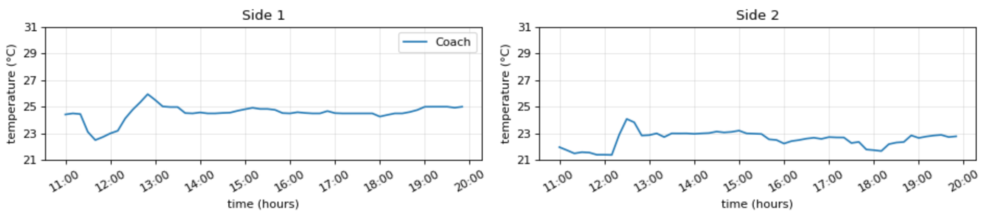
Then figure 9b, which had both sides of the coach classified by the model as *normal*, except for a the 16:00 to 18:00 region of side 2, which was labeled as *hot*. All other HVACs show similar readings during the same period, but are not labeled the same. Neither the Linear SVC model nor the Logistic Regression model labels this period as *hot*.

Finally, figure 9c shows some readings on side 2 which should be labeled as *hot* starting at about 14:00 until the end of the day. The models label this period as follows:

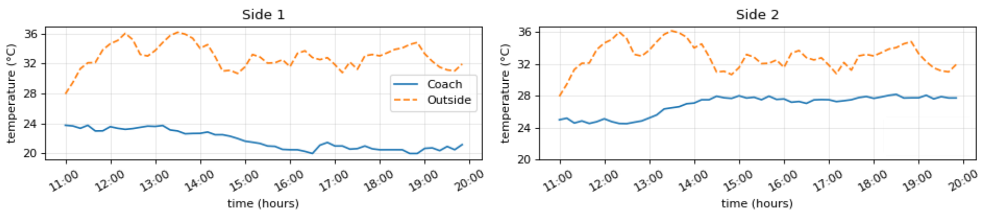
Logistic Regression	Fully labeled as *normal*, except for a single *hot* reading in the period between 14:00 and 16:00.
Linear SVC	Labeled as *hot* from 13:00 onward.
Nearest Neighbors	This entire period is labeled as *hot*, except for a single reading: The period between 16:00 and 18:00 is labeled as *normal*.



(a) Coach A



(b) Coach B



(c) Coach C

Figure 9: Temperature graphs for some incorrectly labeled readings, taken from different trains. Solid blue lines indicate internal temperature readings, whereas the dashed orange lines represent the measured external temperature.

6 Discussion

All three models appear to perform well with the chosen features. The use of hyperparameter tuning ever so slightly increased their performances, but the effect is minimal since the performance was already good to begin with. All three hyperparameter-optimized models have an AUC value of 0.99, which is exceptional. The DET curves in 7b on page 20 show that the performance is more focused on producing a minimal number of False Positives.

The reason why the AUC values are so exceptionally good may be because it was mostly the more extreme cases that were manually labeled. These cases tended to have temperatures above $30\,^{\circ}\mathrm{C}$ and readings which clearly deviated from the norm. It was also these more extreme cases that were used to obtain the ROC curves and DET curves. This means the model is good at picking out the HVACs with obvious defects. Not much can be said about HVACs on the boundary between working and malfunctioning, based on just these curves.

For that matter, we turn to the unprocessed results as presented in section 5.4 on page 21. These results were cherry-picked to demonstrate some points about the performance of the different models close to the decision boundary. With the labeling of figure 8 on page 21 it is demonstrated that all three models can indeed label cases where an HVAC is clearly malfunctioning. The data in figure 9 on the preceding page gives more information about performance around the decision boundary. The decision boundary seems to lie somewhere in the range of $26\,^{\circ}\mathrm{C}$ to $30\,^{\circ}\mathrm{C}$ for all three models.

Figures 9a and 9c give a glimpse into the boundary behaviour of the three models. Though it appears to be much the same for all three models, a distinction can be made based on figure 9c: The Linear SVC model is the only one that labels this case correctly, while the Nearest Neighbors model is close. The Logistic Regression model appears to have its decision boundary at some higher value, since it mostly labels the period as *normal*.

The Nearest Neighbors model has more quirks, as is evident from the labeling of figure 9b. The labeling around the decision boundary is random to a certain degree, as the labeling of a particular reading depends on the labeled training datapoints nearby. Since the density of these training points is sparse near the boundary area, the class of a new point depends on far-away datapoints. If a closely bunched set of datapoints of one particular label happen to fall within the reach of the datapoint in question, then that label is more likely to be picked. In this case, the *normal* is more likely to be picked because of the class inbalance discussed at the end of section 4.3.2 on page 11: Because the boundary lies between the sets of classes, it will come across more *normal* labels than *hot* labels.

Not much can yet be determined from the unprocessed data in section 5.4 on page 21 in regard to the FPR and FNR (False Negative Rate) at around the decision boundaries of the three models. For the more extreme cases, an FPR of just above $1\,\%$ can be reached by all models at an acceptable FNR of between $2\,\%$ to $10\,\%$. Since the classification which results from the ML model needs some additional processing -as described in section section 4.4.2 on page 15- to determine whether an HVAC should go into maintenance, an FPR of just above $1\,\%$ is good enough.

Based on the above, the choice for which ML model to use comes down to whether datapoints on the decision boundary should be treated in a restrained manner or be labeled as *hot* more quickly. In the former case, Logistic Regression seems to be a good candidate, whereas the Linear SVC model seems appropriate for the latter. The Nearest Neighbors model could also be a good candidate for the latter case, but it may need some additional training with a more balanced labeling and more labels for the less extreme cases.

6.1 Future research

Since the performance of these models is already good, future research need not focus on trying different models. It could instead focus on the processing of incomplete datafiles. This case study solely made use of complete datafiles, no effort was taken to fill in missing data: These datasets were simply dropped. However, many datasets exist in the current pool which are more than 90% complete. Datapoint interpolation is a viable option to fill in this missing data, which could allow for the detection of more broken HVACs.

Only a handful of features were used in the training of this model, perhaps others exist that are well-suited for this train type. On that note, not all train types have data available to calculate the setpoint temperature. More features would need to be found for those train types to give enough information to the ML model in case difference to the median of all HVAC temperatures does not prove to be enough. Features such as the slope of the current reading, temperature differences on previous days or some predefined setpoint temperature could be looked into.

Furthermore, as discussed in section 4.4.2 on page 15, the labels which the trained model creates need some additional processing to determine whether an HVAC should be sent down for maintenance. Singleton events should be filtered out, as those are very likely to be incorrect. Some threshold needs to be found for the number of days that should be checked before marking an HVAC as definitely broken, possibly being dependent on the maximum temperature reached during the period of malfunction.

Finally, it would also be a good idea to compare the model created for this case report to real-world data: Did an HVAC labeled as definitely broken after the additional processing described above actually need the maintenance? If too many False Positives still exist, the model will need to be retrained.

7 Conclusion

The goal of this case study was reached: An accurate ML model was created which is able to classify an HVAC unit as either operational or malfunctioning during a two-hour window. Several different hyperparameter-optimized ML models were presented, depending on the desired accuracy around the decision boundary temperatures of $26\,^{\circ}\mathrm{C}$ to $30\,^{\circ}\mathrm{C}$. The Logistic Regression model seems to be a good candidate for a careful algorithm, whereas the Linear SVC and Nearest Neighbors models are less restrained in that area. All three models perform well for the more extreme cases often characterised by temperatures rising above $30\,^{\circ}\mathrm{C}$. The program created to achieve this goal is well-structured and documented, and may be easily improved or extended.

References

[1] V. Hodge and J. Austin, "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, May 2004, ISSN: 0269-2821. DOI: 10.1023/B:AIRE.0000045502.10941.a9. [Online]. Available: https://eprints.whiterose.ac.uk/767/1/hodgevj4.pdf.

[2] K. L. Tsui, Y. Zhao, and D. Wang, *Big data opportunities: System health monitoring and management*, 2019. DOI: 10.1109/ACCESS.2019.2917891.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Prentice Hall, 2020.

[4] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943. DOI: 10.1007/BF02478259.

[5] T. Mitchell and M. Hill, *Machine Learning*. 1997.

[6] A. Géron, *The Machine Learning Landscape*, 2nd ed., N. Tache, Ed. O'Reilly Media, Inc., Jun. 2019.

[7] *Understanding logistic regression step by step*. [Online]. Available: https://towardsdatascience.com/understanding-logistic-regression-step-by-step-704a78be7e0a (visited on 11/01/2022).

[8] *Support vector machines*. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html (visited on 07/01/2022).