Master's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of control Engineering

Application of decision trees to failure detection in HVAC systems

Bc. Martin Zázvorka

Field of study: Cybernetics and robotics, subfield: Cybernetics and robotics

2020

Supervisor: Ing. Jiří Dostál

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Zázvorka Martin**	Personal ID number:	**457217**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Application of decision trees to failure detection in HVAC systems

Master’s thesis title in Czech:

Využití rozhodovacích stromů při návrhu pravidel pro detekci poruch technických zařízení budov

Guidelines:

1) Study decision trees and evaluate their utility for HVAC failure detection.
2) Validate the principles of selected decision tree methods for a simple scenario and/or modify the approaches to suit complex scenarios.
3) Apply the developed methods to annotated real building data.
4) Compare the methods in terms of accuracy, robustness, and practicality.

Bibliography / sources:

[1] Yan R., Ma Z., Zhao Y., Kokogiannakis G., A decision tree based data-driven diagnostic strategy for air handling units, Energy and Buildings, Volume 133, 2016, Pages 37-45, ISSN 0378-7788
[2] CHaturvedi S., Patil S., Oblique Decision Tree Learning Approaches - A Critical Review, International Journal of Computer Applications, 82, 6-10, 2013, 10.5120/14174-2023

Name and workplace of master’s thesis supervisor:

Ing. Jiří Dostál, Department of Control Engineering, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **10.02.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Jiří Dostál Supervisor’s signature	prof. Ing. Michael Šebek, DrSc. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt Student’s signature

Acknowledgement

I would like to thank Ing. Jiří Dostál for his councel and steering me in the right direction whenever it was needed.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 22. May 2020

....

Abstrakt / Abstract

Diplomová práce se zabývá využitím rozhodovacích stromů při návrhu pravidel pro detekci poruch technických zařízení budov. V první části této práce jsou představeny koncepty strojového učení a jsou popsány rozhodovací stromy. Na základě teoretických poznatků jsou vybrány vhodné metody rozhodovacích stromů a to CART, Random Forest a šikmé rozhodovací stromy (OC1). V dalších sekcích je představena implementace vybraných metod a jsou popsány doporučené postupy, jak dané metody používat a nastavit jejich parametry. V poslední části diplomové práce jsou vybrané metody porovnány na sadách reálných dat z technických zařízení budov. Jednodušší část testování proběhla na otopných okruzích a následně byly metody použity na diagnostiku vzduchotechnický jednotek.

Klíčová slova: Diagnostika, Rozhodovací stromy, Náhodný les, Klasifikace

This thesis seeks to analyze the usage of decision trees in diagnostics rule design in HVAC. At first, theoretical concepts of machine learning and decision trees are introduced. Viable methods are selected based on theoretical research. These methods are CART, Random Forest and oblique decision tree classifier (OC1). Implementation of selected methods is described as well as their limitations and guide to parameter tuning. Lastly, methods are compared using real-world datasets from HVAC systems. The comparison starts at more straightforward tasks from heating circuits and then follow up with problems from the air handling units domain.

Keywords: Diagnostics, Decision trees, Random Forest, Classification

Contents /

Tables / Figures

Chapter 1
Introduction

HVAC (heating, ventilation, and air conditioning) systems are nowadays present in almost every building. Quality of control and mechanical conditions of such systems have an impact on occupant's comfort, energy usage and maintenance cost. Undetected failure can lead to increased operating costs even when comfort can be still preserved. Without the use of diagnostics, such faults can stay undetected for a long time. Many buildings have SCADA (Supervisory control and data acquisition) systems for online collection and storage of operating data. With the use of diagnostics operating data can be analysed to exploit control faults and detect unusual changes in systems performance that are caused by mechanical faults.

In many applications HVAC systems, especially AHU (air handling units), are not controlled optimally. Heat recovery is underutilised in cooling dominated buildings and switching between cooling and heating is common. Detection of such faults can lead to significant energy savings.

Machine learning methods can be used for fault detection [3] [18]. While the use of black-box machine learning methods can be very effective, it does not provide any explanations of trained models which could be checked or used by an expert.

Grey-box machine learning methods can be used on labelled data to describe system behaviour in the moments of faults. HVAC systems often consist of tenths or hundreds of sensors. Grey-box machine learning methods can be used to detect significant features, that can be analysed by an expert later on.

Designing diagnostic rules is very time consuming and requires expert knowledge in a given HVAC domain. Machine learning methods pose a new approach that can help experts and perhaps, in some cases, substitute them entirely. Machine learning methods can also be used to evaluate existing diagnostic rules. A team from UCEEB (University centre for energy efficient buildings) extensively diagnoses AHU units with a broad set of diagnostic rules based on their expert knowledge. Up to 100 experiments were conducted over the past year. One of the ambitions of this thesis is to diagnose faults from this experiment with decision trees and compare the resulting model with diagnostic rules.

1.1 Goals and structure of the thesis

This master thesis is focused on the applicability of decision trees in HVAC systems concerning interpretability. In the control domain interpretability of trained models is as important as their performance.

The assignment points of this thesis are the following:

- Study decision trees and evaluate their utility for HVAC failure detection.
- Validate the principles of selected decision tree methods for a simple scenario and/or modify the approaches to suit complex scenarios.

- Apply the developed methods to annotated real building data.
- Compare the methods in terms of accuracy, robustness, and practicality.

The thesis starts with an introduction and goals of this thesis. Following chapter focuses on the introduction to machine learning. Chapter 3 contains theoretical research on decision trees. Implementation of selected methods and examples of trained models are described in Chapter 4. Parameter tunning is tackled in Chapter 5. Chapters 6 and 7 describe decision tree usage on data from real HVAC systems. Finally, results are concluded in Chapter 8.

Chapter 2
Machine learning

In this chapter basic principles of machine learning are described as well as its basic taxonomy and properties.

Machine learning, with its data–driven methods, offers a different approach to engineering problems. Due to experience obtained throughout the years, data–driven methods are now used alongside with logic–based expert systems. This section will focus on some basic properties of machine learning methods, counter the question of when it is appropriate to use them over a logic–based system and primary machine learning taxonomy will be introduced.

Conventional engineering design flow starts with modelling of the desired system. System modelling requires domain knowledge as well as information on process and measurement noises. Some of the required information might not be available due to many reasons such as unknown sensor parameters, technical or military secrets. If enough information is available, a mathematical model describing the behaviour of the given system can be created. Based on the mathematical model, an algorithm solving the given task is developed. If the model describes the real-world system accurately, the developed algorithm can provide performance guarantees.

Machine learning methods replace the complex task of system modelling by data acquisition. With data–driven methods resulting algorithm can often be developed without any model just with the usage of hypothesis classes and large dataset representing system behaviour.

A compromising option is to use the best of both worlds. Machine learning methods can significantly benefit from additional prior information. This information can consist of sensor accuracies, physical limitations of system behaviour, parameters or additional knowledge. Necessary information such whether machine was incorrectly calibrated last week can be crucial for the performance of machine learning methods.[28]

2.1 Machine learning taxonomy

Machine learning methods can be divided into three main classes:

- **Supervised learning**
  The input of supervised learning is a set of data tuples. Tuples consist of attribute parameters and a label. An example of supervised learning would be classification of cars into insurance categories (labels) based on their age and engine size (attributes). The result of supervised learning is mapping between attribute space and label space.

- **Unsupervised learning**
  Unsupervised learning is a machine learning technique that works with data without labels. Unsupervised learning algorithms such as clustering are used to get structure or patterns in training data.

- **Reinforcement learning**
  In contrast to supervised learning where all needed or at least all usable information is already contained in the training set, reinforcement learning uses an agent approach. It is also referred to as semi–supervised learning and is usually modelled as a Markov decision process. Learning is done via agent interaction with an environment to maximise reward function.

2.2 Machine learning usage

This section will provide some guidance on when it is appropriate to use machine learning methods. The focus is on supervised learning because other approaches are not utilised in this thesis. There are many papers and scientific work that successfully tackle the theory behind supervised learning. Most basic methods, such as decision trees or naive Bayes classifiers, are understood and described very well. [27] There are also sources for deep networks and similar alternatives. [32] Based on [6] and [28] these are the criteria which could help you determine when to use supervised machine learning:

- **Other methods fail due to model inaccuracies or algorithm flaws**
  The conventional engineering approach, focused on a mathematical representation of problems, has its limitations, such as model inaccuracies or complexity problems. Machine learning methods might succeed even in cases where the conventional approach fails.
  - **Model inaccuracies**
    As described in 2.1, there are many problems with complex systems modelling. Many variables or even physical principles could be unknown, which causes model problems.
  - **Algorithm problems**
    Although the model is accurately representing the real system developed algorithmic solution can be too complex to be implemented on given hardware. Many machine learning algorithms, such as decision tree classifiers, are much faster to use than to train. In some cases, machine learning methods can be utilised on less powerful devices.
- **Given task does not require the use of logic or domain knowledge**
  Such a task would probably not benefit from the mathematical model approach, so machine learning is a good alternative. Machine learning methods can be, in some cases, simple to use and not as time–consuming as the usage of traditional methods.
- **Given task can be solved by black–box fashion**
  In some applications, such as fault detection, explanations of how a decision was made can be as valuable as the decision itself. If it is not the case, the use of a black box learned algorithm with sufficient performance is a valid option.
- **The task of data acquisition is easy or at least possible**
  This condition is more of a restriction than a guideline. Deployment of a machine learning algorithm is limited by data acquisition. Complications with data acquisition can be caused by a low sampling rate of sensors or insufficient excitation of the system. For this reason, the system will not be represented by data enough, and the whole task will be more complicated.
- **Given task does not necessarily need safety or performance guarantees**

Machine learning methods mostly cannot provide advanced guarantees as model–based algorithms. Some warranties can be, to some extent, provided by numerical simulations.

A decision tree model is represented by a tree graph that consists of 3 types of nodes:

- **Root node** has no incoming edges and it is unique.
- **Internal node** is a node somewhere in the tree which has a single incoming edge and some outgoing edges. The number of outcoming edges varies for multiple decision tree types.
- **Leaf node** is a node with single incoming edge and none outgoing edges. Every leaf node represents a class label.

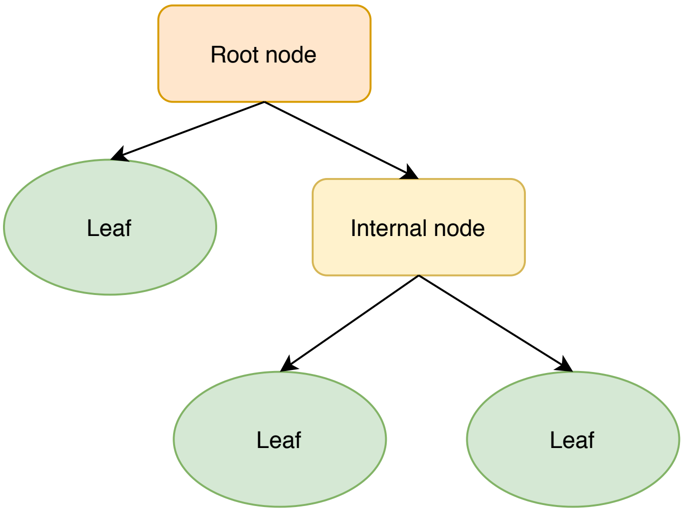


Figure 2.1. Structure of a decision tree model.

Nodes that are not leaves hold a splitting condition. Splitting conditions are functions of attributes of classified objects and results in outgoing edges that lead to another node. After the decision tree model is induced on training data classification of new objects is straightforward. Starting on the root node, the classification path is determined by splitting conditions based on the object's attributes. After going through all nodes with splitting conditions, path ends at a leaf node that holds a class label. This class label is used as the result of classifying the examined object.

Chapter 3
Decision trees

Decision tree classifiers are supervised machine learning methods that classify objects using a tree–like model of decisions. Decision trees are widely used due to their simplicity and easy interpretation. A decision tree can be trained as a classifier by computer algorithms, but they also can be made by hand to describe processes with conditional control statements. Transparent interpretation of decision trees allowed their usage in medicinal diagnostics, where other machine learning methods, such as neural networks, are often unusable because of their black–box nature. [21]

3.1 Decision tree induction

Finding an optimal decision tree is an NP–hard problem. However, there are efficient algorithms that can induce suboptimal trees with excellent performance. Decision tree induction is generally much less demanding on data quantity and computational resources than deep neural networks and many other machine learning methods.

3.1.1 Hunt's algorithm

Hunt's algorithm recursively induces decision trees. It is straightforward and is the core of many advanced and better–known decision tree algorithms. [11] Let D be training set of data with vector of attributes $a = (a_1, a_2, .., a_n)$. This training set consists of objects from classification classes c. As we induce a decision tree model, every node with splitting conditions splits the training set into subsets. A subset of training set associated with node N is denoted D_N. Hunt's algorithm can be recursively defined as follows:

- **Choose leaf type**
  If all objects in D_N have the same classification c, node N is a leaf node with classification c. If not, N is an internal node, and D_N must be further split into smaller subsets.
- **Select an attribute splitting condition**
  New child node is created for every outcome of a selected splitting condition. Most decision trees use binary conditions, but it can vary. For every new node, D_{new} assign the subset of D_N based on the condition outcome.

3.1.2 General splitting options

There are multiple options how to split objects based on their attributes. For better clarity attribute types and splitting conditions are showcased on an example of guitar classification. Attributes can be categorical such as guitar kind in or numerical, for example, guitar neck length or the number of frets. Based on the number of outcomes, splitting conditions are distinguished as binary or multiway. Some decision tree algorithms such as CART (classification and regression tree) use only binary splits. In order to use categorical multiway splits in CART, splits must be substituted by all combinations of binary splits between two subsets of categories. Examples of these splits are shown in Figs. 3.2 and 3.1. Numerical attributes can be treated as a categorical binary split with conditions such as

$$a_i > treshold \tag{1}$$

where a_i is an attribute of classification object in binary numerical split 3.4, or multiway split 3.3 where objects are split into multiple result intervals.

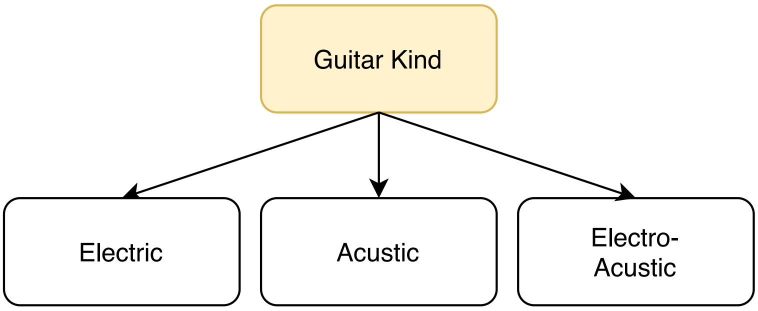


Figure 3.1. Example of cathegorical attribute multiway split.

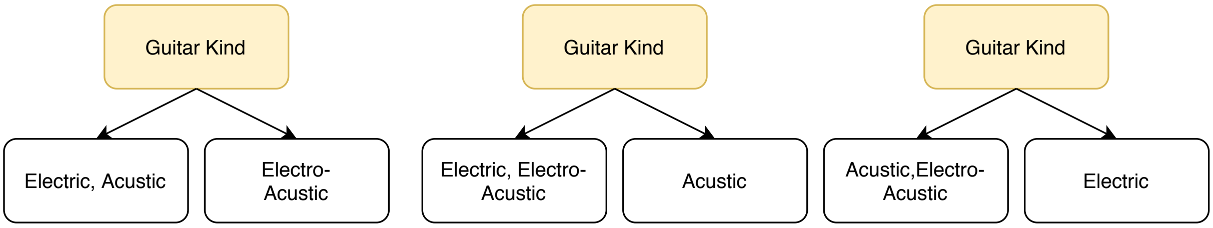


Figure 3.2. Example of cathegorical attribute binary split.

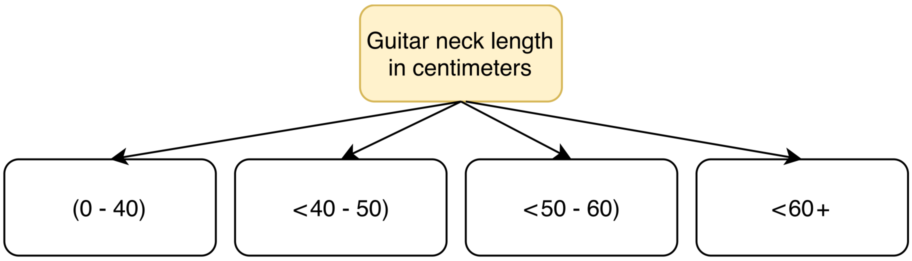


Figure 3.3. Example of numerical attribute multiway split.

3.1.3 Splitting criteria

After a general splitting option is chosen, there is a question of how to evaluate the goodness of a splitting condition. Many splitting criteria are mentioned in literature [4], [29]. Particular criteria typically outperform others on some specific datasets. Most commonly used impurity functions are the Gini (2) that is used by decision tree algorithms introduced by Breiman [5] and Gelfand [12] and the entropy (3) used by Quinlan's ID3, C4.5 and others. [23]

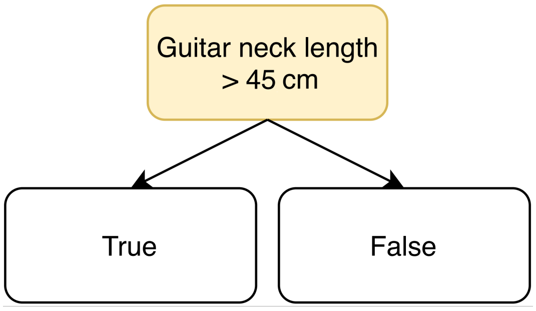


Figure 3.4. Example of numerical attribute binary split.

$$\phi(p) = \sum_j p_j(1 - p_j) \tag{2}$$

$$\phi(p) = -\sum_j p_j log(p_j) \tag{3}$$

Where $p_0, ..., p_j$ are node proportions and $\phi(p)$ is impurity function.

Important note for this thesis is that for a small number of classes, all splitting criteria should give very similar results. When a diagnosis of a specific error is made by classification into two classes (normal state and error state), the selection of splitting criteria should be negligible from accuracy perspective. Entropy is more demanding to compute due to logarithm in equation (3), so Gini criteria should be preferred to save computational power. This conclusion is discussed in detail in [4].

When dealing with a higher number of classification classes, splitting criteria should be chosen according to the desired properties of splits. Gini construct splits where majority class separation is emphasised. This behaviour can lead to highly unbalanced splits at the begging of a decision tree. Entropy prefers even splits into 50/50 groups. Entropy splits are often chosen from a larger group of equally good splits at the moment of selection, where optimality depends on the future evolution of given subtrees. Breiman [4] suggests that entropy splits should be done with a look–ahead approach.

3.2 Pruning

An unconstrained decision tree inducing algorithm will result in a huge decision tree with many splits and nodes. Training data can be noisy or just very hard to describe with available splitting conditions. Example of such data is shown in Fig. 3.7. Basic decision tree inducing algorithms (Hunt's 3.1.2) create new splits as long as classification objects are not separated into pure nodes. This complex decision tree suffers from overfitting. They are hard to interpret and inaccurate on objects that were not part of the training process. There are two options on how to solve this issue. The first is to terminate the decision tree algorithm prematurely to limit the final tree size. This option is called a pre–pruning. The second option is post–pruning the tree, which prunes the decision tree after it is generated. This option is also known as backward pruning. Pruning a decision tree increases computational efficiency as well as classification accuracy. [19]

3.2.1 Pre–pruning

Pre–pruning restricts tree expansion in non–significant directions. The restriction is done by terminating conditions. A simple terminating condition is for example the minimal number of objects for a given node. When the inner node's splitting condition affects a meagre part of objects, the resulting split will not be statistically significant, and it can likely be omitted. Good terminating condition is a minimum impurity decrease. This condition is used in CART.

Pre–pruning is done not only to increase the performance of classifiers but also to reduce the complexity of tree building. Pre–pruning must be applied with caution because it can reduce performance when done excessively. What seems like insignificant last split on training data can be the difference between good and excellent classifier on real data. [30]

3.2.2 Post–pruning

Post–pruning methods remove non–significant branches from decision trees. This branches could reflect the effect of noise on training data. Such branches harm classification accuracy and are not desirable. The most simple and clear method for post–pruning is reduced error pruning proposed by Quinlan [23].

A complete decision tree is tested against a validation dataset. This test gives a base accuracy score. Post–pruning is done iteratively as follows:

- **Inner node selection**

  An inner node is selected as a candidate for pruning. This can be done randomly or systematically over all nodes.

- **Removal of a subtree**

  The selected inner node is converted into a leaf node with a prevailing classification class. This results in a subtree cut–off.

- **Performance evaluation**

  Pruned tree accuracy is evaluated against the primary tree. If the error rate of the pruned tree is equal or smaller than before, the inner node is kept as a leaf node. Example is shown in Fig. 3.5.

The reduced error pruning method has linear computational complexity and results in smaller trees with similar or better accuracy than most pruning methods [1].

Next, more advanced and time–consuming option on how to post–prune a tree is tree raising. This method erases all except one child node of a given subtree. The root of that subtree is then replaced by its only child. Objects which were originally in erased subtrees must be distributed to other nodes based on their attributes. Note that on tree raising example shown in Fig. 3.6 from [30] nodes 1, 2, 3 are marked 1', 2', 3' because they also contain all objects from 4 and 5.

Post–prunning alone can preserve some combined predictors, where two attributes which alone do not seem to split data efficiently are very informative when combined. For this reason, most decision tree algorithms emphasise post–pruning rather than pre–pruning. [30]

3.3 Decision tree algorithms

In this section, some of the important algorithms will be briefly presented. There are many open–source implementations as well as many modifications to original

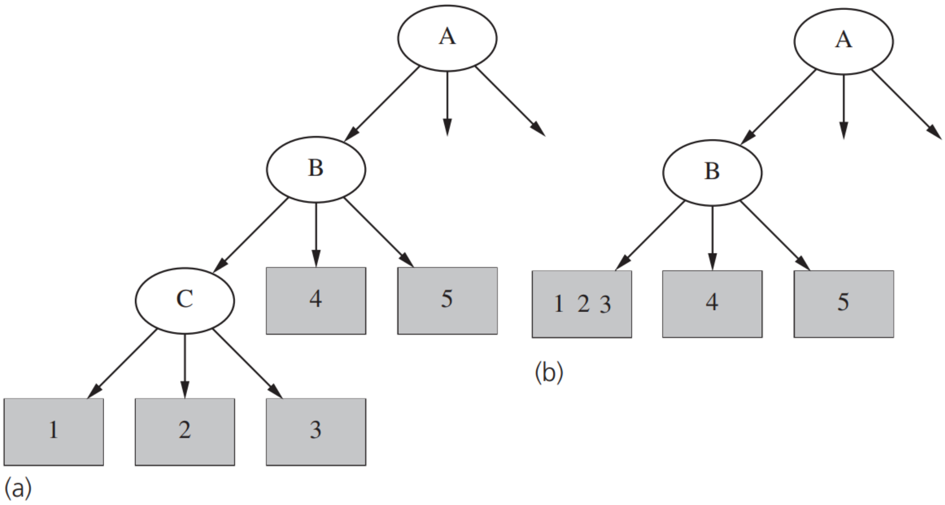


Figure 3.5. Example of subtree erasing, where node C is replaced by a leaf node. Leaf 123 contains all data from previous nodes 1, 2, 3 with classification of majority.

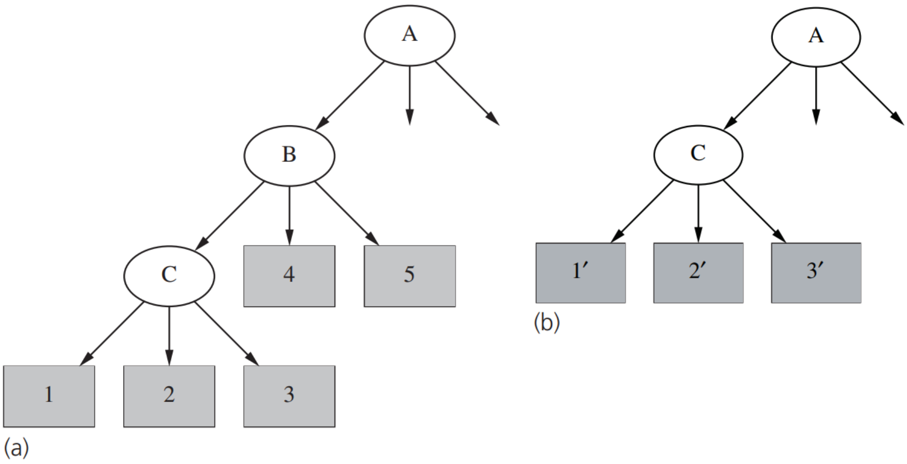


Figure 3.6. Example of subtree raising, where node C is "raised" to subsume node B. [30]

algorithms. For example, the later used decision tree algorithm from Scikit–learn package is an optimised version of the CART algorithm. [20]

3.3.1 ID3

ID3 is the first decision tree inducing algorithm that sparked international interest. [24] Compared to other more recent algorithms, it is considered very simple. ID3 works with categorical features and multiway splits. It does not handle missing values or numerical values. Information gain is used for splitting criteria. A tree grows until all objects are divided between leaves with given classification classes or information gain for best split is not greater than zero. The biggest drawback of ID3 was its restriction to categorical data.

3.3.2 C4.5

C4.5 was published in 1993 by Quinlan [26] as an extension to his previous algorithm ID3. It is probably one of the most popular decision tree inducing algorithms. [31] C4.5 uses information gain as splitting criteria and multiway splits similarly to ID3. It is highly popular because it resolved many ID3 problems. A system for discretisation of continuous numerical attributes was introduced. The original approach was modified throughout the years to improve C4.5 performance on numerical attributes. [25] Continous values are partitioned into multiple intervals based on thresholds which are determined locally. C4.5 is one of the first decision tree algorithms which defined its behaviour when attribute values are missing. When an attribute value is missing, all options are explored. A prediction is made for each subnode, and classification class with the most significant density is chosen as the result.

C5 also exists, but it is released under a proprietary license. It is capable of building smaller trees while using less memory with superior accuracy results.

3.3.3 CART

CART is along with C4.5 one of the most used algorithm for decision trees. Compared to C4.5, it utilises Gini as a splitting criteria (2) and can process numerical attributes. It was proposed by Leo Breiman [5]. CART uses a greedy search and has no termination conditions. The complete tree is built and then post–pruned. CART has many modifications as well as open-source implementations in major languages as Python, C, and R. CART can do linear splits by using one attribute at a time. Surrogate splits handle missing values, and some implementations allow misclassification costs. CART, especially in a Random Forest, has proved to be very efficient and accurate on many real–life applications, but as a single decision tree classifier has its weaknesses, for example it can be unstable. Only a small change in the training dataset results in a completely different tree. This behaviour happens because every split is affected by the change of the parent split, and a small change can result in a different tree root. One of the disadvantages discussed later in section 3.4 is that CART cannot efficiently classify objects with linearly dependent attributes. These disadvantages are not only CART specific but also apply to many others.

3.4 Oblique Decision trees

Decision tree induction algorithm such as C4.5, ID3 or CART are also called orthogonal, because they use splits orthogonal to features. This property leads to great comprehensibility of induced trees, but there are applications where orthogonal splits produce very complex trees with poor performance. Such application is classification

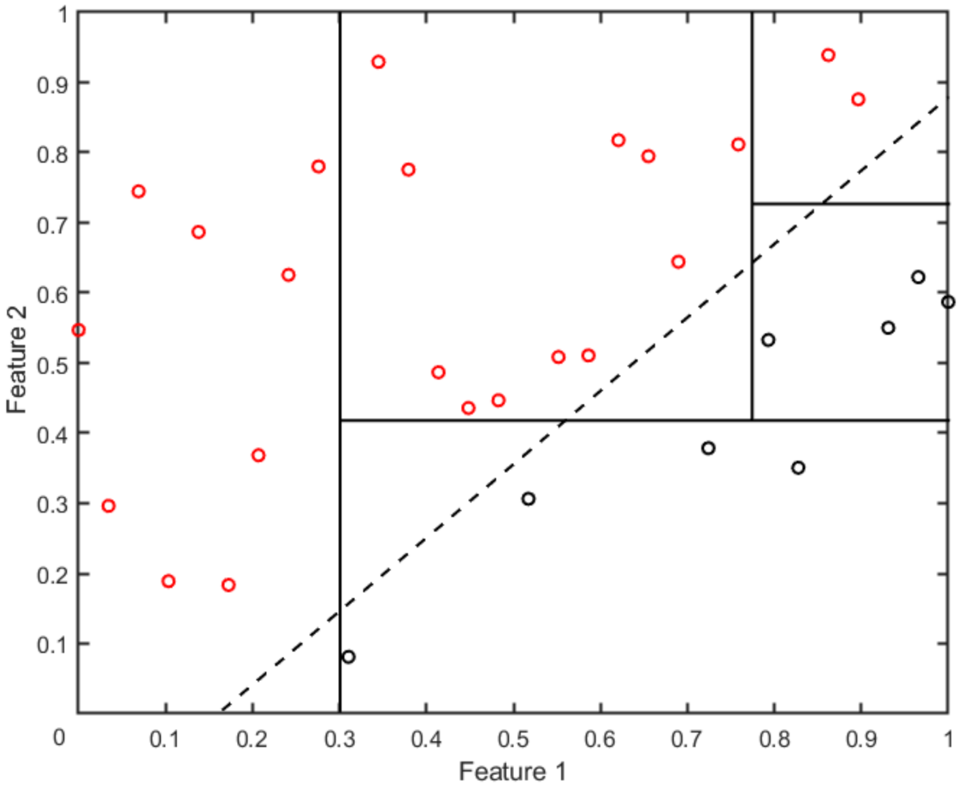


Figure 3.7. Comparison of classification using oblique and axis paralel splits on object with 2 dependent features.

task described in Fig. 3.7. Oblique decision trees are modification decision trees that make use of oblique splits.

Oblique decision trees have shown accuracy improvement when working with dependent features. Oblique trees are typically much smaller and at least as accurate as orthogonal decision trees. This property is obvious because an oblique decision tree can perform the same mapping function as an orthogonal decision tree, but it can not be said vice versa.

3.4.1 CART–LC

The first oblique decision tree algorithm was CART–LC introduced in 1984 [5], where LC stands for linear combinations. The idea was to use a linear combination of multiple features rather than just one at the time. CART–LC algorithm creates hyperplanes by finding locally optimal coefficients that maximise the goodness of split at every decision node. Splits are fully deterministic. Therefore coefficients estimation can get stuck in a local minimum. [8] Hyperplanes can be described by equation

$$\sum_{i=1}^{n}(\omega_i D_{a_i}) + \omega_0 > 0, \tag{4}$$

where a_i are attributes of object D with coefficients $\omega_1, ..., \omega_n$, n is number of attributes. Constant ω_0 is the treshold to be determined from the data.

Oblique splits are more difficult to interpret compared to univariate splits. To preserve as much comprehensibility backwards–deletion procedure is proposed. This procedure simplifies splits used in the final model by neglecting attributes with low impact on final classification performance.

3.4.2 OC1

OC1 is a widely used decision algorithm that was developed to tackle crucial problems of CART–LC. OC1 uses CART–LC's gradient optimisation methods for coefficients estimation and also employs stochastic elements. After coefficients estimation converges to a local minimum, the resulting hyperplane is perturbed in random directions. This perturbance, together with multiple random restarts of coefficients estimation, leads to superior performance results. Trees induced by OC1 are smaller and more accurate compared to CART–LC. OC1 has limited number of perturbations for each hyperplane which leads to time constraints on tree induction. OC1 spends only polynomial time on each node in the tree. CART–LC does optimisation of coefficients one at the time without a given time limit. OC1 does not exploit any advanced pruning or stop–splitting criteria. As explained in [17], a simple impurity threshold condition for cutting off subtrees performed well enough. No impurity measure is declared to be superior to others. The best measure is determined by the data for a given task.

3.5 Random Forest

Random Forest is a decision tree modification that was introduced by Breiman in 2001. [13] Random Forest was created to utilise some of Breiman's older research such as CART method combinations [5] and bagging [4]. Random Forest developed into a framework of models that have great practical success.

Random Forest algorithm use multiple decision trees as independent models. Prediction is made by averaging over all used models. The induction of particular decision trees varies among algorithms. The method used in the original Random Forest algorithm is to utilise feature subsets that can be used for splitting. This restricted feature set is chosen randomly for every split in every decision tree. Modified algorithms that use some knowledge about training data rather than pure randomness for choosing feature subsets also exist. Each decision tree is trained separately.

Despite many successful applications of Random Forest, underlying mathematical properties are still not well understood. There are two main properties of Random Forest to discuss: convergence and consistency. Breiman proved the convergence of classification error of random forest algorithm alongside with existence of its upper bound. [13] Consistency for decision tree algorithm has not yet been established. [10]

3.5.1 Random Forest parameters

When building a Random Forest, many parameters need to be set. Some of these parameters are crucial such as the choice of decision tree algorithm, which will train individual trees. The choice here could be randomised ID3, CART or oblique decision tree algorithm, OC1. Other parameters are more of a tuning tool for a given application. Well known and widely used algorithm for Random Forest induction commonly have parameter tuning guidelines. These recommendations are usually based on real–life usage experience. Tuning parameters are the ones used by decision tree algorithms as well as parameters used for forest induction.

Parameters for tree algorithms contain the minimal size of leaves, size of feature subsets for splitting, splitting criteria, maximum tree depth etc. The most important parameter for forest induction is the total number of trees. Breiman's convergence proof points in the direction of creating as many trees as possible, but other sources suggest different options. [2] Based on [22], the biggest improvement of performance occurs within the first 100 trees made. This result was also confirmed in an article [14]. Fig. 3.8 from article [14] shows that the classification error rate decreases with an increase in the number of decision trees. After reaching the 100 trees, the classification error trends to be stable (approximately 0.01). With a growing number of trees, further improvements slowly diminish, so one of the options on how to save computational power is to stop forest induction when the last N new trees do not improve performance.

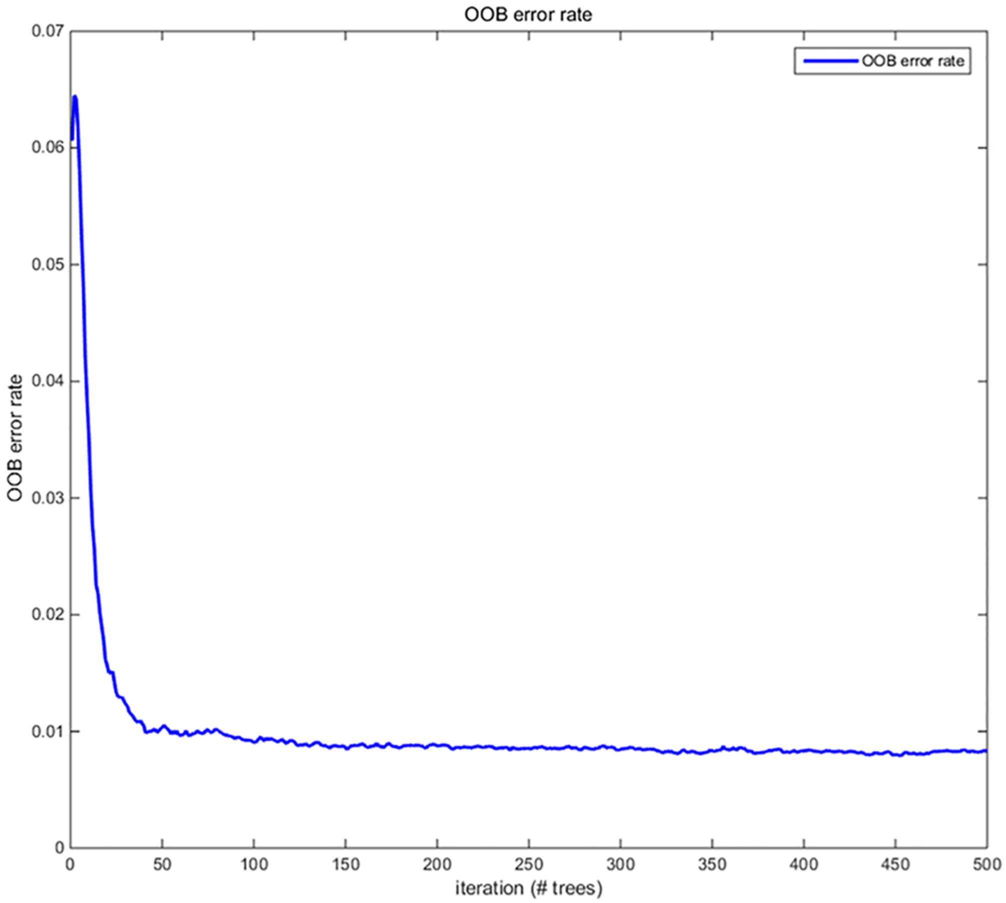


Figure 3.8. OOB error rate curve based on the number of decision trees. [14]

The performance of Random Forest is most of the time superior to estimators based on a single decision tree. A significant drawback of using Random Forest is the reduction of final model comprehensibility. A Random Forest frequently contains a large number of decision trees, so its usage is more of a grey to black–box fashion.

Chapter 4
Implementation

The goal of this thesis is to assess the usability of various decision tree algorithms in the domain of HVAC systems diagnosis. Python was chosen as the primary programming language for this thesis due to its popularity in the machine learning domain, but other languages such as Cython and C were used. For usability assessment, it is necessary to solve the following tasks:

- **Determine dataset structure and create tools for dataset creation.**
- **Implementation of the primary decision tree algorithm.**
- **Implementation of the Random Forest algorithm.**
- **Implementation of the oblique decision tree algorithm.**

Datasets used in this thesis consist purely from real–world data provided by company Energocentrum Plus. Dataset structure is described in section 4.1. The structure of decision tree model evaluation and comparison is proposed in section 4.2. Every implemented decision tree algorithm is described in its own section. These sections also contain examples of exported models with guidance on how to read them.

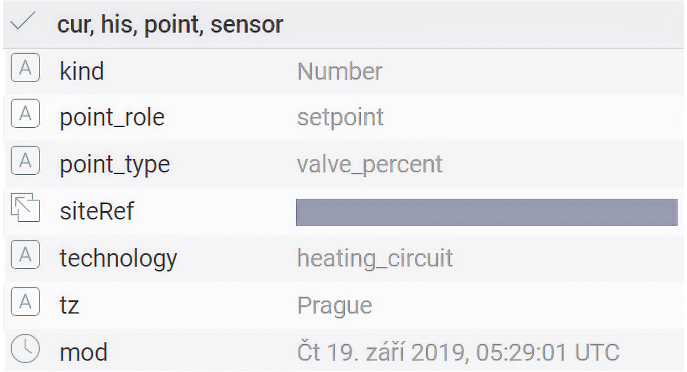
4.1 Dataset creation

Datasets for decision trees must contain tuples of features and class labels. The class label for a normal state is 0. The fault (alarm) state is labelled as 1. Datasets are created from data of multiple HVAC systems. Apart from features and class labels, every record contains origin identification. This identification string allows cross–validation between subsets from different sites or equipment. Example of dataset structure is given in Table 4.1.

identification	feature 1	feature 2	feature 3	class label
`circuit1`	1	52	32	0
`circuit1`	0	53.25	12	0
`circuit2`	1	12	11	1

Table 4.1. Structure of dataset.

Datasets are exported from Skyspark. Skyspark is an analytical tool developed by SkyFoundry LLC. Data in Skyspark consist of records such as site or equipment records and data points. Metadata denote Skyspark records. Additional metadata fully describe data points. Metadata are stored in the form of tags. Tags can store various data types such as booleans, strings, numbers or references. Example of datapoint's tags is in Fig. 4.1. The first row contains tags which are markers. From given tags, it can be identified that this data point is a sensor of valve position in a heating circuit.



✓	cur, his, point, sensor	
A	kind	Number
A	point_role	setpoint
A	point_type	valve_percent
	siteRef	[redacted]
A	technology	heating_circuit
A	tz	Prague
🕓	mod	Čt 19. září 2019, 05:29:01 UTC

Figure 4.1. Tags of Skyspark datapoint.

Tag *point_role*: *setpoint* indicates that the valve position is requested value rather than the actual measurement.

Data points contain reference tags which describe connections between individual points. This approach is very convenient for creating datasets. It is possible to gather data from the HVAC system with a single function. Functions that export datasets from Skyspark into .csv files were implemented in scripting language AXON[1].

4.2 Decision tree evaluation

Decision tree methods are used for fault detection. This task is a form of binary classification. The trained model performance is evaluated based on prediction accuracy. Prediction accuracy is the fraction of predictions which model predicted correctly. Prediction errors can be divided into false–negative and false–positive errors.

Trained models of decision tree methods can be compared based on accuracy. Decision tree methods, except for Random Forest, can also be compared based on the tree size. The decision tree models can also be compared by interpretability. This comparison requires domain knowledge and can be very subjective.

4.3 Implemented decision tree methods

There are three main approaches of how decision trees are utilized. The primary one is the single feature split decision trees which are also called orthogonal. More advanced approach is to use oblique decision trees. The last commonly used method is a Random Forest. Implementations of selected algorithms from these groups are described in the following sections.

4.3.1 Orthogonal decision trees

There is a wide variety of open–source decision trees algorithms. Selected decision tree algorithm is from the Python package scikit–learn [20]. This package uses an optimized version of the CART algorithm which does not support categorical variables or missing values. Both of these restrictions do not interfere with the intended use. Scikit–learn package contains few datasets often used as a benchmark and many functions for dataset handling. Scikit–learn is very popular, and many other machine learning methods and projects use compatible API (Application programming interface) [7].

[1] https://axoniq.io/

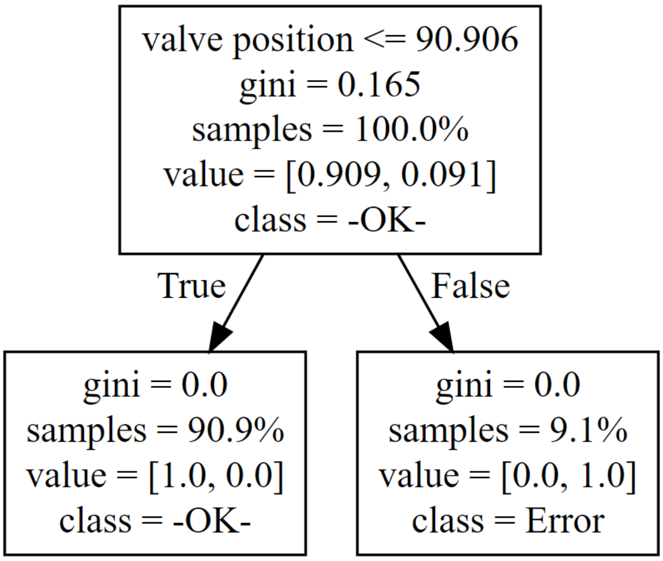


Figure 4.2. Example of decision tree exported to graphviz file.

The input of used algorithm CART consists of 2–dimensional numpy array of features, 1–dimensional numpy array of classification classes, list of feature names and class labels. The trained model can be exported into a text file or more conveniently into a graphviz[1] (.gv) file. Example of graphviz file visualization is shown in Fig. 4.2.

The first decision tree node is a root node with a splitting condition on a feature named valve position. All records with valve position smaller or equal to 90.906 proceed to the right node. Records with greater valve position proceed to the left. Gini value for the given split is 0.165. Attribute samples show the percentage of records which are affected by this node. This value can be used to detect significant paths in a decision tree. Attribute value shows how data of the given node are split into classification classes. In this example, the root node contains 0.909 of class OK records and 0.091 class Error records. Both leaf nodes are pure, with only one class present. Last attribute class shows the majority class in a given node. If a node is a leaf node, the attribute class contains the final classification.

4.3.2 Random Forest

Random Forest algorithm is also implemented in the scikit–learn package. Decision tree algorithm used in a Random Forest is the same as described in section 4.3. Due to the nature of the Random Forest model, it is complicated to fully understand the trained model. Random Forest is used only for accuracy comparisons, and trained models are not visualized in any way.

4.3.3 Oblique decision trees

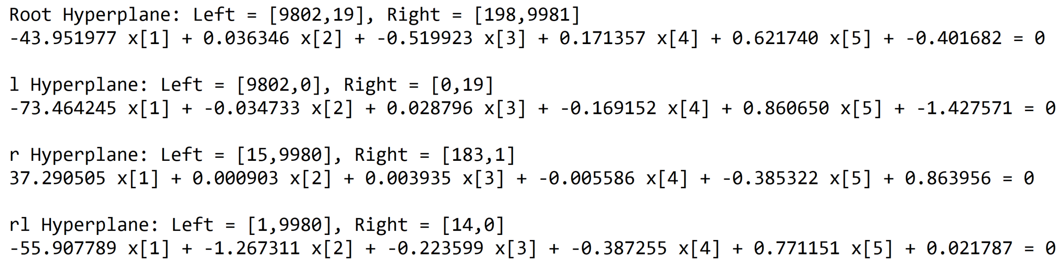
Oblique decision tree implementation is based on the original OC1 source code. With the use of Python API from Andriy Mulyar [16] it is possible to use CART with linear combinations, OC1 algorithm and OC1 algorithm with an option of orthogonal splits in Python. Source code downloaded from Andriy Mulyar consist of original C code, Cython[2] bridge between C and Python and Python demo. Model training is done via

[1] https://www.graphviz.org/

[2] https://cython.org/

the original C library. Evaluation and exports of trained models are done predominantly in Python.

All implemented algorithm works with the same inputs. Cython part of the code accepts Python scikit–learn compatible datasets with the exception of feature names and classification classes. An oblique decision tree is represented by the equations of corresponding splitting hyperplanes. Example of oblique tree representation is shown in Fig. 4.3.



```
Root Hyperplane: Left = [9802,19], Right = [198,9981]
-43.951977 x[1] + 0.036346 x[2] + -0.519923 x[3] + 0.171357 x[4] + 0.621740 x[5] + -0.401682 = 0

l Hyperplane: Left = [9802,0], Right = [0,19]
-73.464245 x[1] + -0.034733 x[2] + 0.028796 x[3] + -0.169152 x[4] + 0.860650 x[5] + -1.427571 = 0

r Hyperplane: Left = [15,9980], Right = [183,1]
37.290505 x[1] + 0.000903 x[2] + 0.003935 x[3] + -0.005586 x[4] + -0.385322 x[5] + 0.863956 = 0

rl Hyperplane: Left = [1,9980], Right = [14,0]
-55.907789 x[1] + -1.267311 x[2] + -0.223599 x[3] + -0.387255 x[4] + 0.771151 x[5] + 0.021787 = 0
```

Figure 4.3. Oblique decision tree representation using hyperplane equations.

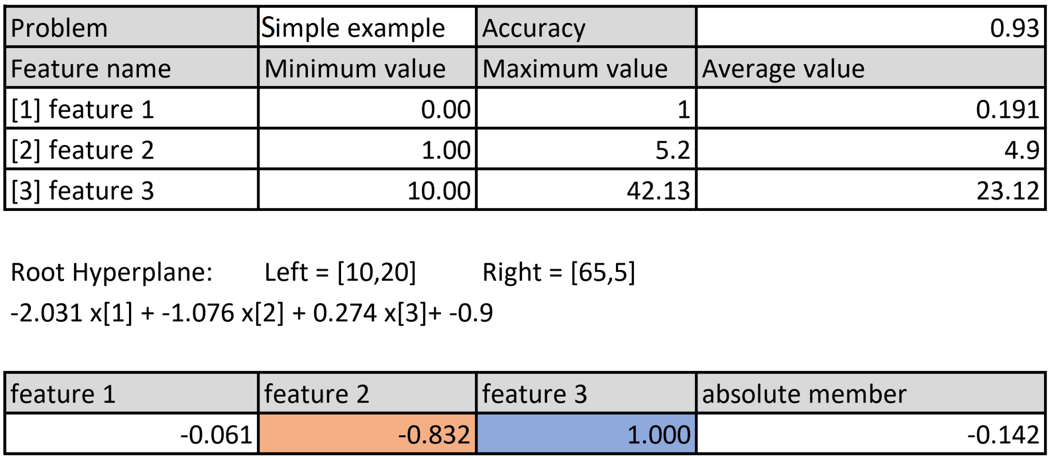
Every inner node of an oblique decision tree is fully described by its location within the tree, data split, and hyperplane equation. The root node is denoted by the word Root. Other nodes are localized by a sequence of l and r letters which stand for left and right if decision tree is searched from the root node in a depth–first manner. Symbol $x[n]$ denotes the n-th feature.

Records are split based on their relative position to the splitting hyperplane. Records above the splitting hyperplane proceed to the right node. Records below proceed to the left node.

Oblique decision trees can be saved into and loaded from text files with methods from the original OC1 library. Trained models occupy limited storage space and can be modified by hand. For example, a fully grown tree can be trained by original code and pruned later.

Original representation of oblique decision tree is fully descriptive but not exactly easy to understand. For this reason, Python methods that provide more complex export into .csv file are implemented. The convenient way to analyze oblique decision tree is to identify essential features for given splits. A simple proposed method for importance evaluation is to use average feature values and coefficients of given splits. When evaluating splitting condition feature coefficients are multiplied by feature values. A binary feature with values limited to 0 and 1 is going to have a much higher coefficient than features such as temperature or pressure with values of higher orders. Important features can be determined using average feature values. After the multiplication of coefficients with average feature values resulting values in splitting conditions are scaled–down to the range of <-1, 1>. The scaling coefficient is the highest positive or the lowest negative number. This approach allows for quick detection of important values while preserving ratios of features importance. Example of selected export is shown in Fig. 4.4.

Exported oblique decision tree is trained on a dataset containing three features. This example shows that feature with the smallest coefficient can be the most significant. For this reason, it is crucial to identify feature value ranges when interpreting decision trees. All feature values are multiplied by coresponding average values. Minimal value is represented by feature 2 where -1.076 is multiplied with 4.9 resulting in -5.2724. Maximum value is represented by feature 3 where 0.274 is multiplied by 23.12 resulting



Problem	Simple example	Accuracy	0.93
Feature name	Minimum value	Maximum value	Average value
[1] feature 1	0.00	1	0.191
[2] feature 2	1.00	5.2	4.9
[3] feature 3	10.00	42.13	23.12

Root Hyperplane: Left = [10,20] Right = [65,5]

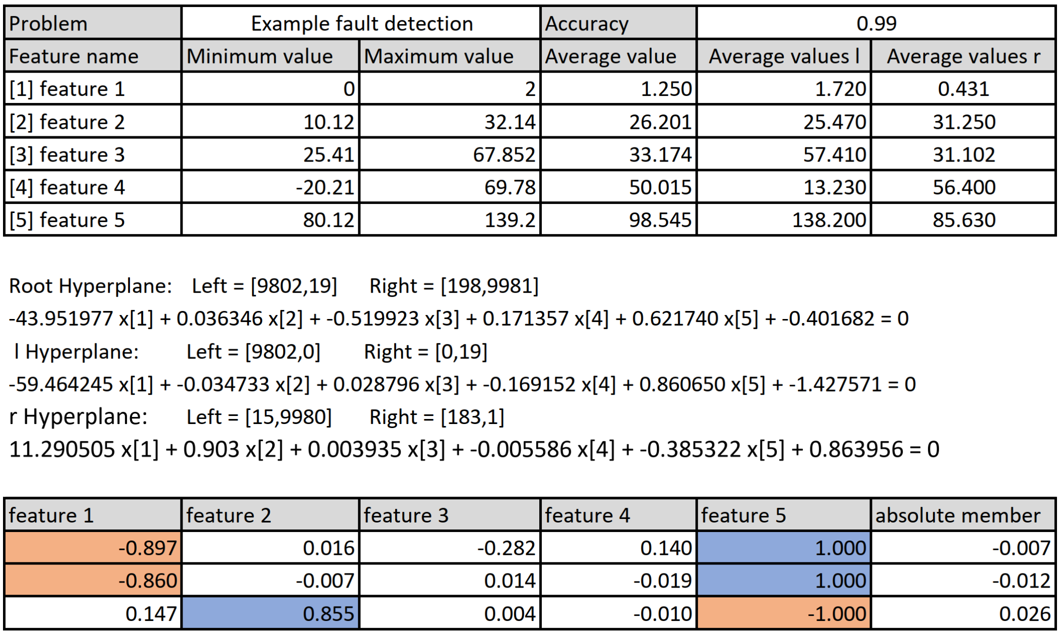
-2.031 x[1] + -1.076 x[2] + 0.274 x[3]+ -0.9

feature 1	feature 2	feature 3	absolute member
-0.061	-0.832	1.000	-0.142

Figure 4.4. Example of feature importance analysis export - root only.

in 6.3349. All values are scaled–down by extreme with higher absolute value which is 6.3349 in this case. For clarity, features with a positive impact on splitting result are coloured blue, and features with negative impact are coloured orange.

Feature importance can be evaluated on every inner node of the oblique decision tree. For trees with multiple nodes average values must be computed for every node. Collums are denoted the same way as inner nodes with l and r letters. Export of such non–trivial tree is shown in Fig. 4.5.



Problem	Example fault detection		Accuracy	0.99	
Feature name	Minimum value	Maximum value	Average value	Average values l	Average values r
[1] feature 1	0	2	1.250	1.720	0.431
[2] feature 2	10.12	32.14	26.201	25.470	31.250
[3] feature 3	25.41	67.852	33.174	57.410	31.102
[4] feature 4	-20.21	69.78	50.015	13.230	56.400
[5] feature 5	80.12	139.2	98.545	138.200	85.630

Root Hyperplane: Left = [9802,19] Right = [198,9981]

-43.951977 x[1] + 0.036346 x[2] + -0.519923 x[3] + 0.171357 x[4] + 0.621740 x[5] + -0.401682 = 0

l Hyperplane: Left = [9802,0] Right = [0,19]

-59.464245 x[1] + -0.034733 x[2] + 0.028796 x[3] + -0.169152 x[4] + 0.860650 x[5] + -1.427571 = 0

r Hyperplane: Left = [15,9980] Right = [183,1]

11.290505 x[1] + 0.903 x[2] + 0.003935 x[3] + -0.005586 x[4] + -0.385322 x[5] + 0.863956 = 0

feature 1	feature 2	feature 3	feature 4	feature 5	absolute member
-0.897	0.016	-0.282	0.140	1.000	-0.007
-0.860	-0.007	0.014	-0.019	1.000	-0.012
0.147	0.855	0.004	-0.010	-1.000	0.026

Figure 4.5. Complex example of feature importance analysis export.

From the values and colour highlight of the first feature in Fig. 4.5, it can be seen that this feature is essential in root splitting condition. The average value of the first feature increased from 1.250 to 1.720 for the left node and decreased from 1.250 to 0.431 for the right node. The average values can be used for a quick overview of how the records are separated based on features values. For further understanding, it is convenient to go through the decision tree and look at the data subsets various nodes of the classification path.

Chapter 5
Parameter settings

All implemented decision tree algorithms have various parameters that need to be set. Reasonable ranges for various parameters are recommended in this chapter. The process of finding optimal parameters for a given task is often referred to as a parameter tuning.

5.1 CART and Random Forest parameter tuning

Optimal CART parameters are hard to set because they change accordingly on data. In this thesis, classification is binary. Therefore splitting criteria should not play any role. Gini criterium is used in all decision trees because of its fast evaluation. This conclusion confirms theoretical assumptions made in section 3.1.3. Trained models should work as a grey-box, where the main focus is on comprehensibility. Decision tree depth must be limited to a reasonable level, which might differ for various tasks. In the best–case scenario, the vast majority of alarm class data should follow the same classification path. For detailed parameter tuning see this article [15] or the original paper [5].

The Random Forest consists of many CART decision trees. Finding parameters for CART tree in the Random Forest is at least as challenging as in CART alone. Random Forest is used only for accuracy comparison, so there is no ambition to look into individual trees. The number of classifiers trained is set to 100. This number is chosen based on theoretical research [22], [14] and also on the recommended default value by the scikit–learn library. Accuracy comparisons between CART and Random Forest in this thesis are always made on models trained with the same parameters.

5.2 Oblique decision tree methods comparison

There are three possible algorithms to choose from: CART with linear combinations, OC1 and OC1 with the possibility of orthogonal splits. In this section, all algorithms are compared based on their accuracy and tree depth. The best algorithm is selected after a series of tests, and reasonable ranges for parameters are recommended.

The first test is going to test tree accuracy for various sizes of training dataset. Decision trees should detect state, where two feature values are compared, and another feature value is above a threshold. A real–world example of such classification would be underheating while the building is occupied or detection that a car is turning to one side based on engine revs and wheels rotation speeds. Training dataset is systematically reduced from 50 000 of each classification class down to 100 each. The number of restarts and number of perturbation is set to 10. Importance of these parameters is discussed later. Results of this test are shown in Fig. 5.1.

From theoretical point of view, an oblique split should always surpass orthogonal split because orthogonal split can be described as an oblique split with zero weights on all but one feature. OC1 and OC1 with orthogonal splits had the same accuracy for all datasets. After further testing on other datasets, it seems that OC1 almost exclusively prefer oblique splits. OC1 orthogonal will not be part of further testing.

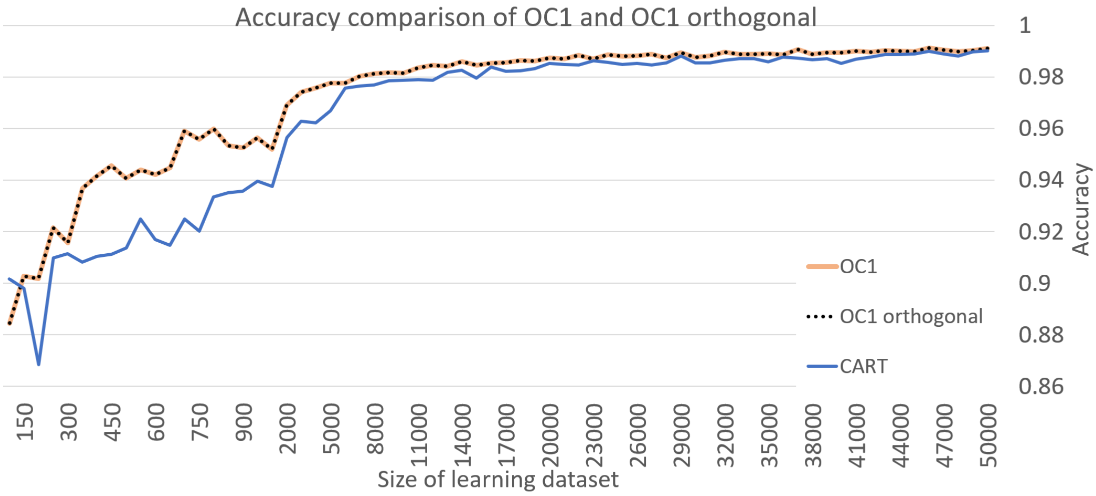


Figure 5.1. Accuracy comparison of CART, OC1 and OC1 with orthogonal splits with different training dataset size.

CART with linear combinations shows lower accuracy across all datasets greater than 250 records of both classification classes. The difference between OC1 and CART accuracy for big datasets is minimal (0.5 %). In the following test accuracy of both induced tree is compared on each level. Due to stochastic properties of OC1 algorithm is started from 10 different random states, and resulting accuracies are averaged. Results are shown in Fig. 5.2. OC1 induced tree is considerably more accurate with small tree depth which is excellent for comprehensibility. Larger decision trees with more levels are more likely to suffer from overfitting. Based on Occam's razor principle, trees build with OC1 algorithm should generalize problems better and are expected to be preferable. From now on in this thesis, all comparisons between oblique trees and other methods will use OC1.

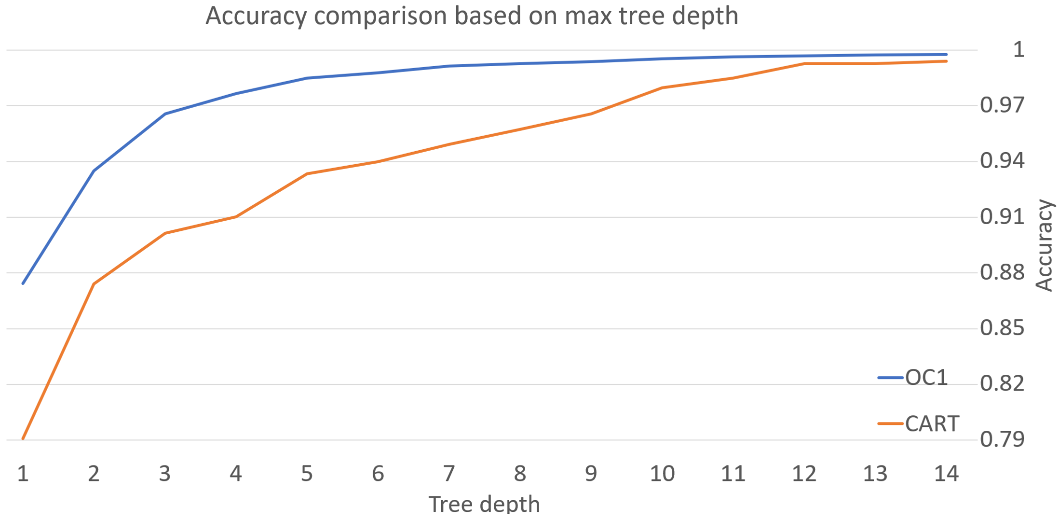


Figure 5.2. Accuracy comparison of CART and OC1 based on tree depth.

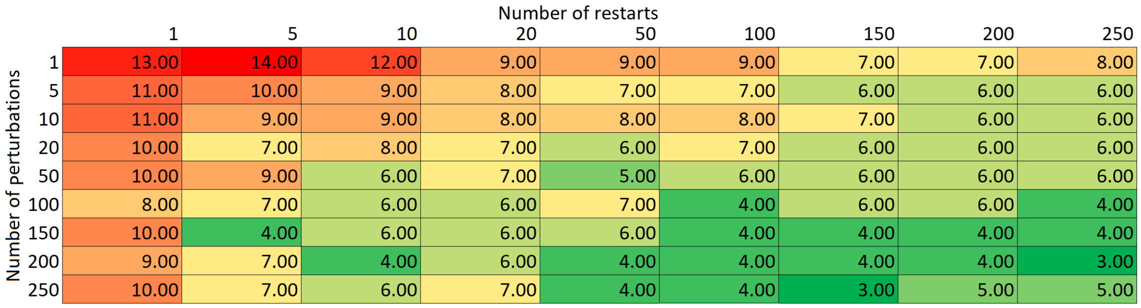
5.3 OC1 parameter tuning

OC1 algorithm has few tuning parameters. Basic parameters are defined in OC1.h header file. These parameters with corresponding values are shown in Table 5.1.

parameter name	value
MAX_COEFFICIENT	1.0
MAX_NO_OF_ATTRIBUTES	4020
MAX_DT_DEPTH	50
MAX_NO_OF_STAGNANT_PERTURBATIONS	10
MAX_CART_CYCLES	100
TOO_SMALL_THRESHOLD	2.0
TOO_SMALL_FOR_ANY_SPLIT	3
TOO_SMALL_FOR_OBLIQUE_SPLIT	2 * no_of_dimensions

Table 5.1. Default configuration of OC1 parameters.

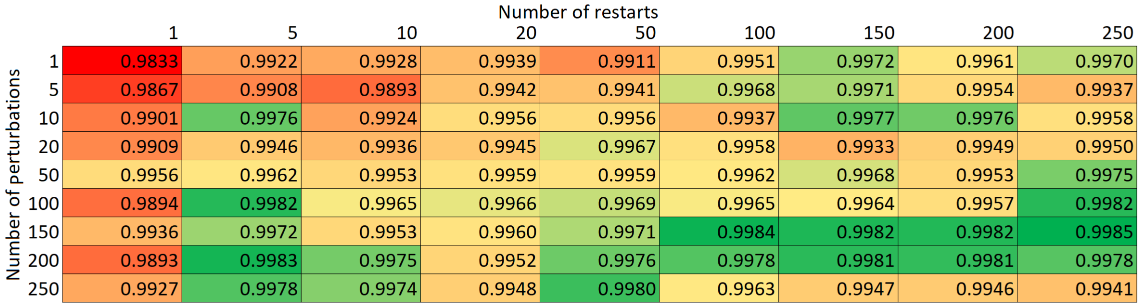
Basic parameters are set in the original code by Breiman. These values are used for all experiments and tests. Other configurable parameters of OC1 are the number of restarts and the maximum number of hyperplane perturbations. The following test is done on the dataset containing five features. Alarm state in this dataset compares two features with each other and other two features with a threshold. OC1 algorithm is run with multiple settings from one restart and one perturbation up to 250 restarts and 250 perturbations. Resulting heatmap showing the depth of trained trees is shown in Fig. 5.3. Accuracy heatmap is shown in Fig. 5.4.



	Number of restarts								
Number of perturbations	1	5	10	20	50	100	150	200	250
1	13.00	14.00	12.00	9.00	9.00	9.00	7.00	7.00	8.00
5	11.00	10.00	9.00	8.00	7.00	7.00	6.00	6.00	6.00
10	11.00	9.00	9.00	8.00	8.00	8.00	7.00	6.00	6.00
20	10.00	7.00	8.00	7.00	6.00	7.00	6.00	6.00	6.00
50	10.00	9.00	6.00	7.00	5.00	6.00	6.00	6.00	6.00
100	8.00	7.00	6.00	6.00	7.00	4.00	6.00	6.00	4.00
150	10.00	4.00	6.00	6.00	6.00	4.00	4.00	4.00	4.00
200	9.00	7.00	4.00	6.00	4.00	4.00	4.00	4.00	3.00
250	10.00	7.00	6.00	7.00	4.00	4.00	3.00	5.00	5.00

Figure 5.3. Heat map of decision tree depth based on the number of restarts and perturbations.

Using heatmap visualization, it is clear that the increasing number of perturbations and restarts have a positive effect on the tree depth. This result is expected because more perturbations can only improve splitting properties of each node. The number of restarts seems to be as important as the number of perturbations, but that is probably highly dependent on a given problem. High–dimensional problems with many local minima will probably require more restarts than more straightforward tasks.

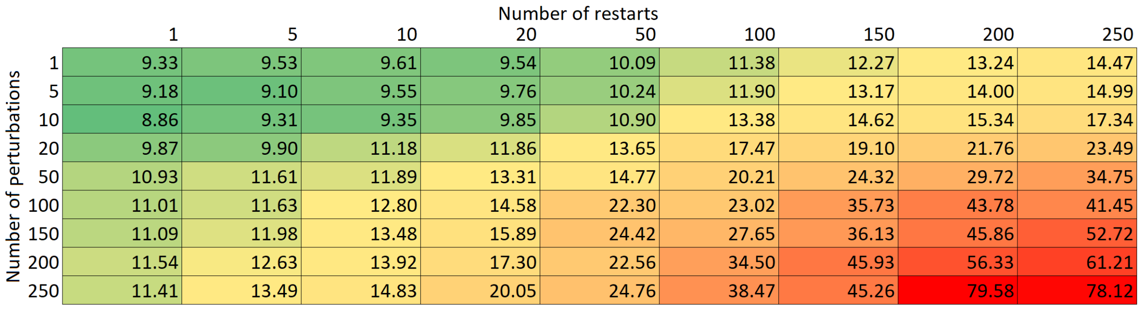


	Number of restarts								
Number of perturbations	1	5	10	20	50	100	150	200	250
1	0.9833	0.9922	0.9928	0.9939	0.9911	0.9951	0.9972	0.9961	0.9970
5	0.9867	0.9908	0.9893	0.9942	0.9941	0.9968	0.9971	0.9954	0.9937
10	0.9901	0.9976	0.9924	0.9956	0.9956	0.9937	0.9977	0.9976	0.9958
20	0.9909	0.9946	0.9936	0.9945	0.9967	0.9958	0.9933	0.9949	0.9950
50	0.9956	0.9962	0.9953	0.9959	0.9959	0.9962	0.9968	0.9953	0.9975
100	0.9894	0.9982	0.9965	0.9966	0.9969	0.9965	0.9964	0.9957	0.9982
150	0.9936	0.9972	0.9953	0.9960	0.9971	0.9984	0.9982	0.9982	0.9985
200	0.9893	0.9983	0.9975	0.9952	0.9976	0.9978	0.9981	0.9981	0.9978
250	0.9927	0.9978	0.9974	0.9948	0.9980	0.9963	0.9947	0.9946	0.9941

Figure 5.4. Heat map of decision tree accuracy based on the number of restarts and perturbations.

The preprunning conditions of OC1 determined by basic parameters stop the building process when nodes are pure or too small to be split further. Smaller trees are the ones that managed to split data more efficiently with a simpler model. These trees are likely to be more accurate. Smaller trees are expected to exceed bigger trees, especially on cross–validation with data from different equipment. They are more likely to display general system behaviour rather than only overfit data. The expected behaviour is supported by the results shown in heatmaps, but accuracy is high even for medium–sized trees.

Decision tree training time heatmap is shown in Fig. 5.5. Training time is affected by both number of restarts and the number of perturbations approximately in the same manner. All tests were done on a computer with the intel i7–9750h processor.



Number of perturbations \ Number of restarts	1	5	10	20	50	100	150	200	250
1	9.33	9.53	9.61	9.54	10.09	11.38	12.27	13.24	14.47
5	9.18	9.10	9.55	9.76	10.24	11.90	13.17	14.00	14.99
10	8.86	9.31	9.35	9.85	10.90	13.38	14.62	15.34	17.34
20	9.87	9.90	11.18	11.86	13.65	17.47	19.10	21.76	23.49
50	10.93	11.61	11.89	13.31	14.77	20.21	24.32	29.72	34.75
100	11.01	11.63	12.80	14.58	22.30	23.02	35.73	43.78	41.45
150	11.09	11.98	13.48	15.89	24.42	27.65	36.13	45.86	52.72
200	11.54	12.63	13.92	17.30	22.56	34.50	45.93	56.33	61.21
250	11.41	13.49	14.83	20.05	24.76	38.47	45.26	79.58	78.12

Figure 5.5. Heat map of decision tree training time [s] based on the number of restarts and perturbations.

Recommended values from this test are at least 100 restarts and 100 perturbations. This recommendation is applicable only for a reasonably big dataset. OC1 algorithm is utilizing only CPU core. Tree building time can easily get into hours when training decision trees on bigger datasets.

Due to substantial time demands, training dataset size importance is evaluated. OC1 algorithm is used on a balanced dataset with samples of each class ranging from 100 to 50 000 with 100 increments. Algorithm parameters are set to 10 restarts and 10 perturbations. Each step is also repeated ten times to evaluate the deviation of accuracy. The resulting accuracy mean, and the standard deviation are shown in Figs. 5.6 and 5.7.

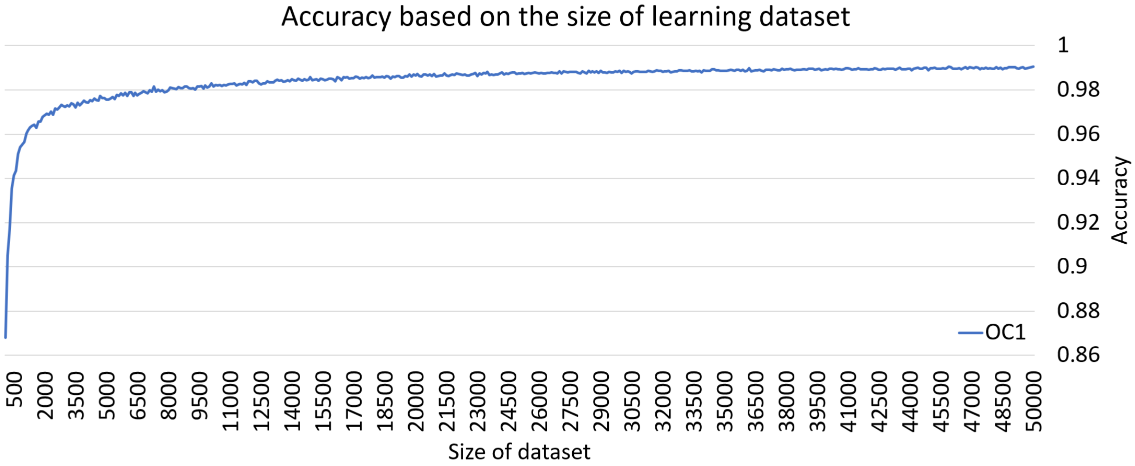


Figure 5.6. Accuracy of OC1 based on the size of training dataset.

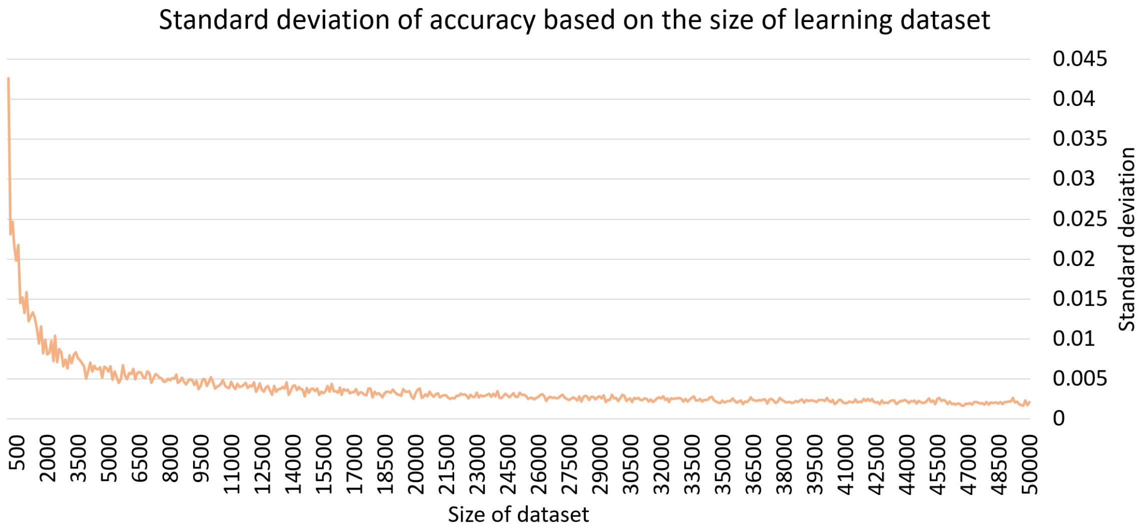


Figure 5.7. Standard deviation of OC1 classifier accuracy based on the size of training dataset.

Based on the results from previous experiments, it is highly recommended to use datasets with at least 10 000 samples of both classes. If there is enough computational power or time available, it is probably best to use the biggest possible training set. Computational time requirement based on the dataset size is shown in Fig. 5.8.

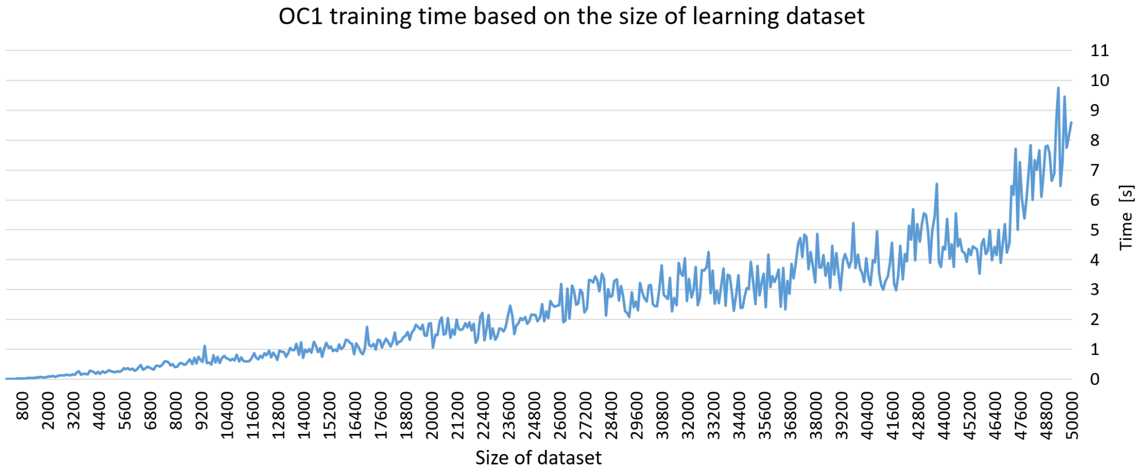


Figure 5.8. Training time of OC1 classifier accuracy based on the size of training dataset.

Training time is proportional to the number of features. The main issue is the size of training dataset. It is also worth mentioning that these times in Fig. 5.8 are achieved on a problem which resulted in smaller trees. When it is hard for OC1 to fit training data, computational times become significantly longer, because the number of nodes in each following level is up to double. In general, it is a good idea to start with smaller datasets and parameter values to evaluate time consumption and iterate from there.

Chapter 6
Methods comparison

In this chapter, decision tree methods are compared using three simple problems from the domain of heating circuits. Heating circuits are part of almost every building. Faulty control of heating circuits can lead to energy waste and discomfort. Heating circuits are one of the modest HVAC systems and provide an excellent environment for simple tests. Heating circuits usually contain four sensor data points and control datapoint with requested temperature. Schematic of a heating circuit with feature labels is shown in Fig. 6.1.

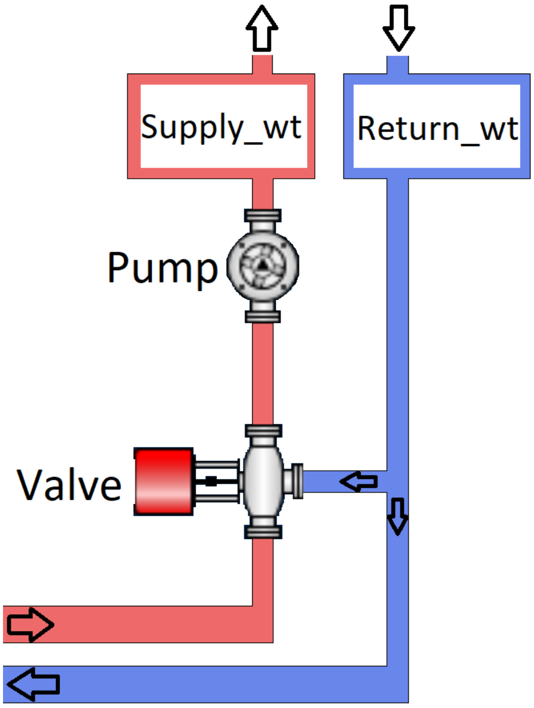


Figure 6.1. Schematic of heating circuit with feature labels.

Feature names are used in the visual interpretation of trees in shortened form. Complete list of features is:

- **pump** – a binary signal of pump state (1 – on, 0 – off),
- **return_wt** – the temperature of the water returning from a heating circuit [°C],
- **supply_wt** – the temperature of the supply water coming from threeway valve [°C],
- **supply_wt_req** – requested supply temperature [°C],
- **valve** – valve position [%].

The first detection problem is valve saturation. Valve saturation requires decision trees to detect if two features are above a given threshold. The *pump* state must be 1 and the *valve* position greater than 99%. All methods are expected to achieve high accuracy and interpretability on this problem.

The second detection problem is underheating. Underheating should be detected when *supply_wt* is under *supply_wt_req* by more than 0.5 °C. This problem should test CART classification performance on dependent features.

The last detection problem combines both valve saturation and underheating and should demonstrate decision tree usage on a more complex problem.

Heating circuit dataset was exported from SkySpark. It is composed of over 8.5 million records from approximately 100 different heating circuits. All records are from the year 2019. Heating circuits are sampled with a sampling period of 5 minutes, but the timestamp is not used for classification.

Based on the properties of OC1 discussed in section 5.3, training dataset size is set to 10 000 samples for each classification class. With the dataset made of over 8.5 million records, it is beneficial to use large validation dataset. The size of validation dataset is 100 000 for each classification class in order to quarantee same validation dataset size across all diagnosed problems. Records are assigned to training and validation datasets randomly.

6.0.1 Valve saturation

Valve saturation in a heating circuit is generally a sign of fault within the system. Valve saturation is defined as a state when *valve* position is over 99 % and the *pump* is on. This problem should suit CART decision trees because the evaluating condition can be expressed as a logical conjunction of two single feature splits.

Resulting CART induced decision tree has two nodes as expected. The decision tree shown in Fig. 6.2 is accurate and easy to interpret.

Random Forest algorithm achieved perfect accuracy as well as CART.

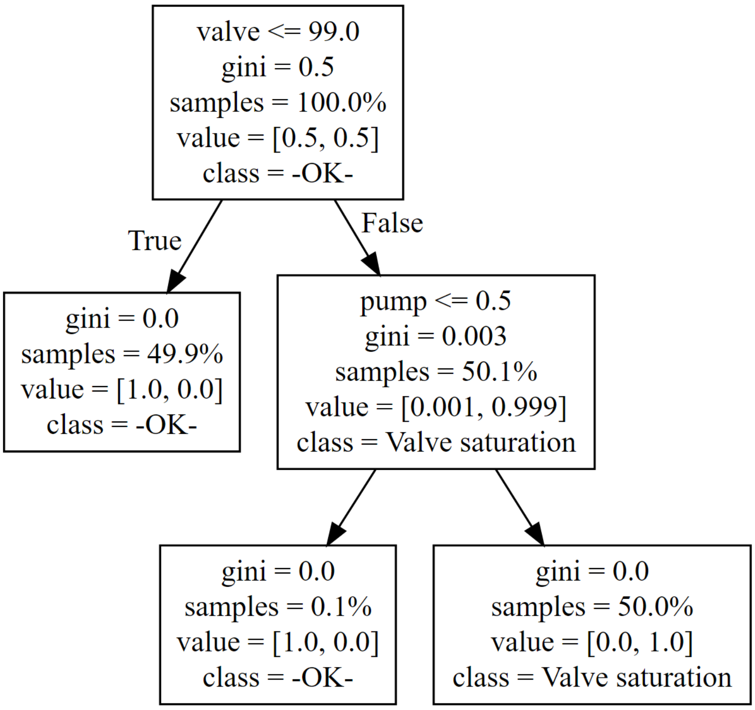
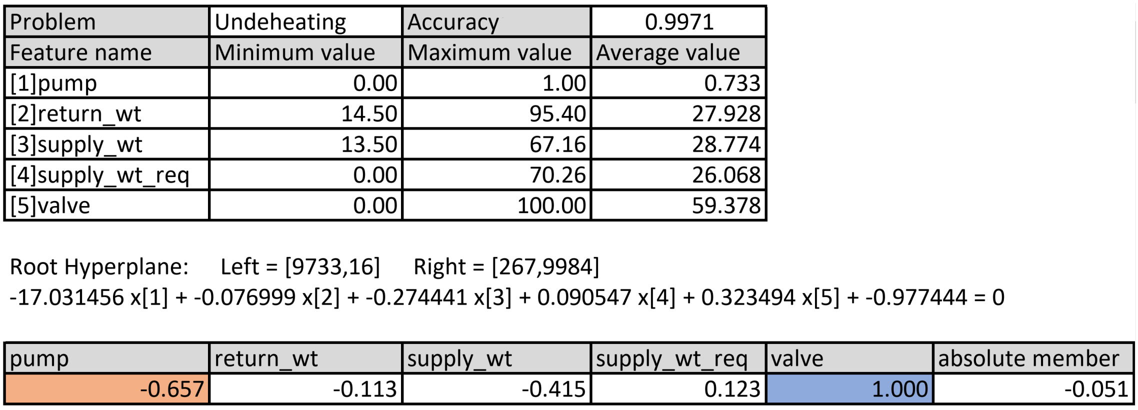


Figure 6.2. CART induced decision tree for valve saturation problem.

An oblique decision tree is able to split records based on both *valve* position and *pump* state in a single split. This approach generally leads to smaller trees. The oblique tree requires additional nodes to classify all training records. The induced tree is shown in Fig. 6.3. Root node splitting condition is analyzed by coefficient scaling. The results are shown in Fig. 6.4. Most significant features are *pump* state and *valve* position. Unexpectedly *return_wt* is also quite significant. Accuracy of an oblique tree consisting only from the root node is 0.985645.

```
Root Hyperplane: Left = [9733,16], Right = [267,9984]
-17.031456 x[1] + -0.076999 x[2] + -0.274441 x[3] + 0.090547 x[4] + 0.323494 x[5] + -0.977444 = 0

l Hyperplane: Left = [0,16], Right = [9733,0]
167.295052 x[1] + 0.226616 x[2] + 0.274562 x[3] + -0.104502 x[4] + -1.985245 x[5] + 5.725792 = 0

r Hyperplane: Left = [251,1], Right = [16,9983]
-86.773194 x[1] + 0.003008 x[2] + -0.031606 x[3] + 0.034386 x[4] + 0.878198 x[5] + -0.334850 = 0

rr Hyperplane: Left = [14,0], Right = [2,9983]
62.650217 x[1] + 0.942421 x[2] + 0.676243 x[3] + 0.389375 x[4] + -0.747570 x[5] + -0.196499 = 0
```

Figure 6.3. OC1 induced oblique decision tree for valve saturation.



Problem	Undeheating	Accuracy	0.9971
Feature name	Minimum value	Maximum value	Average value
[1]pump	0.00	1.00	0.733
[2]return_wt	14.50	95.40	27.928
[3]supply_wt	13.50	67.16	28.774
[4]supply_wt_req	0.00	70.26	26.068
[5]valve	0.00	100.00	59.378

Root Hyperplane: Left = [9733,16] Right = [267,9984]
-17.031456 x[1] + -0.076999 x[2] + -0.274441 x[3] + 0.090547 x[4] + 0.323494 x[5] + -0.977444 = 0

pump	return_wt	supply_wt	supply_wt_req	valve	absolute member
-0.657	-0.113	-0.415	0.123	1.000	-0.051

Figure 6.4. Analysis of feature importance in root node splitting condition.

In theory, it is possible to find an oblique decision tree with just a root node, which would have 1.0 accuracy as CART. The reason why OC1 did not find this tree is probably a high resolution of *valve* position. All data is taken directly from SkySpark, and *valve* position has up to 8 decimal places accuracy. Training and validation dataset were rounded up to 1 decimal place in order to test this hypothesis. Rounded datasets in combination with a high number of hyperplane perturbations and restarts resulted in a perfect oblique tree with just root node and accuracy 1.0. This hyperplane decision tree is shown in Fig. 6.5. Rounding decimal places has enormous impact on this problem, but it does not seem to be significant for other tested datasets. Rounding of values is not used for any other dataset in this thesis.

```
Root Hyperplane: Left = [0,10000], Right = [10000,0]
-99.0 x[1] + 0.0 x[2] + 0.0 x[3] + 0.0x[4] + -1.00 x[5] + 198.01 = 0
```

Figure 6.5. Oblique decision tree for rounded datasets of valve saturation.

All partial results are summarized in Table 6.1.

Method	Model size	Accuracy
CART	2 nodes	1.0
Random Forest	100 estimators	1.0
OC1	root node	0.9856
OC1	4 nodes	0.9996
OC1 rounded	root node	1.0

Table 6.1. Accuracy comparison of different methods on valve saturation.

6.0.2 Underheating

Faulty control or mechanical defects can result in different supply water temperature than requested. The most common variant of this problem is underheating. Underheating can cause discomfort, and it is often an indication of failure within the system. Underheating in this test is defined by *supply_wt* + 0.5° < *supply_wt_req* while the *pump* is on. This problem should exploit the CART's inability to compare feature values.

CART induced decision tree depth was limited to 5 layers. The resulting tree is shown in Fig. 6.6. Root node utilized the *pump* state to filter out most records from a normal class. Records with running *pump* are further split based on *supply_wt* and *supply_wt_req*. These splits try to approximate feature comparison on given values.

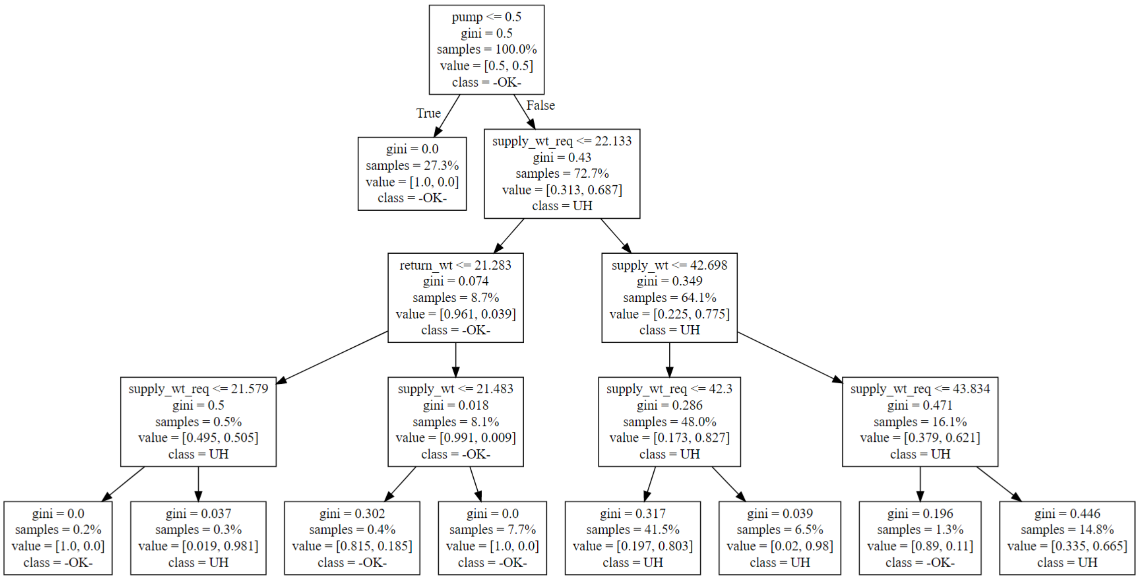


Figure 6.6. CART induced decision tree for the underheating problem (UH).

This approach does not achieve high accuracy and is hard to interpret. What might be surprising is that unpruned decision tree is more accurate even though it consists of over 800 nodes. Decision trees of this size are hardly ever interpretable and usually suffer from overfitting, which causes lower accuracy. In this test, nodes in lower tree levels are approximating comparison between *supply_wt* and *supply_wt_req*. As shown in Fig. 3.7, it is hard to fit oblique split even with an enormous number of binary splits.

Classification paths of this CART tree are dividing feature space into big areas far away from the true splitting line, where classification is easy and small areas near the true splitting line. A dataset with many records near the splitting line forces CART to

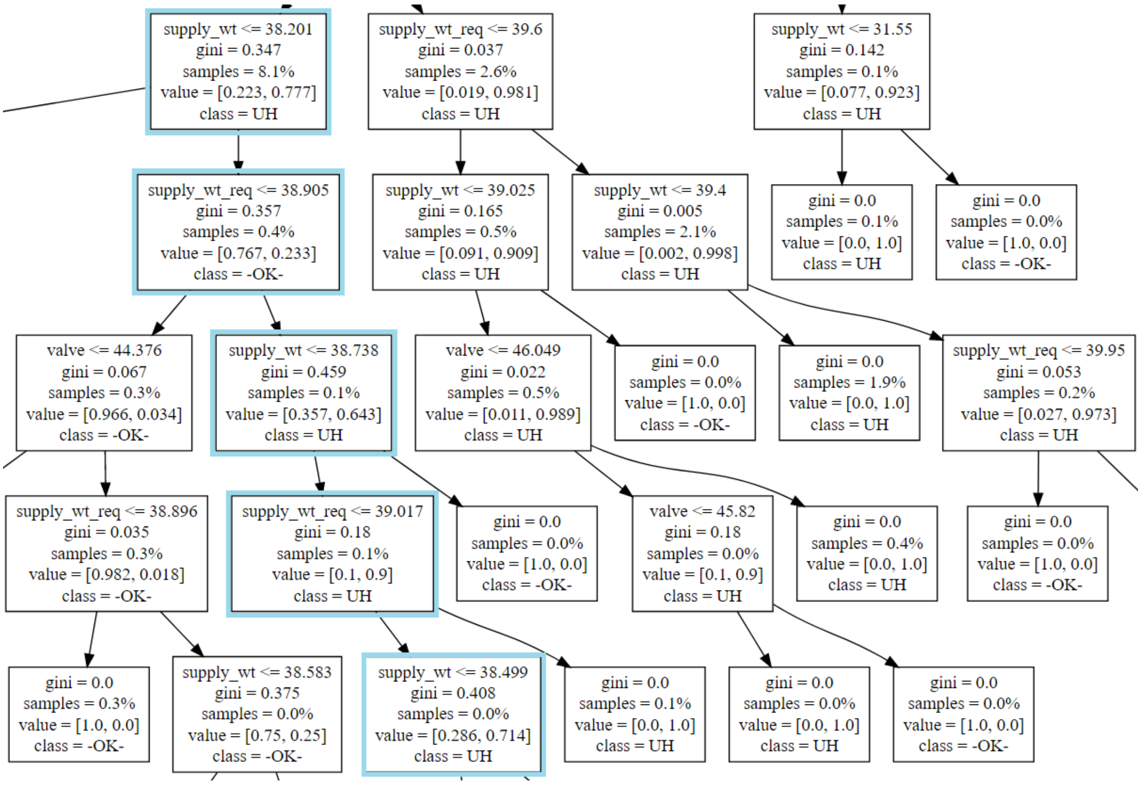


Figure 6.7. Section of unpruned decision tree for underheating problem.

create an enormous amount of layers. Classification path from the unpruned tree that is highlighted with a light blue colour in Fig. 6.7 is described using a scatter plot in Figs. 6.8 and 6.9.

In Fig. 6.8, the subset of records from training dataset affected by a given classification path is plotted using a scatter plot. True evaluation condition is represented by a red line. Decision tree splitting conditions are represented by black dashed lines. In following Fig. 6.9, parts of the scatter plot are highlighted in different colours based on which split cut off given part. The first four splitting conditions cut off areas in the order: red, green, blue and yellow. The last splitting condition divides remaining records into two pure leaves. The example shown in this figure demonstrates that the oblique splitting condition is approximated with an accuracy determined by training dataset. Small training datasets lead to poor accuracy, and big datasets lead to enormous trees.

The induced oblique decision tree is smaller, has superior accuracy and is easier to interpret than the CART tree. The resulting oblique decision tree with feature importance analysis is shown in Fig. 6.10. Root node filters records from normal classification class very effectively. The left node contains another splitting condition which further split records based on *supply_wt*, *supply_wt_req* and the *pump*. Oblique decision tree should have all the properties needed to solve this problem completely.

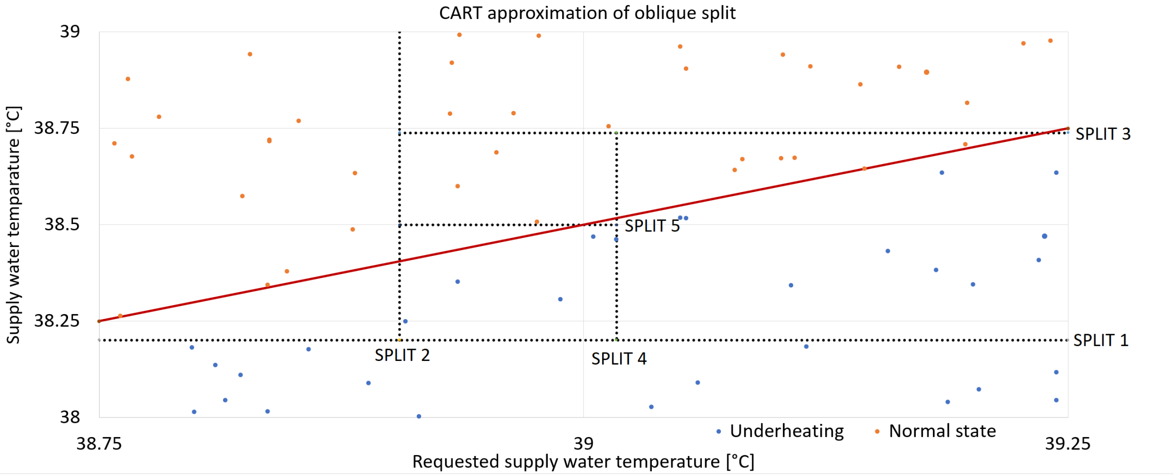


Figure 6.8. Visualization of the decision path of CART tree in underheating problem.

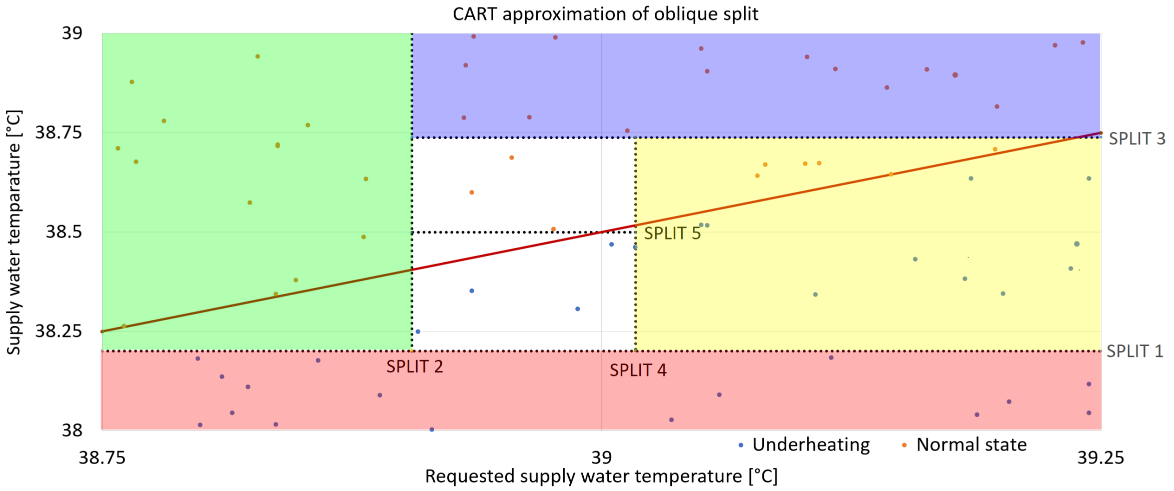
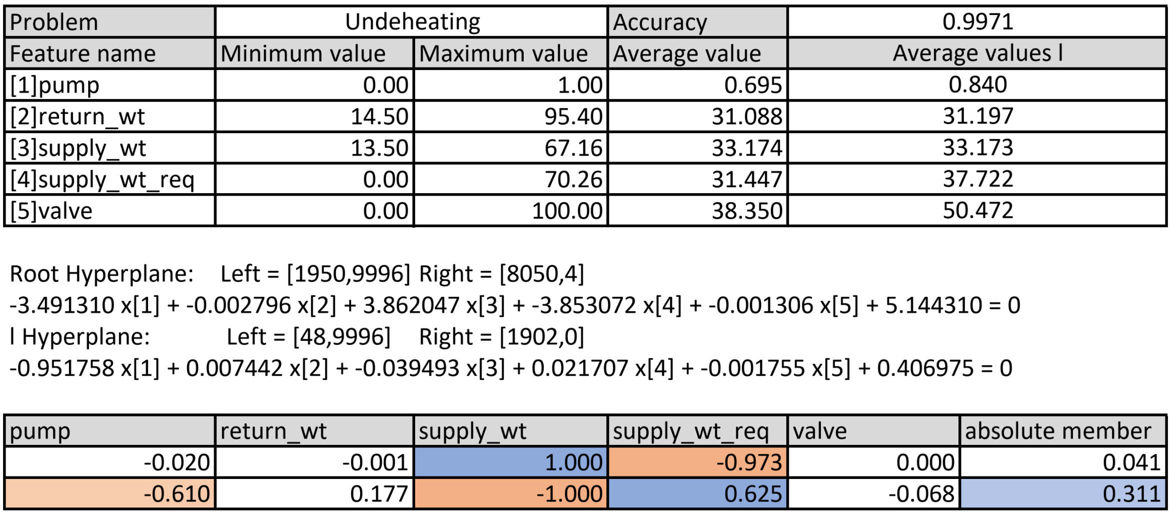


Figure 6.9. Visualization of the cutaway areas in the decision path of CART tree in underheating problem.



Problem	Undeheating		Accuracy	0.9971
Feature name	Minimum value	Maximum value	Average value	Average values l
[1]pump	0.00	1.00	0.695	0.840
[2]return_wt	14.50	95.40	31.088	31.197
[3]supply_wt	13.50	67.16	33.174	33.173
[4]supply_wt_req	0.00	70.26	31.447	37.722
[5]valve	0.00	100.00	38.350	50.472

Root Hyperplane: Left = [1950,9996] Right = [8050,4]

-3.491310 x[1] + -0.002796 x[2] + 3.862047 x[3] + -3.853072 x[4] + -0.001306 x[5] + 5.144310 = 0

l Hyperplane: Left = [48,9996] Right = [1902,0]

-0.951758 x[1] + 0.007442 x[2] + -0.039493 x[3] + 0.021707 x[4] + -0.001755 x[5] + 0.406975 = 0

pump	return_wt	supply_wt	supply_wt_req	valve	absolute member
-0.020	-0.001	1.000	-0.973	0.000	0.041
-0.610	0.177	-1.000	0.625	-0.068	0.311

Figure 6.10. Oblique decision tree for underheating.

What are the records that are misclassified?

The validation dataset of 200 000 records is split by decision tree among the nodes in the same ratio as in the training dataset. There are three leaf nodes. These nodes are referred to as r, ll and lr corresponding to their position within the decision tree. The total number of misclassified records is 598. The majority of these errors (539) ends up being classified by leaf ll, and the rest (59) is classified by leaf r. Leaf node lr classifies all records correctly. Records from validation dataset are filtered according to classification paths of ll and r leaves in order to detect any similarities between misclassified records.

Misclassifications – leaf r

All misclassifications made by leaf r are false–positive. Misclassified records can be split into two groups. The bigger group contains records, where *supply_wt* is right under threshold value for underheating while temperatures are above average. The relatively high coefficient for *return_wt* in combination with high temperature causes an incorrect classification. Maximum *return_wt* in validation dataset is 95.4 °C. Whenever *return_wt* reached 95.4°C, *supply_wt* is at a much lower temperature. This extremely high temperature is present in most misclassified records. This value is very likely to be a sensor fault. The smaller group of errors contains records where high *valve* position (90 – 100 %) is combined with the small difference between *supply_wt* and *supply_wt_req* (0.1 – 0.4 °C). Most controllers in heating circuits do not use that aggressive valve control near the requested value. These records are not represented in dataset sufficiently. This type of misclassification is responsible for 15 errors from a total of 200 000 records.

Misclassifications – leaf ll

All misclassifications made by leaf ll are false–negative. A small part of misclassified records contain records where *supply_w* is higher than *supply_wt_req* by more than 2 °C. The majority of errors falls into the last group. This group contains records where the difference between *supply_wt* and *supply_wt_req* is very close to the threshold. The average difference from threshold from over 500 cases is 0.0046 °C with the highest value 0.00660 °C. In heating circuits temperature sensors often work with the accuracy of 0.1 °C or worse.

The analysis of misclassified records shows the importance of quality of training datasets and limitations of the split search methods. This analysis also shows the importance of grey-box nature of decision trees. Expert opinion can be used to asses each classification branch.

All partial results are summarized in Table 6.2.

Method	Model size	Accuracy
CART	depth 5 – 16 nodes	0.8612
CART	unlimited – 842 nodes	0.9809
Random Forest	depth 5 – 100 estimators	0.8669
Random Forest	unlimited – 100 estimators	0.9737
OC1	2 nodes	0.9971

Table 6.2. Accuracy comparison of different methods on underheating.

6.0.3 Valve saturation and underheating

The last problem from the domain of heating circuits is the combination of valve saturation and underheating. This problem contains aspects which might benefit

CART with feature comparison suitable for oblique decision trees. Valve saturation and underheating can be a sign of undersized or suboptimally performing heat source. Detection of such behaviour can be used for predictive maintenance before further damage to the heat source is done. Alarm state is defined by *valve* position greater than 99%, *supply_wt* being more than 0.5°C lower than *supply_wt_req* and the *pump* state on. For clarity valve saturation and underheating is referred to as VSU.

Results from this experiment solidify the experience acquired from the past two tests. CART induced decision tree has no problems detecting valve saturation and pump state, but struggles to fit feature comparison with single feature splits. *Pump* and *valve* position splits significantly reduce the number of records that proceed to the lower parts of the decision tree. This reduction provides higher accuracy compared to the underheating problem. CART induced tree pre–pruned to the depth of five levels is shown in Fig. 6.11.

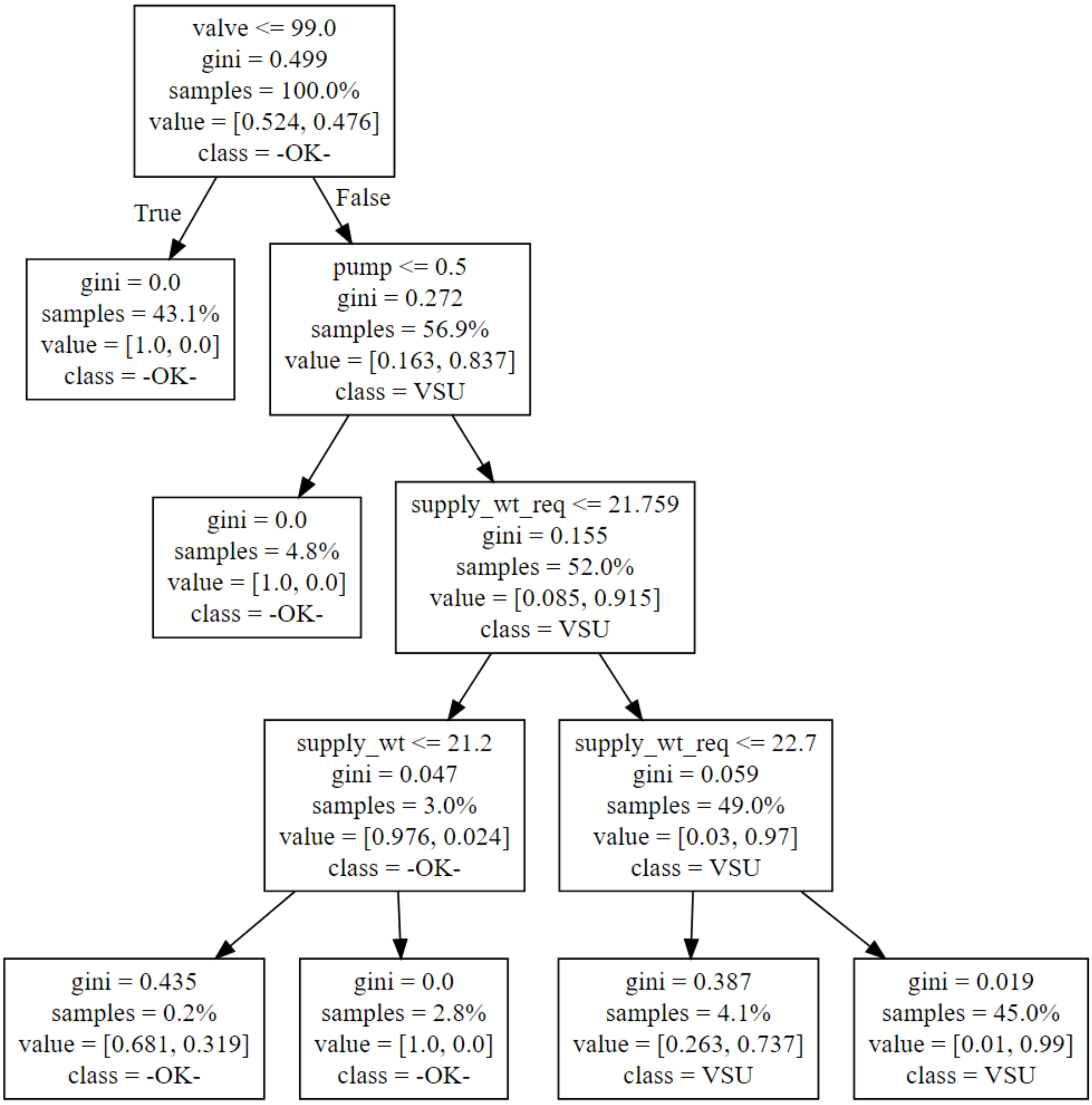
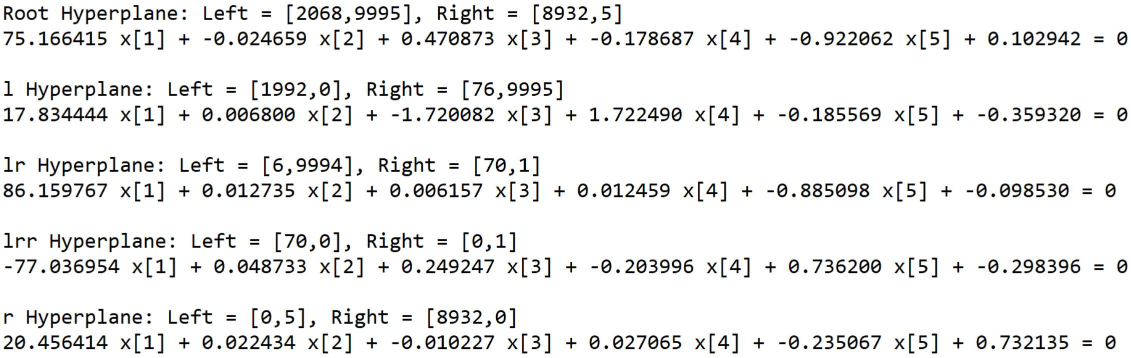


Figure 6.11. CART induced decision tree for valve saturation and underheating problem.

Fully grown CART tree starts with pump and valve nodes and then follow the same structure as the decision tree from the underheating problem. The unpruned tree is more accurate than the pruned one, but with over 150 nodes, it is difficult to interpret.

Random Forest performance is very similar to CART. With little accuracy advantage which also stands only for pruned trees, there is not much motivation to use Random Forest over CART for this problem.

The oblique decision trees show superior accuracy and interpretability to both CART and Random Forest. Oblique decision trees, as well as CART, have higher accuracy compared to the underheating problem. The fully grown oblique tree is shown in Fig. 6.12.

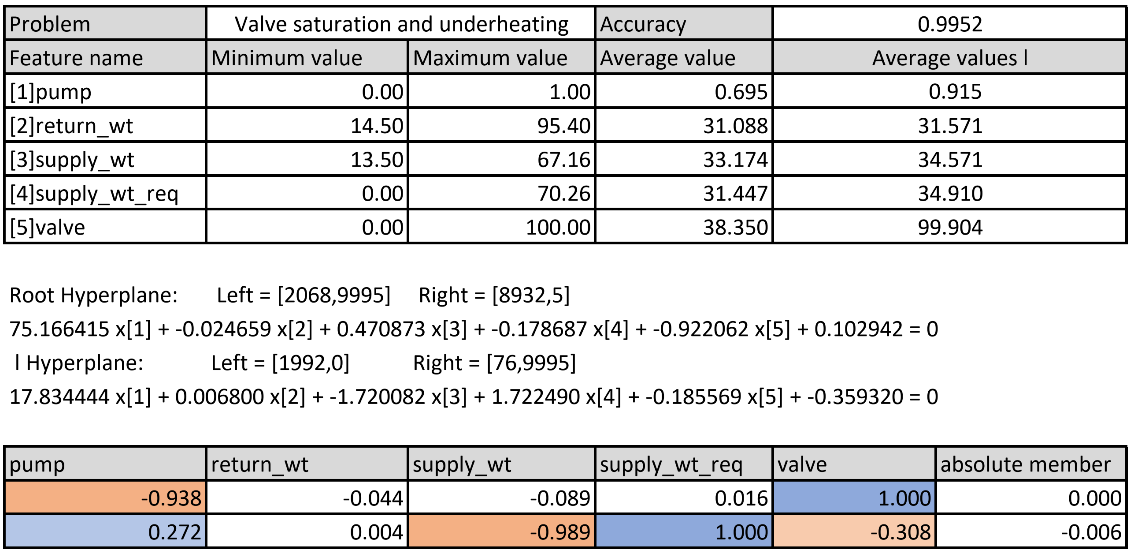


```
Root Hyperplane: Left = [2068,9995], Right = [8932,5]
75.166415 x[1] + -0.024659 x[2] + 0.470873 x[3] + -0.178687 x[4] + -0.922062 x[5] + 0.102942 = 0

l Hyperplane: Left = [1992,0], Right = [76,9995]
17.834444 x[1] + 0.006800 x[2] + -1.720082 x[3] + 1.722490 x[4] + -0.185569 x[5] + -0.359320 = 0

lr Hyperplane: Left = [6,9994], Right = [70,1]
86.159767 x[1] + 0.012735 x[2] + 0.006157 x[3] + 0.012459 x[4] + -0.885098 x[5] + -0.098530 = 0

lrr Hyperplane: Left = [70,0], Right = [0,1]
-77.036954 x[1] + 0.048733 x[2] + 0.249247 x[3] + -0.203996 x[4] + 0.736200 x[5] + -0.298396 = 0

r Hyperplane: Left = [0,5], Right = [8932,0]
20.456414 x[1] + 0.022434 x[2] + -0.010227 x[3] + 0.027065 x[4] + -0.235067 x[5] + 0.732135 = 0
```

Figure 6.12. CART induced decision tree for VSU.

Based on the misclassification analysis similar to the underheating problem, most classification errors are caused by records extremely close to the underheating threshold.

While oblique decision trees are much smaller than CART thus more manageable to explain, an oblique split can be much more difficult to understand fully. First two nodes of the oblique decision tree correctly classify all but 82 records from training dataset. The results of feature importance analysis of these splits are shown in Fig. 6.13.



Problem	Valve saturation and underheating		Accuracy	0.9952
Feature name	Minimum value	Maximum value	Average value	Average values l
[1]pump	0.00	1.00	0.695	0.915
[2]return_wt	14.50	95.40	31.088	31.571
[3]supply_wt	13.50	67.16	33.174	34.571
[4]supply_wt_req	0.00	70.26	31.447	34.910
[5]valve	0.00	100.00	38.350	99.904

Root Hyperplane: Left = [2068,9995] Right = [8932,5]

75.166415 x[1] + -0.024659 x[2] + 0.470873 x[3] + -0.178687 x[4] + -0.922062 x[5] + 0.102942 = 0

l Hyperplane: Left = [1992,0] Right = [76,9995]

17.834444 x[1] + 0.006800 x[2] + -1.720082 x[3] + 1.722490 x[4] + -0.185569 x[5] + -0.359320 = 0

pump	return_wt	supply_wt	supply_wt_req	valve	absolute member
-0.938	-0.044	-0.089	0.016	1.000	0.000
0.272	0.004	-0.989	1.000	-0.308	-0.006

Figure 6.13. Feature importance analysis for oblique decision tree pruned to two nodes for VSU.

From feature importance, it can be deduced that the first split combines condition on *pump* state with the condition on *valve* saturation. The second splitting condition further divides records based on the difference between *supply_wt* and *supply_wt_req*. This approach closely describes evaluating condition with just two conjugated oblique splits. Combining multiple single–feature splitting conditions into a single oblique split shrinks tree size, but it is not intuitive and might be hard to understand with an increasing number of features.

All partial results are summarized in Table 6.3.

Method	Model size	Accuracy
CART	depth 5 – 10 nodes	0.9838
CART	unlimited – 158 nodes	0.9977
Random Forest	depth 5 – 100 estimators	0.9851
Random Forest	unlimited – 100 estimators	0.9976
OC1	2 nodes	0.9952
OC1	5 nodes	0.9993

Table 6.3. Accuracy comparison of different methods on VSU.

6.1 Heating circuits decision tree usability conclusion

Oblique decision trees are superior to other methods in cases of underheating and VSU. CART decision trees can be trained using the same datasets, and training takes very little computational time compared to OC1. As demonstrated with valve saturation problem, CART can provide very small and descriptive decision trees on some problems. Due to minimal amount of extra effort required, it is worth to train CART tree just for comparison and other insight into the studied problem. Oblique trees show better interpretability and performance and are suitable for more challenging tasks. Decision tree methods can be successfully used for heating circuit diagnostics.

Chapter 7
Application in air handling units domain

After decision tree method comparison on the easier HVAC system in Chapter 6, oblique decision trees alongside with CART are tested on problems from AHU domain. There is an ongoing project[9] in UCEEB that tries to diagnose AHU based on a set of expert–designed rules. Series of tests were conducted in order to create datasets to evaluate the performance of designed rules. This dataset was created prior to the idea of using machine learning methods. Some pitfalls of using this dataset with decision tree methods are described in the following section. Decision tree methods are used on various selected faults.

7.1 Dataset description

The dataset consists of normal state records and records of various faults. Faults are recorded on up to 6 AHU units for the duration of 2 hours. AHU units data are sampled with a sampling period of 5 minutes. Records representing the normal state of AHU units are from periods where was no testing done. The resulting dataset consists of tenths of thousands of records of normal states and up to 144 records of each fault.

The use of decision tree methods requires records to be consistent in the number of features. Some of the AHU units have extra sensors that are not present in others. Therefore, these extra features are omitted. An example scheme of an air handling unit is shown in Fig. 7.2.

The dataset consists of the following features:

- **fanE_log** – exhaust air fan position (1 – open, 0 – closed),
- **fanF_log** – fresh air fan position (1 – open, 0 – closed),
- **AHU_log_req** – run request signal from control system,
- **AHUS_vfr** – volumetric flowrate,
- **hreBy_con** – bypass position,
- **heater_log** – heater state (1 is running),
- **heater_vlv** – heater valve position,
- **heaterOut_tw** – measured water temperature after heater,
- **chiller_vlv** – chiller valve position,
- **chillerOut_tw** – measured water temperature after chiller,
- **oda_ta** – measured outdoor air inlet temperature,
- **sup_ta** – measured outlet air temperature,
- **zone_ta** – measured space temperature,

- **eta_ta** – measured exhausted air inlet temperature,
- **eha_dmp** – air damper position on exhaust air outlet (1 – open, 0 – closed),
- **oda_dmp** – air damper position on fresh air inlet (1 – open, 0 – closed).

In this application decision tree methods build models that try to split records accurately and straight-forwardly. The induced decision trees have accuracy 1.0 on all selected faults and consist only from the root node. These results are suspicious, and further analysis shows that all faults were recorded on Fridays. Requested measured space temperature was the same on all weekdays except for Friday. This means that the decision tree could perfectly distinguish any faults just by checking this temperature. CART induced tree is shown in Fig. 7.1.

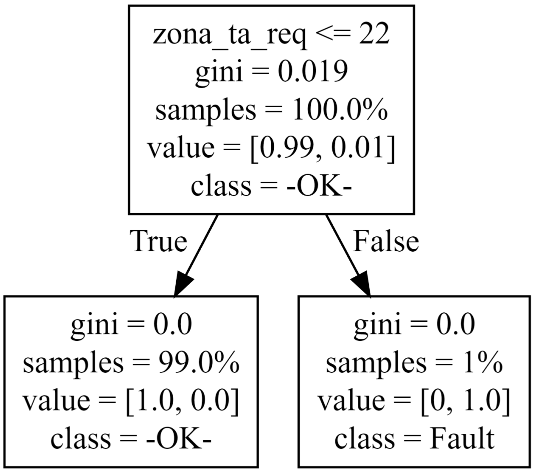


Figure 7.1. CART induced tree for AHU dataset that uses just requested temperature that has nothing to do with recorded fault.

This example demonstrates how easy it is to produce a dataset that might be suitable for the expert–design analytical approach but useless for machine learning methods. The requested temperature was removed from the dataset.

7.2 Shut down fans while heating

This test was conducted on all AHU units and is represented by 84 faulty records. Fault state is defined by at least one of the fans being shut down while the AHU unit is heating. CART induced tree is shown in Fig. 7.3.

Interpretability of this decision tree is hard. Two–thirds of fault states are classified based on *eha_dmp* and *AHU_log_req*, and just one third takes fans into account. Accuracy of this decision tree is 0.998, but the accuracy is highly misleading due to the size of the dataset.

The trained oblique decision tree consists of just two nodes. Root node successfully filters out 43 out of 45 fault states in the training dataset. The second node further splits records into a large pure group of normal states and a small group of just one normal and two fault records. The interpretability of the root node decision is important. Feature importance analysis of a subtree containing just the root node is shown in Fig. 7.4.

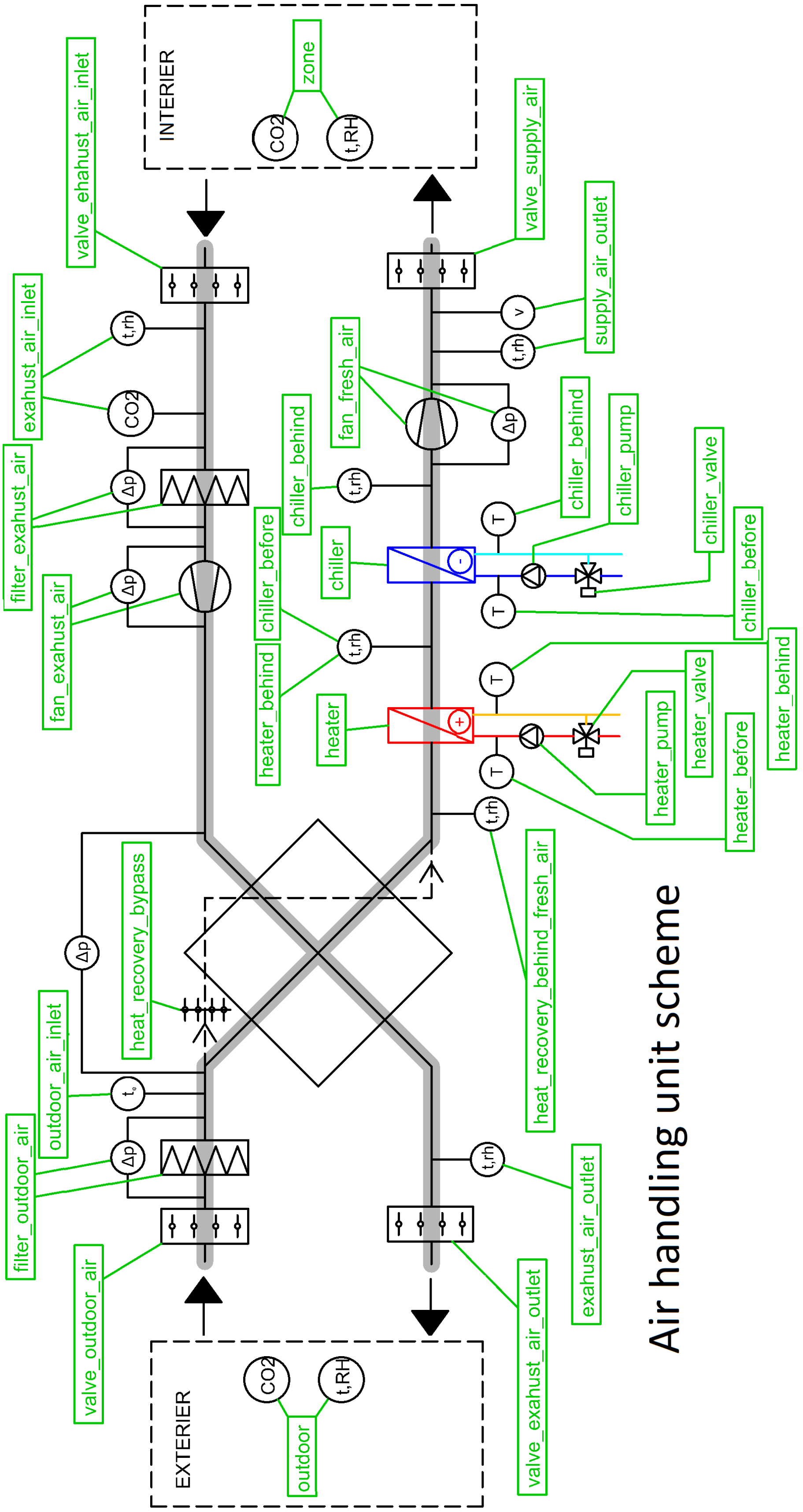


Figure 7.2. AHU scheme with sensors and actuators labels.

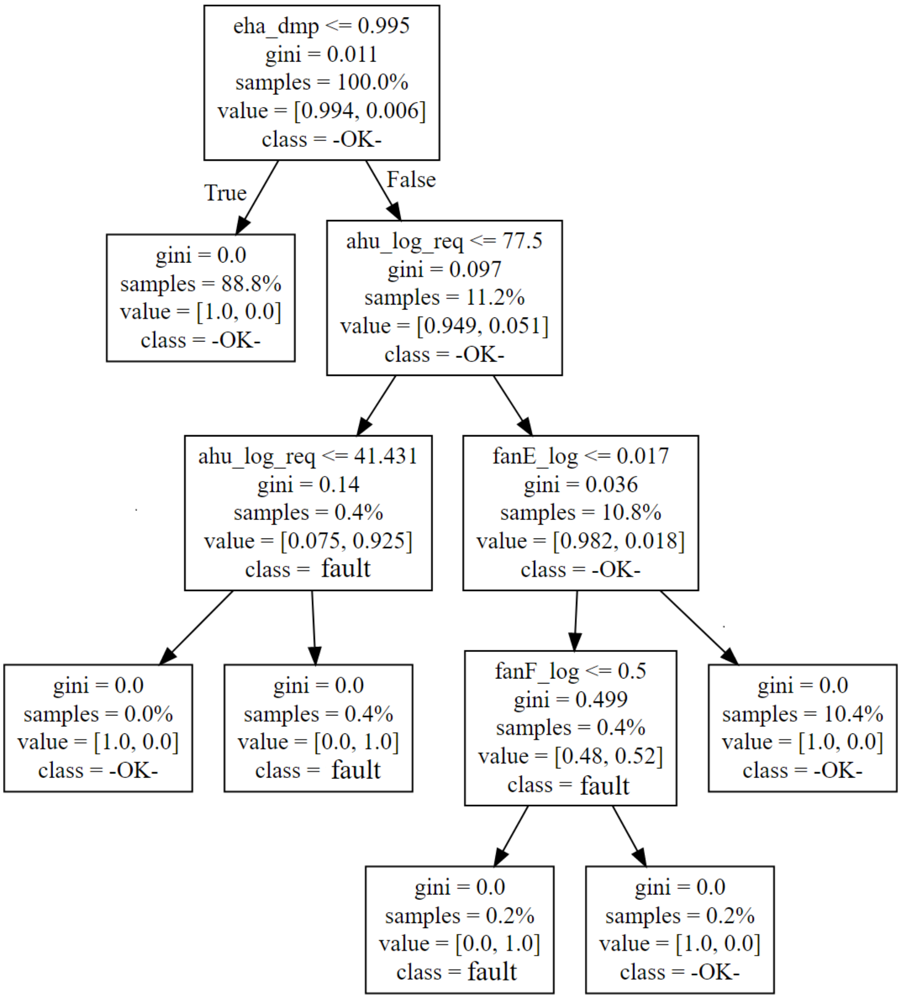


Figure 7.3. CART induced tree for shut down fans while heating fault.

Full feature importance is extensive; therefore, features with low importance are not displayed. It is clear that the root node is affected by both $fanE\_log$ and $fanF\_log$ as well as $chillerOut\_tw$ and $heater\_log$.

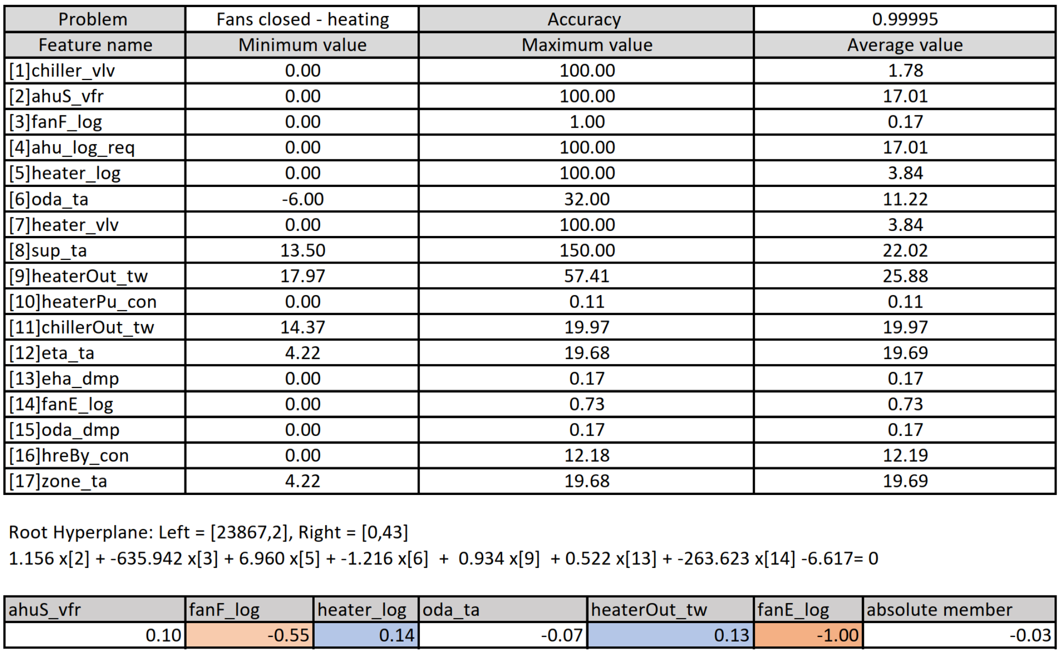
From the feature weights in root splitting condition, it can be deduced that at least one of the fans must be closed and $heater\_log$ must be on. This explanation follows the real fault and shows that this decision tree can be used for diagnostics of such fault.

7.3 Pressure sensor fault

The second experiment was designed to demonstrate the fault of a differential pressure sensor. This fault leads to different states of $fanE\_log$ and $fanF\_log$. CART induced tree is shown in Fig. 7.5.

This decision tree has accuracy 0.995, but its interpretation is impossible. Both $fanE\_log$ and $fanF\_log$ are not present in any splitting condition.

The oblique decision tree consists of two nodes. Root node splits all fault records with five normal states into one branch and the rest of the normal states into the other. The second node filters out the remaining five normal states. Feature analysis shown in Fig. 7.6 is done only on the root node. Features with very low importance are not displayed.



Problem	Fans closed - heating	Accuracy	0.99995
Feature name	Minimum value	Maximum value	Average value
[1]chiller_vlv	0.00	100.00	1.78
[2]ahuS_vfr	0.00	100.00	17.01
[3]fanF_log	0.00	1.00	0.17
[4]ahu_log_req	0.00	100.00	17.01
[5]heater_log	0.00	100.00	3.84
[6]oda_ta	-6.00	32.00	11.22
[7]heater_vlv	0.00	100.00	3.84
[8]sup_ta	13.50	150.00	22.02
[9]heaterOut_tw	17.97	57.41	25.88
[10]heaterPu_con	0.00	0.11	0.11
[11]chillerOut_tw	14.37	19.97	19.97
[12]eta_ta	4.22	19.68	19.69
[13]eha_dmp	0.00	0.17	0.17
[14]fanE_log	0.00	0.73	0.73
[15]oda_dmp	0.00	0.17	0.17
[16]hreBy_con	0.00	12.18	12.19
[17]zone_ta	4.22	19.68	19.69

Root Hyperplane: Left = [23867,2], Right = [0,43]

1.156 x[2] + -635.942 x[3] + 6.960 x[5] + -1.216 x[6] + 0.934 x[9] + 0.522 x[13] + -263.623 x[14] -6.617= 0

ahuS_vfr	fanF_log	heater_log	oda_ta	heaterOut_tw	fanE_log	absolute member
0.10	-0.55	0.14	-0.07	0.13	-1.00	-0.03

Figure 7.4. Feature importance analysis of oblique tree for shutdown fans while heating fault.

Maximum values for some temperature features suggest that normal state records in the dataset contain some corrupted records. Values of *zone_ta* should not reach 150 °C. Higher quality dataset could improve the performance of decision tree methods. The major issue is probably the low number of fault records.

7.4 Bypass stuck in the open position

The last problem from AHU domain is bypass stuck in the open position. This fault was simulated using false sensor data. This means that the control system, as well as data in the dataset, think that bypass is working properly, even though it is stuck in a fully open position. CART induced decision tree achieves excellent accuracy, but after further inspection, it is clear that it does not truly model the problem. Decision tree uses combinations of upper and lower limits on various features to extract fault records. This behaviour is described in section 6.0.2 and demonstrated in Fig. 6.8. This problem is caused by small dataset and insufficient excitation of the system during fault simulations. The induced tree is shown in Fig. 7.7. Colours highlight combinations of upper and lower limits.

Oblique decision tree resulted in 5 nodes with the accuracy of 0.9990. Analysis of feature importance shows that almost all features have similar importance and oblique splitting conditions are impossible to interpret. Dominant features are the same as in the CART decision tree. Oblique decision trees try to filter out faulty records in the most efficient way. With a small number of faulty records, a decision tree can overfit data rather than describe faulty behaviour of the system.

7.5 Results and AHU usage conclusion

CART and oblique decision trees are trained for three selected problems from the AHU domain. The first problem is the detection of shutdown fans while the AHU

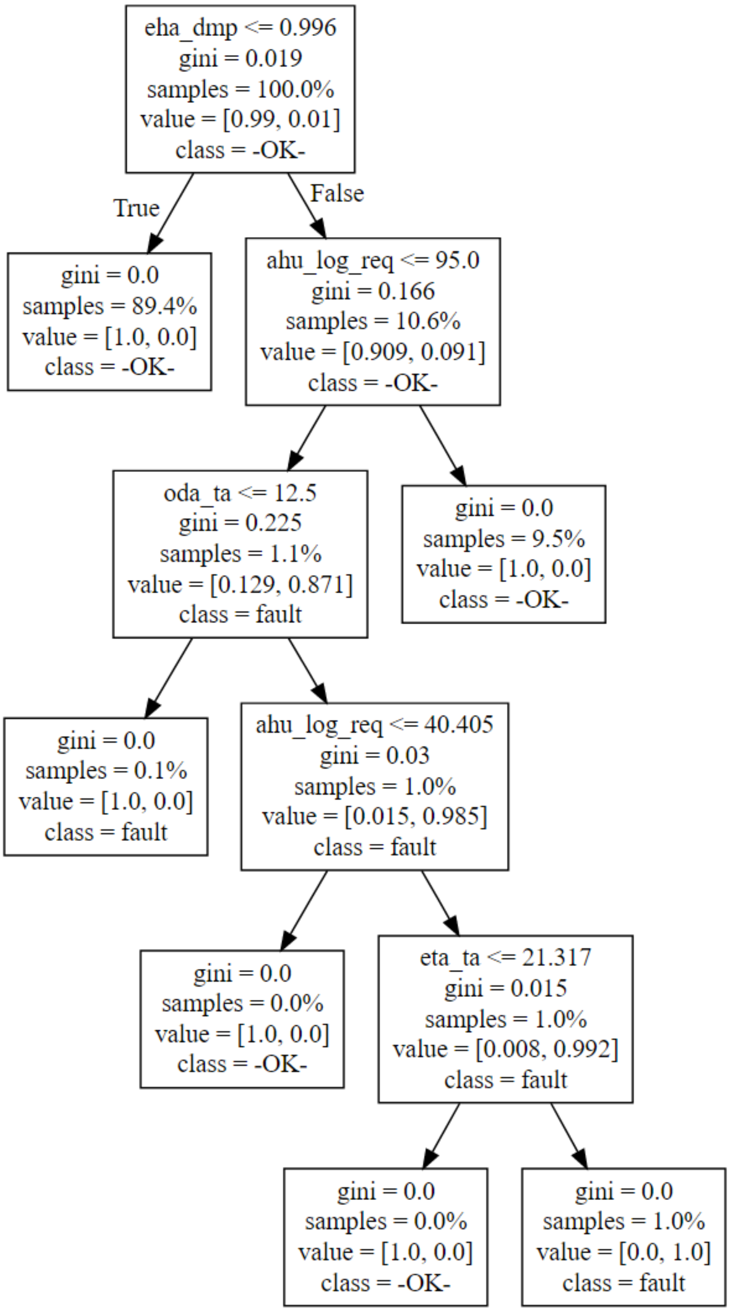


Figure 7.5. CART induced tree for pressure sensor fault.

is heating. CART decision tree contains splitting conditions on $fanE\_log$ and achieves high accuracy, but it can not be interpreted. The oblique decision tree has even higher accuracy. Feature importance analysis highlights expected features of the root node. These features contain both fans signals as well as features that can be used to detect heating such as $heater\_log$.

Both further problems do not result in any usable models. All trained decision trees have accuracy over 0.995 but are not applicable in the real-world. Further analysis of trained models shows that decision trees use signals that have nothing to do with the faults and just overfit the data. CART ability to filter records based on upper and lower limit splits on some features such as temperatures is shown in Fig. 7.7. Resulting

Problem	Pressure sensor fault	Accuracy	0.99993
Feature name	Minimum value	Maximum value	Average value
[1]chiller_vlv	0.00	100.00	1.215
[2]ahuS_vfr	0.00	100.00	10.399
[3]fanF_log	0.00	1.00	0.092
[4]ahu_log_req	0.00	100.00	10.399
[5]heater_log	0.00	100.00	2.911
[6]oda_ta	-6.00	33.91	10.662
[7]heater_vlv	0.00	100.00	2.911
[8]sup_ta	13.50	150.00	24.880
[9]heaterOut_tw	17.97	57.41	30.821
[10]heaterPu_con	0.00	1.00	0.066
[11]chillerOut_tw	15.31	79.96	22.869
[12]eta_ta	-1.15	150.00	22.081
[13]eha_dmp	0.00	1.00	0.107
[14]fanE_log	0.00	1.00	0.513
[15]oda_dmp	0.00	1.00	0.107
[16]hreBy_con	0.00	100.00	7.168
[17]zone_ta	-1.15	150.00	22.082

Root Hyperplane: Left = [13112,0], Right = [5,128]
-157.344 x[3]+ 0.759 x[6] + -3.492 x[8]+ 0.653 x[9]+ -0.464 x[10]+ 0.395 x[12] +1.022 x[17] + 9.166 = 0

fanF_log	oda_ta	sup_ta	heaterOut_tw	chillerOut_tw	eta_ta	zone_ta	absolute member
-0.17	0.09	-1.00	0.23	-0.12	0.10	0.26	0.11

Figure 7.6. Feature importance analysis of oblique tree for pressure sensor fault.

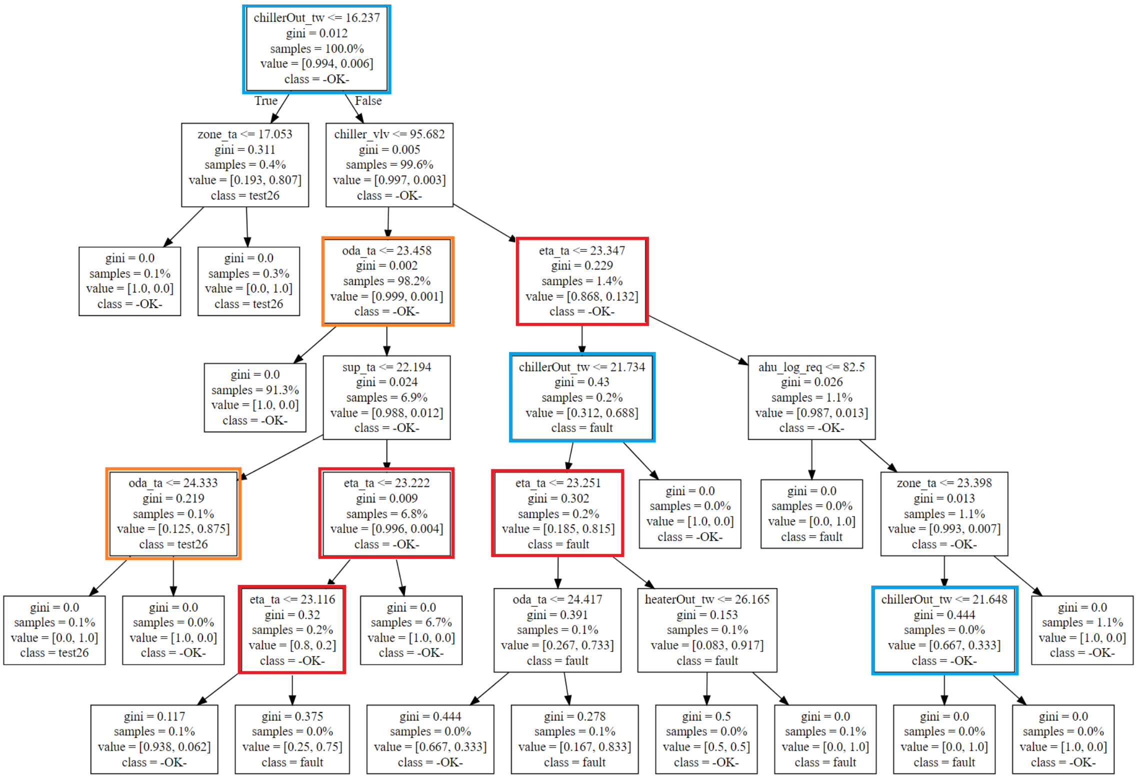


Figure 7.7. CART induced tree for pressure sensor fault.

models analysis leads to the conclusion that the training dataset is too small for the application of the decision tree methods.

The vital advantage of grey–box nature of decision tree models lies in detecting when trained models are not usable despite their high accuracy. It is impossible to evaluate trained models without proper validation dataset or the ability to analyse trained models. The results from conducted tests show the advantage of decision trees over black–box methods in this application.

Chapter 8
Conclusion

Nowadays, expectations of energy efficiency and reliability of many HVAC systems are higher than before. It is requisite to use diagnostics to meet these requirements. Faulty systems can lead to wasted energy, discomfort or even danger. Expert time is costly and limited, which calls for automated diagnostics using diagnostic rules or methods based on models.

This thesis investigates the usability of decision trees in fault detection in HVAC. At first, decision tree principles were explained alongside with their drawbacks and advantages. Based on literature, decision tree modifications such as Random Forest and oblique decision trees were presented.

Selection of decision tree methods was implemented using Python library scikit–learn and OC1 classifier code. Implemented methods were compared on a real–world dataset from the heating circuit domain that consists of over 8 million records. Decision tree methods were also utilised on AHU dataset from UCEEB. This usage tested whether it is possible to provide an alternative to an expert–designed set of diagnostic rules using already existing dataset.

CART induced trees were able to achieve good accuracy on most problems, but their interpretability was problematic. HVAC systems and process control in general work with feature comparison and dependent variables. Dependent variables are, for example, requested and measured values of various temperatures, pressures or positions. CART is not able to handle these problems and preserve the interpretability of trained models.

Oblique decision trees were superior to CART in both accuracy and interpretability. In heating circuit domain OC1 induced trees contained a small number of nodes. They were easy to interpret and had excellent accuracy. In AHU domain trained decision trees revealed that despite the great accuracy (over 0.995) they are not usable in this application. Due to the small size of the dataset decision trees managed to classify records based on features unrelated to the detected fault. Notwithstanding its limitations, one of the AHU faults demonstrates well that oblique decision trees can also be successfully used on more complex systems.

In general, decision trees especially oblique decision trees seem to be an excellent tool for diagnostic rule design. Its grey–box nature allows experts to check how is trained model working and compare splitting conditions with existing diagnostic rules. Interpretability of trained models allows decision tree models to work as a foundation for expert–design rules. Trained models can also be manually tweaked based on expert experience.

Decision trees usage has its limitation in dataset size. The tested AHU dataset consists of simulations from multiple units in the duration of hours. The biggest challenge of decision tree usage is the creation of a sufficiently large and diverse dataset. The solution to this problem could be cooperation with manufacturers that might be able to produce sufficient datasets.

References

[1] Anyanwu, M., and Shiva, S. Comparative analysis of serial decision tree classification algorithms. *International Journal of Computer Science and Security 3(3)* (09 2009).

[2] Barman, R. K., Saha, S., and Das, S. Prediction of interactions between viral and host proteins using supervised machine learning methods. *PLoS ONE 9*, 11 (Nov. 2014), e112034.

[3] Beghi, A., Brignoli, R., Cecchinato, L., Menegazzo, G., Rampazzo, M., and Simmini, F. Data-driven fault detection and diagnosis for HVAC water chillers. *Control Engineering Practice 53* (Aug. 2016), 79–91.

[4] Breiman, L. Bagging predictors. *Machine Learning 24*, 2 (Aug 1996), 123–140.

[5] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[6] Brynjolfsson, E., and Mitchell, T. What can machine learning do? workforce implications. *Science 358*, 6370 (2017), 1530–1534.

[7] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122.

[8] Chaturvedi, S., and Patil, S. Oblique Decision Tree Learning Approaches - A Critical Review. *International Journal of Computer Applications 82*, 13 (Nov 2013), 6–10.

[9] D.ADAMOVSKY, L.DOBIASOVA, V.MAZANEC, O.NEHASIL, and J.SIROKY. Expert system for analysis of air handling units in practice. *Journal of Heating, Ventilation, Sanitation 3*, 1 (feb 2019), 131–133.

[10] Denil, M., Matheson, D., and Freitas, N. D. Narrowing the gap: Random Forests in theory and in practice. In *Proceedings of the 31st International Conference on Machine Learning* (Bejing, China, 22–24 Jun 2014), E. P. Xing and T. Jebara, Eds., vol. 32 of *Proceedings of Machine Learning Research*, PMLR, pp. 665–673.

[11] Esposito, F., Malerba, D., Semeraro, G., and Kay, J. A comparative analysis of methods for pruning decision trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 19* (06 1997), 476 – 491.

[12] Gelfand, S. B., Ravishankar, C. S., and Delp, E. J. An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 2 (Feb 1991), 163–174.

[13] L.Breiman. *Machine Learning 45*, 1 (2001), 5–32.

[14] Li, K., Yu, N., Li, P., Song, S., Wu, Y., Li, Y., and Liu, M. Multi-label spacecraft electrical signal classification method based on DBN and Random Forest. *PLOS ONE 12*, 5 (May 2017), e0176614.

[15] Mukesh, M. How to tune a decision tree? https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680. Accessed: 2020-04-20.

[16] Mulyar, A. sklearn-oblique-tree. https://github.com/AndriyMulyar/sklearn-oblique-tree/, 2019.

[17] Murthy, S. K., Kasif, S., and Salzberg, S. A System for Induction of Oblique Decision Trees. *arXiv e-prints* (Jul 1994), cs/9408103.

[18] Najafi, M., Auslander, D. M., Bartlett, P. L., Haves, P., and Sohn, M. D. Expert system for analysis of air handling units in practice. *Applied Energy 96* (Aug. 2012), 347–358.

[19] Patel, N., and Upadhyay, S. Article: Study of various decision tree pruning methods with their empirical comparison in weka. *International Journal of Computer Applications 60*, 12 (December 2012), 20–25. Full text available.

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

[21] Podgorelec, V., Kokol, P., Stiglic, B., and Rozman, I. *Journal of Medical Systems 26*, 5 (2002), 445–463.

[22] Probst, P., and Boulesteix, A.-L. To tune or not to tune the number of trees in Random Forest. *Journal of Machine Learning Research 18*, 181 (2018), 1–18.

[23] Quinlan, J. Simplifying decision trees. *International Journal of Man-Machine Studies 27*, 3 (1987), 221 – 234.

[24] Quinlan, J. R. Induction of decision trees. *MACH. LEARN 1* (1986), 81–106.

[25] Quinlan, J. R. Improved Use of Continuous Attributes in C4.5. *arXiv e-prints* (Feb 1996), cs/9603103.

[26] Salzberg, S. L. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning 16*, 3 (Sep 1994), 235–240.

[27] Shalev-Shwartz, S., and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[28] Simeone, O. A Very Brief Introduction to Machine Learning With Applications to Communication Systems. *arXiv e-prints* (Aug 2018), arXiv:1808.02342.

[29] Taylor, P. C., and Silverman, B. W. Block diagrams and splitting criteria for classification trees. *Statistics and Computing 3* (1993), 147–161.

[30] Witten, I. H. v., Frank, E., and Hall, M. A. *Data mining : practical machine learning tools and techniques.* Burlington, 2011.

[31] Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., and Steinberg, D. Top 10 algorithms in data mining. *Knowledge and Information Systems 14*, 1 (Jan 2008), 1–37.

[32] Zhang, A., Ballas, N., and Pineau, J. A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning. *arXiv e-prints* (Jun 2018), arXiv:1806.07937.

Appendix A
Abbreviations

AHU	Air handling unit
API	Application programming interface
CART	Classification and regression tree
CPU	Cetral processing unit
ČVUT	Czech Technical University in Prague
FEL	Faculty of electrical engineering
HVAC	Heating, ventilation, air conditioning
ID3	Iterative Dichotomiser 3
ID4.5	Iterative Dichotomiser 4.5
OC1	Oblique clasifier one
OOB	Out of bag classification
SCADA	Supervisory control and data acquisition
UCEEB	University centre for energy efficient buildings
VSU	Valve saturation underheating

Appendix B
Contents of the enclosed CD

The electronic version of the diploma thesis is present on enclosed CD. This CD also includes source code, datasets and results of all tests mentioned in the presented work.