**Title**
Continuous Commissioning of Buildings: HVAC Fault Detection and Diagnosis



**Author**
Shia, James

**Publication Date**
2018

Peer reviewed|Thesis/dissertation

**Continuous Commissioning of Buildings: HVAC Fault Detection and Diagnosis**

By

James Alexander Shia

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in Charge:

Professor David M. Auslander, Chair

Professor Chris Dames

Professor Haiyan Huang

Fall 2018

**Continuous Commissioning of Buildings: HVAC Fault Detection and Diagnosis**

# Abstract

Continuous Commissioning of Buildings: HVAC Fault Detection and Diagnosis

by

James Alexander Shia

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor David M. Auslander, Chair

It is almost certain that all systems contain faults, and Heating, Ventilation, and Air Conditioning (HVAC) systems are no exception. About 41% of total energy consumption in the U.S. of year 2014 is used for heating and air conditioning, that is about 40 quadrillion ($10^{15}$) British thermal units (BTU)! In the past, fault detection and diagnosis (FDD) has been commissioned by man, which is costly and inefficient. If FDD can be done automatically and continuously, a great amount of energy could be saved and our buildings would become more sustainable.

People have been working on HVAC FDD for years, and will continue to make progress. However, buildings are complicated, and so are the HVAC systems which comes with them. Thanks to the fast growth of computers and developments of machine learning algorithms in recent years, we have more tools to work with.

In despite of many researches done on HVAC FDD, we have noticed that very few publications have focused on scalability and low cost. In order to address this challenge, we will propose an approach which focuses on control data. In this thesis, we will be relying on simulation data because of data access issues and the lower cost for experiments. A list of machine learning algorithms are introduced for data exploration and analysis, a control data focused model free approach is presented as well, and finally, FDD is carried out by implementing anomaly algorithms.

# Contents

# List of Figures

# List of Tables

# Acknowledgments

I would like to thank my advisor Professor David M. Auslander for his guidance and supervision throughout my years at UC Berkeley. This work would not have been completed without his help. He has provided insightful advice and pointed out some blind spots of my work. This has both inspired and encouraged me, keeping me motivated and on-track. A true mentor in my academic studies.

I would also like to thank Professor Chris Dames, Professor Haiyan Huang for being my committee members and giving me great suggestions to my work, helping me to have my work become more complete and well-organized, Professor Van P. Carey for his advice to keep me in the right direction. Also, I would like to thank the department for understanding my difficulties and the help to keep me in progress. I consider myself very fortunate and privileged to meet with all these great people.

I am thankful to UC Berkeley for providing such a great environment full of great people to learn from. I would like to thank anyone who has provided me any form of help to my work.

Last but not the least, I would like to thank my family for always being encouraging and supportive. Their support has been essential to me and is the main reason I am able to seek for further education. Words are not enough to express how grateful I am.

# 1 Introduction

## 1.1 Energy usage of HVAC systems

Buildings play an important role in human civilizations. They are a symbol of technology advancement, a style of culture, and a shelter for people. The most important thing is that buildings provide indoor environments to meet with our day to day needs. For example, libraries provide quiet places to study and read, meeting rooms for people to discuss things, cold rooms are used for food storage, negative pressure rooms for special treatments, and so on.

As the scale of buildings grow larger to provide more space capacity, the task for maintaining the indoor environment grows in scale as well; thus, Heating Ventilation and Air-conditioning (HVAC) systems, become more complicated and power consuming. In the U.S., more than 40% of the total energy consumption is used for heating and air conditioning purposes in general households. This is shown in figure 1.1.

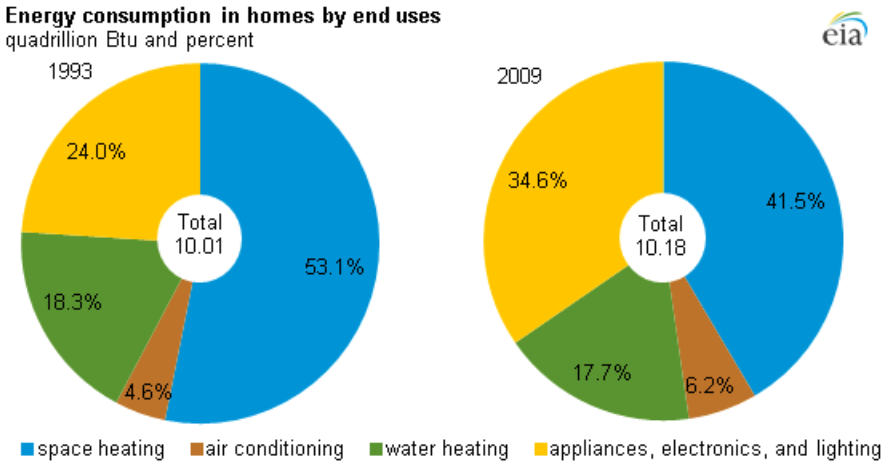



**Figure 1.1:** Energy consumption in residential homes (U.S. Energy Information Administration, Residential Energy Consumption Survey).

Due to improvements in insulation, new sustainable materials, and more efficient designs, newer buildings generally have better HVAC system energy efficiencies. According to U.S. Energy Information Administration's (EIA) most recent Residential Energy Consumption Survey (RECS), homes built in 2000 and later consume only 2% more energy on average than homes built prior to 2000, despite being on average 30% larger [1, 2]. See figure 1.2. The general trend of energy usage is similar for commercial buildings as well. See figure 1.3.

From the figures we see that the overall percentage of energy consumption for HVAC systems has gradually dropped from more than half to around 40% over the past 20 years. Still, the amount of energy consumed is considerably high! In 2014, 41% of total U.S. energy consumption was consumed in residential and commercial buildings, or about 40 quadrillion ($10^{15}$) British thermal units (BTU).

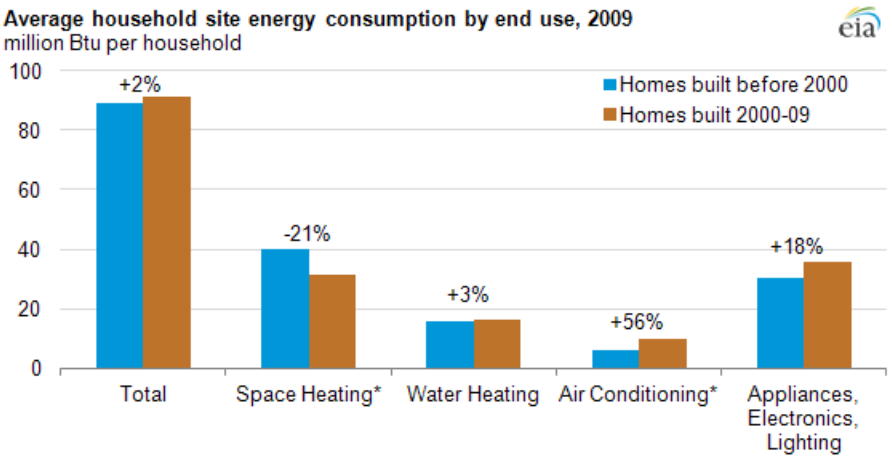



**Figure 1.2:** Energy consumption changes in residential homes (U.S. Energy Information Administration, Residential Energy Consumption Survey).

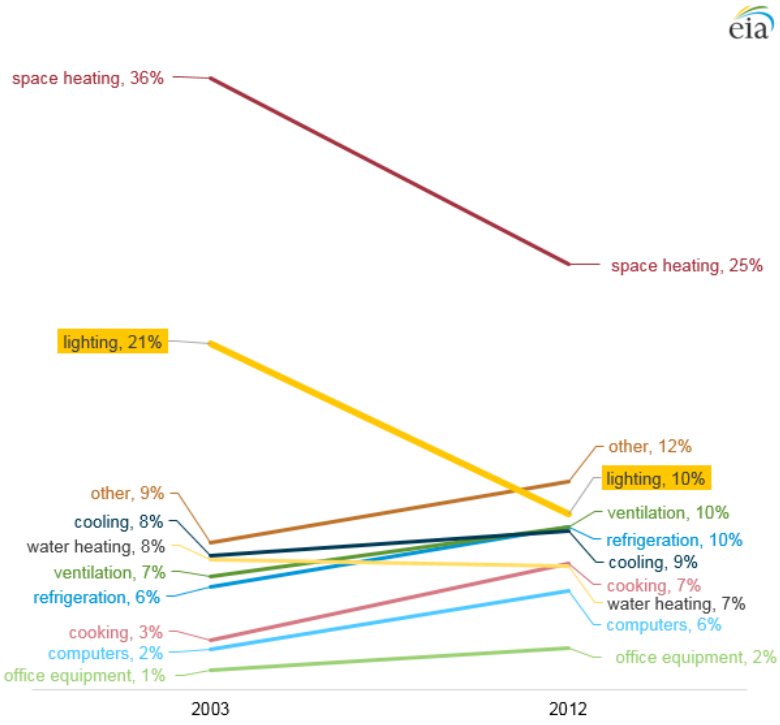



**Figure 1.3:** Energy consumption changes in commercial buildings (U.S. Energy Information Administration, Commercial Buildings Energy Consumption Survey).

## 1.2 Motivation

HVAC systems have been playing an important role in maintaining the indoor environments and have become more complex than before. Ideally, every building has its HVAC system designed and made specifically, and assuming all its components are installed properly. In practice, the real cases are usually not perfect and systems have faults from the very first day of installment. In addition, after years of usage, due to the lack of maintenance, many HVAC systems that seem to work properly have internal faults which results in a large amount of waste in energy. Even a small percentage of waste adds up to a large amount over time and due to the fact that HVAC systems are implemented almost everywhere.

Under normal conditions, people usually do not notice and forget about the quietly working HVAC system. Once people are aware of it is when something feels wrong, which usually indicates some serious problems, and the repair is expensive. This can be avoided by scheduled routine checks and maintenances. The best strategy is to prevent failures by spotting the faults when they are minor and easy to fix. However, maintenance can be expensive as it requires a lot of labor. As the scale of HVAC systems grows larger, the cost and difficulty in maintenance increases significantly, and continuous commissioning by man is no longer a preferred option.

As mentioned, HVAC systems consume a huge amount of energy. Our goal is simple: We want to build a system that can help us save energy by finding faults in HVAC systems to avoid unnecessary energy losses at a low expense. Making our buildings become more sustainable and environmentally friendly. Preventing waste of energy is as important as harvesting sustainable new energy.

Before we look into the wide selection of methods and approaches, we may want to get more familiar with the problem we want to solve by answering the questions:

- *Why is fault detection important?*
  As mentioned, cost is high for commissioning by man. The larger the system the higher the cost for maintenance. The cost would decline if we could detect the faults.

- *Why does fault consume energy?*
  There are various kinds of faults; however, they mean the system is not operating well and all are undesired, e.g., a leak in the air duct or a jammed damper, causing the system to have extra heat loss for a room that needs heating.

- *System is still working but faulty?*
  We can almost be certain to say that all systems have faults. However, HVAC systems are designed to handle extra loadings, and their control systems tend to 'hide' faults while they compensate faults and use up more energy.

One last step before we start our search of possible methods is that we should set up our goals. Yes, we just mentioned and described our motivation; however, 'building

a Fault Detection and Diagnosis (FDD) system to save energy' may be somewhat too general. We would like to be more specific with our goal in order to make sure that our FDD approach is reasonable and fits our purposes. The following four major features are listed as our goals; we would like our FDD system to be:

- **Scalable**
  Applicable to most modern buildings which have installed control systems and sensors.
- **Reliable**
  Delivers credible results with high accuracy.
- **Automatic**
  The system has the ability to detect and diagnose faults by itself - continuous commissioning.
- **Economical**
  Once installed, no expert nor any system development member is needed to maintain the system.

Scalability is something we would prefer; that is, a system can be adapted to various kinds of HVAC systems instead of being model specific. The system should need to be set up only once without too much trouble. Without scalability this would be more like a project focused on one certain building (case by case), and would have to redo everything for each building. Reliability is important to almost anything we would work with. We do not want to receive false alarms (false positive) all the time, nor do we want our system to remain silent and not report errors when faults do exist (false negative). See 1.2.1 for false positives and false negatives. Continuous commissioning is something important that differs new modernized HVAC systems from conventional ones. That is, our FDD system should be able to continuously monitor the HVAC system automatically and report any faulty behaviors without any personnel needed for the task. Once set up, the system should be running on its own. The main purpose of conducting FDD is maintaining HVAC systems in order to save unwanted waste of energy. Whatever we do, it should lower our overall cost, or else it would be meaningless. Building a FDD system will reduce the cost of labor and work for routine maintenance and checks, which are expensive.

### 1.2.1 False positives and false negatives

In statistics, we often set up a test to test whether our null hypothesis $H_0$ is true or false. In hypothesis testing [3], there are two possible errors that may occur; they are named as Type I error and Type II error. Type I error happens when we reject the null hypothesis $H_0$ when $H_0$ is true, while Type II error happens when we fail to reject the null hypothesis $H_0$ when $H_0$ is false. Their relation can be easily understood by

putting them into a table as shown in table 1.1. Type I error and Type II error are also known as false positives and false negatives.

The null hypothesis $H_0$ is determined arbitrarily. Usually people choose the null hypothesis for the case which is more important, causes more serious consequences if incorrectly rejected, easier to approach ... etc. For our purpose here, we choose the null hypothesis $H_0$ as the system is operating faultlessly; since there are almost an infinite number of possible faults that may occur to the system. This fact also means that in order to compute the false negative rate, we have to know the kind of fault we are dealing with and their characteristics before we are able to determine their values. In other words, there is a false negative rate for each kind of possible fault. Figure 1.4 gives us the idea of this concept. For different faults, there are different probability density profiles (blue curve).

| | $H_0$ is true | $H_0$ is false |
|---|---|---|
| Reject $H_0$ | *Type I error* | Correct |
| Fail to reject $H_0$ | Correct | *Type II error* |

**Table 1.1:** Type I & Type II errors.

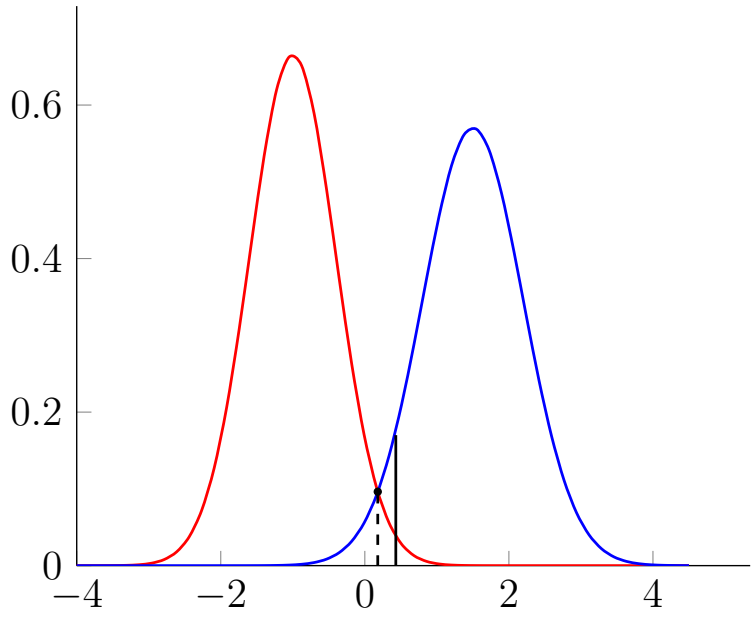



**Figure 1.4:** False positives and false negatives. Assuming the red curve is the density function of a faultless system and the blue curve is the density function of a faulty system. The false positive rate $\alpha$ (Type I error) is the area formed by the red curve, black line, and the x-axis. The false negative rate $\beta$ (Type II error) is the area formed by the blue curve, black line, and the x-axis.

It is impossible to eliminate false positives and false negatives, since they are the

nature of probability. Nevertheless, for a given set of null hypothesis $H_0$ and alternative hypothesis $H_A$, we have a choice to determine the critical/threshold value. It is like determining where the black line (not the dashed line, the dashed line is just an intersection) should be located in figure 1.4. If we want to have a relatively low false positive rate, the trade-off would be an inevitably higher false negative rate, and vice versa. How and where should we draw the line at? This depends on the problem we are dealing with and how we value the consequences if these errors happen.

## 1.3 Fault detection and diagnosis

There are a variety of different Fault Detection and Diagnosis (FDD) methods introduced in numerous publications [4–6]. They provide different views and approaches to the problem. These methods can be classified into three categories: quantitative, qualitative, and history-based methods. Each category can be further classified as shown in figure 1.5.

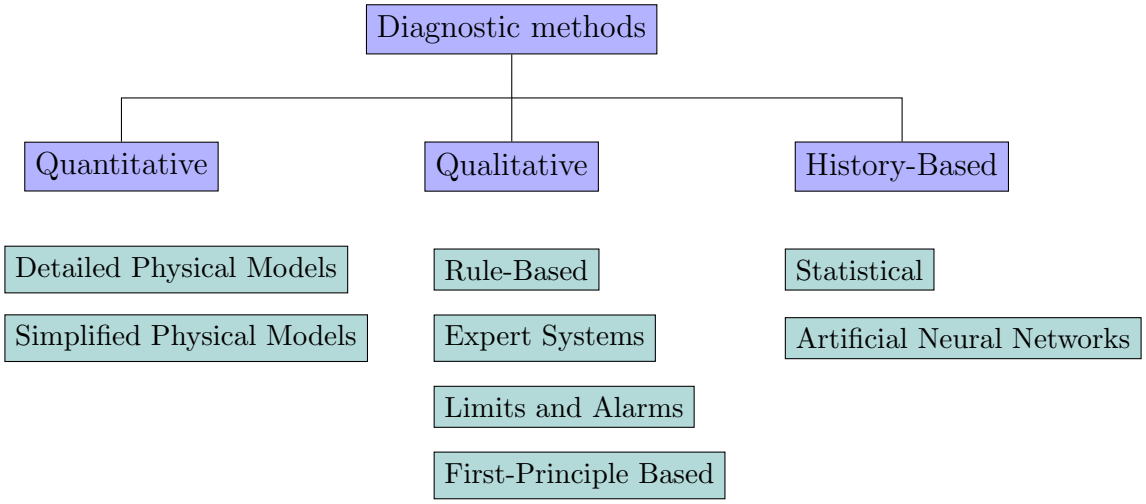



**Figure 1.5:** Fault detection and diagnosis methods.

Figure 1.5 gives us a general idea of the FDD methods, there may be more or a mixture of multiple methods out there. A general overview of FDD methods is discussed in [7, 8]. We should note that all methods are somewhere in the scalability-accuracy spanned space; that is, usually there is a trade-off between scalability and accuracy. This relation is shown in figure 1.6. Note that the dashed line and the shaded area is just drawn to show this concept, the trade-off relation is not necessarily exact like what is drawn.

It is probably safe for us to say that in order to practice FDD, some sort of baseline (reference) is needed. However, how to recognize any patterns or find such a comparable reference is the key. All FDD methods try to make use of the information gathered as much as possible. Some use prior knowledge and/or logic rules while some use historical data. We will do a brief and simple comparison of quantitative, qualitative, and

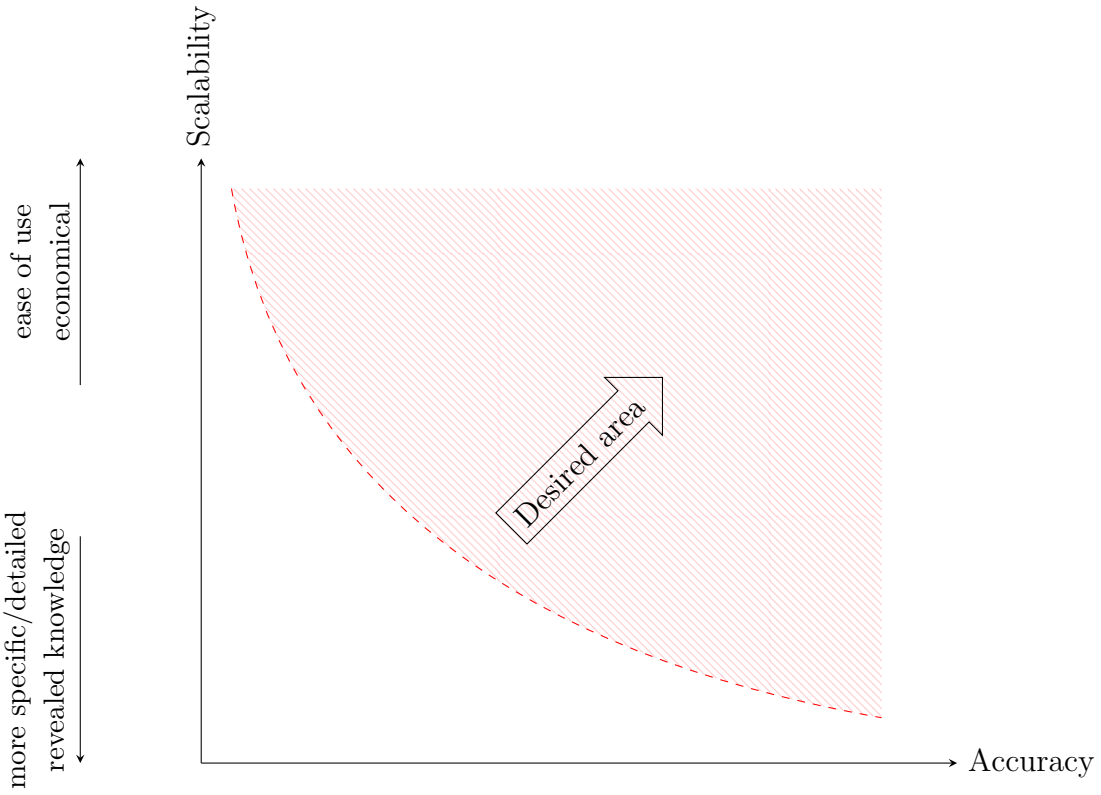



**Figure 1.6:** Showing the trade-off of FDD methods trend conceptually.

history-based methods.

Advantages of quantitative methods are:

- Models are more detailed and therefore provide more overall information.
- Unlike qualitative physical models, detailed data and information are available with more accuracy.
- Unlike history-based models, they don't need historical data.

Quantitative methods usually provide the most information and are tailor-made. This comes with the cost that more computation is needed. Also, such models are more constrained to specific cases and cannot be reused easily, that is, not scalable.

Advantages of qualitative methods are:

- If some prior knowledge is known, they are relatively easier to set up and check. The process is more transparent and less computation is needed.
- Unlike quantitative physical models, detailed set ups are not needed. Is potentially more scalable.

- Unlike history-based models, they don't need historical data.

Generally speaking, qualitative methods are straightforward and obey physical laws. They give us certain rules to follow while keeping the models simple.

Advantages of statistical methods are:

- They are history-based models, if data are handled correctly, the model can learn from the data by itself.
- Unlike quantitative physical models, detailed set ups are not needed. Are potentially more scalable.
- Unlike qualitative models, thresholds can be learned or trained instead of a prior determined one, which may not be suitable. Has potentially better scalability.

In short, statistical methods tend to adapt to various situations. This meets our requirement for the system to be scalable. Also, thresholds are set by the model itself, saving the cost of hiring people to do it.

Here we have given a rough picture of how FDD methods are categorized and their characteristics. For the moment, we will give it a pause. Further discussion of Fault Detection and Diagnosis will be continued in section 3.

## 1.4 Deployment of our FDD approach

Our work is based on HVAC system simulations. Details on building HVAC models are introduced in section 2. In order to achieve scalability and lower the cost, we will be adopting a data-driven approach focusing on control system variables. To our knowledge, HVAC FDD is mostly carried out focusing on thermodynamic/physical variables, neglecting the control system (or at least not shown explicitly). We think that variables of the control system provide many crucial information of the system. Many researches focus on sensor data, which rely on the accuracy and well-being of the sensors. Sensors may be faulty themselves or costly to install (absence of sensor) at times. In comparison, control signals are generated by the controllers/control system; thus, such problems don't exist for control data. Further discussion on control data can be found in section 3.3.

Scalability has often been left out in HVAC FDD. Many FDD methods are model specific and need to be tuned and set up, that is, system specific. Also, they more or less assume there are known faults and somehow know their patterns/characteristics. The system specific set up and prior knowledge make them less scalable. Most control systems make use of PID controllers and on/off switches, and they are somewhat similar in how they behave (universal). Fewer assumptions made the better, only those that are common should be considered.

Due to our focus on scalability and control system data, other benefits such as easy installation and low cost follow along. As mentioned, this approach is not system

specific; therefore, the set up process does not require to know much of the HVAC system. We only need to be able to access data. Because there is no need of tuning involved, nor is the method designed for a specific HVAC system. The cost is cheaper to deploy.

Since we are using simulation data, we have the benefits of producing faultless data, which is not likely to happen in real cases. In addition, we can run experiments on the same models to simulate identical buildings under different conditions, which is not possible for real buildings. If methods work for ideal cases, they are likely to work in the real world. That is, we have controlled experiment conditions.

The FDD approach we have adopted is described in section 3 and our proposed FDD road map is listed in section 3.7. The set up process and experiments are demonstrated in sections 4, 5, and 6. Other than general input weather data and its preprocessing work, we see that having access to the control system data is the only set up needed. Our work shows how a simple FDD approach can be deployed for HVAC systems.

# 2 Building model and data

## 2.1 Building and data access

Before getting our hands dirty and dive into fault detection and diagnosis for HVAC systems and starting to look up what we should do with data, our first question becomes 'Which specific HVAC system are we dealing with?' and 'Where do we find and gain access to data?'

Here comes the problem; one would find that gaining access to HVAC systems and its data is not as simple as just walking into a building and asking for them. Both the building itself and its HVAC system built with it are private properties, and are usually made and commissioned by private companies, not to mention the data. These datasets are not easily accessible and are not opened to the public like highway transportation statistics data. We are simply saying that datasets come with a cost; they are not free. In addition to data access, our goal is to build a FDD system that is scalable. That is, we want our FDD system to be able to handle data from not only one specific building, but most modern commercial buildings. This means we would need to gain access to multiple buildings and their HVAC system data.

Under such circumstances, a reasonable and viable solution is to depend on simulation data. By implementing a simulation model, we would not only be able to collect all the data we need, but also gather data from various buildings and systems with some adjustments to our model. Here we would like to introduce Modelica[1] (see section 2.2), "a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents."

## 2.2 Modelica

Modelica is a programming language widely used for modeling and simulations for engineering systems [9–11]. Among all of the programming languages out there, Modelica is chosen as the preferred language for us to work with due to some of the features that Modelica provides.

First, one of the special features that Modelica differs itself to other programming languages is that it is equation oriented instead of statement oriented. A simple example would help us understand what this means:

$$\begin{cases} x = y + z \\ y = x - z \\ z = x - y \end{cases}$$

If we run the set of equations above in a common programming language (such as Python or Matlab), each equation in the set is considered doing something completely

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

different. However, these three equations are considered the same in Modelica! This is due to the fact that the 'equal' sign, '=,' is actually assigning the value on the right-hand side to the variable on the left-hand side in normal programming environments, while Modelica takes the 'equal' sign to be 'equal' as we use normally for math derivations. If we put in all three equations into our Modelica model, we would be giving redundant information. Any one of them is enough and works the same.

The second main feature is that Modelica handles differential algebraic equations (DAE) for us. This feature simplifies our task of building a simulation model significantly, especially when models grow large and become complex. While one is building a simulation model, one is actually 'describing' a system with math equations based on physical laws. To be more specific, one is using differential equations, algebraic equations, or both (DAE) to model a system. Solving an ordinary differential equation (ODE) or algebraic equation (AE) usually isn't too hard, thanks to the hard work of mathematicians and the software packages being developed. Nevertheless, when it comes to DAEs, things get complicated and difficult. Also, it is quite common that a model is formed of DAEs.

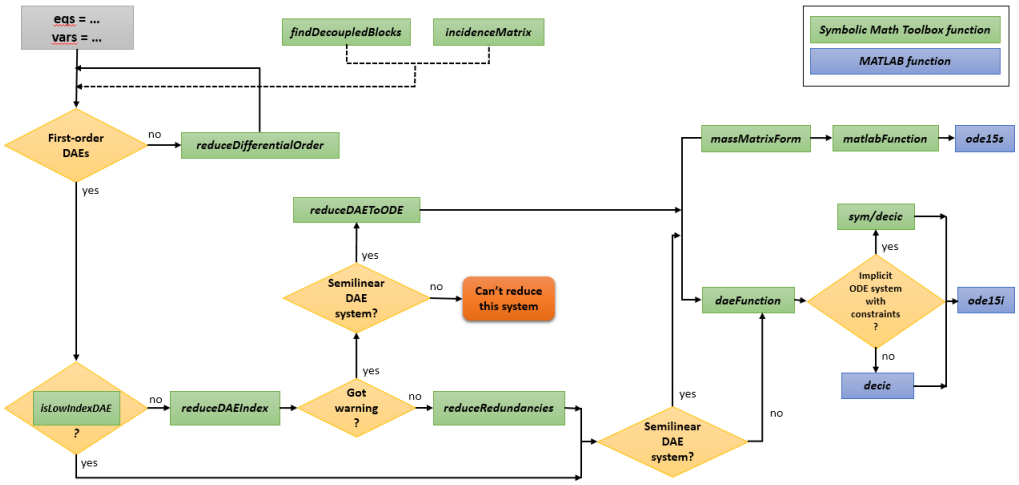



**Figure 2.1:** Solving DAEs in Matlab (DAE flowchart). Source: Matlab.

Solving a set of DAE may not be too big a deal. Take the Matlab working environment for example, first we would have to reduce, decouple, and transform the DAEs into ODEs and AEs. Then we would pass these equations to their corresponding solvers; this is shown in figure 2.1. In other words, we are reformulating the problem into a form that is compatible with our toolboxes and solvers. If the set of DAEs is solvable we would then get the results we want. Suppose we are trying to build a simulation model using Matlab, we could follow the flowchart in figure 2.1 for a simple model. However, when it comes to building a system, the model is usually much larger in scale and has many sub-models included, and for each sub-model there may be some sub-sub-models ... etc. Therefore, as the system grows large and its complexity increases,

doing simulations and solving these DAEs become intimidating. With Modelica, we don't have to worry too much about this problem, as long as we keep our models simple and representative, and follow the general rules of Modelica, it is likely that Modelica will handle the DAEs for us properly[2].

Another feature of Modelica is that it handles both causal and acausal models. We may not be using this feature for our work here, but it is a special feature of Modelica and is worth mentioning.

## 2.3 Our HVAC model

HVAC systems are usually built and commissioned by private companies. They are designed and built specifically for a certain building in order to fit the requirements and needs of its use. This means that every building out there has a different HVAC system. The good news is that the HVAC system is 'tailor-made' for each building, but the bad news is that maintenance will also be 'tailor-made,' in other words, costly.

There are hundreds and thousands of HVAC systems out there. In order to start our work, we will have to start with one model. Hence, we will start with a model which should be as simple as possible to save time and effort, but also complicated enough to reflect the features and characteristics of HVAC systems working in real buildings. We start with a model shown in figure 2.2. In this model, we have a simple HVAC model that has one room, one AHU, one cooling system, one heating system, and a few other components. This simple model contains the basic features of what a HVAC system does, which are heating, cooling, and ventilating.

Now that we have a targeted simple HVAC system model, we will try to build it in the Modelica environment for simulation purpose.

## 2.4 Building our model in Modelica

As mentioned in 2.2, Modelica is a programming language which has some special features that are designed and used for many engineering simulations. There is a list of Modelica simulation environment software packages, such as Dymola and SimulationX, which are commercially available and popular within the community, while OpenModelica and JModelica.org are free to the public [10, 12, 13]. In this thesis, we will be using both OpenModelica and JModlica.org as our working environment. This choice is made due to free accessibility, compatibility, and ease of use.

We start with building simple models in OpenModelica because of its friendly user interface and has OMEdit, a model editor and connector with a graphical user interface (GUI) that comes with OpenModelica. Note that the interface, functions, solvers, and

[2]Modelica is still under development. New releases improve the interface and solvers, fix bugs, add in new features, and so on. With that being said, Modelica still has bugs and compatibility issues. It's best to test out one's model with care and make sure the libraries being used are compatible with each other.

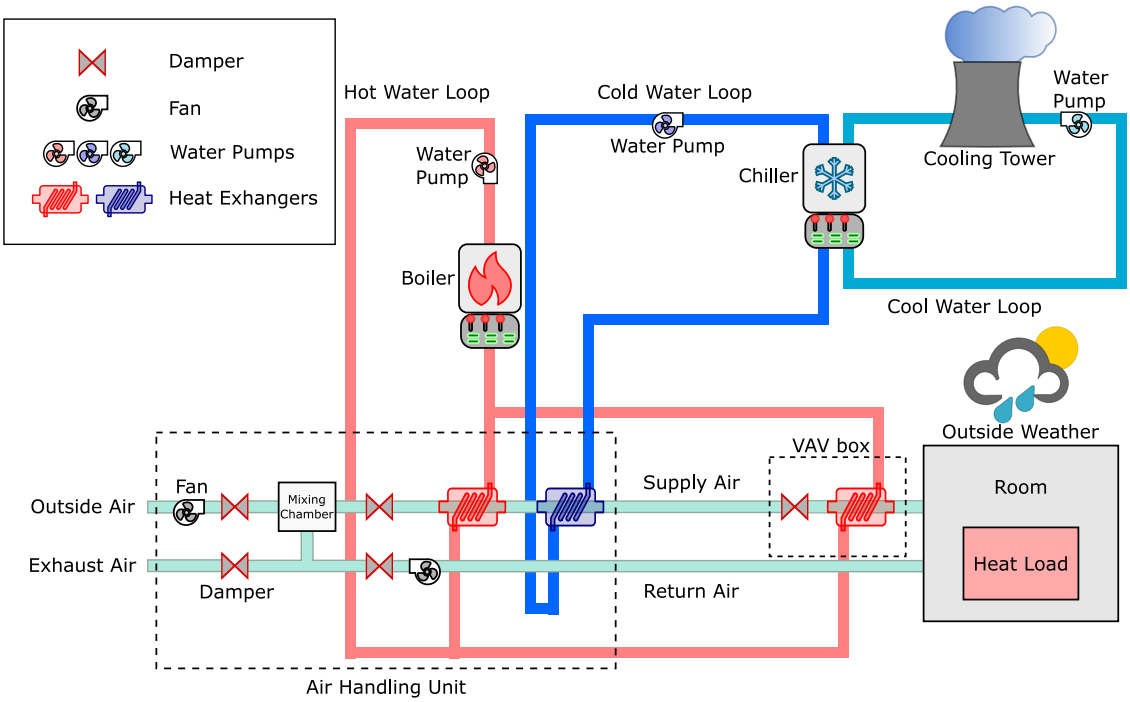



**Figure 2.2:** Our targeted HVAC system model.

features may vary across different Modelica simulation environments; however, they all use the same syntax and support the Modelica Standard Library.

In order to build the HVAC system, we would have to start with the most basic components. The strategy is simply 'divide and conquer,' it would be almost impossible to build a large system without having its components and subsystems first. One would have to build all the components, such as fans, air ducts, dampers, heat exchangers, ... etc (as in figure 2.3), making sure all the linking ports have their properties balanced, media which are used well defined, parameters fine tuned and set, and check if they are compatible with one another. This is a lot of work and very time consuming. Fortunately, people have noticed this problem and have been working on libraries to save work and time by setting standards and making the models reusable.

In practice, it is best for us to take advantage of these libraries. One reason is that people have already spent a huge amount of time and effort to develop and maintain these libraries, saving us from going over all the trouble and work. Another reason is that sticking with these standard libraries, it is more likely the model we build will have better compatibility with other libraries in the future.

In addition to the Modelica Standard Library[3], we will be using the Buildings Library[4] for most of our models. It is being developed and maintained by the Lawrence

[3] *Modelica Standard Library* is the standard and basic library that is free and comes with all Modelica simulation environment pacakges.

[4] A library that is built on top of the Modelica Standard Library. It includes models for HVAC systems, controls, heat transfer, ... etc. See http://simulationresearch.lbl.gov/modelica/ for more

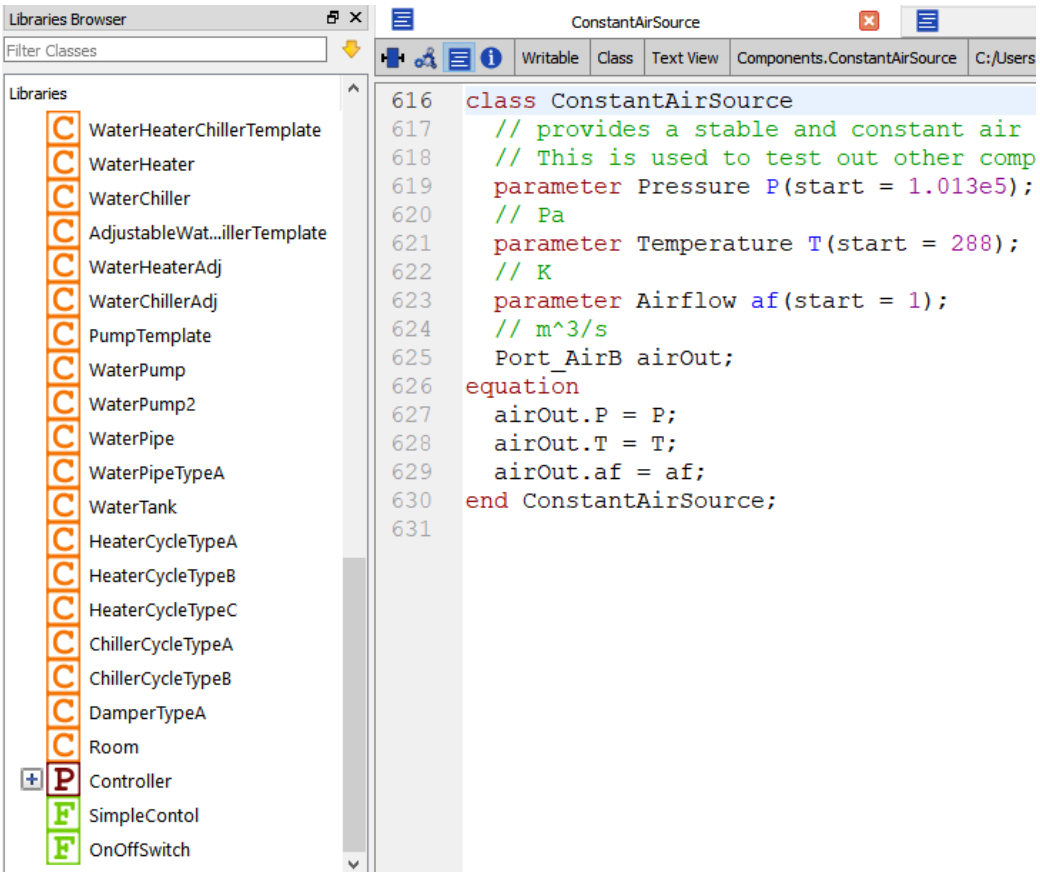



**Figure 2.3:** Building components in OMEdit. Making all of the component models by ourselves is not recommended due to the time and effort spent, and compatibility issues with other libraries. We have made these models for practice and testing purposes. Some of the components are still built ourselves to fit our usage.

Berkeley National Laboratory (LBNL) and funded by the U.S. Department of Energy. We have chosen to build our model based on this library is due to the fact that it contains many component models of HVAC systems. Also, it is compatible with the standard library. However, this library is being developed and tested in Dymola, a commercial software package; therefore, some of its models are not compatible with OpenModelica. Some of the errors are reported, and hopefully the issues will be resolved soon. Most of our work will be using OMedit to build up the model, for it provides a much more user friendly GUI, and run our simulations in JModelica.org environment. We have chosen to run simulations in JModelica.org, this is because that it is much lighter; thus, loads and runs faster. The second reason is because of its compatibility with the Modelica Buildings Library. Another reason is that its UI uses Python; this makes running simulations and exporting data in batches easy.

Although we have these simulation environment software packages and libraries as

details.

useful tools which speed up our efficiency and reduce the amount of work significantly, we still need to put all the models together, test all components, and make sure all the settings are correct [14]. We build our model brick by brick and have resulted with a simulated HVAC system shown in figure 2.4. Some of the subsystems, such as the air handling unit (AHU), cooling and heating system are shown in figures 2.5, 2.6, and 2.7.

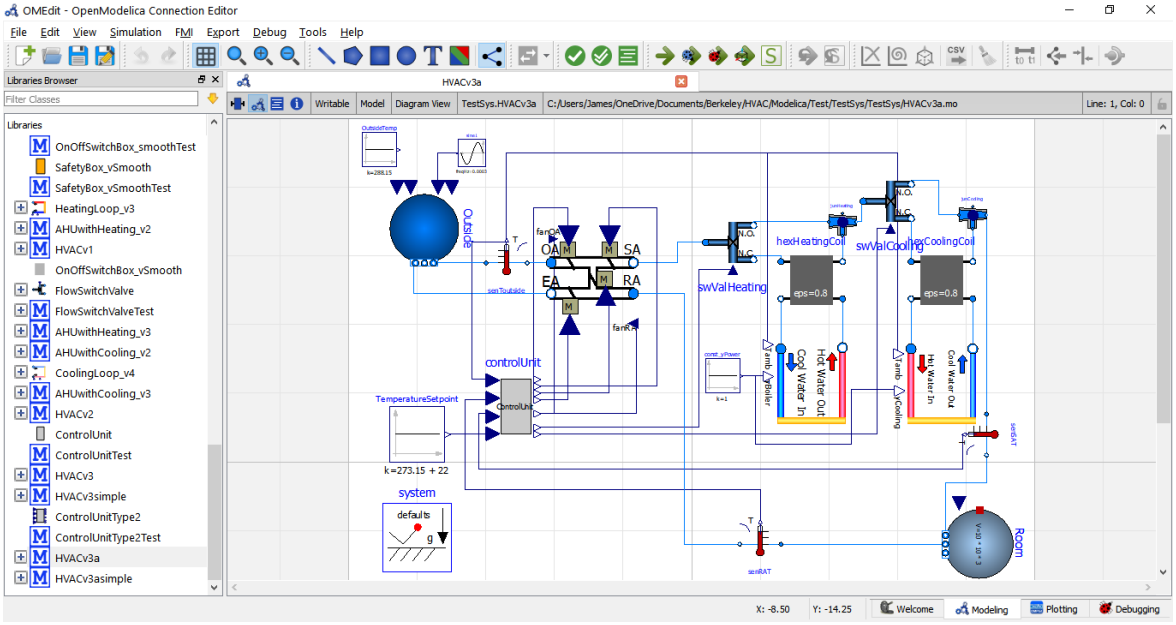



**Figure 2.4:** Our HVAC model in OMEdit.

## 2.5 Model verification and validation

In previous subsections, we have introduced using Modelica briefly as a simulation tool for our later on HVAC FDD work. We are relying on simulation data due to the fact that we do not have access to real commercial buildings and building one (or multiple ones) of our own is way too expensive and impractical. However, while relying on simulation data, we need to be aware of the common pitfalls we may encounter [9], which are: Pygmalion effect, Procrustes effect, and forgetting the model's level of accuracy[5]. We try to avoid falling into these pitfalls by conducting verification and validation tests.

Simulations are done by using mathematical models to describe the physics of a system. Verifications and validations of model reliability are very hard in general; however, there are still ways for us to check how good our model performs to some extent. First, we can try to look into how our system components are modeled. Most

[5]Pygmalion effect means to become too enthusiastic about a model and forget all about the experimental frame. Procrustes effect means forcing reality into the constraints of a model.

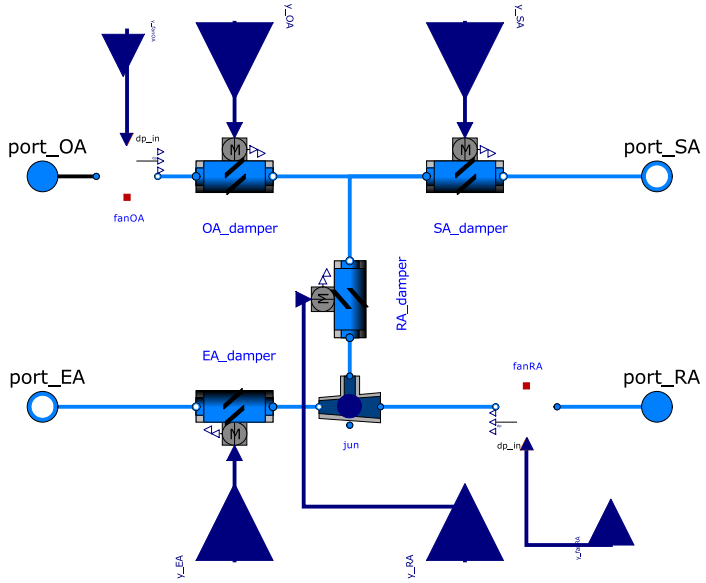



**Figure 2.5:** AHU model in OMEdit.

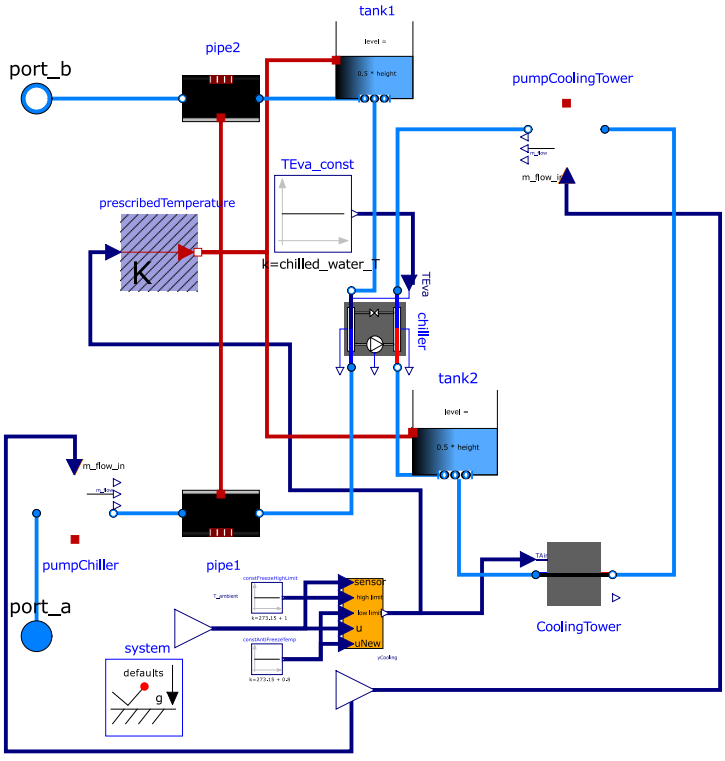



**Figure 2.6:** Our cooling system model in OMEdit.

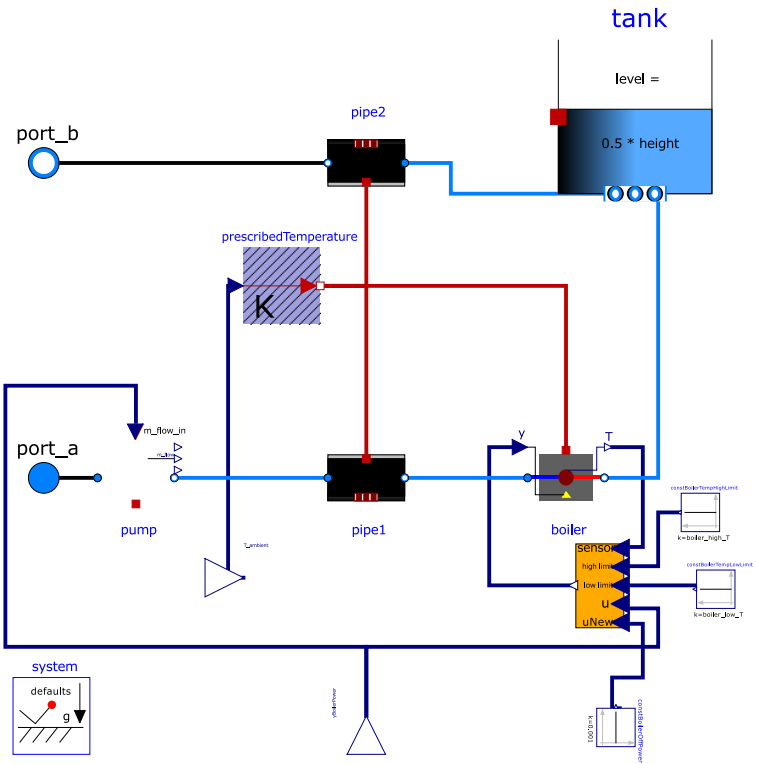



**Figure 2.7:** Our heating system model in OMEdit.

of the components we have used are from the Buildings Library which is built on top of the Modelica Standard Library (MSL). By checking Modelica's official website, we can find a list of publications[6] and different libraries applied for all kinds of engineering areas. As for our task in HVAC systems, a list of publications can also be found in the Buildings Library webpage[7]. This somehow gives us a sign that these libraries do have credibility to some point and people are willing to use them for their work. Nonetheless, we would still have to look into the model structures to be convinced that they are valid. It would be tedious and redundant for us to elaborate all the components used, we will only demonstrate a few examples here. Of course, while we were building the HVAC system model, we tested and looked into the components before putting them together. For example, we would like to put a heat exchanger into our model; we first look into the libraries to find one. Suppose we have picked a constant effectiveness type from the library, in order to understand how it is modeled, it would be best to read the documents first[8]. According to the documents, we see that this model uses the simple heat transfer equation:

$$Q = Q_{max}\varepsilon$$

[6]https://www.modelica.org/publications

[7]http://simulationresearch.lbl.gov/modelica/publications.html

[8]http://simulationresearch.lbl.gov/modelica/releases/latest/help/Buildings.html. For new users, it is also recommended to read the user guide.

where $\varepsilon$ is the constant effectiveness and $Q_{max}$ is the maximum heat that can be transferred. Its Modelica code is shown below:

```
model ConstantEffectiveness "Heat exchanger with constant effectiveness"
  extends Buildings.Fluid.HeatExchangers.BaseClasses.PartialEffectiveness(
    sensibleOnly1 = true,
    sensibleOnly2 = true,
    final prescribedHeatFlowRate1=true,
    final prescribedHeatFlowRate2=true,
    Q1_flow = eps * QMax_flow,
    Q2_flow = -Q1_flow,
    mWat1_flow = 0,
    mWat2_flow = 0);

  parameter Modelica.SIunits.Efficiency eps(max=1) = 0.8
    "Heat exchanger effectiveness";

end ConstantEffectiveness;
```

Due to the simplicity of this model, we believe that most people can understand the code shown. There are different variants of heat exchanger components included in the library, such as `DryEffectivenessNTU`[9], which uses the NTU (Number of Transfer Units) method[10]. If none of the models meet one's needs, one can always build a new model on top of existing models (Modelica is an object-oriented language) or from scratch. Note that the constant effectiveness heat exchanger model shown here is built on top of (extends) another basic model `PartialEffectiveness`, so it is actually more complicated than the code shown here.

The heat exchanger component cannot work on its own; to test it out, we would need to hook it up with other components. The simplest example would be connecting it with flow sources.

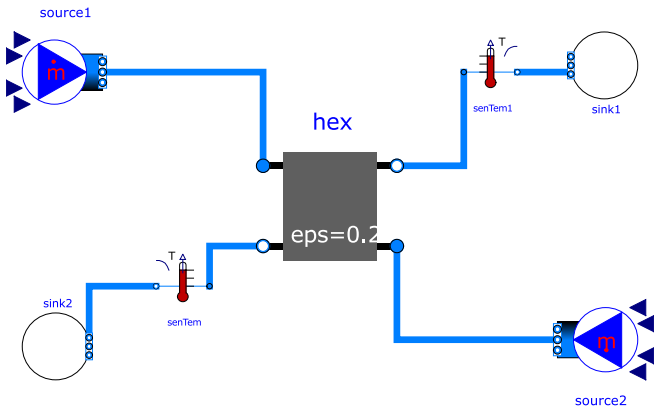



**Figure 2.8:** A simple heat exchanger test example in Modelica.

[9]The Modelica code for this model is much more complicated and would take up too much space to show it here. However, all components can be looked up from the library or documentations.

[10]NTU method can be found in most heat and mass transfer textbooks, e.g., Chapter 13-5 (page 690) of *Heat Transfer: A Practical Approach 2nd Edition by Yunus A. Çengel*. This can also be found on Wikipedia: https://en.wikipedia.org/wiki/NTU_method.

Note that in flow systems, every component, which deals with fluids, must redeclare its `medium` (e.g., air, water, and R134a). Each type of `medium` is modeled in other classes. Take water for example, all its properties are defined and modeled in `package Water` which has more than 1000 lines of code; we will just show the constants in its definition:

```
constant Modelica.Media.Interfaces.Types.TwoPhase.FluidConstants[1]
  waterConstants(
  each chemicalFormula="H2O",
  each structureFormula="H2O",
  each casRegistryNumber="7732-18-5",
  each iupacName="oxidane",
  each molarMass=0.018015268,
  each criticalTemperature=647.096,
  each criticalPressure=22064.0e3,
  each criticalMolarVolume=1/322.0*0.018015268,
  each normalBoilingPoint=373.124,
  each meltingPoint=273.15,
  each triplePointTemperature=273.16,
  each triplePointPressure=611.657,
  each acentricFactor=0.344,
  each dipoleMoment=1.8,
  each hasCriticalData=true);
```

In figure 2.8, icons at the upper left and lower right are another type of component. They are fixed mass flow sources. The lower left and upper right components are fixed boundary (flow) sources. The gray box is a constant effectiveness heat exchanger we have just mentioned. Two flow thermometers are added to the exits of the heat exchanger in order to monitor output flow temperatures. Depending on the design, each component will require different inputs; we will have to assign them carefully or errors would occur (Errors incurred by missing values forgotten to fill in happen quite often. Unrealistic inputs usually lead to computational errors as well.).

By checking with the physical models of the components, we can verify that the physics seem to be reasonable and consistent with what is written in textbooks. However, we would also need to be cautious while putting components together and building a bigger component or system. After verifying the physics contained within the components, input values would be next for us to verify.

Input values are mostly determined by looking up handbooks [15–17] or online resources[11]. Take air flow for example, according to *Ventilation for acceptable indoor air quality ANSI/ASHRAE Standard 62.1*, for office areas, the minimum ventilation air flow rate would be 8.5[L/s] per person. Let's assume an office room has a dimension of $10 \times 10 \times 3$ (length, width, and height in meters), and a typical office cubicle usually has a size of $6' \times 8'$ or $8' \times 8'$ (in feet). By doing the math, we would result in an estimation of 20-30 people in this office room. We would tend to be on the conservative side, so we will end up with a minimum air ventilation of $8.5 \times 30 = 255$[L/s]. This tells us we would need to set our air flow rate to be at least 255[L/s]. Nevertheless, this is only considering air ventilation, our HVAC system should also be able to remove heat

[11] e.g., https://www.engineeringtoolbox.com/

dissipated from human bodies[12] and other equipment or facilities within the room (e.g., lights, computers, ... etc). If a centralized HVAC system is used (such as the ones we used in section 5), we know that the supplied air is usually set to 13°C for centralized systems, and 20 BTU (British Thermal Unit, $\sim$ 1055 Joules) per hour for each square foot is needed in general[13]. We can then get a rough estimation of the heating and cooling power by carrying on this kind of estimation with the aid of handbooks and online resources. Some simulation tests are run to verify our input values are valid or reasonable by checking if the system variables respond within a desirable range (within a magnitude of order).

Another quick check is by looking at the simulation result plots. We will show this by adjusting our example shown in figure 2.8 a little, because the model in figure 2.8 has two prescribed temperature and mass flow sources of water flows, we would expect a steady state output result, which is a little boring and not showing the dynamics of our heat exchanger. To make things more interesting, we will test the heat exchanger with a model shown below in figure 2.9.

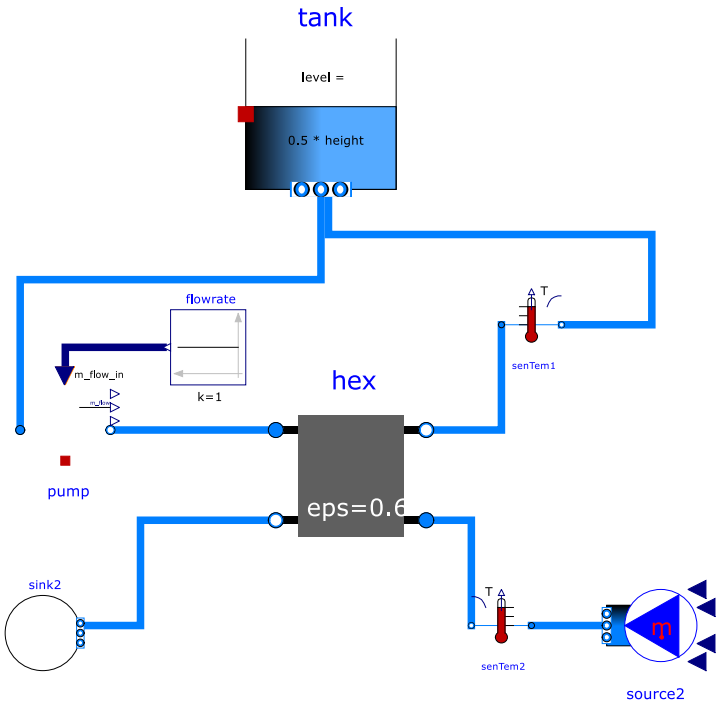



**Figure 2.9:** A heat exchanger test example with a water tank in Modelica.

We replace the water source on the top with a water tank and a water pump. In order to verify this model quantitatively, we will now assign parameter values to our components in this model. Our water tank will have a height of 2$[m]$ and an area of

[12]Usually close to 100 Watts and could go up to around 120 Watts. This can be easily estimated by calculating the overall calories a normal person intakes a day.

[13]https://www.energy.gov/energysaver/room-air-conditioners

$2[m^2]$. The water level will be set to half of the height, which is $1[m]$. So we will have a total of $2[m^3]$ water, which is $2000[kg]$ (we are approximating the water density to $1000[kg/m^3]$) of water in the upper loop. The pump[14] is set to provide a constant mass flow at $1[kg/s]$ using the `flowrate`[15] block to provide a control signal. We have chosen a constant effectiveness type heat exchanger and set its constant effectiveness $\varepsilon$ to 0.6. Water source at the lower right corner provides a $3[kg/s]$ constant water flow at 80[°C], and our initial water temperature in the water tank is set to 20[°C]. The lower left is just a water sink and we don't have to worry about it. After running the simulations, we get the water temperature profile shown in figure 2.10.

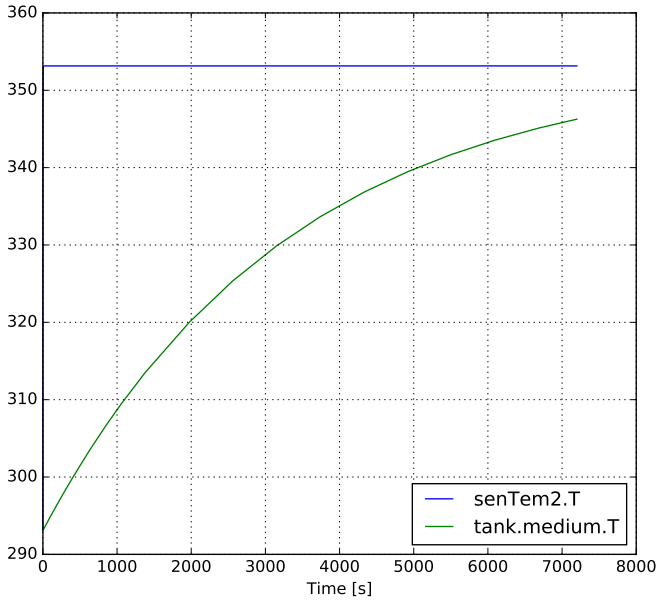



**Figure 2.10:** The simulation results of our water tank temperature in Modelica (using JModelica with simulation time of 7200 seconds). The y-axis is water temperature in Kelvins[K].

The Modelica code for this model looks like:

```
model HeatExchangerTest2
  package water = Buildings.Media.Water;
  Buildings.Fluid.HeatExchangers.ConstantEffectiveness hex(redeclare
      package Medium1 = water, redeclare package Medium2 = water,
      dp1_nominal = 100, dp2_nominal = 100, eps = 0.6, m1_flow_nominal = 1,
      m2_flow_nominal = 3);
  Buildings.Fluid.Sources.MassFlowSource_T source2(redeclare package Medium
       = water, T = 273.15 + 80, m_flow = 3, nPorts = 1);
  Buildings.Fluid.Sources.FixedBoundary sink2(redeclare package Medium =
      water, nPorts = 1);
```

[14]The pump component is also in the Buildings Library.

[15]This is a constant source signal block component. It can be found in the Modelica Standard Library.

```
  Buildings.Fluid.Sensors.TemperatureTwoPort senTem1(redeclare package
      Medium = water, m_flow_nominal = 1);
  Buildings.Fluid.Sensors.TemperatureTwoPort senTem2(redeclare package
      Medium = water, m_flow_nominal = 3);
  Modelica.Fluid.Vessels.OpenTank tank(redeclare package Medium =
      water,crossArea = 2, height = 2,nPorts = 2, use_portsData = false);
  Buildings.Fluid.Movers.FlowControlled_m_flow pump(redeclare package
      Medium = water,dp(start = 100), m_flow(start = 1), m_flow_nominal = 1)
      ;
  Modelica.Blocks.Sources.Constant flowrate(k = 1);
equation
  connect(senTem1.port_b, tank.ports[2]);
  connect(tank.ports[1], pump.port_a);
  connect(hex.port_b2, sink2.ports[1]);
  connect(senTem2.port_b, hex.port_a2);
  connect(source2.ports[1], senTem2.port_a);
  connect(flowrate.y, pump.m_flow_in);
  connect(pump.port_b, hex.port_a1);
  connect(hex.port_b1, senTem1.port_a);
end HeatExchangerTest2;
```

In order to verify this result, we can solve this problem analytically. Assuming the water temperature in the tank is $T_1$ and mass flow is $\dot{m}_1$. The hot water has a temperature $T_2$ and a mass flow $\dot{m}_2$. Water heat capacity is denoted as $c_p$. We start with calculating the maximum possible heat transfer rate $\dot{Q}_{\max}$:

$$\begin{aligned} & \dot{Q}_{\max} = \dot{Q}_{1\,\max} = \dot{Q}_{2\,\max} \\ \Rightarrow \quad & \dot{m}_1 c_p (T - T_1) = \dot{m}_2 c_p (T_2 - T) \end{aligned}$$

we then get the output temperature $T$ as

$$T = \frac{1}{\dot{m}_1 + \dot{m}_2}(\dot{m}_1 T_1 + \dot{m}_2 T_2).$$

We now can use the relation $Q = Q_{\max}\varepsilon$ given by the constant effectiveness heat exchanger:

$$\begin{aligned} \dot{Q} = \dot{Q}_{\max}\varepsilon &= \dot{m}_1 c_p (T - T_1)\varepsilon \\ &= \varepsilon \dot{m}_1 c_p \left( \frac{\dot{m}_2}{\dot{m}_1 + \dot{m}_2} T_2 - \frac{\dot{m}_2}{\dot{m}_1 + \dot{m}_2} T_1 \right) \\ &= \varepsilon \frac{\dot{m}_1 \dot{m}_2}{\dot{m}_1 + \dot{m}_2} c_p (T_2 - T_1) \end{aligned}$$

this heat flow will then result in heating up the water in the water tank; thus we can write:

$$\dot{Q}_1 = M c_p \frac{dT_1}{dt} = \varepsilon \frac{\dot{m}_1 \dot{m}_2}{\dot{m}_1 + \dot{m}_2} c_p (T_2 - T_1)$$

where $M$ stands for the total mass of water in the water tank. This is a first order ODE (Ordinary Differential Equation) and can be solved easily with the use of change

of variables by letting $\Theta = T_2 - T_1$, our equation becomes:

$$\frac{dT_1}{dt} = \varepsilon \frac{\dot{m}_1 \dot{m}_2}{(\dot{m}_1 + \dot{m}_2)M}(T_2 - T_1)$$

$$\Rightarrow \quad \frac{d\Theta}{dt} = -\varepsilon \frac{\dot{m}_1 \dot{m}_2}{(\dot{m}_1 + \dot{m}_2)M}\Theta$$

solving for this:

$$\Theta = Ae^{-Bt}$$
$$\dot{\Theta} = -ABe^{-Bt}$$

plug these two back to the ODE:

$$-ABe^{-Bt} + A\varepsilon \frac{\dot{m}_1 \dot{m}_2}{(\dot{m}_1 + \dot{m}_2)M} e^{-Bt} = 0$$
$$\Rightarrow \quad B = \varepsilon \frac{\dot{m}_1 \dot{m}_2}{(\dot{m}_1 + \dot{m}_2)M}$$

making use of the initial condition that $T_1(t = 0) = T_{1,\text{init}}$; hence, we have

$$\Theta(t = 0) = T_2 - T_{1,\text{init}}$$
$$\Rightarrow \quad \Theta = (T_2 - T_{1,\text{init}}) \exp\left(-\varepsilon \frac{\dot{m}_1 \dot{m}_2}{(\dot{m}_1 + \dot{m}_2)M} t\right)$$

replacing $\Theta$ with $T_2 - T_1$, we finally get:

$$T_1 = T_2 - (T_2 - T_{1,\text{init}}) \exp\left(-\varepsilon \frac{\dot{m}_1 \dot{m}_2}{(\dot{m}_1 + \dot{m}_2)M} t\right)$$

To show the result quantitatively, we just have to plug in the values as in our set up:

$$\begin{aligned} T_{1,\text{init}} &= 20 \\ T_2 &= 80 \\ \varepsilon &= 0.6 \\ \dot{m}_1 &= 1 \\ \dot{m}_2 &= 3 \\ M &= 2000 \end{aligned}$$

We use a Python script to help us plot out this analytical solution:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 N = 101 # number of computational points
5 eps = 0.6 # heat exchanger effectiveness
```

```
m1_dot = 1 # m1 flow rate
m2_dot = 3 # m2 flow rate
Cp = 4 # heat capacity, [kJ/(kg K)]
M = 2000 # tank water mass
T1_init = 20 # ambient temperature
T2 = 80 # hot water temperature

t = np.linspace(0,7200,N) # time
c = eps*m1_dot*m2_dot/(m1_dot+m2_dot)/M # exponential constant

T1 = T1_init*np.ones(N,)
T2 = T2*np.ones(N,)

T1 = T2 - (T2-T1)*np.exp(-c*t)

# plot
plt.plot(t,T1,label='T1')
plt.plot(t,T2,label='T2')
plt.xlabel('time[sec]')
plt.ylabel('Temperature[degC]')
plt.legend()
plt.show()
```

and the plot is shown in figure 2.11. We see that figure 2.11 shows the same trend as figure 2.10. However, we have made some approximations, such as assuming water heat capacity is a constant with respect to temperature in this temperature range and approximating the density to be constant at $1000[kg/m^3]$ (which both are not true), while the Modelica Standard Library has a more sophisticated water model. Therefore, the results are still slightly different.

The components can be easily modified as one wishes, or one can simply replace a component with another one, as long as the connecting ports and basic parameter settings are compatible. For example, we can simply swap out the constant effectiveness type heat exchanger to a dry effective NTU type from the library.

As the system model grows larger, it becomes hard and impractical to keep track of all outputs. It is best to check small components at early stages before putting them together. However, we could check how our model behaves when there are small changes in the parameter values. As stated in [9]:

*if the simulated behavior is not very sensitive to small variations in the model parameters, there is a good chance that the model fairly accurately reflects the behavior of the real system.*

This happens to be true with our HVAC model. Number values are first checked with handbooks or online, then small experiment tests on components are carried out, and some quick estimation is done. After filling in the parameters, we check different values which are within an order of magnitude range and see how the system behaves. Most of the time our models perform as expected and usually we tend to put

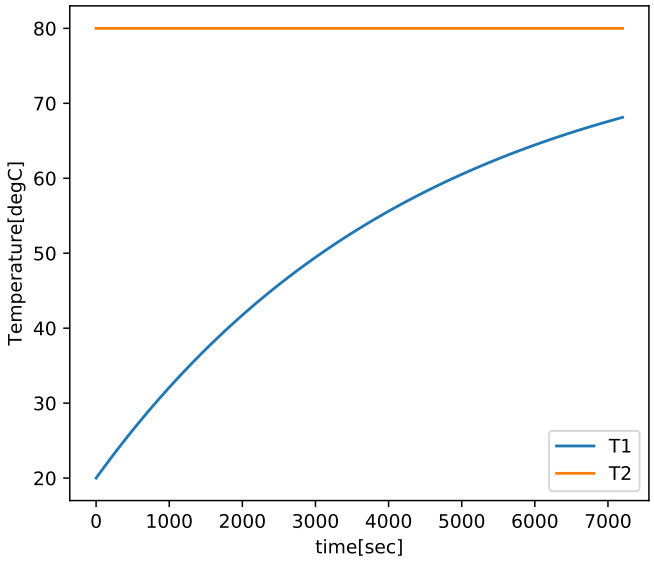



**Figure 2.11:** Water temperature (water tank) profile verification of our analytical solution.

in a parameter that is more conservative, which means a system with more reserved capacity. The models we have built are still considered simple; however, our goal is to have data access to HVAC systems which are meaningful; hence we tend to keep things simple for us to finish our work within reasonable time.

One more thing to note is that in the libraries we have used, there usually are user guides, examples, and validation tests for the components in the package. For instance, [18] explains how the fan and pump models are built. Users can assign different fan curves if needed; however, for our case, we will stick to the defaults for building a sophisticated model is not our goal.

# 3 Fault detection and diagnosis

In section 1.3 we have given a brief introduction to FDD methods and how they differ. In order to meet our requirements in section 1.2, history-based models would be favorable to us.

Modernly built commercial buildings with modern HVAC systems installed are equipped with sensors to monitor how the system is functioning. If we have access to the systems, we have access to setpoints and sensor data. In section 2 we have already dealt with this problem.

In the past, FDD is often approached by using model based methods; that is, people build a physical or simulation model of the system and compare how the model differs with the system behaviors. Thanks to the development and popularity growth of the topic, 'Machine Learning,' data-driven approaches have started to catch people's attention.

In the following subsections, we will present a brief introduction to the set up of our work (sections 3.1, 3.2, and 3.3), then an introduction to the data analysis tools we will be using (sections 3.4, 3.5, and 3.6), and finally our road map to our FDD approach (section 3.7).

## 3.1 Data-driven FDD approach

In recent years, due to the improvement in computer processing power and statistical tools, machine learning has proven to be very powerful and useful for applications in many areas. However, these statistical tools are not a panacea for all problems. The famous quote from George Box, '*All models are wrong but some are useful*,' is telling us in some way that statistical models are not 'plug and play,' we cannot simply feed all the data we gathered into some simple model and expect the results to be satisfying. Models are useful only when our problem fits with the assumptions of the models in the first place. Our challenge here is how and what to do with the data (see section 2) we have. With an abundant collection of statistical models and tools out there, which do we choose and how do we implement them?

For data-driven HVAC FDD, in order to explain the simple idea, we can basically break it down into two parts. For the first part, the idea is, given some input data, we build some model[16] to generate some measure, which is simply some output value(s). The other part is using the input data to establish some threshold or a detection criterion. We then compare the measure with the threshold to determine whether the system is faulty or not. This is simply shown in figure 3.1.

Usually, people try to model the input data with some mathematical or statistical model, such as HMM (Hidden Markov Model) [19], Bayesian networks [20, 21], ARMA models [22], GP (Gaussian Process) [23], FCD (Fractal Correlation Dimension) [24] ... etc., and then along with some threshold/detection criterion set up, depending on

[16]The word *model* here means a mathematical or statistical model; we are not referring to the HVAC model, such as the one we mentioned in section 2.

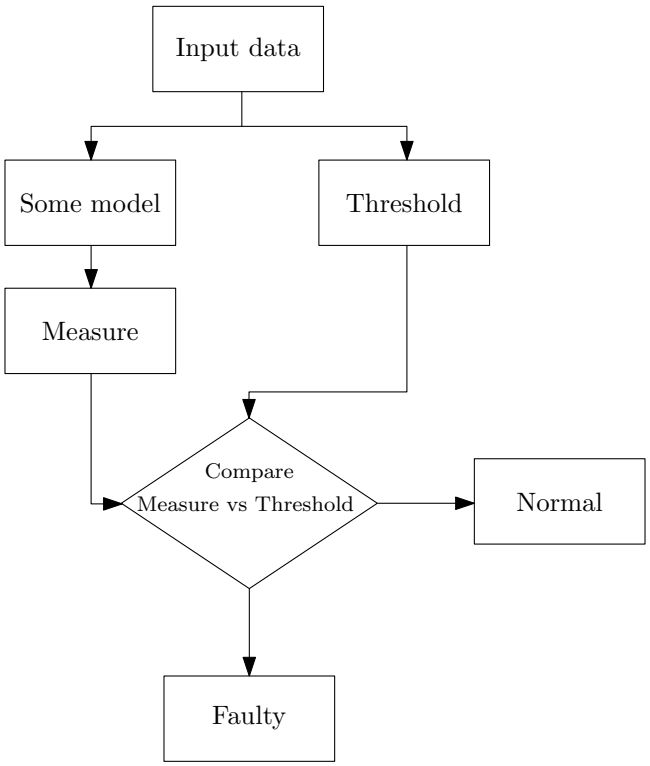



**Figure 3.1:** Basic idea of a fault detection and diagnosis approach.

the model formulation. Common ones are PCA (Principal Component Analysis) [25–31], SVM (Support Vector Machine) [32–35], OC-SVM (One Class SVM) [23, 36], ANN (Artificial Neural Networks) with fuzzy logic [37–41], ... etc. Moreover, in order not to fall into the 'data rich, information poor' situation, dimensionality reduction methods such as PCA, PLS (Partial Least Squares) [30], mRMR (minimal-redundancy-maximal-relevance) [32, 42, 43], and so on are adopted. Dimensionality reduction not only shrinks down our datasets to prevent them from bogging down our computing machines, but also specifies the hidden features of the data.

## 3.2 Time-series data

The data we collect are from a simulation model (or a real HVAC system). The data is in a sequence form; that is, for each variable $x_t$ at each time step $t$, $x_t$ has some real number value. Quoting from Wikipedia[17]:

*A time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data.*

[17]https://en.wikipedia.org/wiki/Time_series

The difference between time-series data and non time-series data is that time-series has a 'time order.' This time order feature of time-series turns the data into some high dimension form (a time-series with $N$ data points can be viewed as a vector of length $N$, that is $x \in \mathbb{R}^N$) and makes it much more difficult to handle.

In section 3.1 we have mentioned many methods how people dealt with HVAC FDD. The most well-known time-series model is probably the ARMA (Auto Regression Moving Average) [44] model. Many analysis tools such as Fourier transform, wavelet transform, and their variants are also widely used. Many other methods and tools that are not specifically designated for time-series are also commonly used, for example, PCA, SVM, HMM, SSM (State Space Models), fuzzy logic [45], Bayesian network, GP, ... etc.

One would find that time-series analysis is a big topic with numerous methods and approaches. There are even researches regarding comparing multivariate time-series with different dimensions such as [46]. Before we dive into the time-series analysis area and try to do some analyses on our data by picking a mathematical model, we should be considering what we want to achieve by inspecting the data.

In HVAC systems, there are all kinds of time-series data. Sensor data, such as, room temperature, air flow rate, motor speed, damper positions, and so on; control signal data, cooling and heating controls, for example; weather data, including temperature, wind speed, pressure, humidity, ... etc.

## 3.3 Control data

In section 1.2, we have mentioned that we want some FDD to be scalable, reliable, automatic, and economical. In short, we would like the overall cost of FDD to be low. We try to achieve this by looking for possible ways to improve HVAC FDD.

To lower the cost of FDD, we would want it to be scalable; this means that detailed modeling of a specific HVAC system would not be our choice (Though very accurate, it is very expensive as well). We would like something that is general to most, if not all, HVAC systems. HVAC systems installed in buildings are all different, but they all do one same thing, that is, to provide a controlled indoor environment, including thermal comfort and air quality. This is done by controlling temperature, pressure, humidity, and ventilation ($CO_2$ concentration level). The components in HVAC systems may be different in design, models, numbers, performance, ... etc; nevertheless, they all work similarly; the indoor environment is simply controlled by heating, cooling, and air ventilation with controllers. What all HVAC systems have in common are the controllers. Although the control system in HVAC systems varies, the basic control components are generally the same, either an on/off switch or a PID controller[18] is used. Other types of switches are rare; even if they are used, on/off switches and PID controllers are likely to be used in the system as well. Moreover, for a working HVAC

[18]PID controllers are used due to the fact that they are simple, easy to tune, and work fairly well in almost all applications. More complicated designs for controllers may work better for some cases; however, the benefits gained usually don't justify the higher cost and requirement for more training.

system, we suppose all the design and set up have been carefully engineered with a lot of work and effort put into it. Hence, a working HVAC system means its control system works as well. Having this in mind, and since the control system collects all the input data and outputs all the control signals (manipulated variables) after running computations over some rules, we claim that the control system data holds important information of the overall HVAC system. Figure 3.2 shows a simple block diagram of a control loop.

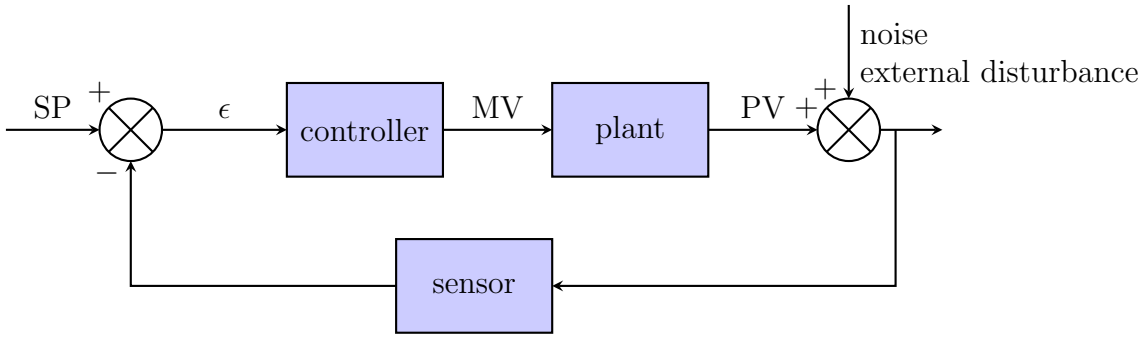



**Figure 3.2:** Block diagram of a control loop.

In this figure, there are basically only five variables: SP (setpoint), $\epsilon$ (error), MV (manipulated variable), PV (process variable), and sensor data. If we put aside noise for the moment, there are only SP, $\epsilon$, MV and PV (same as sensor data) four variables. In the controller's point of view, these four variables are the only information needed to make the HVAC system work. No matter how large the system grows or how complicated it becomes, how the control system works is the same.

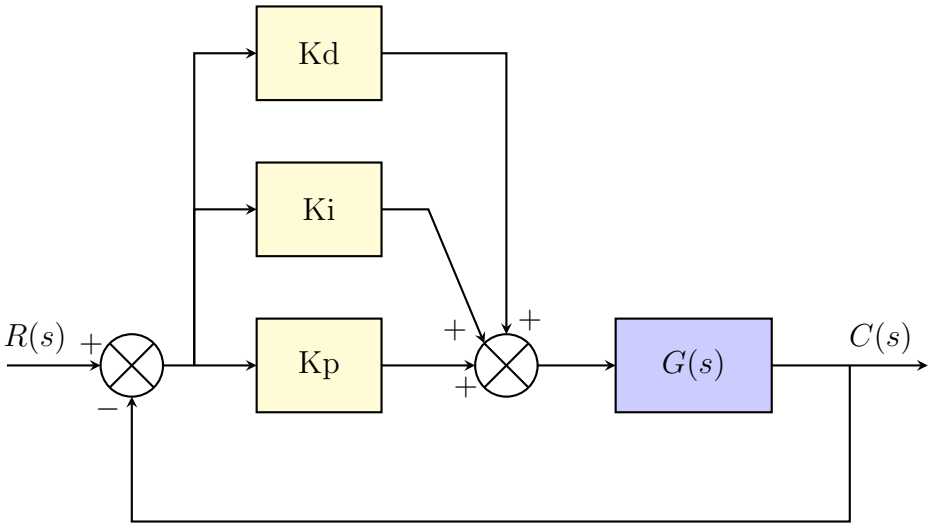



**Figure 3.3:** Block diagram of a PID controller.

The controller in figure 3.2 is either an on/off switch or a PID controller (for most cases if not all). A PID controller block diagram is shown in figure 3.3.

However, an HVAC system is not closed, it is not isolated from the outside environment. External data (including weather data, noise, and heat load) comes in and interacts with the system, so now an external variable label should be located where the 'noise' label is in figure 3.2.

Most HVAC FDD approaches found in publications monitor the external variables and sensor data closely. People try to find their relationships and patterns with all kinds of mathematical/statistical models. We here suggest that instead of inspecting external variables and sensor data alone, we should focus more on the manipulated variable data. There are a few reasons why we should focus on the control data. First, every HVAC system has controls; though they may have some intricate designs and/or complicated connections, they are basically comprised of on/off switches and PID controllers. The overall picture of this concept is again shown in figure 3.2. Figure 3.4 depicts the concept of control loops in the HVAC system.

Second, solely depending on weather and sensor data, one could encounter sensor failure and/or inaccuracy problems. That is, an anomaly data point could be caused by a faulty component, a failing sensor, or both. If control data is not included, it would be difficult to tell if the datasets we collected are reliable. Research regarding sensor failures, such as [47], requires additional system-specific information, which we do not prefer due to the scalability trade-off.

Another reason why we should focus on control data is that we are targeting hidden faults of an HVAC system. It has been mentioned a few times in papers that hidden faults [29] is something hard to deal with, but to our knowledge, so far no one has really explicitly stated how hidden faults are handled[19]. We believe this is because that control systems hide faults. This is the nature of control systems, for they are designed to meet the requirements of user settings by making sure the process variables are controlled and maintained at setpoint values. For example, a heater is controlled by a thermostat to keep the temperature in a room warm during winter. Suppose the fan has been clogged by dust causing the air flow to decrease; thus, leading to a decreased heating performance. The thermostat then senses the temperature is lower than expected; therefore keeping the heater on for longer periods to compensate the lost heating performance. This means that the control system is making the overall system to work harder to ensure the process variables are in the desired range. HVAC systems are usually designed to have some reserve capacity in order to take care of different loadings. When faults, which are not serious, occur, they are being hidden by the control system and users won't notice any difference; hence, no one would report or complain about them. However, energy is still being wasted; faulty components may get worse and fail eventually in the long run, which could be very expensive to repair. When it comes to hidden faults, monitoring external and sensor data alone would not be as effective as monitoring MVs (control data), for the controllers have a nature

[19]Most papers don't even mention about hidden faults; that is, a distinction of faults (hidden or not hidden) probably isn't made. In addition, some approaches are implicitly assuming some knowledge of faults are known (supervised learning) while some don't (unsupervised learning). This can usually be distinguished by looking at the machine learning algorithms used.

of hiding faults, and inspecting their behaviors directly[20] makes more sense. Sensor data may be able to catch some anomaly behavior for a short period of time before the controllers compensate it. On the other hand, MVs will continuously reflect the controllers' attempts to hide faults, making MVs a better indicator to hidden faults.

Finally, with MV data (control data) available, it should help us differentiate the anomaly behaviors with system transient states. All machines and systems we build have different operating modes; even the most simple ones have at least two states, on and off. For instance, the state of a heater is on while the temperature of a room is below the setpoint, and the state is off if the temperature is higher than the setpoint. When the scale of a system grows larger, it's likely that there are multiple operating modes, which may be a combination of different states of its subsystems and/or components. Take a simple HVAC system for example; imagine two different scenarios for the exact same building. The first scenario is during a hot sunny summer day; without a doubt, the building gets hot inside and the HVAC system tries to cool the interior by turning on the cooling subsystem. The second scenario is people holding a big event in the building during fall season; the outdoor weather is cool and comfortable; however, the indoors get hot and stuffy due to the large number of people attending the event, causing a lot of heat and rising $CO_2$ levels. The HVAC system responds to this situation by turning on both the cooling and ventilation (fans and dampers) subsystems. In this example, the HVAC system starts cooling the building in both cases, but only increases the ventilation for the second case. This is because for the first case, the $CO_2$ level is normal and further ventilation is not needed; increasing hot outside air intake will force the cooling system to work harder and increase power usage. As for the second case, outside air is cool and indoor $CO_2$ level is relatively higher; increasing the ventilation by letting more outside air in would not only help air ventilation, but also reduce the work load of the cooling system. From this example, we see that though the HVAC system is cooling the building for indoors temperature which is higher than the setpoint, it may be working in different operating modes. If we only monitor external and sensor data, one major challenge would be how to determine which operating mode the system is in. We could guess by historical training data; still, it may be vague at times and we could guess wrong, especially when there are multiple operating modes; thus, reporting false positive faults. Another problem would be the transient states between two operating modes. These data points don't belong to either operating modes and could look very similar to faulty behaviors. If not taken with special care, these data points are often being marked as anomaly points, leading to false positives, or even causing trouble to train the FDD at the beginning.

Due to the reasons stated and discussed above, we propose that including and focusing on control data could be beneficial to our FDD task.

[20] Let's say some component (plant in figure 3.2, or the HVAC system in the dashed box of figure 3.4) has a glitch and is not functioning normally. External data should be independent of the HVAC system and sensor data is the result of external data interacting with the HVAC system, making it to be indirectly related and less representative to hidden faults.

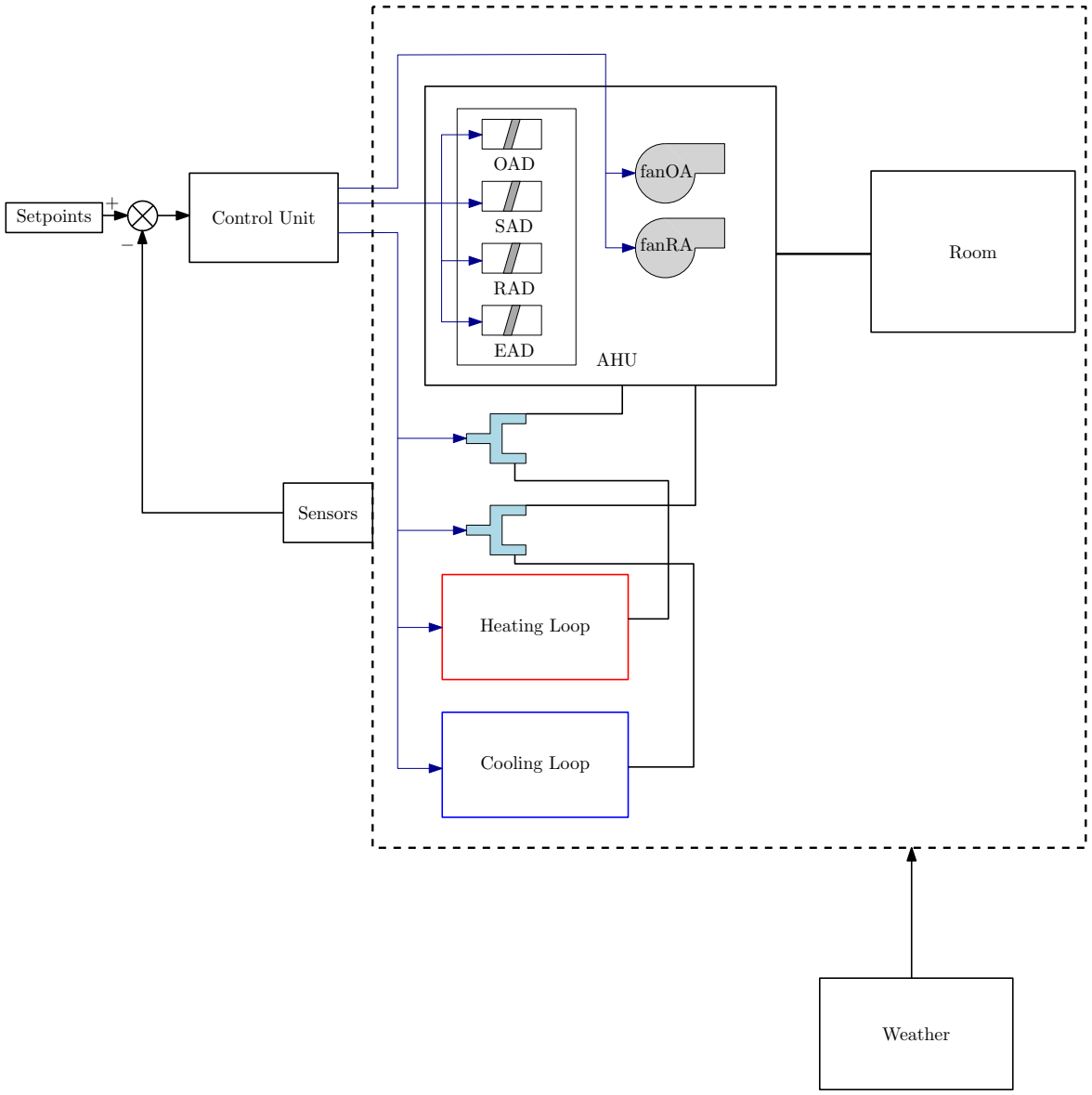



**Figure 3.4:** A simplified control loop concept of a HVAC system.

## 3.4 Time-series clustering

Given a dataset of time-series, without much prior knowledge about the data itself, clustering is one of the approaches we may choose. Since we are dealing with fault detection and would like our approach to be scalable, which means we are not dealing with a specific model; thus, we would not know how faults will affect the system nor how the system will react to faults.

As mentioned, we do not have much prior knowledge of a specific HVAC system model type besides what are commonly seen in most HVAC systems. With the large amount of data generated from the system, clustering serves as a good tool helping us to learn more about the data.

Clustering (see section 3.4.3), a category of unsupervised statistical analysis methods, is commonly used to explore data. There are a number of different clustering methods; however, there are generally four types of clustering methods: connectivity-based clustering, centroid-based clustering, distribution-based clustering, and density-based clustering. One of the most well-known algorithms of each type are hierarchical clustering, k-means clustering, Gaussian mixture models, and DBSCAN respectively. Each of them has its strengths and drawbacks.

While clustering algorithms serve as powerful tools for exploring conventional data, most of them are not able to handle time-series data. Nevertheless, researchers have come up with fixes and adjustments to the existing clustering algorithms, making them compatible with time-series data. Moreover, some approaches are designed specifically to handle time-series data. Here we will give a brief overview on how time-series clustering works.

Generally speaking, there are a few challenges when dealing with time-series data. First, time-series data are often far larger than memory size and consequently are stored on disks. This leads to an exponential decrease in speed. Second, time-series data are high dimension, which makes handling these data difficult for clustering algorithms and also slows down the speed. Third, some similarity measure is needed to carry out the clustering task. There are a plethora of papers about time-series clustering [48–51]; they can be classified into three types: whole time series clustering, subsequence time-series clustering, and time point clustering. This is shown in figure 3.5. Our work will be focusing on whole time-series clustering.

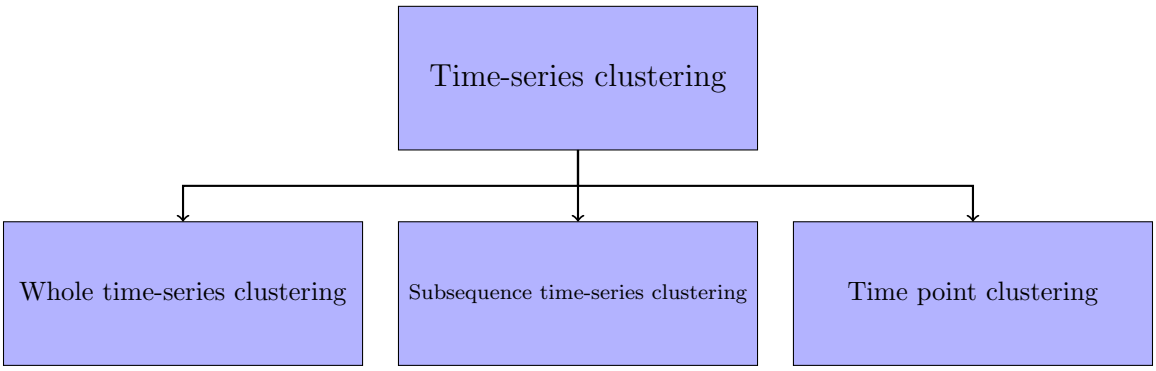



**Figure 3.5:** Time-series clustering types.

The time-series clustering task can be split into four components [48]:

1. Time series representation
2. Distance measure(similarity/dissimilarity)
3. Clustering prototype
4. Time series clustering

### 3.4.1 Time-series representation

Before doing any clustering, we would have to determine how we want our time-series data to be represented to clustering algorithms. One can choose to simply use the raw time-series data, reduce the dimensions, or apply some math model to represent the data.

Time series representations:

1. Raw data
2. Data adaptive - In this category, a common representation will be chosen for all items in the database that minimizes the global reconstruction error (e.g., PCA, SVD, SAX)
3. Non-data adaptive - In contrast, these methods consider local properties of the data, and construct an approximate representation accordingly (e.g., PAA,DWT)
4. Model based - Relies on the assumption that the observed time series was created based on basic model. The aim is to find the parameters of such a model as a representation (e.g., HMM,ARMA [44])
5. Data dictated - The compression ratio is defined automatically based on raw time series such as clipped (e.g., Clipped [52])

### 3.4.2 Distance measures

In order to compare time-series data, a distance measure (or some kind of similarity/dissimilarity measure) is required. Again, there are a variety of distance measures. How and which we should choose depend on the time-series characteristic itself, length of time-series, our objective, and the time-series representation used. We will introduce some of the most commonly used and competitive ones. There is a long list of distance measures, and they can be classified into different types, such as shape-based, compression-based, and feature-based [48–51]. Our work will be focusing on dealing with raw time-series data; therefore, we will be focusing on their corresponding distance measures.

---

#### 3.4.2.1 Euclidean distance

For most people, the first thing that comes to mind when it comes to a distance measure is probably the Euclidean distance. Let's say we have two time-series $x$ and $y$ of equal length $T$; that is, $x$ and $y$ are $\in \mathbb{R}^T$. Then the Euclidean distance can be easily calculated:

$$\begin{aligned}\text{Euclidean distance}(x, y) &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_T - y_T)^2} \\ &= \sqrt{\sum_{t=1}^{T} (x_t - y_t)^2}\end{aligned}$$

Or in vector notations, this is simply:

$$\text{Euclidean distance}(x, y) = ||x - y||_2$$

where $|| \cdot ||_2$ stands for the $L^2$ norm.

The Euclidean distance is simple but very useful, and has a tendency to be neutral that it does not favor any direction, which means it is blind to trends/correlations. Although the Euclidean distance seems simple and straightforward, it actually turns out to be surprisingly competitive in time-series clustering [53, 54] when it comes to comparing their shapes. Other cases such as applications in high-dimensional space, the Euclidean distance is probably not so good a choice [55].

#### 3.4.2.2 Manhattan distance (Absolute distance)

The Manhattan distance is similar to the Euclidean distance with little difference in

its mathematical form. Given two time-series $x$ and $y$ of equal length $T$, such that $x$, $y \in \mathbb{R}^T$:

$$\begin{aligned}\text{Manhattan distance}(x, y) &= |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_T - y_T| \\ &= \sum_{t=1}^{T} |x_t - y_t|\end{aligned}$$

In vector notations:

$$\text{Manhattan distance}(x, y) = ||x - y||_1$$

where $|| \cdot ||_1$ stands for the $L^1$ norm.

The Manhattan distance, which is also known as the absolute distance, taxicab geometry, city block distance, or $L^1$ norm, gives us similar results for our usage in comparing weather data. However, the Manhattan distance is widely used as a heuristic[21] technique in optimization problems.

#### 3.4.2.3 Minkowski distance

Minkowski distance ($L^p$[22] norm) is a generalization of both the Euclidean and Manhattan distances. Its mathematical definition is:

$$\begin{aligned}\text{Minkowski distance}(x, y) &= (|x_1 - y_1|^p + |x_2 - y_2|^p + \cdots + |x_T - y_T|^p)^{\frac{1}{p}} \\ &= \left( \sum_{t=1}^{T} |x_t - y_t|^p \right)^{\frac{1}{p}}\end{aligned}$$

where $p \geq 1$[23]. Figure 3.6 shows a plot of $||x||_p = 1$ for different $p$ values.

[21] While working on optimization problems, we would sometimes need to work with cardinalities, which is not a convex problem. A non-convex problem is relatively hard to solve; hence, people try to relax the problem by solving the $L^1$ norm instead.

[22] $L$ stands for Lebesgue spaces.

[23] If $p < 1$, it is not a metric.

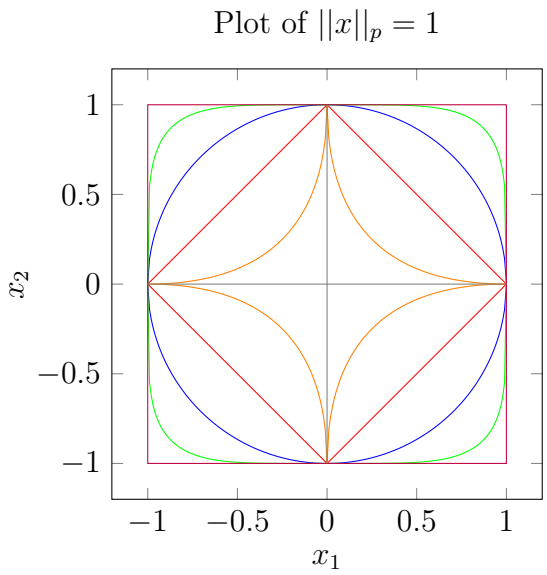



**Figure 3.6:** Unit spheres in 2 dimensional space for different $L^p$ norms, $||x||_p = 1$. $p = 0$, 0.5, 1, 2, 5, $\infty$ are plotted in gray, orange, red, blue, green, and purple colors respectively.

#### 3.4.2.4 Dynamic time warping (DTW)

Dynamic time warping [56] has been used for voice recognition in the past, and is still used as a similarity measure for time-series data. What dynamic time warping does is non-linearly aligning two sequences $x$ and $y$ ($x$ and $y$ don't have to be same in size, $x \in \mathbb{R}^N$, $y \in \mathbb{R}^M$) by minimizing their differences in shape. A warping path $p$ is found between the two sequences, defining how they are being matched. This is done by creating a cost matrix $C$, which its entity $C(n, m)$ is the cost of $x_n$ and $y_m$. A cost function can be chosen arbitrarily; usually the Euclidean distance is the default choice.

$$C(n, m) := dist(x_n, y_m)$$

where $dist$ is the cost function, $x_n$ is the $n$th element of sequence $x$, and $y_m$ is the $m$th element of sequence $y$. For a given warping path $p$, the warping cost is:

$$c_p = \sum_{l=1}^{L} c(x_{n_l}, y_{m_l})$$

where $L$ is the size of the warping path $p$, that is, $p = (p_1, p_2, \ldots, p_L)$. The dynamic time warping distance is then defined as:

$$DTW(x, y) = \min_p(c_p(x, y))$$
$$p^* = \arg\min_p(c_p(x, y))$$

for $p$ is the warping path and $p^*$ is the optimal warping path that minimizes the DTW distance. This optimization problem is solved by dynamic programming [57–59]. Defining the accumulated cost matrix $D$ as:

$$D(n, m) := DTW(x(1 : n), y(1 : m))$$

where $x(1 : n)$ and $y(1 : m)$ represent the subsequences of sequence $x$ from step 1 to $n$ and sequence $y$ from step 1 to $m$, respectively. The accumulated cost matrix $D$ is then computed:

$$D(n, m) = \min[D(n-1, m-1), D(n-1, m), D(n, m-1)] + c(x_n, y_m)$$

and the optimal warping path $p^*$ can then be determined by the algorithm in reverse order:

$$p_{l-1} = \begin{cases} (1, m-1) & \text{, if } n = 1 \\ (n-1, 1) & \text{, if } m = 1 \\ \arg\min\limits_{(n,m)}[D(n-1, m-1), D(n-1, m), D(n, m-1)] & \text{, otherwise} \end{cases}$$

where $p_1 = (1, 1)$ and $p_L = (N, M)$ (boundary conditions). Once $D$ and $p^*$ are solved, we get our DTW distance:

$$DTW(x, y) = D(N, M)$$

DTW distance takes the shape of a sequence into account and is a good measure for comparing sequential data. The drawback of DTW is that it is computationally expensive compared to measures such as the Euclidean distance. Because of this, some DTW variants have been developed in order to speed up the computation process and save time. For example, FastDTW [60], a multi-scale DTW variant that gives an approximated optimal result with a reduced computational time.

To show an example of how DTW works, we take two weather temperature profile data. In figure 3.7, the plot in the first row shows two weather profiles. The green lines between them indicate how they are being aligned by the warping path $p^*$. The plot in the second row shows what it would be like if both sequential data were unfolded into a linear space; also, DTW and FastDTW are compared. The bottom two plots in the figure show how it would look like when sequential data1 is non-linearly warped to data2's space and data2 non-linearly warped to data1's space, respectively. Figure 3.8 shows the DTW and FastDTW warping paths. The two paths are plotted on top of a contour plot (heat map) of the cost matrix $C$. We can see that the optimal warping path walks in the 'valley,' which is a path with the lowest cost, of the contour plot. The FastDTW warp path tends to stay closer to the diagonal region due to its multi-scale approximation. Figures 3.9 and 3.10 show the DTW warp of two pulse data instead; this clearly shows how DTW matches the shape of sequences.

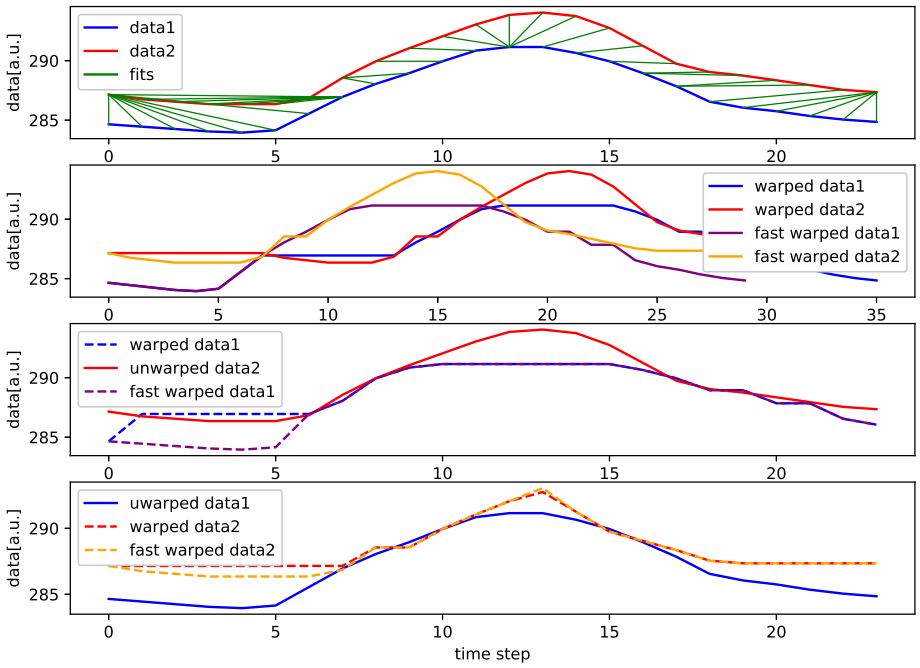



**Figure 3.7:** DTW warping two weather temperature profile data.

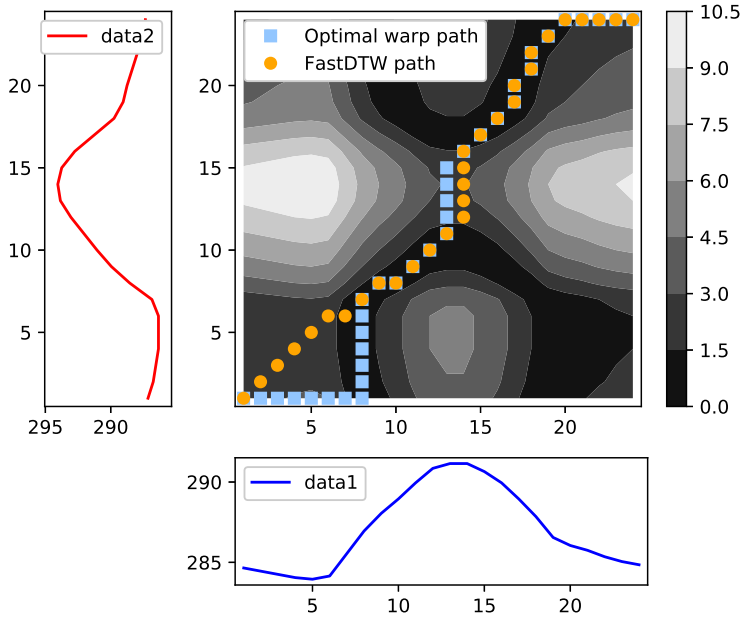



**Figure 3.8:** DTW warping path of two weather temperature profiles.

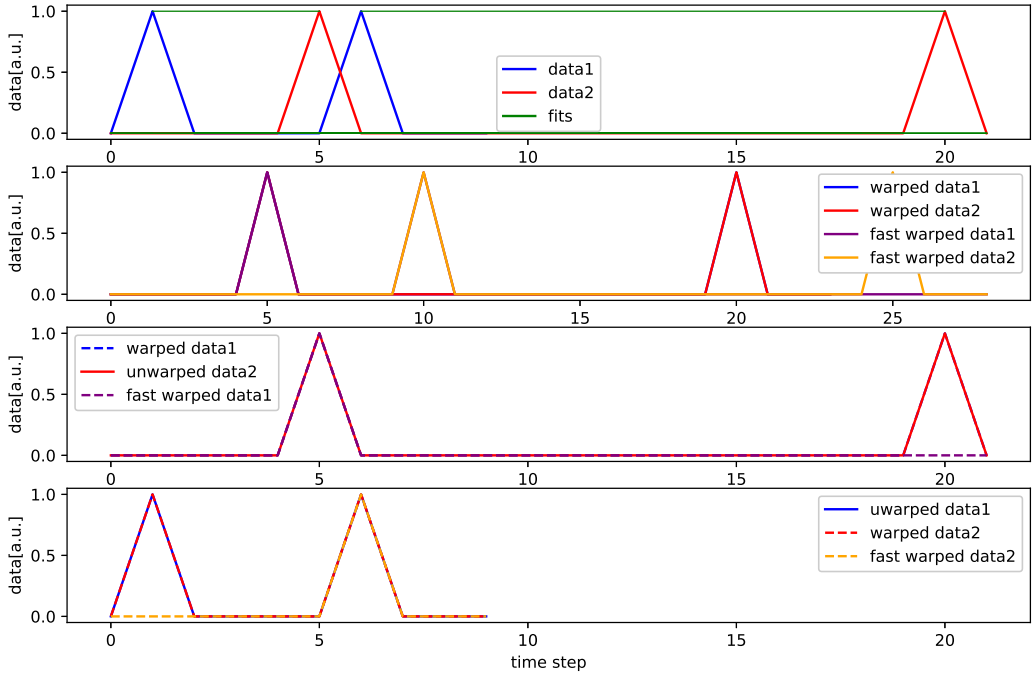



**Figure 3.9:** DTW warping two pulse data.

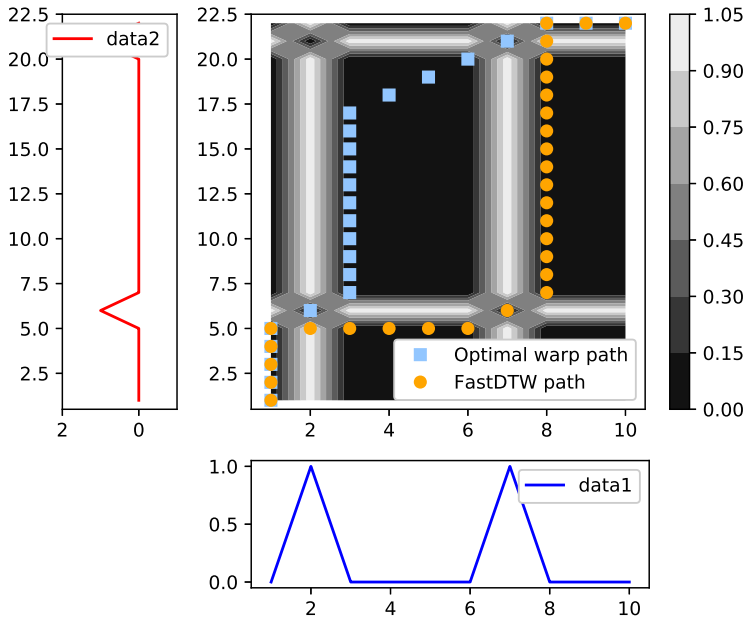



**Figure 3.10:** DTW warping path of two pulse data.

#### 3.4.2.5 Mahalanobis distance

If we say DTW distance takes the shape of a sequential data into consideration, Mahalanobis distance takes the correlation into account. Given two time-series data $x \in \mathbb{R}^N$ and $y \in \mathbb{R}^M$:

$$\text{Mahalanobis distance}(x, y) = \sqrt{(x-y)^\top C^{-1}(x-y)}$$

where $C$ is the covariance matrix of $x$ and $y$.

---

There are many other distance measures, cosine similarity, Jaccard/Tanimoto coefficient, Pearson correlation coefficient, averaged Kullback-Leibler divergence, longest common subsequence (LCSS), and probability-based distance function to name a few. They all have different features and their usage depends on the problem itself. The choice of distance measures is also affected by the time-series representation used. For instance, if one chooses to use a model based representation, such as HMM, ARMA or GP, the distance measure used would probably be comparing the model parameters [23]; and if the data is modeled with some probability distribution, Kullback-Leibler divergence or some probability-based distance function are likely to be chosen. In [53], a comparison of using different distance measures for building energy time-series data clustering is demonstrated. The paper has come up with an evaluation method for the distance measures and concluded that "Euclidean distance as the best similarity metric to obtain good general solutions in raw-data-based time series clustering" and "has been the most successful in the evaluation test and in finding the best clusters." In addition, other papers, such as [54], that are working on time-series data other than building energy also came up with similar conclusions that Euclidean distance is a very competitive measure.

### 3.4.3 Clustering algorithms

As mentioned, there are basically four kinds of clustering types, each having its own pros and cons. They are powerful tools for exploring and learning about new data. Which clustering methods we should choose to use is like picking the right tools for a certain task; it depends on the problem itself. Here we will mention some of the methods that we have tried out and found useful for our work.

#### 3.4.3.1 K-means clustering

K-means clustering [61–63] is one of the most popular clustering algorithms. Its concept is simple and easy to understand; we only need to iterate over two steps until the result converges. The idea is, given $k$ clusters in a feature space $\mathbb{R}^M$, our task would be assigning all the data points to their nearest cluster. The location, or center/centroid,

of each cluster is determined by the mean of all data points in the cluster. However, we are only provided with the information: the number of clusters $k$ and data points in the feature space. If we know the centroids of each cluster, we could easily determine the assignments for each data point. On the other hand, if we know the assignments of each data point, we could compute their means with ease. What we do is to make a guess for the means (assignments); thus, we can determine the assignments (means). That is, once we know either the means or assignments, we are able to determine the other. By repeating this process iteratively, the means and assignments would converge. K-means clustering is actually minimizing the *distortion measure*[24]:

$$J = \sum_{n=1}^{N} \sum_{i=1}^{K} z_n^i ||x_n - \mu_i||^2$$

where $N$ and $K$ are the number of data points and clusters, $x_n$ is the $n$th data point, $\mu_i$ is the $i$th cluster mean, and $z_n^i$ stands for the assignment of the $n$th data point in the $i$th cluster. $z_n^i$ is defined as:

$$z_n^i = \begin{cases} 1, & \text{if } i = \arg\min_j ||x_n - \mu_j||^2 \\ 0, & \text{otherwise} \end{cases}$$

where the means $\mu_i$'s are determined by:

$$\mu_i = \frac{\sum_n z_n^i x_n}{\sum_n z_n^i}$$

The clustering algorithm is simply computing the two equations shown above.

We see that the k-means clustering algorithm has a 'crisp' or 'hard' classifying assignment. Some variants of k-means, for example, fuzzy c-means [45] and the EM algorithm [61] of Gaussian mixture models, use some 'soft' assignment rule. Furthermore, the k-medoids uses the *medoids* of data points instead of the means. Given $N$ data points in a feature space and a distance function *dist*, the medoid is calculated as:

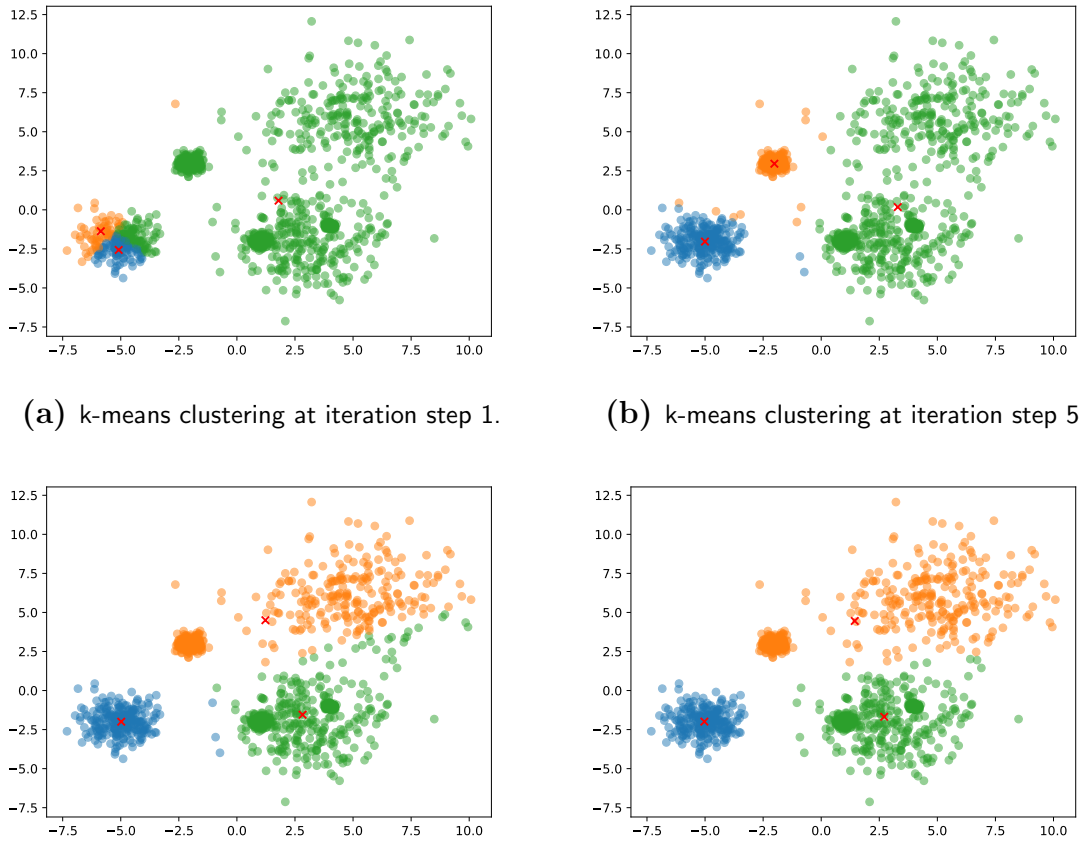
$$m^* = \arg \min_{m \in \{x_1, x_2, \cdots, x_N\}} \sum_{i=1}^{N} dist(x_i, m)$$

An example of k-means applied to synthetic data is shown in figure 3.11. We have set parameter $k = 3$. We can see how the cluster means (marked as red crosses in the figure) move around and converge eventually.

#### 3.4.3.2 Mean shift

Mean shift [64] algorithm is often used in computer vision or image processing, such

[24]The Euclidean distance is usually used as the default distance measure for k-means clustering.



**(a)** k-means clustering at iteration step 1.

**(b)** k-means clustering at iteration step 5.

**(c)** k-means clustering at iteration step 9.

**(d)** k-means clustering at iteration step 12.

**Figure 3.11:** K-means clustering example with $k = 3$. Iteration steps of 1, 5, 9, and 12 are shown. Cluster centers are marked with red crosses. Python code used for this example can be found in appendices D.3 and D.4.

as tracing a moving object in a video; however, it can also be used as a clustering technique. The major use of mean shift for us is that mean shift finds 'modes' of a dataset. In section 3.4.3.1, we mentioned that k-means clustering starts with an initial guess of either the means or assignments, and then iterate the steps until clusters converge. Nonetheless, we have never mentioned how the number of clusters, $k$, is determined, which is the challenging part of k-means. We would need to have some kind of prior knowledge or insight about the problem or data itself. For instance, if our data is seasonal, setting $k = 4$ seems to be reasonable. The challenge here is caused by the nature of clustering, for we conduct clustering algorithms due to the fact that we do not know the data well enough, or else, we would be doing classifications instead. With the use of mean shift, we would be able to determine the number of clusters from the density distribution of the data points.

The idea of mean shift is for each data point, we compute the mean of all data

points within a given radius from a data point. We then 'shift' the data points to their new locations, which are the mean points we just computed. The 'shift,' is the difference of the mean and location of the original data point. Iterating this process until the data points converge, we then find the 'modes' of the dataset. Data points are then clustered to the 'modes' they converge to. This shifting iteration is illustrated

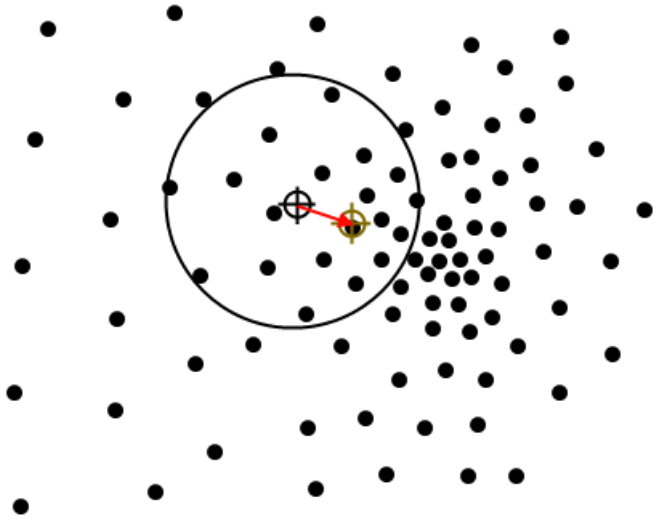


**Figure 3.12:** An iteration of mean shift. The data point is moved to a new location determined by the mean of all points inside the black circle. Source: https://goo.gl/qC6JP9.

in figure 3.12.

There are different kernel functions used to calculate the means; given a kernel function $k(x)$, we can calculate the mean:

$$\mu(x) = \frac{\sum_{x_i \in N(x)} k(x_i - x) x_i}{\sum_{x_i \in N(x)} k(x_i - x)}$$

where $x$ is the data point, $N(x)$ is the neighbors of $x$, and $\mu(x) - x$ is the mean shift. Popular kernels are the step kernel:

$$k(x) = \begin{cases} 1, & \text{if } x \leq r \\ 0, & \text{otherwise} \end{cases}$$

and the Gaussian kernel:

$$k(x) = \exp\left(-\frac{x}{2\sigma^2}\right).$$

#### 3.4.3.3 DBSCAN

Density-Based Spatial Clustering of Applications with Noise, or DBSCAN [65, 66],

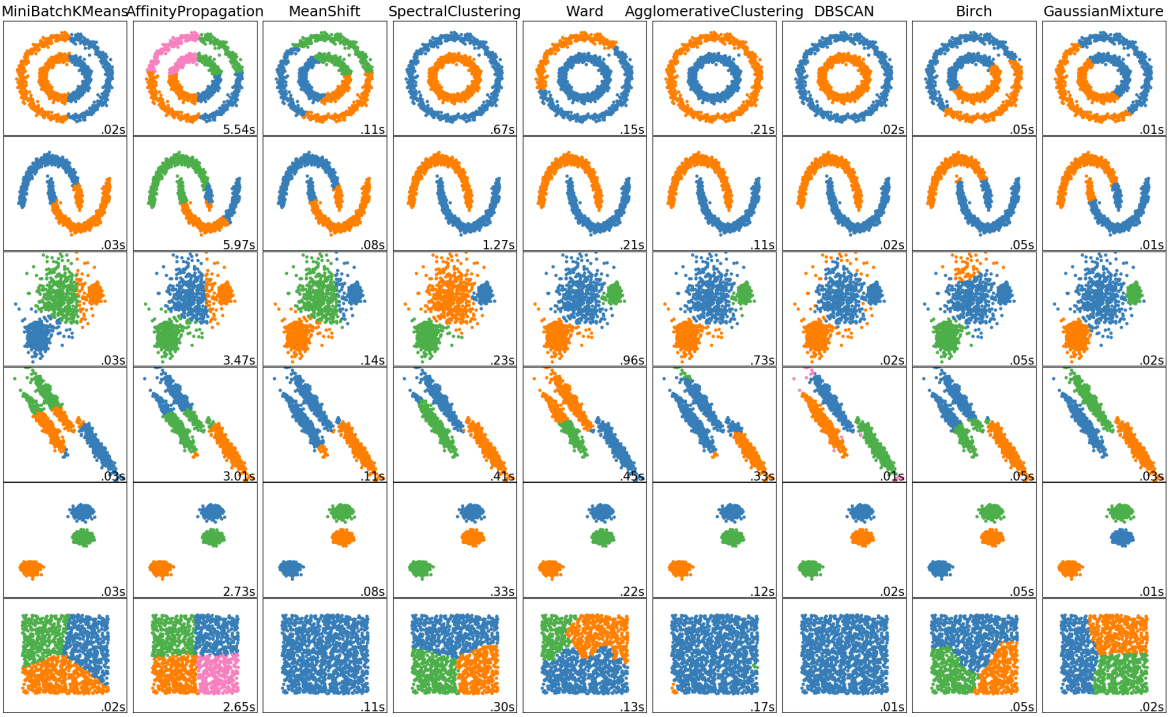


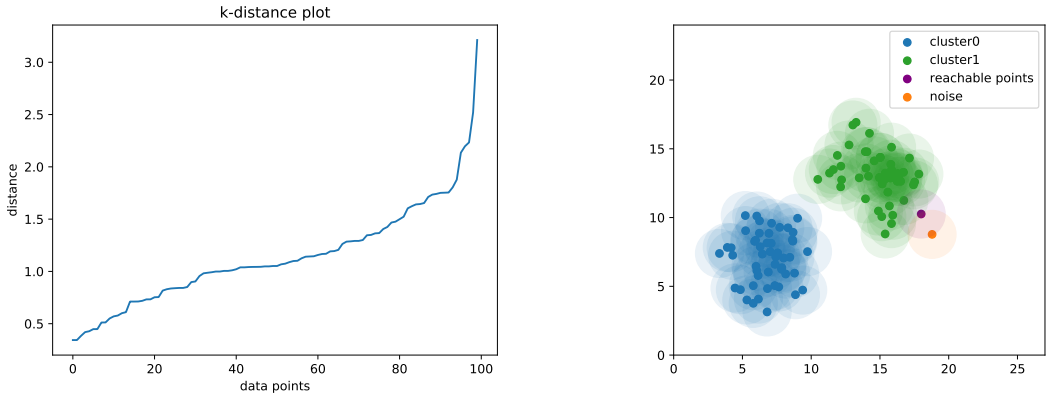

**Figure 3.13:** A comparison of different clustering algorithms with synthetic data. Source: scikit-learn, https://goo.gl/nNUstV.

is a density-based clustering algorithm. Compared to centroid-based clustering algorithms, e.g., k-means, DBSCAN has the advantage of being able to cluster data points distributed in 'irregular-shapes' in its feature space with a much higher accuracy. Figure 3.13 shows a comparison of different clustering algorithms applied to different sets of synthetic data. The result in the figure shows us the characteristics of how data is treated for each algorithm.

The basic idea of DBSCAN is putting the data points into three different categories. That is, a data point is either a core point, reachable point, or noise. DBSCAN requires two parameters, *MinPts* and $\epsilon$. If a data point has at least a number of *MinPts* within the radius distance of $\epsilon$, then it is considered as a core point. If a data point is reachable[25] from a core point but is not a core point itself, it is then a reachable point. If a data point is not reachable from any core points, it is then considered as an outlier and treated as noise. In order to determine these two parameters, one usually starts with assigning a number to *MinPts*. According to [65], for most cases, with a $MinPts \geq 4$ is usually good enough; bigger values do not make much difference. Given a *MinPts*, we can then determine $\epsilon$ by drawing a *k-distance* plot. *K-distance* is the distance from the $k$th nearest neighbor for a data point, and plotting all the *k-distance*'s out for each data point in a(n) ascending/descending order, we would get

[25]Reachable means within the distance of $\epsilon$.

a *k-distance* plot. As an example, with some randomly generated data, a *k-distance* plot is shown in figure 3.14a. By inspecting the plot, we see that there is an 'elbow' at




**(a)** A k-distance plot example. **(b)** DBSCAN clustering scatter plot.

**Figure 3.14:** A DBSCAN example. Random data points are generated and clustered with DBSCAN.

about 1.74 in the distance ($y$) axis. This means that for this set of data points, there is a density change somewhere close to this distance. Normally, we would set our $\epsilon$ to this value, that is, $\epsilon = 1.74$ for this example.

We then run the DBSCAN algorithm with these parameter settings, and end up with the results shown in figure 3.14b. We can see that these data points are separated into two clusters based on their densities. A reachable point, which is reachable to only one (less than *MinPts*=4) core point in cluster 1 (green), is drawn in purple. An outlier, which is not reachable, is drawn in orange. The transparent circles stands for the radii with a length of $\epsilon$.

In order to see this clearer, we generated less data points and did another example, this is shown in figure 3.15.

The same group of authors of DBSCAN has also invented Local Outlier Factor (LOF), an anomaly detection algorithm which we will use and discuss later (see 3.6.1).

#### 3.4.3.4 OPTICS

Closely related to DBSCAN, Ordering Points To Identify the Clustering Structure (OPTICS) [67] can be thought of as an enhanced version of DBSCAN by taking care of density varying datasets, where DBSCAN falls short. The advantage of OPTICS is that it gets rid of those hard to determine parameters in other clustering methods, e.g., the number of clusters $k$ for k-means, or the radius parameter $\epsilon$ in DBSCAN, and handles density varying datasets that are problematic to most clustering tools except hierarchical algorithms. Besides, it also provides us information on how the data structure looks like, something similar to hierarchical clustering. OPTICS does

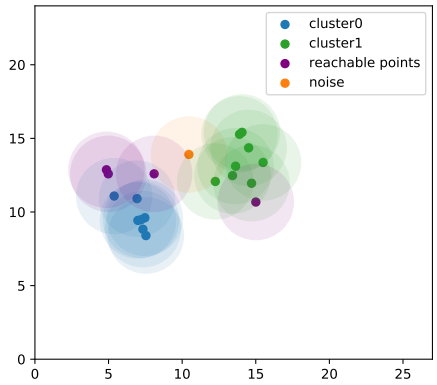



**Figure 3.15:** Another DBSCAN example. Random data points are generated and clustered with DBSCAN.

this by introducing the concept of 'ordering' and 'reachability-distance plots.' The trade-off here is that OPTICS runs slower than DBSCAN due to its ordering and nearest neighbor queries.

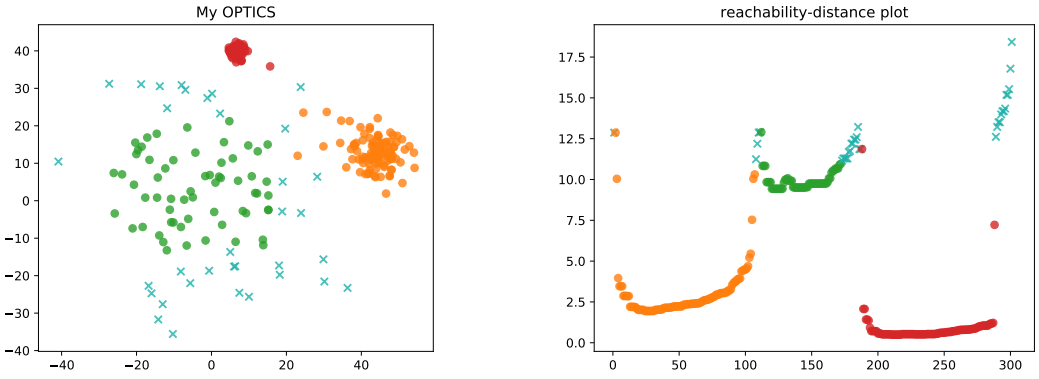
Starting with the first data point in a dataset, OPTICS then looks for the closest $MinPts$ data points from the first data point. The distances are measured using reachability-distance and core-distance:

$$\text{core-distance}_{\epsilon,MinPts}(p) = \begin{cases} \text{Undefined} & , \text{ if } |N_\epsilon(p)| < MinPts \\ MinPts\text{-distance}(p) & , \text{ otherwise} \end{cases}$$

$$\text{reachability-distance}_{\epsilon,MinPts}(p,o) = \begin{cases} \text{Undefined} & , \text{ if } |N_\epsilon(o)| < MinPts \\ \max(\text{core-distance}(o), \text{dist}(p,o)) & , \text{ otherwise} \end{cases}$$

where $N_\epsilon(p)$ is the function that returns the set of neighbors of data point $p$. Note that the reachability-distance is not symmetric in general, though it could be; that is, $\text{reachability-distance}_{\epsilon,MinPts}(p,o) \neq \text{reachability-distance}_{\epsilon,MinPts}(o,p)$. OPTICS then does this data point collecting task iteratively the same as DBSCAN, but with an infinite number of $\epsilon_i$'s that $0 \leq \epsilon_i \leq \epsilon$ and keeping track of the order in a priority-queue.

After getting the order of all data points, we can draw out the reachability-distance plot of the dataset, which provides a lot of information about the data clustering structure. Figure 3.16a shows the clustering result of a randomly generated synthetic dataset (a size of 300 points). Its corresponding reachability-distance plot is shown in 3.16b; the x-axis in 3.16b is the index of the ordered data points. The data points are color-coded circle dots and outliers are marked as crosses. From figure 3.16, we see that the dents in the reachability-distance plot stand for clusters. Since the data points are 'ordered,' a sudden rise in the plot means a longer reachability-distance from the




**(a)** OPTICS clustering. **(b)** OPTICS reachability-distance plot.

**Figure 3.16:** OPTICS clustering with parameters $\epsilon' = 11$ and $MinPts = 15$.

previous point to the current point, which tells us it is a starting of a new cluster. In this example, by reading figure 3.16b, we set $\epsilon' = 11$ and can then extract a clustering result similar to DBSCAN.

However, if we run a DBSCAN on the same exact dataset, we will find that it is hard for us to determine the correct parameters. This is shown in figure 3.17. We have set the radius parameter $\epsilon$ to three different values; each value is an 'elbow' of the k-distance plot in figure 3.17a, and none of the results is satisfying. This is exactly the case, a dataset with various densities, where DBSCAN's weakness reveals.

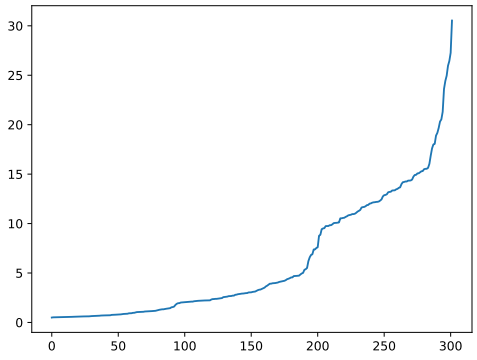
In order to extract clusters from the reachability-distance plot, we could either inspect the plot and set a $\epsilon'$ value cut, or can apply some automatic algorithms to find the 'dents' or 'elbows.' We have implemented[26] the $\xi$-steepness clustering algorithm introduced in [67], which locates the steep upward and downward areas automatically.

The results are shown in figures 3.18 and 3.19. It is interesting that this algorithm not only determines the parameters for us, but also provides a hierarchical cluster structure. We can find that there are smaller clusters within a bigger cluster. This is useful when one is trying to learn about a dataset.

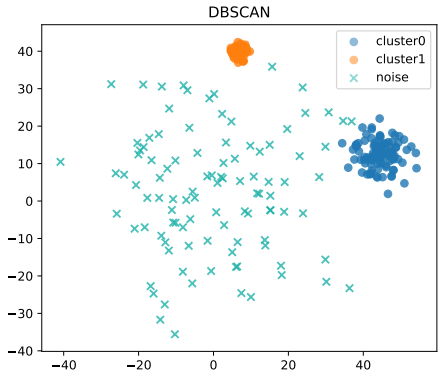
---

When working on clustering, one may encounter cases when the dataset is too large (big data!) to handle that the dataset cannot fit into the memory of a machine. Some algorithms do not have an on-line version to deal with this kind of situation; one would have to look for out-of-core learning methods. For example, mini-batch k-means is an alternative to k-means clustering for large datasets. Data partition techniques,

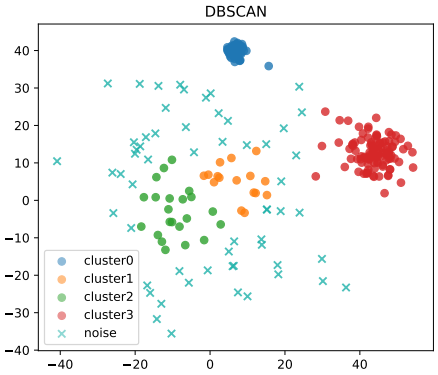
[26] As of the date of writing, no reliable python library was found; however, *scikit-learn* will include this in their next release. Once OPTICS is included, we can verify the results by comparing both implementations. The python code used can be found in appendix D.2.



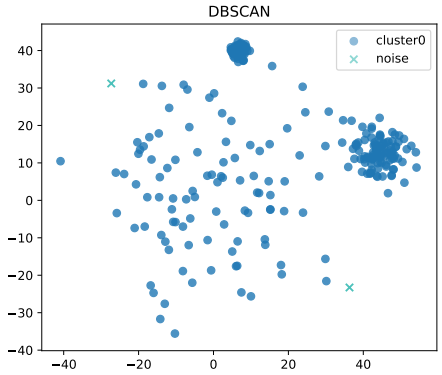
(a) DBSCAN k-distance plot.




(b) DBSCAN clustering with $\epsilon = 4.86$.




(c) DBSCAN clustering with $\epsilon = 9.53$.




(d) DBSCAN clustering with $\epsilon = 15.4$.

**Figure 3.17:** DBSCAN clustering with parameter $MinPts = 15$ .

such as [68], demonstrated distributing clustering for both k-means and DBSCAN by defining the contours of data points in a dataset, using a non-convex polygon generating algorithm [69], provides existing clustering algorithms a way to deal with this problem.

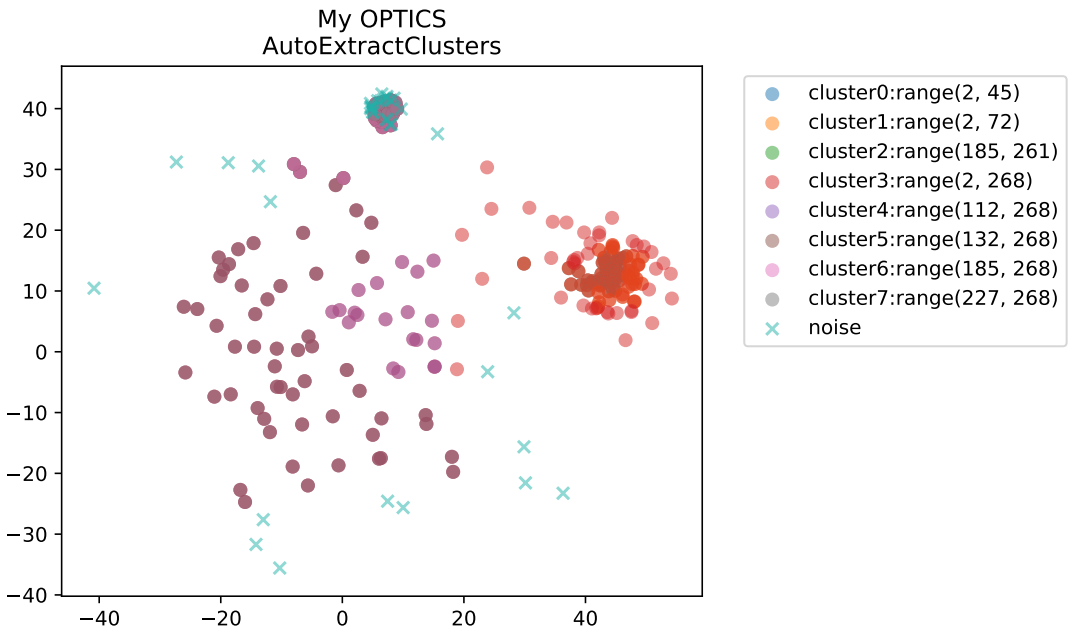



**Figure 3.18:** OPTICS automatic clustering using the $\xi$-steepness clustering algorithm with $\xi = 0.01$.

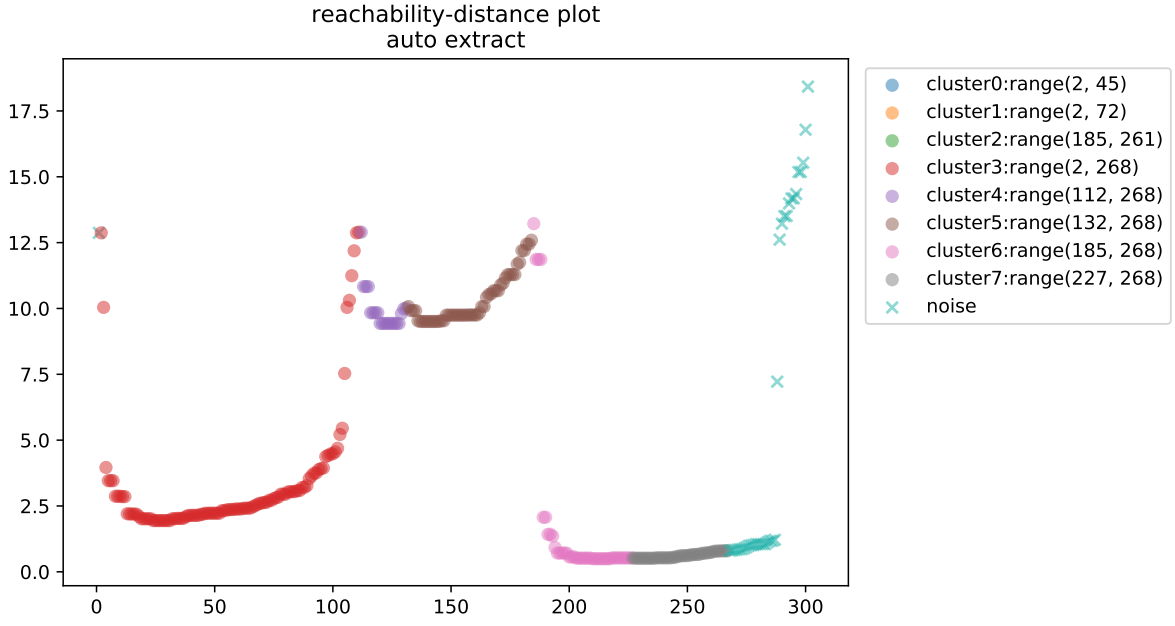



**Figure 3.19:** OPTICS reachability-distance plot.

### 3.4.4 Rank ordering time-series data

Despite the different characteristics of each algorithm, sophisticated clustering algorithms, such as the ones we discussed in 3.4.3, all work well to some extent as they are designed. As shown in figure 3.13, for different datasets, an algorithm may perform better and faster than others for one dataset, but does not necessarily work well for other cases. Algorithm A could be the best choice for dataset A but fail to finish the task in a reasonable time period, then we would probably consider algorithm B instead due to computational feasibility. Sometimes we would even consider using more than one method; a combination of multiple clustering algorithms may help us learn even more about the dataset. Besides that, we may also use multiple methods as some sort of verification for our results.

In addition to sophisticated clustering algorithms, it would be interesting to try out something more simple and straightforward, that is, rank ordering time-series data. The reasons why rank ordering draws our attention are: 1. As stated, simplicity and straightforwardness. There is no obvious reason to not use it. Besides, simple means faster; rank ordering should run faster and take less time than clustering algorithms in general. 2. The main purpose of conducting all these data examinations is to learn how we should compare the input time-series data. When it comes to comparing sequential data, rank ordering seems to fit pretty well. 3. As mentioned, we want our time-series data become 'comparables,' some pattern matching method would serve us better than a clustering algorithm here. Although the boundaries are vague here, rank ordering can both serve as a clustering and pattern matching method.

As simple as it sounds, rank ordering is done by computing the distance from each time-series data to others, using the distance measures we mentioned in 3.4.2. Then we sort the order based on the similarities among these time-series data. We will practice rank ordering in section 4.4.

To conclude, there is no best method; it all depends on the problem itself. We will compare the methods and find the ones that fit our purposes.

## 3.5 Multidimensional Scaling (MDS)

We won't be using multidimensional scaling much, but it will serve as a tool for visualizing our time-series data in another way. MDS is regarded as a member of manifold learning, which focuses on dimensionality reduction. Since our work will mostly be focused on raw time-series data and their distance measures, MDS seems to be a good auxiliary exploration/visualization tool, for it uses similarity/dissimilarity measures as its input. So MDS is basically a method that takes similarity/dissimilarity measures of some data and represents them as distances in a low dimensional space.

For simple cases, one can solve a MDS problem with just a pen, ruler and compass [70]; however, this is not practical for high dimensional data or when dealing with many data points.

MDS is usually formulated as an optimization problem as:

$$\min_{x_1,\ldots,x_N} \sum_{i<j} (dist(x_i, x_j) - d_{ij})^2$$

where $x_i$ and $x_j$ are the $i$th and $j$th rows of our low dimensional data matrix $X \in \mathbb{R}^{N\times M}$, and $d_{ij}$ is the entity of the distance matrix $D = dist(X) \in \mathbb{R}^{N\times N}$ at its $i$th row and $j$th column, which looks like:

$$D = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1N} \\ d_{21} & d_{22} & \cdots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{N1} & d_{N2} & \cdots & d_{NN} \end{bmatrix}$$

and $dist$ is the distance function used, which is usually the Euclidean distance.

The intention here is trying to reconstruct the data points in a low dimension space (usually $M = 2$ for plotting) with as little stress as possible. The term 'stress' used for MDS is also known as error or loss. For example, we use some heat load data we gathered and carry out MDS. From the results shown in figure 3.20, we see

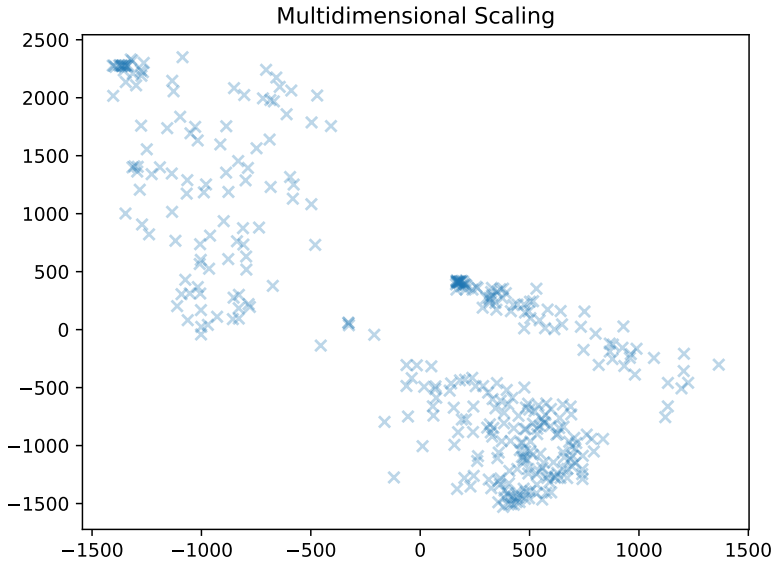
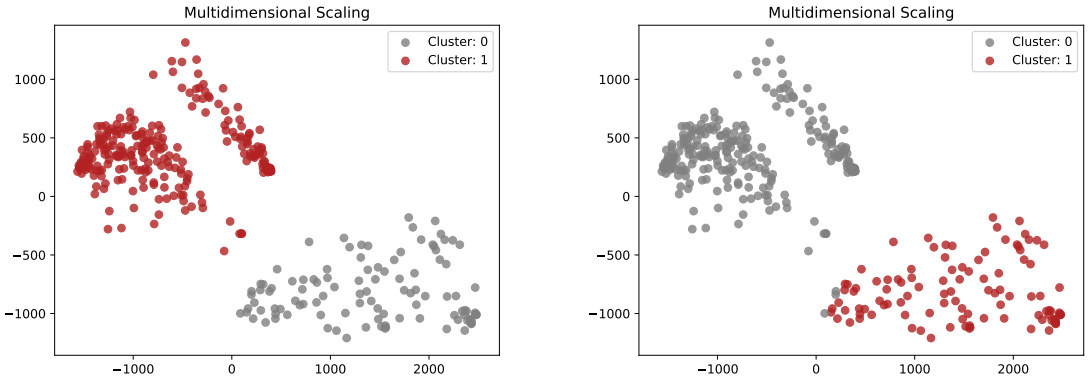



**Figure 3.20:** Multidimensional scaling applied to heat load data.

that each sample of the heat load data, which is time-series data, is compressed to a single data point in a two dimensional plot. Of course, applying a dimensionality reduction method to the data, we will loose some information. But on the other hand, this somehow simplifies the raw data and helps us visualize and explore the dataset. Figures 3.21a and 3.21b show the MDS results of clustered data using k-means and mean shift clustering.




**(a)** Multidimensional scaling applied to heat load data clustered with k-means.

**(b)** Multidimensional scaling applied to heat load data clustered with mean shift.

**Figure 3.21:** Multidimensional scaling with clustering.

## 3.6 Anomaly detection

An anomaly is a process that behaves unusually which its pattern deviates from normal behaviors. Anomaly has many alternative names; people usually use the term, outliers, in statistics; abnormalities, deviants, and discordants are also used interchangeably. Although not necessarily true, in a working physical system with given inputs, an anomaly usually indicates the existent of fault(s), and exceptions happen due to uncertainties in the system. Since we are working on HVAC systems, which is a physical system and should be deterministic, if anomalies are found in our system, this gives us a hint that it is likely to have faults found, too. Therefore, the task of fault detection mainly relies on anomaly detection methods.

It is true that anomaly detection is a big topic and has many applications in a lot of different areas [71–73]. The challenge is how to properly define what is normal? This is still an open question, but people in the probability and statistics field have come up with the concept of hypothesis testing, described in 1.2.1, and this has been adopted for use in anomaly detection. The challenge becomes tougher when we are dealing with unknown anomalies, that is, unsupervised learning. Compared to supervised learning methods, such as SVM [61, 62] and GP [74], our training data does not have data of known faults; hence, it becomes harder to determine the boundaries of normal behaviors. If there is prior knowledge of what faults look like, we then know their patterns and could clearly exclude them from the set of normal patterns.

Principal component analysis (PCA) [75], a popular technique used to examine the components of a dataset, is also often used for dimensionality reduction and as an anomaly detection tool [76]. Research using PCA for HVAC FDD has been conducted [26, 36] giving reasonable results; however, PCA struggles to tell the differences between faults and change of operating modes in a system [8]. This could probably be improved by setting up a more detailed model for the HVAC system, but this would

be going towards the opposite direction of being scalable.

Here we introduce two anomaly detection methods. One is Local Outlier Factor, a density-based approach that is similar to DBSCAN and OPTICS. Another method called Isolation Forest, which is a depth-based approach, is called a forest because it uses a tree data structure in its algorithm.

### 3.6.1 Local outlier factor (LOF)

Sharing similar ideas with DBSCAN and OPTICS, Local Outlier Factor (LOF) [77] is a density-based method; however, it also introduces the 'local' outlier concept compared to 'global' ones[27]. LOF of a data point is computed by comparing its neighboring local reachable densities with its own reachable density. Using LOF's author notations, that is:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{|N_{MinPts}(p)|}$$

where $MinPts$ specifies the minimum number of data points, $N_{MinPts}(p)$ is the set of $MinPts$-nearest neighbors of data point $p$, and $lrd_{MinPts}(p)$ is the local reachable density of $p$:

$$lrd_{MinPts}(p) = \left( \frac{\sum_{o \in N_{MinPts}(p)} reach - dist_{MinPts}(p, o)}{|N_{MinPts}(p)|} \right)^{-1}$$

the reachability distance of $p$ from $o$, $reach - dist_{MinPts}(p, o)$ is defined as:

$$reach - dist_{MinPts}(p, o) = \max(k - distance(o), d(p, o))$$

where $d(p, o)$ is the distance between data points $p$ and $o$, and $k - distance(o)$ is the distance of the $k$th nearest neighbor of point $o$.

The reason why the notion of reachability distance is introduced is because of its smoothing effect and the fact that it improves statistical stability. LOF should have a value comparable or lower than 1 when the data point is considered as an inlier, and a value much higher than 1 when it is an outlier. Given the LOF values, inliers are easily determined for those LOF values which are less than 1; however, one will still have to decide where the outlier determining line is drawn. One method would be to set the threshold at a percentile level, e.g., the 90th percentile; that is, the highest 10% LOF values are marked as outliers.[28]

[27] Another density-based clustering method called OPTICS is closely realted to DBSCAN. OPTICS (see section 3.4.3.4) handles varying densities in the dataset by introducing the reachability distance, similar to what LOF uses.

[28] This is the default setting used for the python library *scikit-learn*. We have implemented LOF based on [77] ourselves getting the exact same results with library *scikit-learn*. See figure 3.22.

What is great about LOF is that it is a density-based method, which works on datasets with irregular shapes[29]. LOF is very adaptive to different datasets for it measures the densities instead of distances. Also, it can be applied to many applications as long as we can come up with a dissimilarity measure that generates a distance matrix for us. Compared with DBSCAN, LOF only takes one parameter $MinPts$. How to

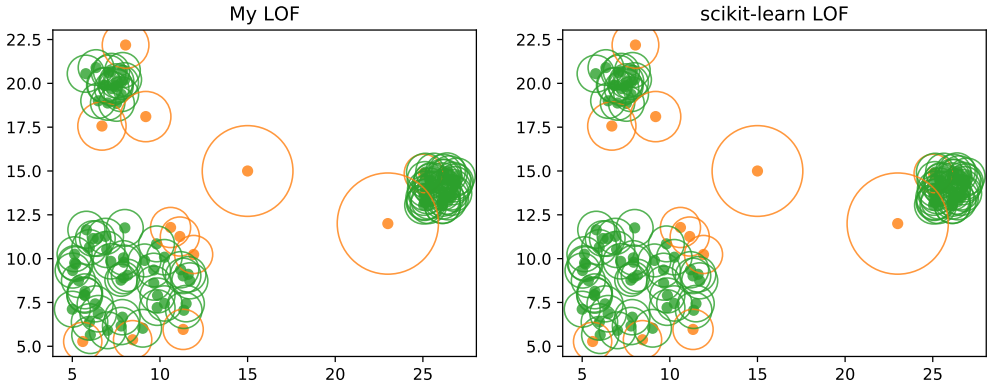



**Figure 3.22:** LOF applied on a testing dataset. Circles around data points represent the LOF values by their radii. Orange colored points are marked as outliers. By comparing the radii, we see outliers have larger LOF values.

determine this depends on the dataset. A general guideline is to choose a value within the upper and lower bounds. For stable results, $MinPts$ should be at least 10 to avoid statistical fluctuations. Also, according to [77] "$MinPts$ can be regarded as the minimum number of objects a cluster has to contain." This means that we would want to choose $MinPts$ to be larger than the number of points close together which we consider as outliers, so these points would not be classified as a cluster. On the other hand, we would choose $MinPts$ to be smaller than the smallest cluster of points we want to be classified as a cluster. That is, if there is a cluster with a number of points less than $MinPts$, they all would be regarded as outliers.

### 3.6.2 Isolation forest (iForest)

Isolation forest was invented with the idea that *"anomalies are 'few and different' and therefore they are more susceptible to isolation."* In plain words, it means that anomaly data points in a feature space are, statistically speaking, easier to isolate. The basic idea is to randomly select an attribute dimension in the feature space and split it at a randomly selected value between the minimum and maximum values of the corresponding attribute dimension. Comparing normal data points with anomaly ones, for most of the time, it should require a greater number of splits (the term path

[29] In the LOF family, there is also a variant called OPTICS-OF, which is based on the same concepts of LOF and OPTICS [78].

length is used for Isolation Forest) before we can isolate a normal data point than an anomaly one.

In practice, Isolation Forest uses an isolation tree data structure, according to its definition:

Definition - Isolation Tree:
Let $T$ be a node of an isolation tree. $T$ is either an external-node with no child, or an internal-node with one test and exactly two daughter nodes $(T_l, T_r)$. A test consists of an attribute $q$ and a split value $p$ such that the test $q < p$ divides data points into $T_l$ and $T_r$.

and the path length:

Definition - Path Length:
Path length $h(x)$ of a point $x$ is measured by the number of edges $x$ traverses an Isolation Tree from the root node until the traversal is terminated at an external node.

The anomaly score is then defined as:

$$s(x, n) = 2^{-\frac{\mathbb{E}(h(x))}{c(n)}}$$

where $c(n) = 2H(n-1) - (2(n-1)/n)$ is the average path length, $H(i)$, the harmonic number, is estimated by $\ln(i) + \gamma$(Euler's constant), and $\mathbb{E}(\cdot)$ is the expectation value.

As for the anomaly score $s$:

$$\begin{cases} \text{If the score } s \text{ is close to 1, then the data point is regarded as an anomaly.} \\ \text{If } s \text{ is much smaller than 0.5, then the data point should be normal.} \\ \text{If } s \text{ is close to 0.5, then the sample does not have any distinct anomaly.} \end{cases}$$

Further details can be found in [79].

Isolation Forest excludes itself from model-based, distance-based, and density-based methods. The advantages of its set up is that it does not need any distance measures; this means that the computational cost of building a distance matrix is saved, which could be a great difference when the dataset is large. In addition, Isolation Forest can work on a partially sampled dataset; thus, making it capable of handling very large datasets.

## 3.7 Proposed FDD approach

In sections 1.3 and 3.1 we have introduced HVAC FDD approaches people have developed and applied. From the literature reviews, we have found that due to the popularity and potential of applications of statistical learning tools, research on data-driven

methods has become a trend. However, most researches are making the assumption implicitly that they more or less have some prior knowledge of the HVAC system they are dealing with. Some approaches using supervised learning tools even have prior knowledge of the kind of faults exist and what their patterns may look like. Research on how faults impact HVAC performance is conducted [80] as well. These papers help us learn more about the connection between faults and our targeted HVAC system; nonetheless, these researches do not account for scalability, which is an important objective of our work.

On the other hand, in the field of HVAC FDD, unsupervised approaches seem to rely heavily on PCA methods. A very large portion of publications apply PCA as both a dimensionality reduction tool and a fault detection method as well. Very few papers have practiced newer anomaly detection tools such as Local Outlier Factor, Isolation Forest, or other tools that have been invented in the past two decades. As mentioned in section 3.6, PCA does not perform well when multiple operating modes exist; also, PCA is not robust to anomalies as it uses the covariance matrix [81]. In practice, false alarms occur often during transient states between operating modes.

Because of the complexity of HVAC systems[30], a dimensionality reduction is needed for data analysis. PCA and PLS are often chosen for this task. It is true that these statistical tools are very powerful and useful in many applications; why they work is basically based on correlations/covariances of the variables. We should remember that correlations do not represent causations, though they are usually indicators of some hidden relations. Even if statistical tools based on correlations, such as PCA, are able to find meaningful connections among the vast number of variables, the lack of meaningful explanation for these connections leads to questionable results. This is normal and acceptable from a data analyst's perspective, since this is considered as exploring and learning about the data/system. However, this may be considered too iffy from an engineer's point of view. Hence, we would propose that we should approach HVAC FDD by focusing on its control system. To our knowledge, HVAC FDD is mostly carried out focusing on thermodynamic/physical variables, neglecting the control system. In addition to the reasons and advantages mentioned in section 3.3, focusing on control data also carries out the dimensionality reduction task for us. This not only reduces the number of variables significantly, but also delivers results with much more meaningful explanations.[31]

Most of the HVAC FDD research work we have found in literature starts with some mathematical or statistical model (see 3.1); some are general models while others are designed for time-series data. That is, most HVAC FDD approaches have worked on model-based time-series representations. A few have used non-data adaptive or data adaptive time-series representations. As of the date of writing, to our knowledge, no HVAC FDD approach has used raw data time-series representation. In [53], time-series

[30] In the simulation model we built using Modelica in section 2, we ended up with more than ten thousand variables. Real systems may be even more complex.

[31] Dimensionality reduction not only simplifies the problem we are dealing with, but also helps us avoid running into the 'curse of high dimensionality' [55, 82, 83].

clustering for building energy patterns using raw data has been carried out, but no FDD related work. Here we would propose to use raw data representation, for we won't be assuming any models, which means that we won't be making strong assumptions on how the data would behave (e.g. If one uses a probability model approach, one would have to assume a probability distribution for the model, such as Gaussian distribution is used for GP models. Using a non-parametric model would introduce another problem, that is, how to determine a good distribution for time-series data); moreover, we would then fill up this untouched HVAC FDD area.

In section 3.7.1, we will explain how we plan to carry out our HVAC FDD task and accomplish our goals.

### 3.7.1 Our goal and proposed road map

The goal for our FDD approach would be scalability; thus, low cost and easy to set up would also be the requirements that come with it. Because there is no need for tuning involved, nor is the method designed for a specific HVAC system, the cost is cheaper to deploy. Also, the approach is not system specific; therefore, the set up process does not require to know much of the HVAC system. We only need to be able to access data. With this in mind, our work flow would be the following:

1. Collect historical weather and heat load data as training input data.
2. Run a data cleaning process, including fixing data sampling rate, discard noisy and system initialization data, approximations for on/off switch control data...etc.
3. Data clustering and exploring.
4. Organize data structure based on clustering/exploring results.
5. Rank order input weather data based on distance measures.
6. Compare and select distance measures.
7. Collect HVAC system data by Modelica simulations based on the historical weather and heat load data.
8. Find and collect the corresponding control data (manipulated variables).
9. Set up normal dataset as a baseline.
10. Collect new test sample data.
11. Run the same data cleaning process for the new sample data.
12. Rank order the new test sample with our normal dataset.
13. Find and collect the corresponding control data for the test sample data.

14. Run anomaly detection algorithms (e.g., LOF) using test sample data with normal dataset.

15. Get fault detection results.

16. Repeat the anomaly detection process and select the optimized parameters with respect to true positive rates and false positive rates[32].

17. Get final optimized fault detection results.

We will be using the HVAC model we built described in section 2, and statistical tools that are introduced in section 3. We will carry out this work flow in section 4; further details and results will be presented as well.

[32]Or draw out a ROC (Receiver Operating Characteristic) curve for comparison. See section 4.6.1.

# 4 A fault detection approach based on raw time-series data

In this section, we demonstrate an approach to conduct HVAC FDD based on raw control variable time-series data as we have mentioned in section 3.7. We will be following the work flow listed in section 3.7.1. A basic HVAC model is introduced in section 2.3; basic statistical models are briefly included in section 3.4, 3.5, and 3.6. Other changes and details will be shown in subsections below, or can be found in the references.

## 4.1 Data source

The first thing to do in order to work on a data-driven approach is to collect data! As we have mentioned in section 2, access to private building datasets is not an option for us. Our HVAC system data will be collected from simulation results of our HVAC system model built in Modelica. The models we use are built using *Modelica Standard Library* (v3.2.3) and the *Buildings Library* (Buildings 4.0.0) in OpenModelica environment (v1.11.0-64bit). Simulations are run with *JModlieca.org* (v2.1). All software packages are executed on a Intel Core i5-2400 3.10GHz CPU with 8.00GB RAM machine running Windows 10 64bit.

As for input data, we have used weather datasets based on TMY3 from NOAA (National Oceanic and Atmospheric Administration)[33]. Our simulations will be based on the bay area (San Francisco) and Boston area weather. Two datasets are chosen to show results for different weather inputs. The locations are chosen due to different weather trends; while the bay area has a rather mild change in weather throughout a year, the east coast of U.S. has an overall wider range of weather change. We will be using the heat load data from the US department of energy, datasets of 'Commercial and Residential Hourly Load Profiles for all TMY3 Locations in the United States'[34], which is a dataset generated by *EneryPlus*[35] using TMY3 (Typical Meteorological Year 3)[36] data. Again, we will choose two different heat load datasets as inputs for comparison. Since our work is mainly targeted for commercial office buildings, we will be using office heat load datasets for the experiments here.

In this section, we will start with a single room model, a HVAC system with a single room, an AHU, a cooling loop, and a heating loop. This set up is shown in figure 4.1.

[33] https://www.climate.gov/maps-data/dataset/past-weather-zip-code-data-table

[34] https://openei.org/doe-opendata/dataset/commercial-and-residential-hourly-load-profiles-for-all-tmy3-locations-in-the-united-states

[35] EnergyPlus is another building energy simulation program funded by DOE. Compared to Modelica, it is more focused on the whole HVAC system in the building for designing. As for our purpose, we need transient time-series data with more freedom to customize different models; Modelica suits us better. See https://energyplus.net/. Some researches use EnergyPlus as a tool, e.g., in [84], EnergyPlus is used to do research on HVAC operational faults.

[36] https://rredc.nrel.gov/solar/old_data/nsrdb/19912005/tmy3/

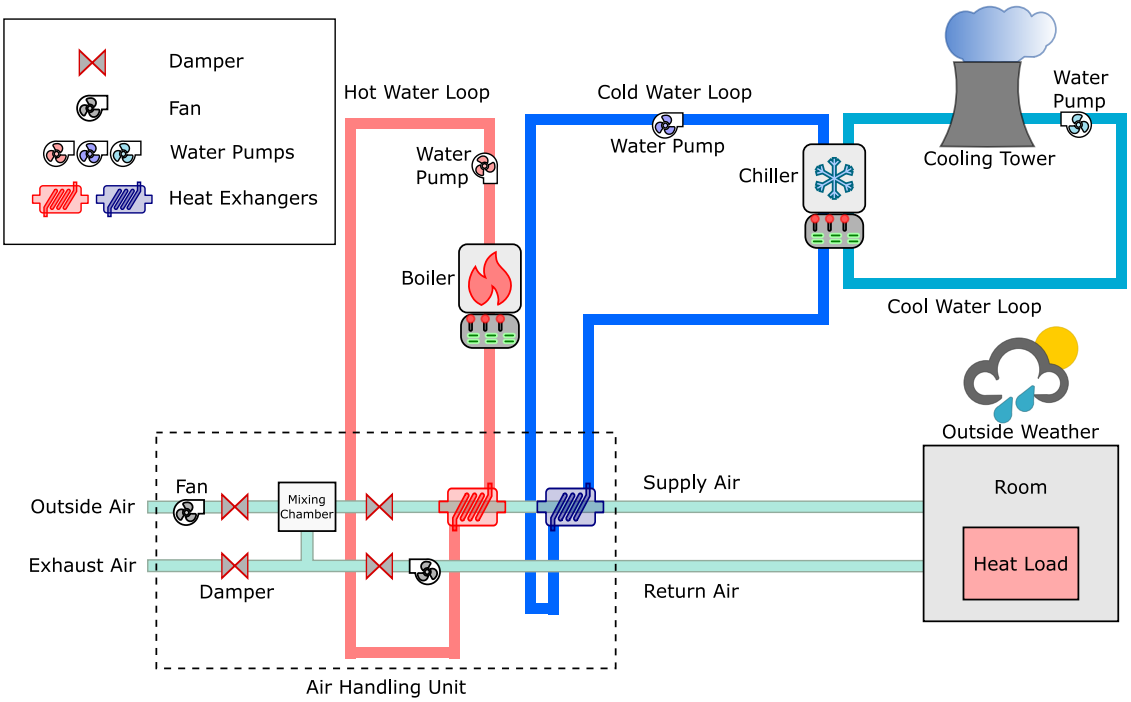



**Figure 4.1:** A simple HVAC model with one room and one AHU.

The corresponding Modelica model is shown in figure 4.2. Here we briefly explain about the Modelica model shown. The circle icon at the upper left is an air flow source with prescribed temperature and pressure values given by a time table component; this time table uses the datasets we obtained from NOAA as we have just mentioned. The circle icon at the lower right is a mixing volume component; we use it to simulate a simple room with heat load. The block which is labeled as system at the lower left is a `system` component. It is used for "system properties and default values (ambient, flow direction, initialization)," it defines the ambient parameters, e.g., gravity, ambient temperature, ... etc. The component which has ports with labels: OA, SA, RA, and EA is the AHU component; its model is shown in figure 2.5. The label 'M' stands for a motor which controls a damper. A list of acronyms can be found in appendix A. Most of the component parameters (library components) we tend to leave them as default values; we fill in required inputs and adjust parameters only if necessary. For example, in this particular model, we have used a constant pressure fan in the AHU; the pressure rise is set to $2000[Pa]$[37] so the air flow rate would be around $1[m^3/s]$, which is about four times of the required ventilation minimum (air flow rate is higher because heat is also being taken away or added in by air; this depends on both weather and heat load), depending on the damper position (controlled by a PID controller). Our default

[37]This number depends on the system friction settings as well; we ended up with this number to ensure that the air flow is sufficient. This fan model also obeys fan affinity laws, so one can adjust the parameter setting accordingly.

room temperature setpoint is set to 22°C.

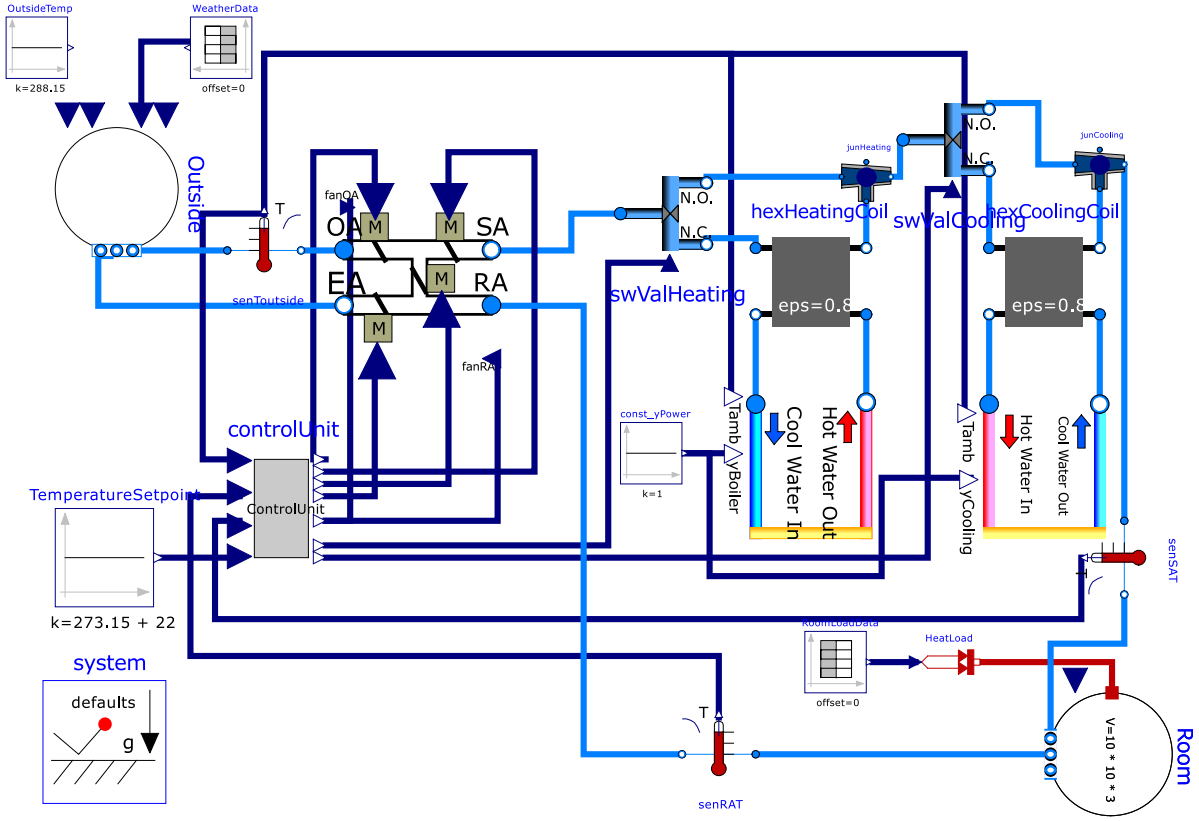



**Figure 4.2:** The corresponding HVAC model of figure 4.1 in Modelica.

## 4.2 Data cleaning

Generally speaking, when doing data analysis, people will spend most of their time working on data cleaning. That is, we have to make sure our data is in a usable, informative, and meaningful form. For instance, most datasets are either collected by man or documented by sensors in the field. This means that it is not uncommon to find errors, missing values, incomplete data, incompatible format, sensor failure, misplaced sensors, wrong installations, noisy data due to environment, ... etc, all kinds of things could happen.

Fortunately, the datasets we need for our HVAC system simulations are available on the Internet. The hourly normal weather datasets we have used are TMY3 Class I, according to NREL, 'Class I sites are those with the lowest uncertainty data'. Therefore, we only need to make sure the time is matched and units are compatible. This is also the case for our heat load data, according to DOE, 'hourly load profile data for 16 commercial building types (based on the DOE commercial reference building models)'.

However, HVAC system variable simulation data are a little different. Since the parameters' initial conditions of the HVAC model won't be exactly the same as input datasets, the system will have an initializing period to put everything into its 'normal' state. Thus, we will remove the first couple of hours to avoid noisy and unstable data. Another thing is that depending on the ODE solver being used, the output results are not necessarily linear in time. This is the case for JModelica.org's default (also recommended) ODE solver *CVode*. We will have to make sure the time intervals are consistent and fixed, for the algorithms we use are based on this assumption. Some data processing task would need to be carried out before applying further work on these output datasets[38].

Furthermore, while working on the datasets, we will have a lot of tables. Conversions between parameters, formats, pivot tables, data dimensions, and units must be done with care, or we may end up with strange and unrealistic results. It would be easier to do it right in the first place than to 'debug' such errors later.

One more thing to take note is that we cannot blindly feed in the data into our simulation model without inspecting the data first. Data cleaning is not just filling in missing data or discarding erroneous data. The data content may be correct, but people could be using them wrongly. One would have to have a good understanding of the dataset to make good use of it. We will show this in the next section how the results could be different if we did not do clustering and explorations before using these datasets.

[38] OpenModelica's default solver gives a fixed time interval for its output. Also, each Modelica editor comes with multiple solvers, the default is usually the recommended one, the user would have to check the solvers before using.

## 4.3 Clustering and exploration of input data

Starting with the temperature profiles, we simply run a DBSCAN clustering algorithm with the weather temperature data. We did not adopt k-means here, because it is difficult to determine the number of clusters. The statistical clustering tools we use here are introduced in section 3.4. The results are shown in figure 4.4. Great, everything works nicely.

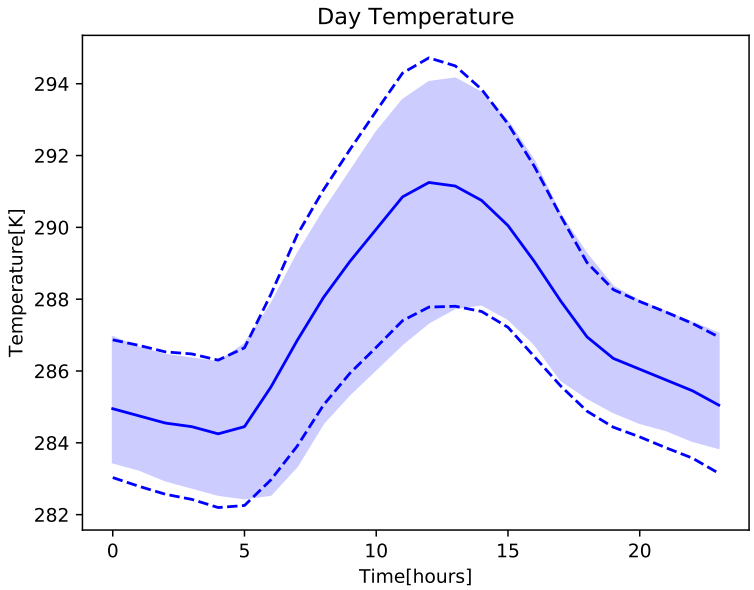
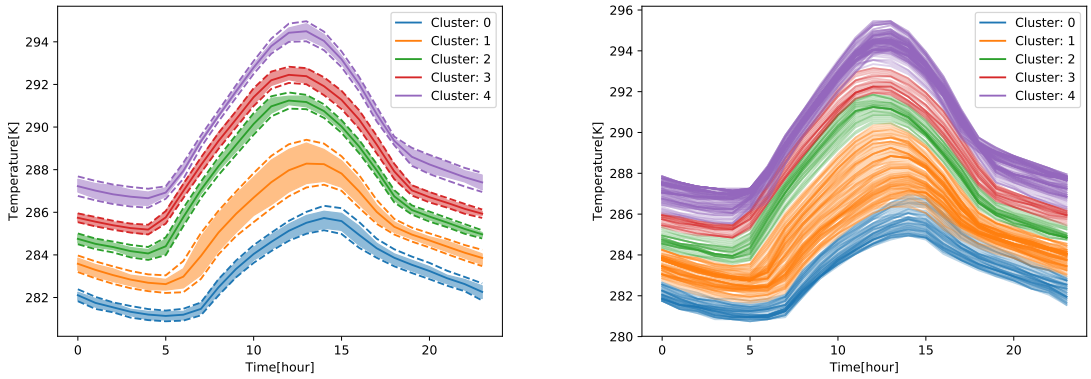



**Figure 4.3:** Temperature hourly data for San Francisco area. The solid line is the average, two dashed curves are one standard deviation from the average, and the color filled area stands for temperature data between the first and third quantile.

At first glance, the results are reasonable and look good. However, we soon realize the results shown in figure 4.4 do not really tell us much. This is actually because the 'shapes' of the temperature profile curves are very similar throughout a year for a fixed regional location. We can arbitrarily group neighboring temperature profiles together and then end up with very similar results that look reasonable. Temperature data is our input data for the HVAC system; thus, how to compare two different temperature data sequences for two days is important. Apparently, from the clustering results we do not gain much information regarding comparable weather data. Nonetheless, we did learn that clustering algorithms are not the tools we are looking for to make two sequential data comparable; what we need here would be a pattern matching technique instead of clustering. We will leave this discussion for later (see section 4.4). Also, the 'shape' is an important feature of the temperature profiles. Collecting more data samples will not help, since the new data samples will be absorbed into their nearest neighboring clusters, making the clusters grow larger; therefore, causing the boundaries to expand.




**(a)** Annual weather data profiles clustered with DBSCAN.

**(b)** Annual raw weather data profiles color coded with DBSCAN clustering.

**Figure 4.4:** Annual TMY3 hourly normal weather data for San Francisco bay area.

This does not help us define the pattern of a data sequence.

HVAC systems are built on top of physics; why they work are because of the physical laws. That is, for inputs given, the outputs should be deterministic. To rephrase this, with the same inputs, we should always end up with the same outputs. However, in real cases, the measurable inputs are the weather data; heat loads generated in and out of the HVAC system are, in general, unlikely to be measurable. As discussed in section 3.3, these inputs, or external disturbances (figure 3.2) in a controller's point of view, are considered as noise to the system. While weather data are measurable, the biggest uncertainty left is the heat load.

For the experiment set up, we are given heat load data for commercial office buildings from the DOE datasets. In experiments and real cases, we would not know how the heat load data would look like (blind test); however, we could still learn some prior knowledge regarding the heat load data's behaviors and trends by inspecting them first. As usual, we look at its average, standard deviation, and quantiles first; see figure 4.5. According to the plot, we see that the heat load seems to have a very wide range, especially during working hours, with about a 100% range difference. This indicates that the heat load is a very big uncertainty factor, and we should be careful while dealing with it.

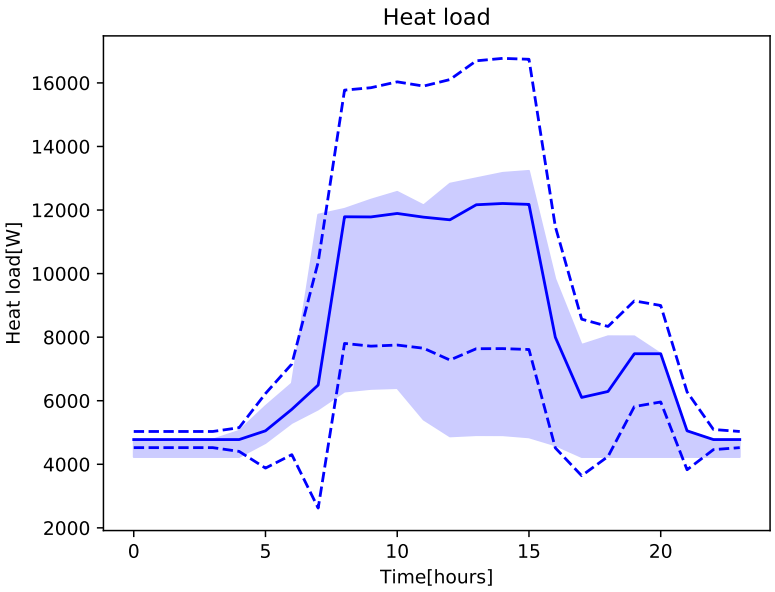
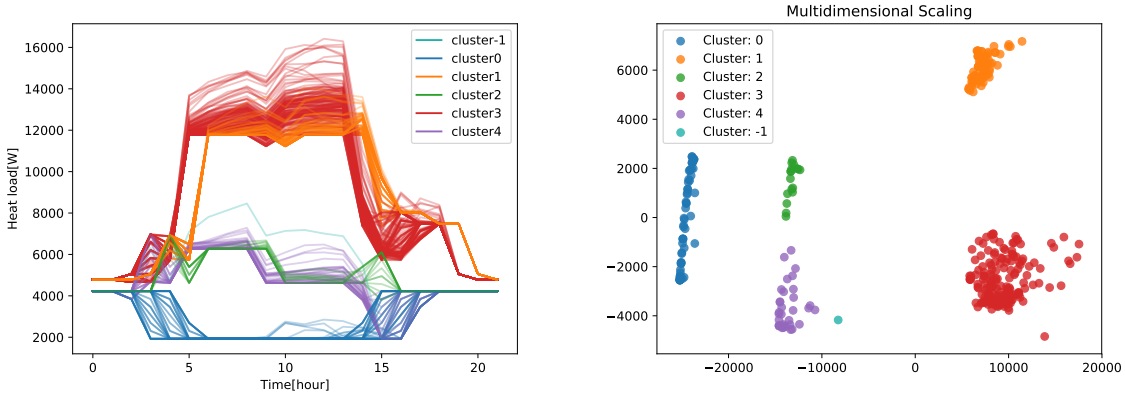



**Figure 4.5:** Heat load hourly data for commercial offices. The solid line is the average, two dashed curves are one standard deviation from the average, and the color filled area stands for heat load data between the first and third quantile.



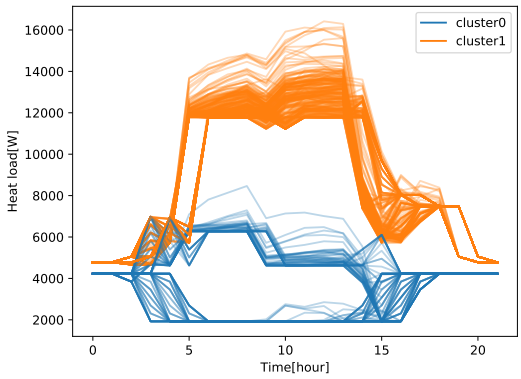

(a) Annual heat load data profiles clustered with DBSCAN.

(b) Annual heat load data interpreted with MDS and color coded with DBSCAN clustering labels.

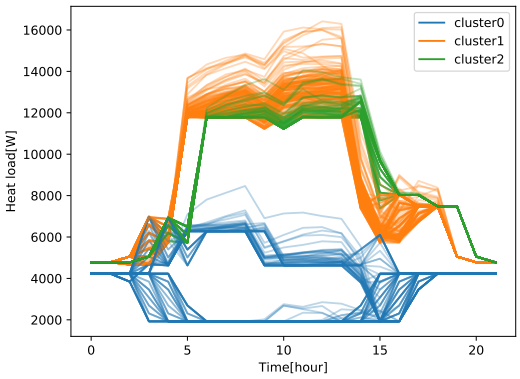
**Figure 4.6:** Annual hourly normal heat load data for commercial offices clustered with DBSCAN.

Again, we run DBSCAN and MDS on the heat load data for further inspection. The results are shown in figure 4.6. By viewing the plots, we soon find that the cause of a big variance is not because of randomness (Yes, there is always some randomness in the data; however, randomness is not the main factor causing the big variance here.);

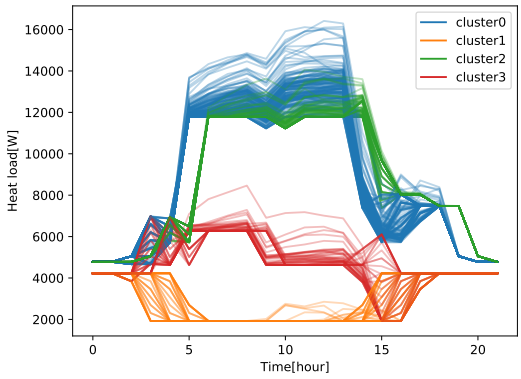
obviously, there are certain patterns or modes in the data. It is the differences among these patterns that are causing the big variance in heat load data. Now that we have an idea about what might be going on with the heat load data, we would like to investigate more before we move on. Next we apply k-means clustering to the data for $k = 2, 3, 4, 5$; the results are shown in figure 4.7.



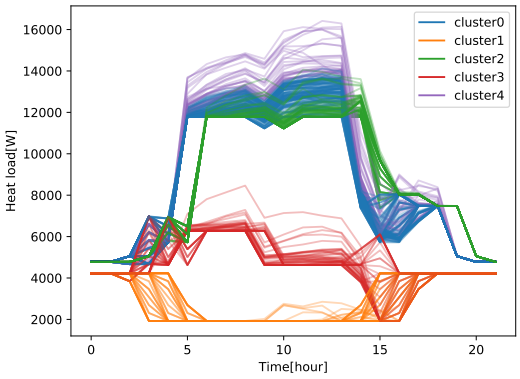

**(a)** Annual heat load data profiles clustered with k-means clustering($k = 2$).




**(b)** Annual heat load data profiles clustered with k-means clustering($k = 3$).




**(c)** Annual heat load data profiles clustered with k-means clustering($k = 4$).




**(d)** Annual heat load data profiles clustered with k-means clustering($k = 5$).

**Figure 4.7:** Annual hourly normal heat load data for commercial offices clustered with k-means.

At this point, one would probably already have guessed why the heat load dataset has these kinds of patterns, and probably have guessed correctly. But before sharing what we believe is the reason, let's do one more clustering. This time we run OPTICS fully automatically and implement the $\xi$-steepness clustering algorithm, only setting the $MinPts = 15$ and $\xi = 0.01$ parameters. In figure 4.8, note that the $\xi$-steepness clustering algorithm generates a hierarchical clustering structure; that is, there could be smaller clusters inside a bigger cluster; thus, a data sample could belong to multiple clusters. This means that a data point is in a smaller cluster that is inside a bigger

one itself. The data points in figure 4.8 are plotted according to the cluster numbering order; therefore, some data points which have multiple labels will only be drawn with one color (with a bigger ordering number), for one color (smaller ordering number) will be covered by the other one. It gets a bit messy to plot out the results with this structure, so we will only plot out the max clusters which are mutually exclusive (no overlaps). This is shown in figure 4.9.

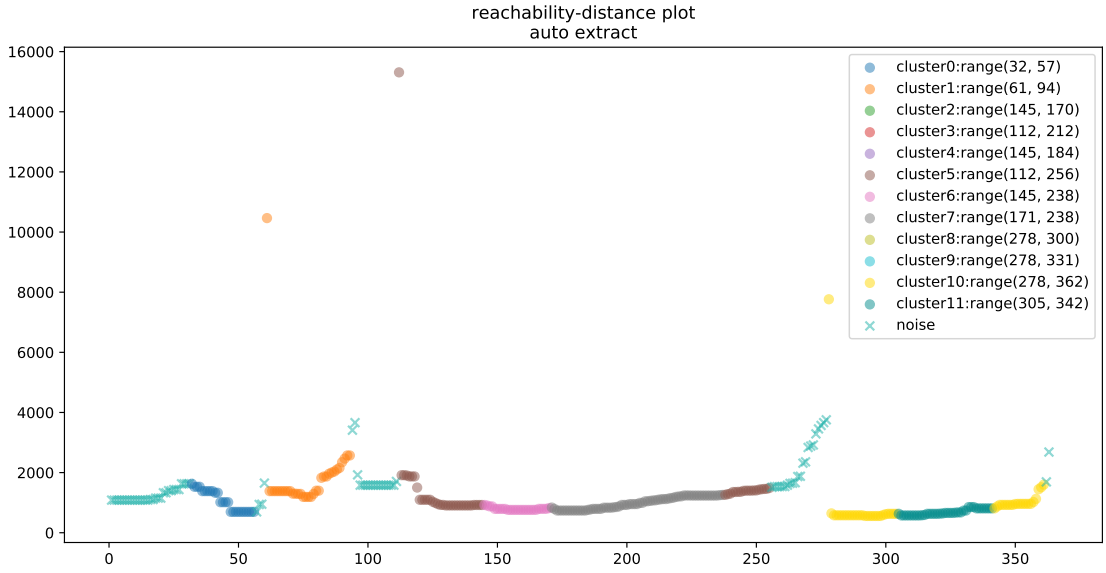
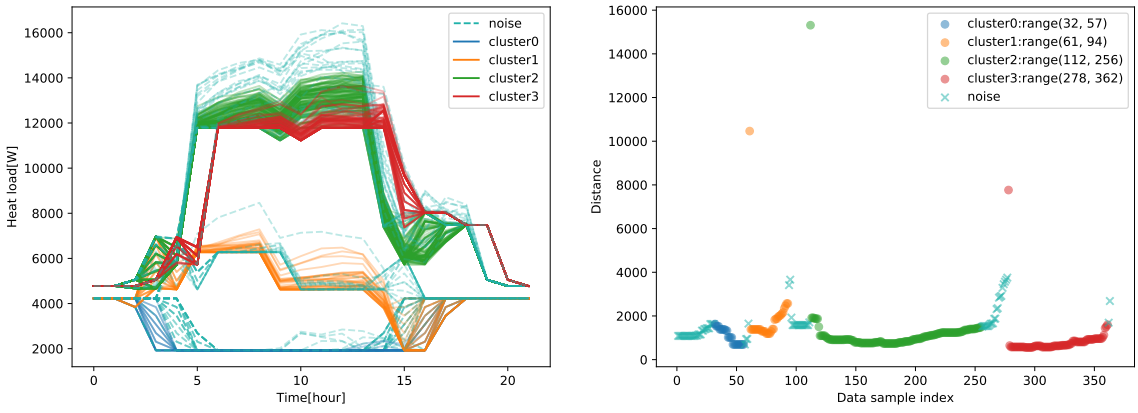



**Figure 4.8:** Reachability-distance plot of OPTICS automatic clustering annual heat load hourly data.




**(a)** Annual heat load data clustered with OPTICS. **(b)** Reachability-distance plot.

**Figure 4.9:** Annual hourly normal heat load data for commercial offices clustered with OPTICS.

It is interesting to find that all of these clustering methods give us very similar

results, but are also slightly different in some ways. Anyways, after doing these explorations, we end up believing these certain patterns are due to weekdays and weekends. Power consumption patterns are different for weekdays and weekends; this seems very reasonable. In order to prove this, we plot out all heat loads based on their day order and cycle over seven days. Each classified heat load data are drawn in a subplot. Figure 4.10 verifies our guess nicely. However, we do spot some of the days in each group that seem like they are different and should belong somewhere else. After cross comparing the dates with a calendar, we have confirmed that those remaining heat load patterns that look like weekend patterns are national holidays. We then list out the weekends and holidays, and label them as non-workdays. The final results are shown in figure 4.12. A monthly average heat load profile is drawn and shown in figure 4.11; we see that there is a seasonal trend in the heat load data, which means heat load is dependent to seasonal weather. Also, we can spot an offset between summer and winter months, a phenomenon resulted from daylight saving time.

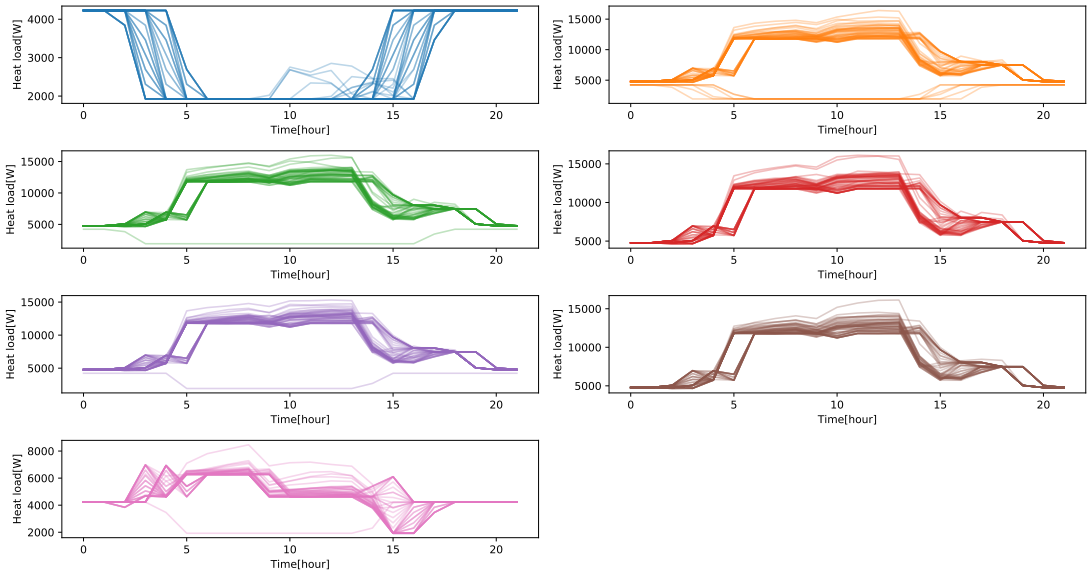



**Figure 4.10:** Heat load hourly data classified in a weekly cycle. Subplots starting from the top left represent Sundays (blue), Mondays (orange), Tuesdays (green), Wednesdays (red), Thursdays (purple), Fridays (brown), and Saturdays (pink).

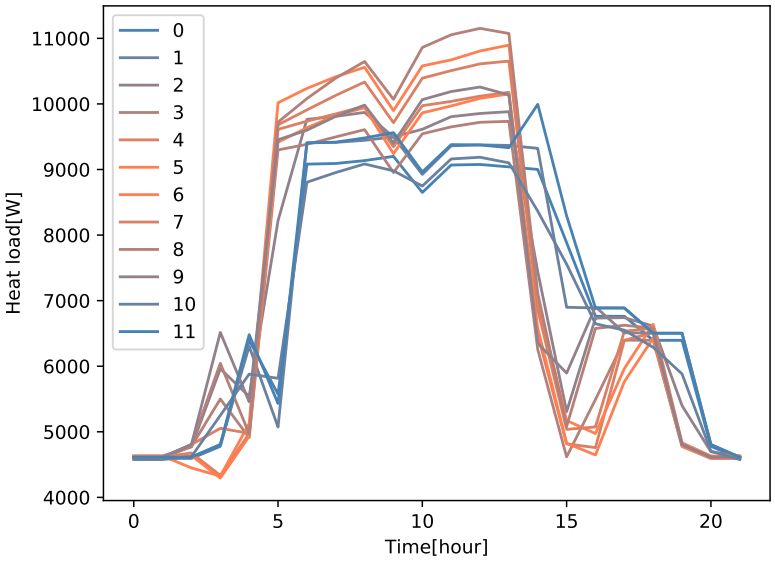



**Figure 4.11:** Monthly average heat load hourly data. Label 0 stands for January, label 1 stands for February, and so on.

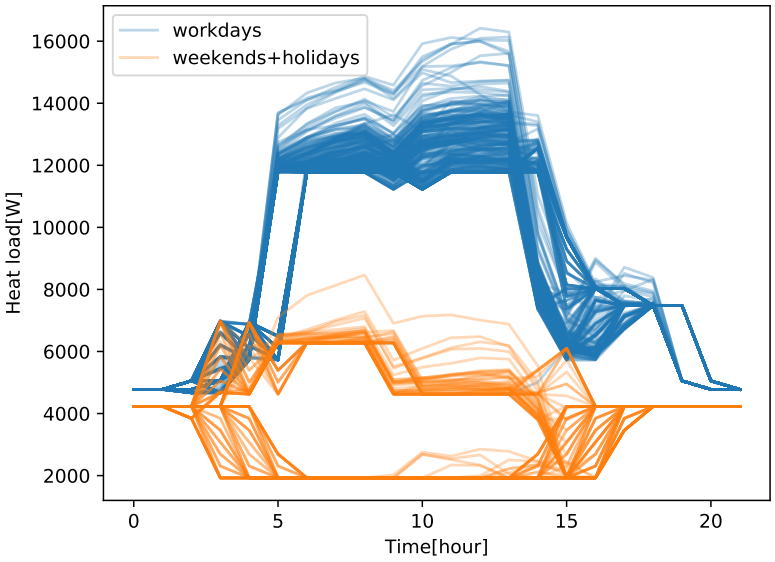



**Figure 4.12:** Heat load hourly data classified as workdays v.s. non-workdays.

A fun fact is that this workday v.s. non-workday result is exactly the same as the result of k-means clustering with a $k = 2$ parameter setting (Yes, we have verified this.). It would be very hard for us to set the parameter value to 2 without any additional

information though. These clustering methods not only were able to classify workdays with non-workdays, they were also successful in telling Sundays and Saturdays apart. Moreover, an offset caused by the daylight saving time is also detected if we set the parameter right, though not that obvious.

Before carrying out the exploring work, we did not have the prior knowledge or information about the heat load dataset. Without knowing all these, blindly feeding all these datasets into our HVAC model and fault detection algorithms, we ended up with very poor results. After knowing there are multiple patterns in the heat load data, we were able to narrow down the uncertainty; therefore, we were able improve our results significantly. The work of this section shows us that having a good understanding of our data is as important as data cleaning.

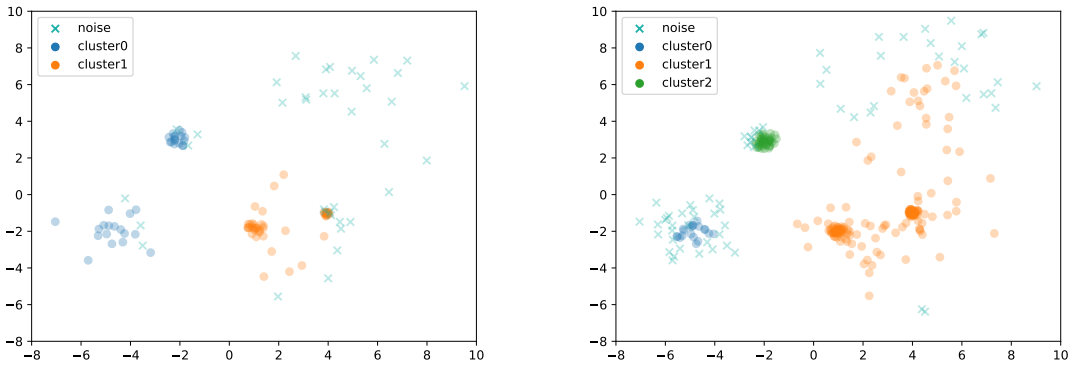
## 4.4 Time-series rank ordering

Our main purpose for time-series rank ordering is to compare the time-series data; this is mentioned in section 3.4.4. In addition, it has been discussed formerly that HVAC systems are built on top of physics; therefore, we are making the assumption that with similar inputs, the system will have similar outputs. Although there may be things which are designed to have very different results given two very similar but not identical inputs, such as a hash function which is widely used for integrity checks, electronic signatures, and block chains, for most physical systems, the response to close inputs do not have kinks or abrupt changes in general. As a result, we are looking for a meaningful way to compare two days of weather data.

Given a weather day data in sequential form, in order to find comparables and make comparisons, we would have to put 'similar'[39] days together. This grouping process is kind of like clustering, but not exactly. The reason why clustering is not a good choice for grouping similar daily data for our case is because for a data sample in a dataset, the distance between itself and other data samples could be very far. This depends on the size and density of the cluster the data sample is in. For example, in figure 4.4b, we see that weather data profiles are clustered into five clusters. It is obvious that cluster 2 (orange) has the largest range/variation by simply reading the plot. Let us look for the largest distance between two data samples within the same cluster; from the figure, we see that in cluster 2, the farthest distance between two data samples are much larger than other clusters. If we pick the top and bottom two data samples from cluster 2, the distance between these two are much larger than their neighbors that happen to be in different clusters. Depending on the data structure, clustering algorithm used, and how many data points are sampled, we would end up having different results; the similar group a data sample is in varies. What's more troubling is that as the size of a dataset grows, it is very likely that the sizes of clusters grow as well, causing dissimilar data samples being put into the same cluster. That is, members of

[39]There are many ways to define similarity; here we will be using distance measures we talked about in section 3.4.2.

a cluster become less and less comparable, while they are suppose to be similar data samples. In order to depict this phenomenon, as an example, we randomly sample 120 and 300 data points from a distribution with multiple modes. In figures 4.13a and 4.13b, 120 and 300 sampled data points are clustered with OPTICS ($MinPts = 15$) and plotted out. We have noticed two things; one is that by increasing the number of data points, cluster 0 in figure 4.13a has split into two smaller clusters. This is because that the number of points in each sub-cluster has surpassed the parameter $MinPts$; based on density estimations, they are considered as independent clusters (cluster 0 and 2 in figure 4.13b). Second, cluster 1 has absorbed neighboring data points and has grown larger. There is nothing wrong here, since this is what clustering algorithms do; however, for our purpose, we do not want clusters grow large as the distance between data points within a cluster may become too large.




**(a)** 120 data points clustered with OPTICS. **(b)** 300 data points clustered with OPTICS.

**Figure 4.13:** Clustering result affected by the number of data samples. Data points are randomly generated in 2D space. Python code used to generated these toy examples can be found in appendix D.4.

Pattern matching techniques, such as SAX (Symbolic Aggregate approXimation, see section 4.4.1), would probably fit our purpose better than clustering, since they are designed to find matches based on similarities. Nevertheless, using sophisticated pattern matching tools is probably an overkill for such a task, and could potentially introduce additional work without great improvements. For example, SAX and iSAX are regarded as very powerful tools without a question; however, what they do to time-series is delivering a discretized approximation output. This is very useful when we have datasets having more than millions of time-series data samples. Approximations of similar patterns can be found much faster than going over the whole dataset. But for our case, weather data is a very specific type of data. From figures 4.3 and 4.4, we find that daily temperature profiles are rather stable, and very similar shape-wise. These profiles have a seasonal trend of offsets going up and down, but their shapes look-alike in general.

We would propose to use rank ordered time-series data for comparing input weather temperature profiles. First, rank ordering is relatively simple and straightforward; we all know that simple means fast in computations. Second, as mentioned, using sophisticated tools for a simple task is not only an overkill, but also introducing more work. Take SAX for example, SAX does not really help us much here, for temperature profiles are similar in general; if we adopt some discretization with a resolution too low, all profiles will be classified as the same symbolic pattern. On the other hand, if we choose a discretization with high resolution, we would be able to tell the profiles apart, but then we won't have comparables, for they are considered different. Moreover, the temperature is usually low during early mornings and after sunsets; symbols reserved for high temperatures for these time periods are basically never used. The opposite is true for low temperature symbols used during noons. Third, rank ordering is actually having a 'cluster' tailor-made for every time-series data sample, meaning each data sample is automatically being at the center of its rank ordered group. This gives us a good comparable group for the data sample is always at the center. Therefore, we think using a simple rank ordering method would be a better choice.

One would find that in the field of HVAC FDD, how to compare two weather data is rarely seen or heard of. This is due to the fact that most approaches found in the literature use model-based methods (see sections 3.1, 3.4.1, and 3.7). One of our goals is to fill up this untouched area without using models to represent the data; thus, improving scalability.

Although we are working on raw time-series data, we still need to use distance measures to compare data profiles. We will perform a comparison of five different distance measures used for rank ordering. We arbitrarily select a day (day 123) as a reference, and compare annual daily temperature profiles with it. A list of days are sorted with an order of the distance measure scores for each distance measure. The closest day would be put at the top of the list, and the most different one would be put at the end. We then compare how similar these lists are. Two comparison methods are conducted and a visualized order similarity plot is shown in figure 4.14.

One way to compare the ordered lists is to find matches in the lists. We start by taking two lists and go through each member in the lists. If members with the same index match, we would add 1 to the accumulative score. Once we have finished with all members, we divide the accumulative score with the total number of members and result with a similarity ratio. All five distance measures are compared to each other, and a similarity ratio matrix is shown in table 4.1. A similar method can be done by computing the distance between members with the same indices in the lists. We would have an overall distance sum after summing over all member distances in the lists. Again, we do this comparison for five distance measures, and the results are listed in table 4.2. In both tables, the last column is the sum for each row; this gives us an idea about how similar/dissimilar each distance measure is compared to other measures.


**Figure 4.14:** Temperature day data rank ordered with different distance measures. Test day is arbitrarily chosen.

| | Absolute | DTW | Euclidean | HighLow | SAX | Sum |
|---|---|---|---|---|---|---|
| Absolute | 1.000 | 0.392 | 0.667 | 0.106 | 0.144 | 2.308 |
| DTW | 0.392 | 1.000 | 0.425 | 0.125 | 0.133 | 2.075 |
| Euclidean | 0.667 | 0.425 | 1.000 | 0.094 | 0.125 | 2.311 |
| HighLow | 0.106 | 0.125 | 0.094 | 1.000 | 0.069 | 1.394 |
| SAX | 0.144 | 0.133 | 0.125 | 0.069 | 1.000 | 1.472 |

**Table 4.1:** Similarity ratio matrix.

| | Absolute | DTW | Euclidean | HighLow | SAX | Sum |
|---|---|---|---|---|---|---|
| Absolute | 0 | 510 | 166 | 2144 | 1138 | 3958 |
| DTW | 510 | 0 | 454 | 2192 | 1194 | 4350 |
| Euclidean | 166 | 454 | 0 | 2264 | 1080 | 3964 |
| HighLow | 2144 | 2192 | 2264 | 0 | 2966 | 9566 |
| SAX | 1138 | 1194 | 1080 | 2966 | 0 | 6378 |

**Table 4.2:** Distance rating matrix.

Figure 4.14 demonstrates the ordering for using these five different distance measures. Compared with the selected reference day, the more similar (dissimilar) the days are, the redder (greener) are the colors. We see that the absolute, Euclidean, and DTW distances look closer than the other two, which is consistent with the results shown in tables 4.1 and 4.2.

From the results, we find that when it comes to rank ordering temperature profiles, these distance measures all deliver similar results. The Euclidean distance seems to work best here, and people have commented that it is a competitive distance measure in many applications [53, 54]. SAX and high-low[40] method gives slightly different ordering results, which is reasonable since SAX uses a discretized approximation for calculations and high-low only uses two scalar values to sort the order. It is surprising that the high-low method works well by just using so little information and computations. No wonder for a local weather forecast, the high and low temperatures are good enough for most people. We believe that this surprising result is due to the fact that temperature profiles are rather stable and similar in shape as we have discussed previously. On the other hand, DTW distance gives us a very similar result to the Euclidean distance. The advantage of DTW is that it is good at measuring dissimilarities shape-wise, with a much higher computational power requirement as a trade-off. However, as mentioned, temperature profiles for a local region is generally similar in shape; DTW won't provide us better results. Furthermore, for long sequential data, DTW distance degenerates to the Euclidean distance [54, 85]. Because of our limited computational resource and very little result differences, we will leave out DTW in later data analyses.

### 4.4.1 Piecewise aggregate approximation (PAA) and symbolic aggregate approximation (SAX)

What Piecewise Aggregate Approximation (PAA) does is rather simple; given a fixed interval (window size), it computes the average of all values within the window for each interval. That's it. PAA can be applied to time-series data as an approximation and coarsen the time intervals. This could be useful for working on signals of on/off switches, for sampling the state of an on/off switch for an instant with an infinitesimal small time interval would not be informative. PAA can also be used as a non-data adaptive time-series representation. Another application of PAA is it is used as a middle step for converting a time-series to SAX.

Symbolic Aggregate approXimation (SAX) [54, 86, 87] is a data adaptive time-series representation. Before transforming a time-series to SAX, it will first be converted to a PAA sequence, and then the PAA sequence will be converted to SAX. SAX is the first symbolic representation for time-series that allows for dimensionality reduction and indexing. Also, due to its characteristic of indexing, it can be used for time-series pattern matching applications [30]. We have used SAX as a time-series representation in section 4.4 for rank ordering temperature profiles.

In order to illustrate this, we take two sequences for example. The first sequence is the output of a sine wave, and the second one will be random numbers. We apply PAA and SAX on them and plot out the results in figure 4.15. The blue solid curve is our sine wave, and the semi-transparent coral solid curve is the sequence of random numbers. The solid green and coral lines contained in bands are the PAA results. We

[40] high-low distance is calculated by taking only the highest (largest number) and lowest (smallest number) temperature data points in the time-series data.

can see that within a window (size is set to 8 here), the line sections are horizontal with the mean value of its corresponding point values. The bands are the SAX results. In this example, we have set the cardinality to be $2^4 = 16$; each interval is assigned to a symbolic value (e.g., a, b, c, etc). Every band contains its corresponding PAA mean value in its interval. SAX then gives a symbolic sequence output.

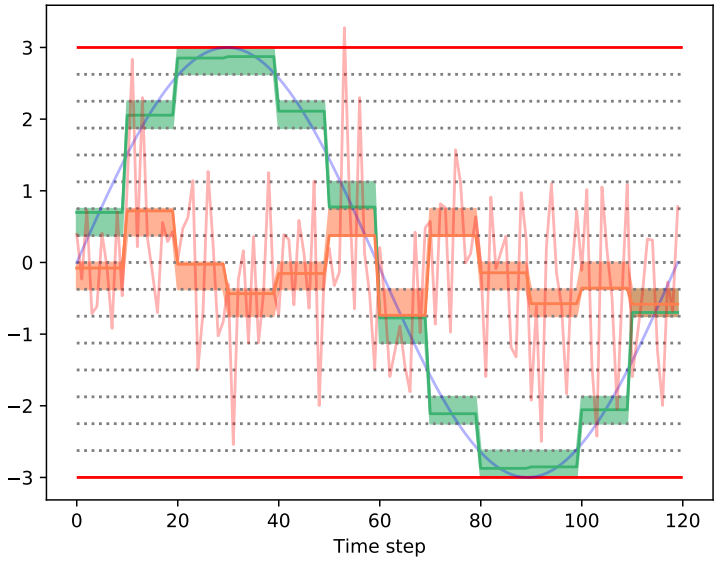



**Figure 4.15:** A PAA and SAX example.

## 4.5 Simulations with Modelica

In this section we will discuss how we set up our model in Modelica for simulations. The reasons why and how to build a Modelica model are discussed in section 2. The model used for this section is shown in figure 4.2. This model simulates a rooftop unit used for a single room with a CAV (Constant Air Volume) set up. Another model variant will be tested in section 5.

The input datasets for our models are cleaned and restructured. Weather data are put into the system as outside environment conditions, which can be monitored by sensors. Heat load data are also plugged into the HVAC model but without sensors monitoring, for most cases in the real world we won't be able to monitor the overall heat load dissipated into a building. Heat load data are used for simulating possible heat load patterns, which should have a similar pattern from day to day and change seasonally with the weather data. Therefore, heat load day data are selected based on the date of weather data (with a random ± 2-day range to introduce some randomness) to reflect seasonal changes, but we will separate non-workdays from the heat load data

| Fault | Fault description |
|---|---|
| Normal | A normal operating dataset for testing purpose. |
| Fault 1 | A stuck/clogged hot water valve in the AHU. |
| Fault 2 | A leaking air duct. Supply air leaking to the outside. |
| Fault 3 | Cooling loop subsystem malfunctioning with a lower efficiency. |
| Fault 4 | A malfunctioning thermostat. Reads the RAT incorrectly by a 0.7°C offset. |
| Fault 5 | Same as fault 2, but with a much minor leak. |
| Fault 6 | Same as fault 4, but with a smaller offset of 0.1°C. |

**Table 4.3:** Simulated faults and descriptions. A more detailed description list can be found in appendix C.1.

in order to avoid large variance and uncertainty of the data, since heat load is strongly dependent on both weather and energy usage.

A normal operating condition dataset is generated by simulating this model for 362 days (approximately a year). This dataset will be used as a baseline/normal reference for the HVAC system. New datasets will be compared with this normal reference by anomaly detection algorithms. Another normal condition dataset with randomly selected dates is generated for true positive tests. We have come up with six different faults and have built variant models to run simulations. The faults are listed in table 4.3. Each of these testing dataset contains at least 100 samples (days) of data with randomly selected dates for weather data and corresponding heat load data with a random $\pm$ 5-day range and excluding non-workdays for their dates. The datasets we used as inputs are described in section 4.1. Note that all faults we have introduced to the HVAC system are hidden faults, that is, they are minor faults to the overall system, and their affections are compensated by the control system, so residents inside the building will not notice any differences.

Simulation results of a selected number of variables are then saved for each day. Not all variables are saved since the total number of variables are nearly 20,000 with 86,400 (secs) for each day. If we stored all data, it would have a size of $20,000 \times 86,400 \times 362$ for a single normal operating dataset alone. This would overwhelm our PC and consume all of our RAM and HD resources, and it would be very time consuming to process all these data. We will save general sensor and control data for our work. A simple PCA is conducted, but the results are not useful; in addition, it is not practical to trace all variables for explanations, nor is it likely for us to convert all variable units and normalize them. Not to mention, these variables may have different properties, e.g., an on/off switch signal may have a high variance, but is probably not a 'principal component' of the system; some data processing should be done first, that is, blindly applying PCA would not be a good idea. This is one of the reasons why we would focus on manipulated variables of the control system.

## 4.6 Fault detection results and discussion

Simulation data of our HVAC Modelica model are collected and stored. The next step would be running fault detection tests on these datasets. We will use anomaly detection algorithms, Local Outlier Factor and Isolation Forest (see section 3.6), for this task.

We construct a rank order list of the training data for each time-series data sample in the testing dataset; the testing data sample is then added to this group to form a testing group sample. Based on this testing group, we look up their corresponding manipulated (control) variables. The two anomaly detection algorithms are then run using the manipulated variables of this testing group sample. Note that we have only used the manipulated variables from the control system (and control components/units); this greatly reduces the number of variables we have to handle while providing meaningful information at the same time. We have run these tests with four different distance measures, the absolute distance, Euclidean distance, high-low distance, and SAX. We have compared these distance measures for rank ordering in section 4.4, we will see whether there will be differences in fault detection results if one distance measure is chosen over the others. Also, fault detection tests are run for four different HVAC datasets using different input settings (two different weather datasets for different regional locations, and two simulated heat load datasets for office buildings are used).

The results are shown in figures 4.19, 4.20, 4.21, and 4.22. From these plots, we have first noticed that all four distance measures deliver very similar results. This is not surprising after we have compared their differences for rank ordering time-series; the similarity in final results of fault detection rates verifies our conclusion discussed in section 4.4. We have found that the high-low distance measure gives slightly lower fault detection rates in all of our experiment set ups; this is probably because using only two values to represent the temperature profile for a whole day is less discernible after all. However, with such little information, results of the high-low distance are already very impressive. On the other hand, by comparing the figures, we have found that the overall fault detection rates seems to be higher for office type 1 compared to type 2, especially for less detectable ones, such as fault 3. In order to check this, we pulled out the heat load datasets and plotted out their profiles; see figures 4.16, 4.17 and compare them with figures 4.5 and 4.12. We see there is obviously a much larger variance for the heat load type 2 dataset; this may be the reason for lower fault detection rates.

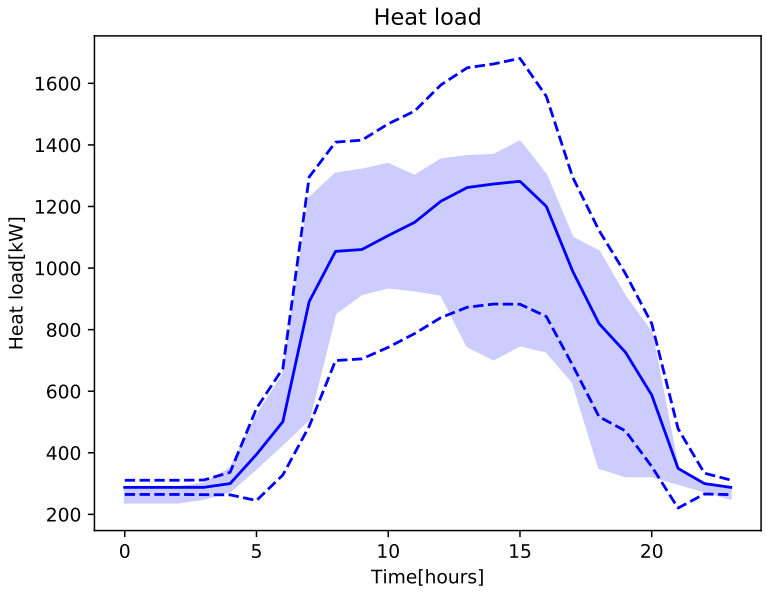



**Figure 4.16:** Summary for heat load data of office building type 2.

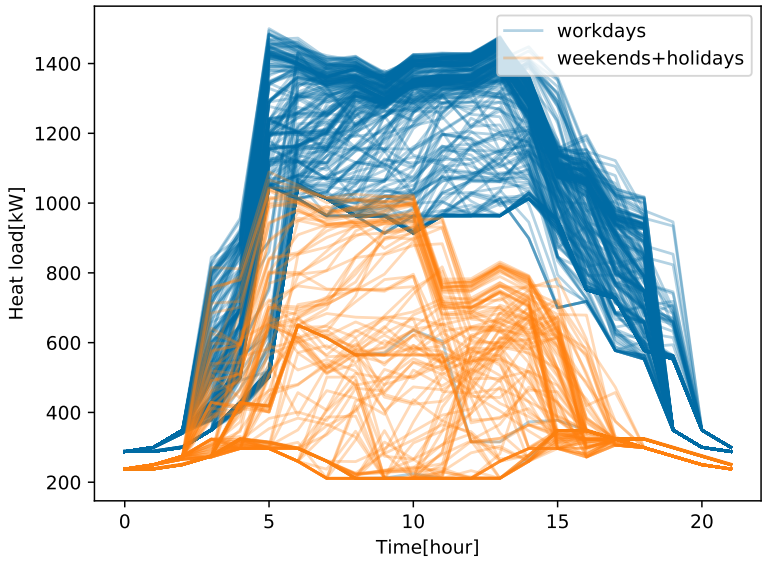



**Figure 4.17:** Heat load of office building type 2 classified as workdays v.s. non-workdays.

### 4.6.1 Parameter selection for anomaly detection algorithms

We have presented the results of our approach for HVAC FDD using raw control time-series data; however, in order to get a better picture of how our approach performs, we will introduce the Area Under Receiver Operating Characteristic (AUROC) [88, 89] curve.

In contrast to Akaike information criterion (AIC), Bayesian information criterion (BIC), and Cross-validation [61, 90][41], which are popular model selection methods used for regression [91] and ARIMA models [44], AUROC is a popular way to present the performance of a classifier. The basic idea of AUROC is to compare the true positive rate (TPR) with false positive rate (FPR) (see section 1.2.1). Let us reuse table 1.1 with some rephrasing and modifications. In table 4.4, we clearly see that there are four possible outcomes from a binary classifier. The diagonal entities, true positive and true negative, are the cases for a classifier to correctly classify data. In order to see how a classifier performs, one would calculate its TPR and FPR:

$$TPR = \frac{\text{Number of true postives}}{\text{Number of true positives + Number of false negatives}}$$

$$FPR = \frac{\text{Number of false positives}}{\text{Number of true negatives + Number of false positives}}$$

| | Actual case is true | Actual case is false |
|---|---|---|
| Predicted as true | True positive | False positive |
| Predicted as false | False negative | True negative |

**Table 4.4:** Possible outcomes of a binary classifier.

If one wants to simplify these two numbers further by combining them as a single curve, which is a function of FPR with an output of TPR, one could use different parameter settings and draw out the ROC curve. This plot shows us with a 'cost' of FPR , how much 'gain' of TPR we get in return. One can use AUROC to optimize[42] a classifier's parameter setting and illustrate its performance simultaneously. An ideal classifier would have a curve going way up to the upper left corner (TPR=1 at FPR=0), and a poor classifier would be close to the diagonal line.

ROC curves of our HVAC FDD approach for all six kinds of faults, two anomaly detection algorithms (Isolation Forest and Local Outlier Factor), and two distance measures (Euclidean and high-low) have been drawn out using 18 different parameter settings, which are shown in figure 4.18.

[41] AIC and BIC penalizes a model for the number of terms used, this is to prevent overfitting, while Cross-validation does this by minimizing the mean squared error. There are many researches comparing these different model selection methods.

[42] Depending on the task one is dealing with, optimize here does not necessarily mean to maximize the TPR/FPR ratio. For example, one may want to minimize FPR even with a loss of TPR in some cases to avoid system interruptions.

Recall that both LOF and iForest give us a score for each of the data points. This score is a measure that describes how likely a data point is an outlier. Nevertheless, similar to a $\lambda$-cut is needed in fuzzy logic; we still need to define a threshold for the decision function. The parameter we are setting here is the threshold of a decision function for these anomaly detection algorithms. Based on different parameter settings, we can clearly see that there are different performance results.

By inspecting the ROC curves, we see that our FDD approach has different fault detection performances for different kinds of faults. The ROC curves for fault 6 is basically lying over the diagonal line; this simply means that fault 6 is not detectable. This is consistent with results of fault detection rates shown in figures 4.19 and 4.20 (4.21 and 4.22 as well). The diagonal line basically means that with one cost of FPR we get one TPR back, so there is no difference to randomly guessing. This is still considered as a classifier, but does not benefit us. In addition, for faults 2 and 5, which are the same type of fault, a fault detection rate drop (about 10%) is noticed, we can also spot this performance drop in their ROC curves. This phenomenon is also true for faults 4 and 6; apparently fault 4 is detected successfully by our method; however, fault 6 is not detectable. Isolation forest seems to perform slightly better for fault 2 and 5 (same type of fault), while LOF works slightly better for fault 4, yet, both methods are comparable.

From these results, we conclude that the fault detection rate depends on the nature of the HVAC system itself, anomaly detection algorithm and parameters used, fault types, and severity of the faults. Strictly speaking, AUROC is used to depict the performances of classification (supervised learning) methods, while our fault detection work is considered as an unsupervised learning method. However, in this section, we are discussing and testing about how our approach works; hence, for testing purposes, we have revealed the fault labels to check the results after running fault detections.

Despite the popularity among the machine learning community, recently some researches have raised some skepticism and started to question the accuracy and representation using AUROC as a classification performance measure. For example, in [92], five reasons have been listed out for doubting the use of AUROC. In [93], the behaviors of ROC curves have been tested on synthetic data using statistical techniques such as cross-validation and bootstrap for small sample sized datasets. However, the potentially questionable issues brought up by these researches should not affect our usage of ROC curves. The issues are mostly regarding modeling probability distributions, and AUROC is not delivering enough information for some of their tasks, which, therefore, could be misleading. For our case here, we are not using 'areas' or some kind of 'score' to benchmark our work; ROC curves are simply used as a tool to visualize and help us select parameters instead of comparing classifiers. In addition, we are not trying to model any probability distributions for our time-series data since we are applying raw time-series representation for our work. If we tried to model probability distributions, we would be working on a model-based approach, e.g., something like Gaussian processes. Probability distributions could affect works regarding classification for some tasks that need more information such as probabilities instead of just a single classifi-

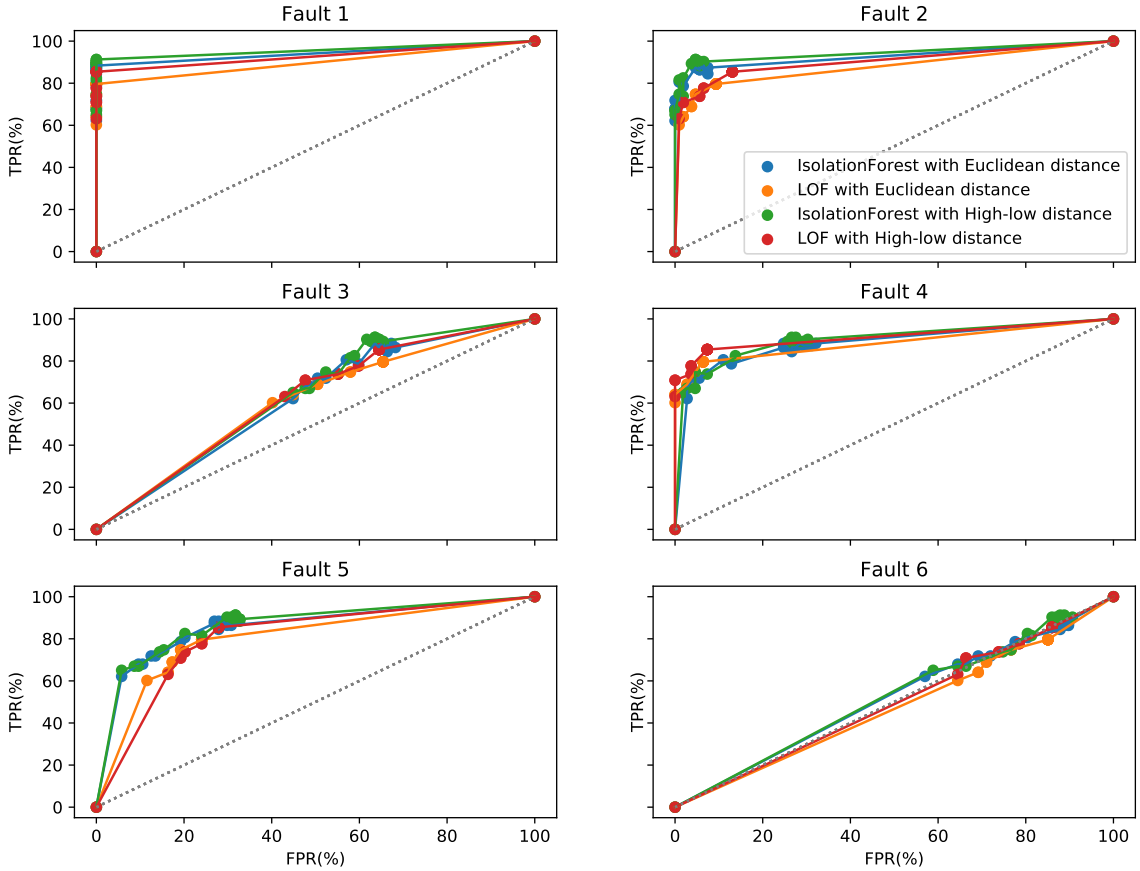
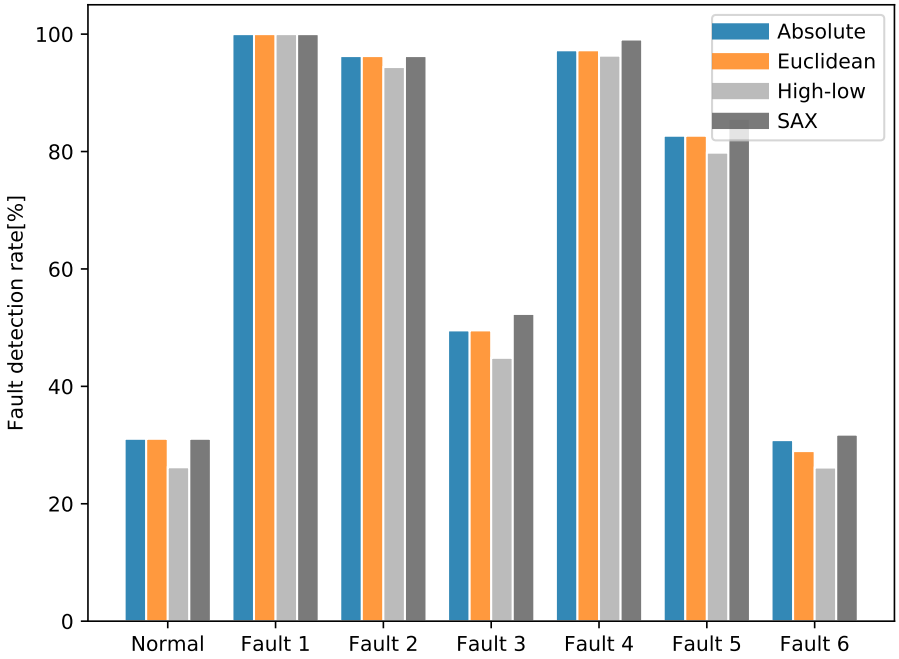



**Figure 4.18:** Receiver Operating Characteristic curves for our HVAC FDD approach.

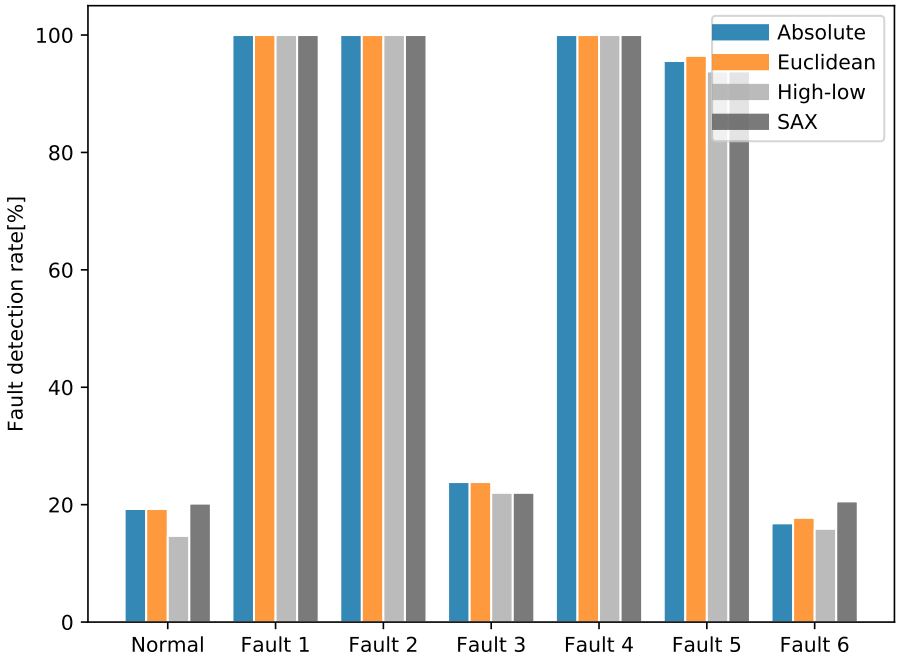
cation label. Nonetheless, we are working on fault detection; a simple normal/faulty result would be the most important. Moreover, we are working on time-series data, that is, high dimensional data at the order of about $10^5$ dimensions; both modeling and presenting probability distributions would be very difficult. It is also mentioned that AUROC tests performance over regions of the ROC space in which one would rarely operate and weighs false positives and false negatives equally. We have only tested the parameters within a reasonable operating range, which is the scattered points shown in our ROC results in figure 4.18, this should be equivalent to a partial ROC plot. Also, we have mentioned that the ROC curve is just a tool to help optimize parameter selection; we choose operating points based on the goal of a task, not by the area, largest accuracy sum, or some kind of single value score. Not to mention that we are assuming the faults are unknown; the reason we reveal them is just for testing purposes. That is, we are working on unsupervised learning, so there are no specific classes of faults defined in the feature space. Hence, it would be impossible to model the probability distributions for we do not have any known classes to model.

To conclude, ROC curves are only used to present and help us to both understand

the performance of our approach and select parameters of our results visually. If one wants to have more information and get the whole picture, it is recommended to look up the raw result data instead of summarized statistics, since any single-valued number would never be capable of representing and containing all information of a large high dimensional dataset.

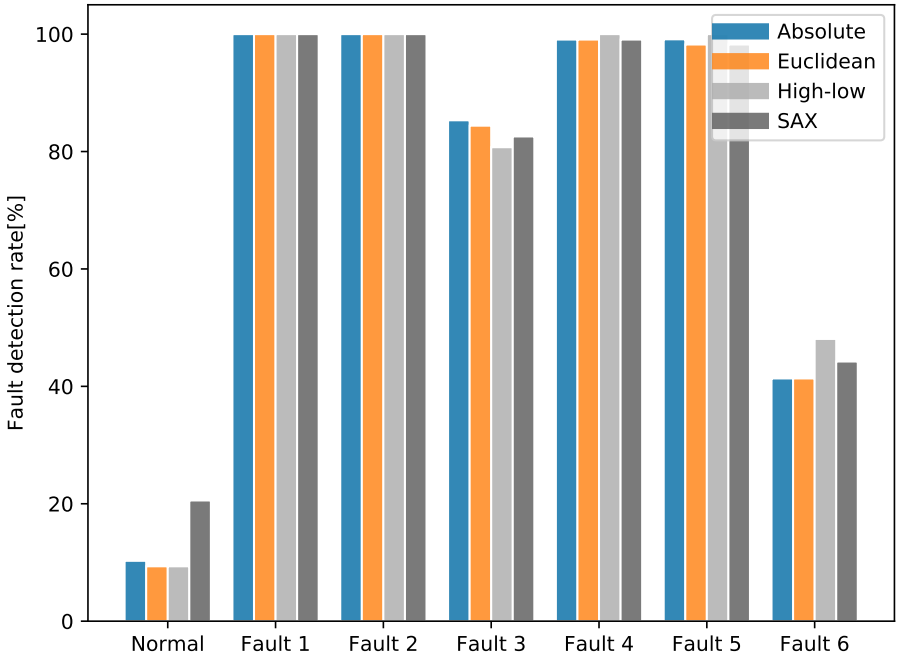



(a) Fault detection rates for Boston weather and office type 1 heat load.

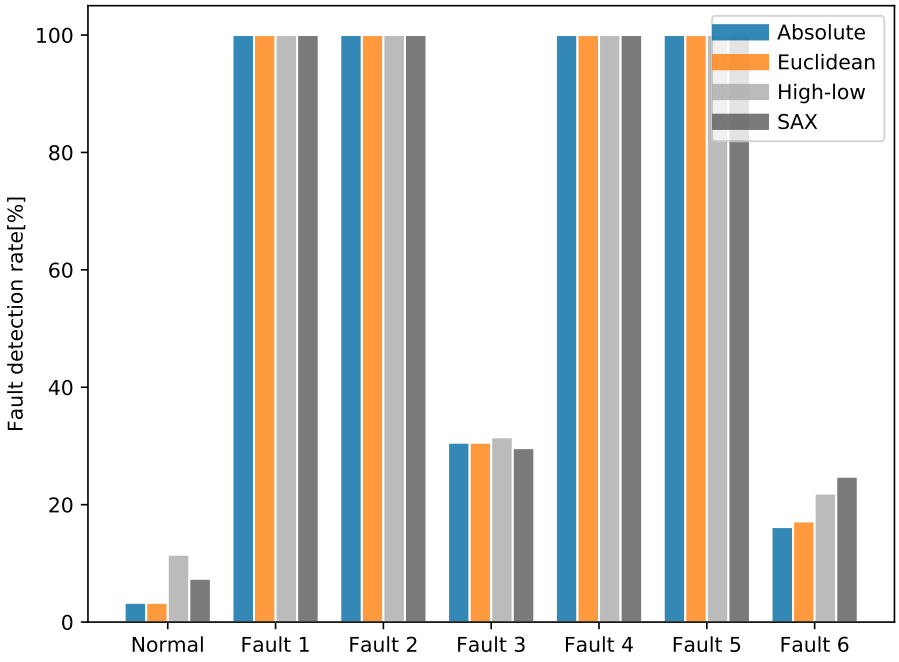



(b) Fault detection rates for Boston weather and office type 2 heat load.

**Figure 4.19:** Fault detection rates for Boston weather with 4 different distance measures using Local Outlier Factor. The fault descriptions are listed in table 4.3.

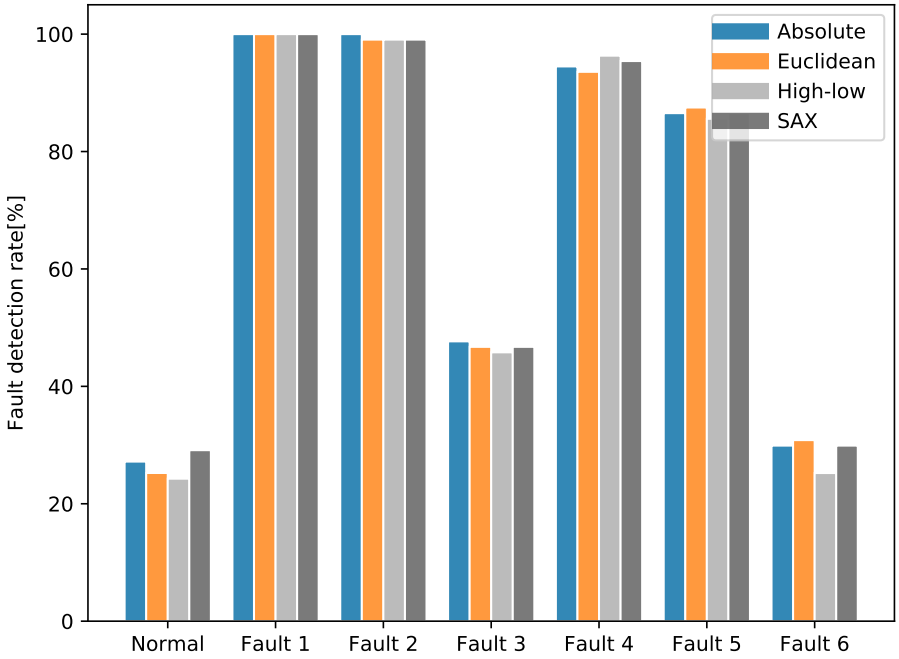



(a) Fault detection rates for San Francisco weather and office type 1 heat load.

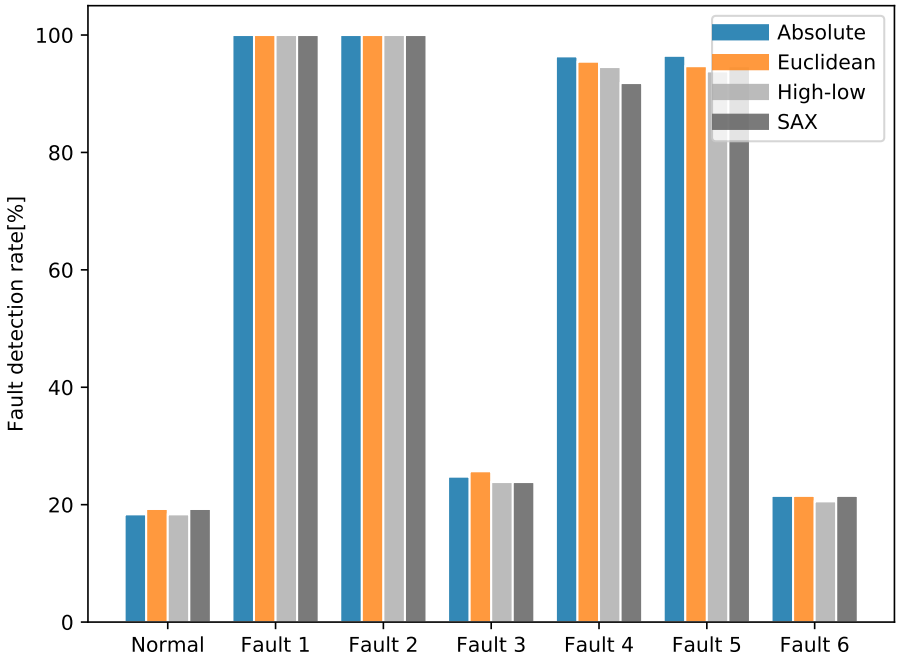



(b) Fault detection rates for San Francisco weather and office type 2 heat load.

**Figure 4.20:** Fault detection rates for San Francisco weather with 4 different distance measures using Local Outlier Factor. The fault descriptions are listed in table 4.3.

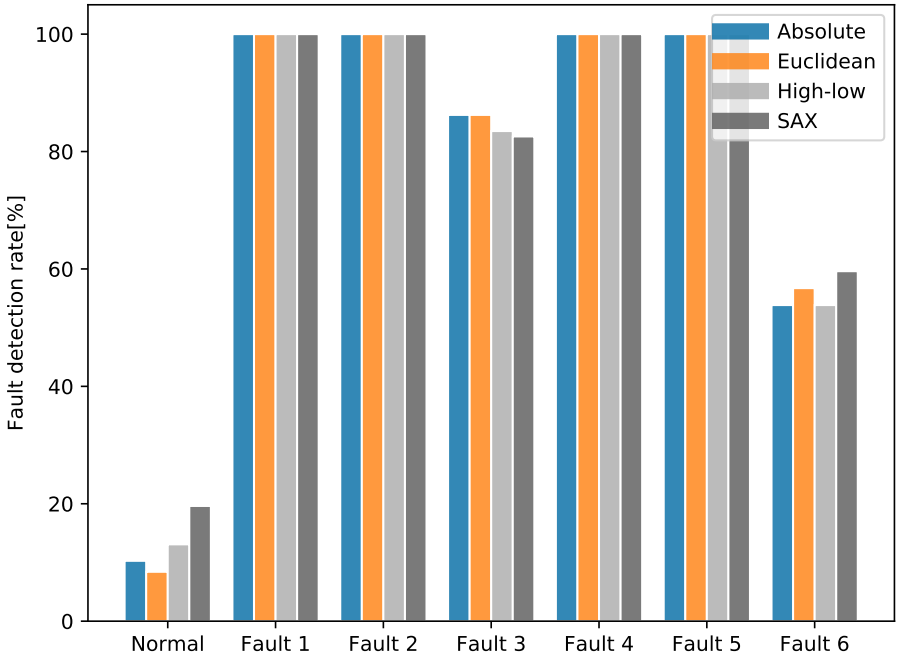



(a) Fault detection rates for Boston weather and office type 1 heat load.

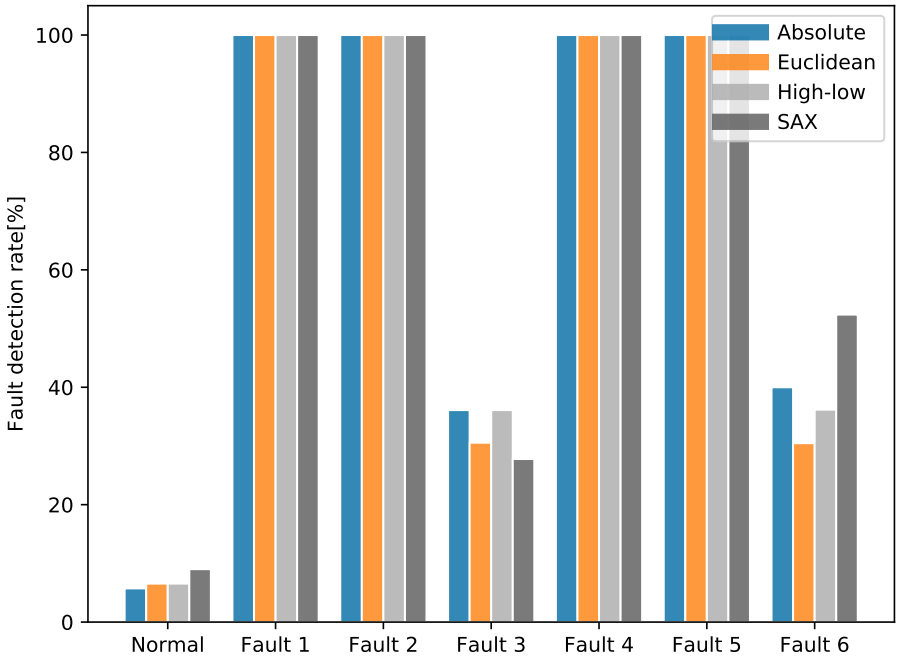



(b) Fault detection rates for Boston weather and office type 2 heat load.

**Figure 4.21:** Fault detection rates for Boston weather with 4 different distance measures using Isolation Forest. The fault descriptions are listed in table 4.3.




(a) Fault detection rates for San Francisco weather and office type 1 heat load.




(b) Fault detection rates for San Francisco weather and office type 2 heat load.

**Figure 4.22:** Fault detection rates for San Francisco weather with 4 different distance measures using Isolation Forest. The fault descriptions are listed in table 4.3.

# 5 Fault detection approach tested on different HVAC models for robustness

A fault detection approach for HVAC systems based on raw control time-series data is demonstrated in section 4, and its results are presented at the end of the section. As shown in the results, our data-driven approach works reasonably without relying on a vast amount of sensor data. In contrast, we rely on the manipulated variables of the system's control data. This approach benefits us by reducing the number of variables significantly; however, the main purpose is to ensure our approach is scalable, for control systems are in general comprised of on/off switches and PID controllers. Moreover, we have used a raw time-series representation without making any model assumptions as model-based representations would. Again, this was chosen, for our intention is to make as little assumptions as possible to ensure scalability. Hidden faults, which are not sensible to residents as the controlled environment (room) has remained the same, were able to be detected. In order to test the robustness of this approach, we will adopt the same steps and work flow using a different model.

## 5.1 A centralized multiple room HVAC system model

In contrast to the model used in the previous section, we will have three rooms with individual VAV (Variable Air Volume) boxes and a central AHU providing constant cool air at 13°C. While we have used a CAV control mechanism to maintain the room environment in our previous model, we will apply a VAV control strategy for our centralized multiple room in this section. Each VAV box will have a cool air intake with constant temperature supplied by the AHU. If higher room temperatures are needed, the cool air will be reheated by the heating coils within the VAV boxes. Each room will be set to different sizes and with their own temperature setpoints, at 20°C, 21°C, and 22°C (all within the normal comfort zone), respectively. In addition, each room will have their own heat load data added to simulate different usages. A hot water loop with a boiler, a cold water loop with a chiller, and a cool water loop with a cooling tower are connected as subsystems in the model. A schematic diagram of this model set up is shown in figure 5.1.

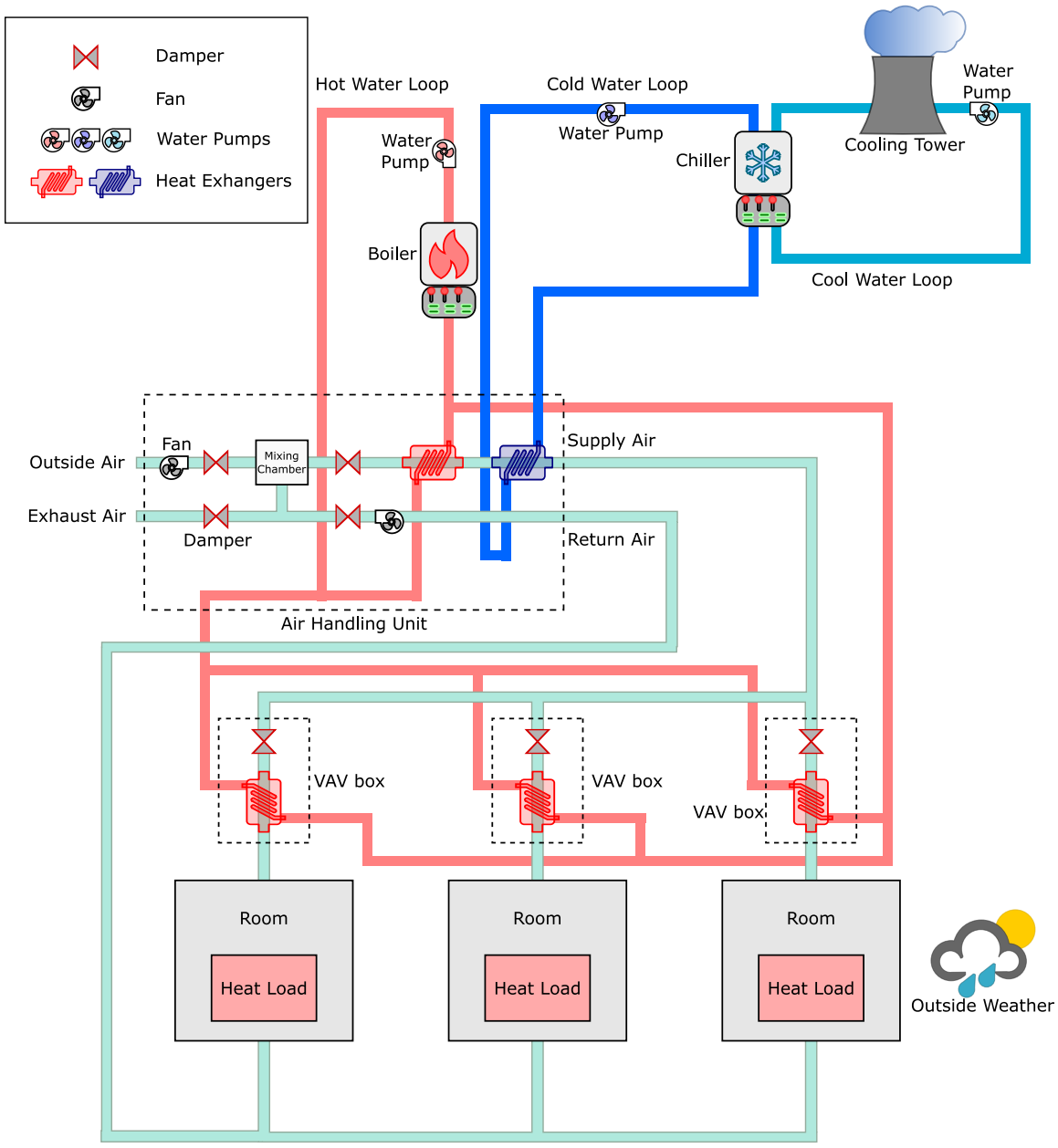



**Figure 5.1:** Our targeted centralized multiple room HVAC system model.

## 5.2 Data source

The input data source we will be using are the same datasets we used in section 4.1. Datasets of TMY3 from NOAA are used for weather data, and 'Commercial and Residential Hourly Load Profiles for all TMY3 Locations in the United States' from DOE are used for heat load data.

As for simulation data, we will use our new centralized multiple room HVAC system model built in Modelica. The simulation environment is the same, using *Modelica*

*Standard Library* (v3.2.3) and the *Buildings Library*(Buildings 4.0.0) in OpenModelica environment (v1.11.0-64bit). Simulations are run with *JModlieca.org* (v2.1). All software packages are executed on a Intel Core i5-2400 3.10GHz CPU with 8.00GB RAM machine running Windows 10 64bit.

## 5.3 Data cleaning

Since we are using the same data source for weather and heat load input data in the previous section, the steps and procedures are the same; we only need to make sure the time is matched and units are compatible.

Again, same as section 4.2, we will remove the first two hours of simulation data to avoid unstable and noisy data during the initialization period. Also, we would have to take note of the data sampling rate, for this model contains more components and is more complicated, which means there are more variables. Thus, we have set a lowered data sampling rate for this model to avoid memory errors. This is because of our computer hardware restrictions. Previous codes used for data cleaning needs to be adjusted before reusing.

## 5.4 Simulations with Modelica

We will skip the steps for input data exploration and time-series rank ordering here, for the steps and procedures are the same and redundant. We will simply adopt the results we have concluded in sections 4.3 and 4.4. Two sets of weather data including San Francisco and Boston area are reused. As in section 4.5, we will classify heat load data into workdays and non-workdays to avoid large variances and uncertainties. A heat load pattern and calendar comparison check is needed to ensure the days are matched. However, since we have three rooms this time, each room will have hourly heat load added based on the input weather date with a $\pm$2-day range of randomness for the training data generation. That is, heat loads for the rooms are different since they are randomly drawn from the heat load dataset within a range. Testing datasets will be generated with a similar way but the heat load data will be drawn using a wider range of $\pm$5-day to introduce randomness while keeping the seasonal dependency of weather data.

A training dataset of a normal operating model comprising 362 days is generated. Another normal operating testing dataset is generated, which is used to test the true positives and false negatives for true positive rates. Five types of faulty HVAC system model variants are built; these models are simulated to generate our faulty datasets. These faults are listed in table 5.1. At least 100 data samples are generated for each testing dataset. The stored variables are adjusted due to the model change. VAV box control variables for each room are collected and saved.

| Fault | Fault description |
|---|---|
| Normal | A normal operating dataset for testing purpose. |
| Fault 1 | A stuck/clogged VAV damper for a VAV box. |
| Fault 2 | A stuck/clogged hot water valve of the heating coils in one of the VAV boxes. |
| Fault 3 | A leaking air duct. Supply air leaking to the outside. |
| Fault 4 | Heating loop with a malfunctioning boiler. Hot water temperature is lower than normal. |
| Fault 5 | A malfunctioning thermostat. AHU supply air temperature is 1°C lower than setpoint. |

**Table 5.1:** Simulated testing datasets and descriptions for our centralized multiple room HVAC system model. A more detailed description list can be found in appendix C.2.

## 5.5 Fault detection results and discussion

Since this section is designed to test out the robustness of our fault detection approach for different HVAC models, we conduct the same procedures as in section 4. That is, the environmental weather is the same, and the heat load datasets are reused; only the HVAC system model is changed. Some minor differences exist due to the different HVAC model used; the rest of the simulation set up is basically the same. However, the manipulated variables are different because of the fact that we have changed the model. Therefore, we would again build a rank order list of the training data for each testing data sample, adding this data sample to the order list and form a testing group. After locating the corresponding manipulated variables data, we run anomaly algorithms on them. As in section 4, we have used four different distance measures for rank ordering. Two different weather datasets (Boston and San Francisco) and two types of office heat load datasets are used.

The results are shown in figures 5.6, 5.7, 5.8, and 5.9. The ROC curves of Boston and San Francisco weather with office type 1 heat load data are shown in figures 5.4 and 5.5. The first subplot (upper left) in the ROC figures shows a ROC curve of a normal operating dataset; we see that the data points basically are on top of the diagonal line, which is as we expected since no faults should be detected in the ideal case. From these result plots, we see that Isolation Forest seems to be more stable than Local Outlier Factor for this set up in both the ROC curves and fault detection rates.

From the results we have demonstrated here and in section 4, we can say that our fault detection approach using raw time-series works and delivers reasonable detection rates. It is true that the overall performance would be affected by many factors, and we are assuming the heat load patterns are in general very similar among work days for a specific building. However, our results suggest that a data-driven fault detection approach using manipulated control variables being scalable, reliable, automatic, and economical is achievable.

### 5.5.1 Parameter selection according to experiment results

Comparisons of different parameter value settings have been carried out and visually demonstrated in the form of ROC curves in figures 4.18, 5.4, and 5.5. As we have mentioned in section 4.6.1, ROC curves will be used as a visualization tool to help us select the parameters. In this subsection, we will list parts of the different parameter values we have used for the anomaly detection algorithms.

The parameter we are talking about here is the threshold parameter (decision function's threshold)[43]. The meaning of this parameter value stands for the proportion of outliers in the dataset. Because both anomaly detection algorithms we have used, LOF and iForest, give us some score of the new data sample after comparing it with the training dataset, a threshold still needs to be assigned in order to tell at what score are the data points considered as outliers. The decision function here is using the distribution percentile of the training dataset. This concept is similar to where we should draw the black solid line in figure 1.4. Hence, a trade-off of a higher fault detection rate versus a lower false alarm rate is set by adjusting this parameter.

Figure 5.3 shows ROC curves with different threshold parameter values labeled for two different weather (Boston and San Francisco) and two different office heat load datasets using LOF as the anomaly detection algorithm. Tabulated data can be found in tables F.1 and F.2[44]. We see that larger variances (which means more uncertainty), e.g., Boston weather with office type 1 heat load data (see figures 4.12 and 4.17 in section 4), leads to poorer overall performance in figure 5.3a. The general trend is the same; that is, by lowering the threshold parameter, we get lower false positive rates by sacrificing true positive rates. Again, how this is chosen depends on the system, inputs, desired sensitivity of our fault detection system, ... etc. However, from our experimented results, a value of 0.007 (99.3 percentile) seems to be a reasonable choice for us, since we would prefer a low FPR for HVAC systems. For more conservative settings, a value of 0.002 (99.8 percentile) is probably the choice to go with. This parameter selection is based on our experiment results. Boston and San Francisco are chosen to reflect different weather trends, where Boston has a wider range of change in weather and San Francisco's bay area is known for its stable weather climate.

[43] In *scikit-learn* library, this parameter is called `contamination`.

[44] We have only listed the tabular data for Boston weather and office type 1 heat load results. This is to avoid using up unnecessary pages. Besides, ROC curves summarize the results for us neatly.

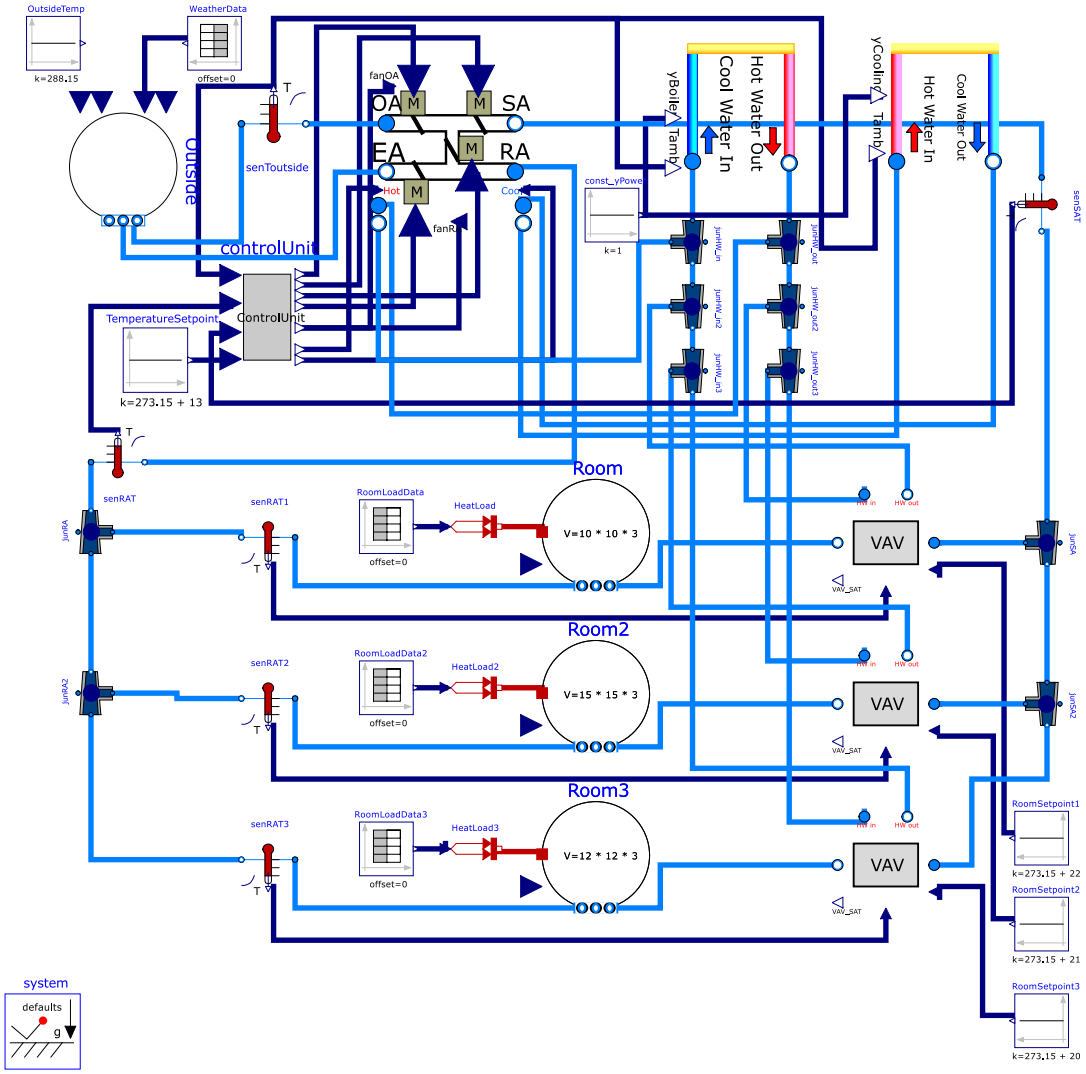
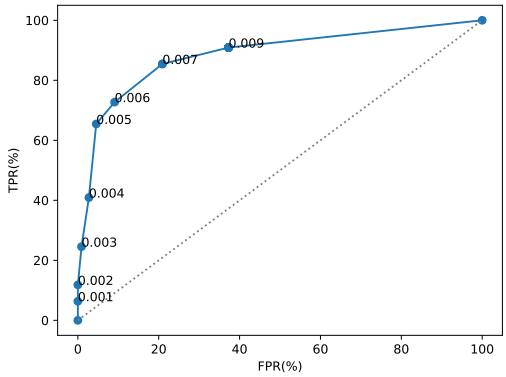
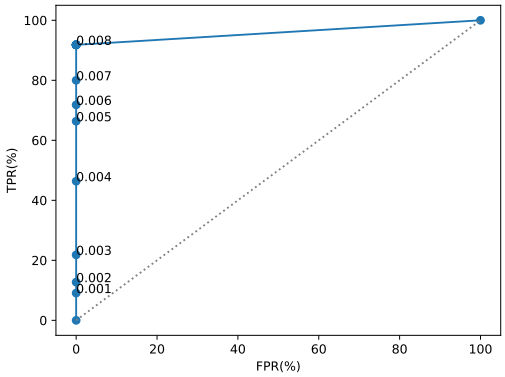


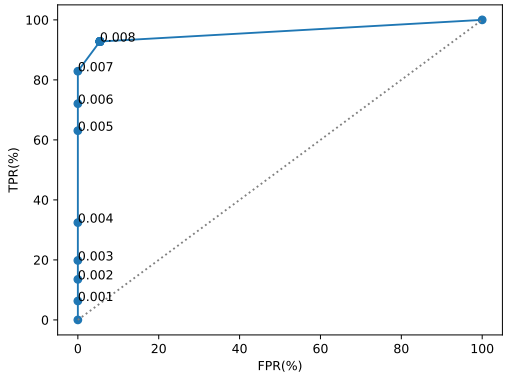

**Figure 5.2:** Our centralized multiple room HVAC system model in Modelica. This is the Modelica equivalent model shown in figure 5.1



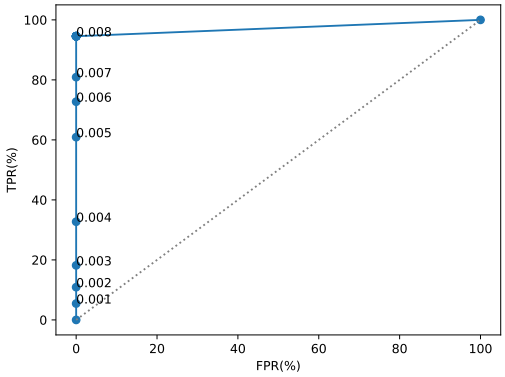

(a) ROC curve of Boston weather and office type 1 heat load data with labeled threshold parameter.




(b) ROC curve of Boston weather and office type 2 heat load data with labeled threshold parameter.




(c) ROC curve of San Francisco weather and office type 1 heat load data with labeled threshold parameter.




(d) ROC curve of San Francisco weather and office type 2 heat load data with labeled threshold parameter.

**Figure 5.3:** ROC curves with parameter value labeled.

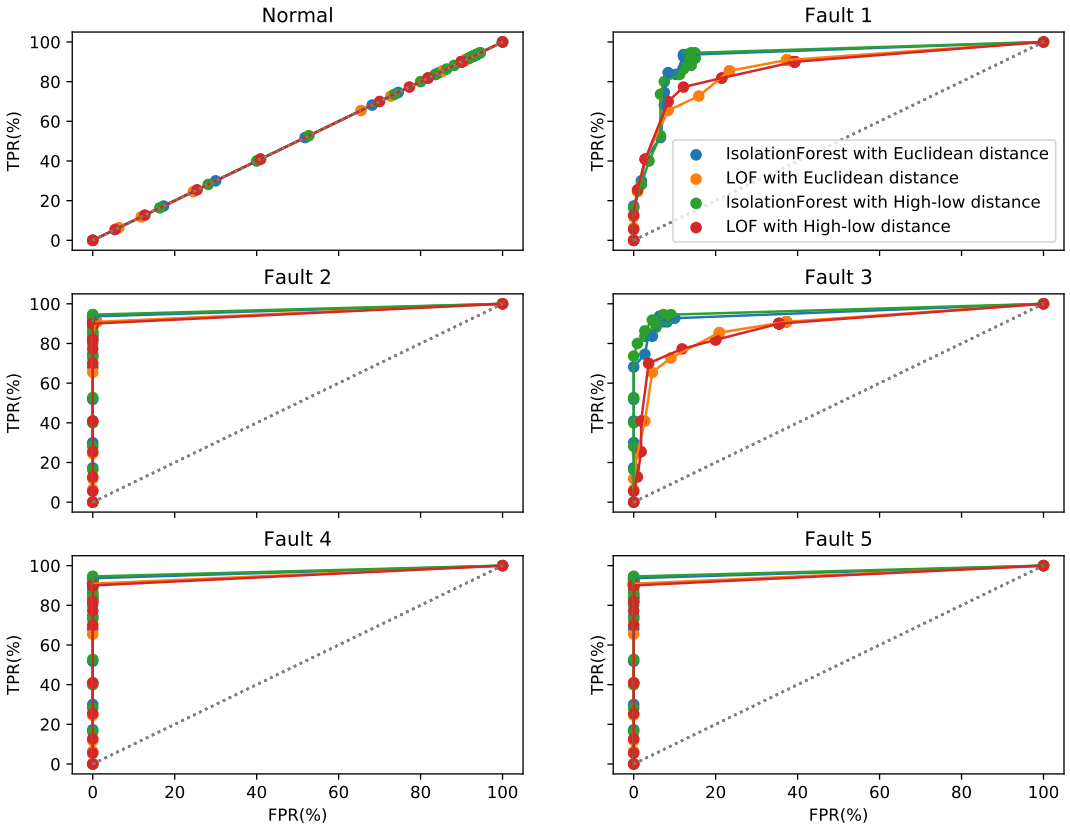



**Figure 5.4:** Receiver Operating Characteristic curves for our HVAC FDD approach for our multiple room model with Boston weather and office type 1 heat load.

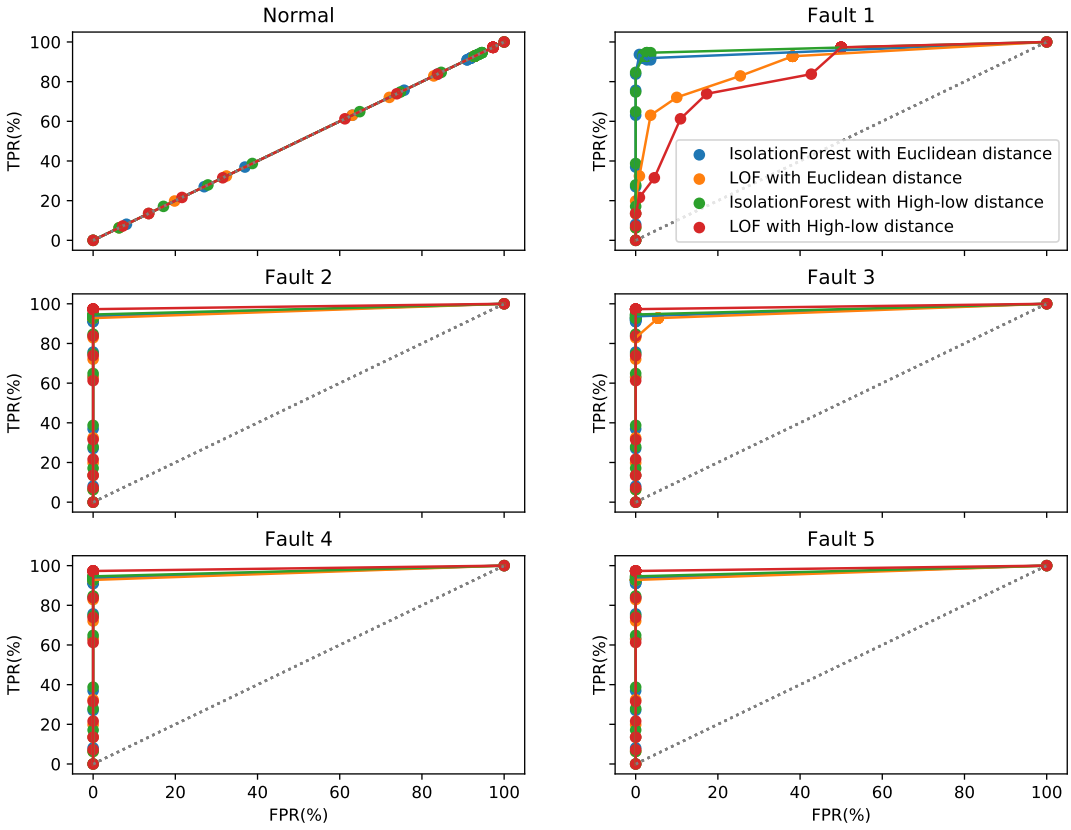
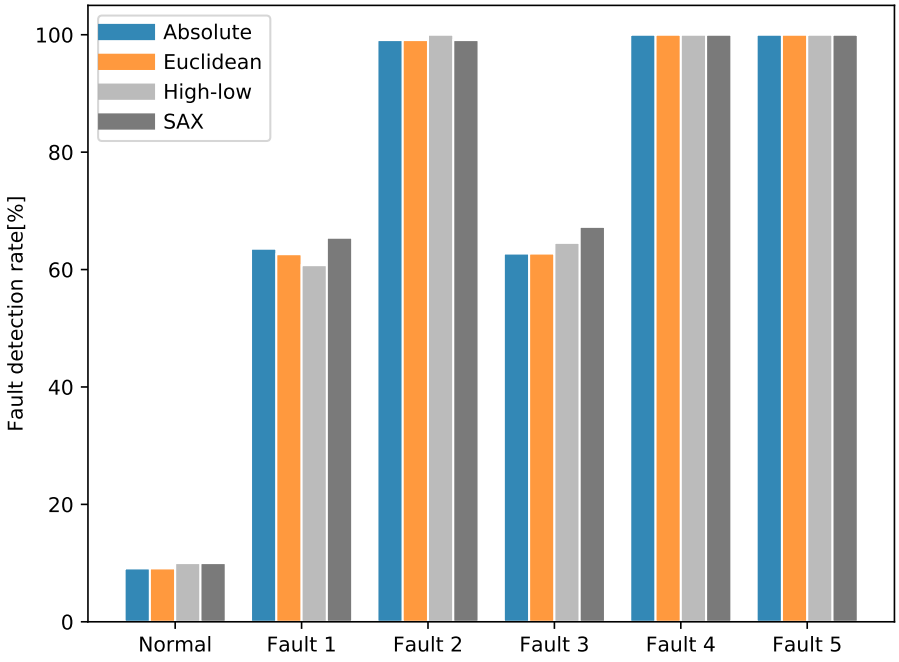
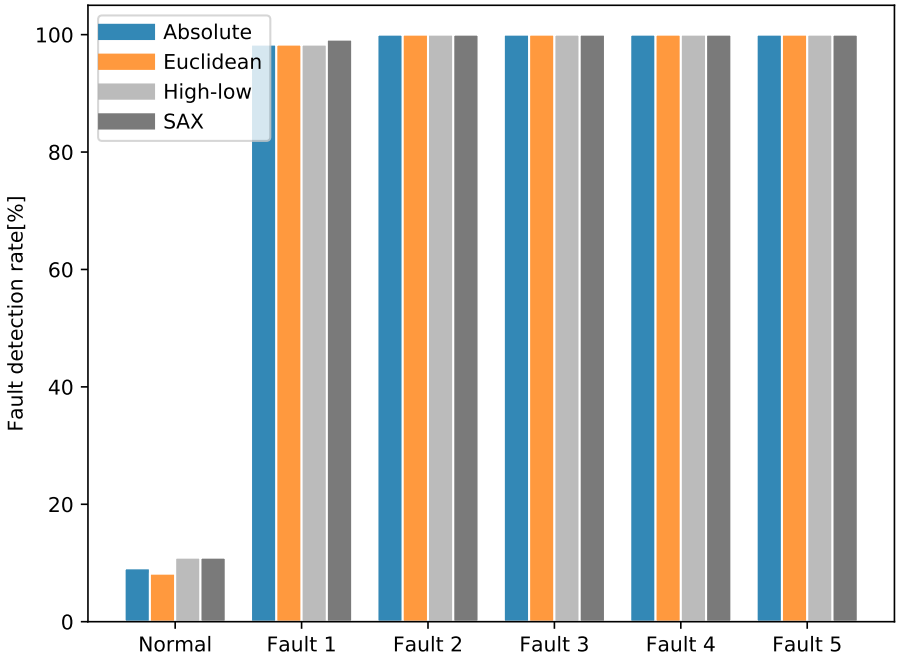



**Figure 5.5:** Receiver Operating Characteristic curves for our HVAC FDD approach for our multiple room model with San Francisco weather and office type 1 heat load.

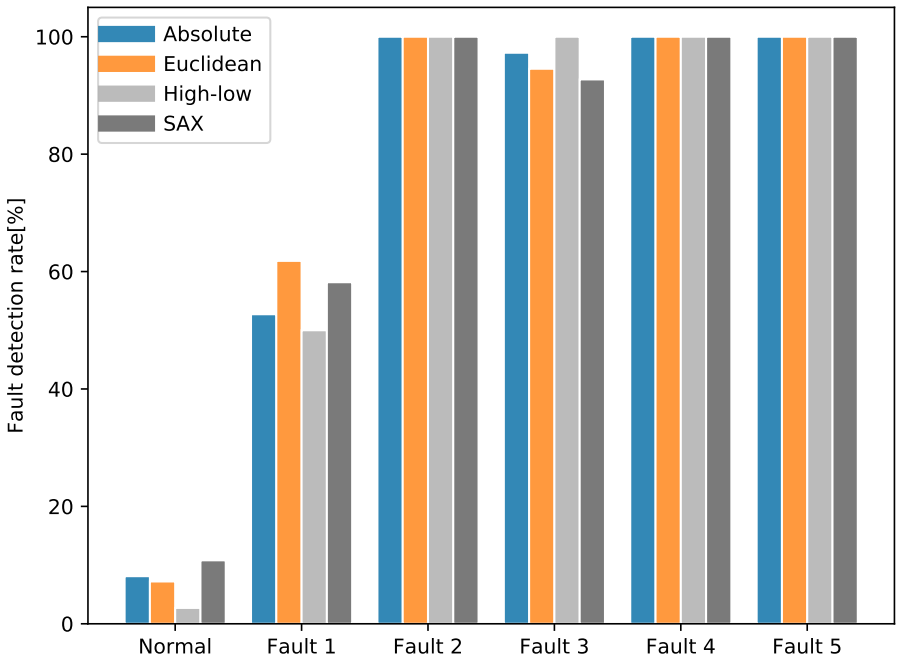



(a) Fault detection rates for Boston weather and office type 1 heat load.

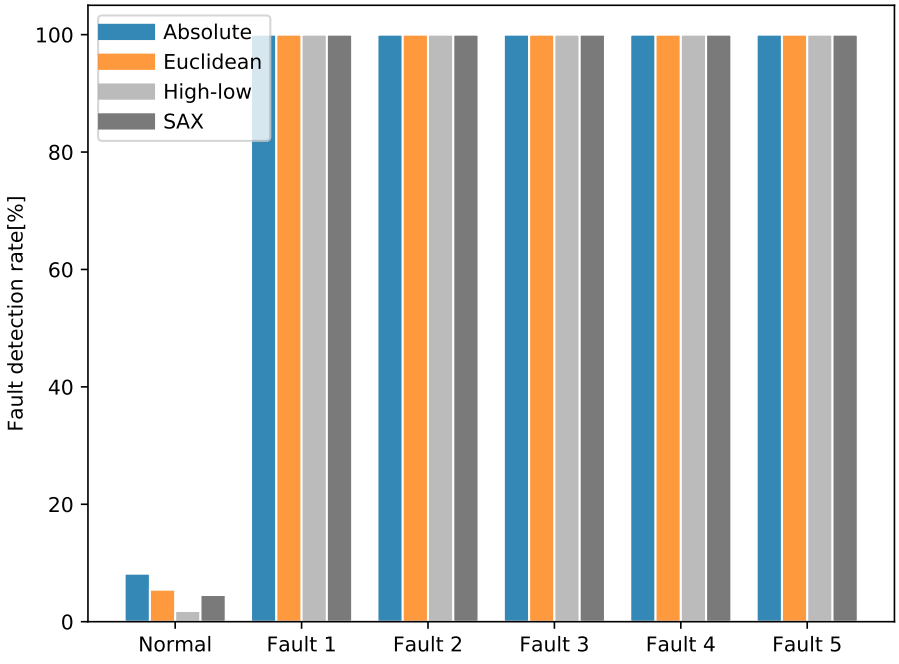



(b) Fault detection rates for Boston weather and office type 2 heat load.

**Figure 5.6:** Fault detection rates for Boston weather with 4 different distance measures using Local Outlier Factor. The fault descriptions are listed in table 5.1.

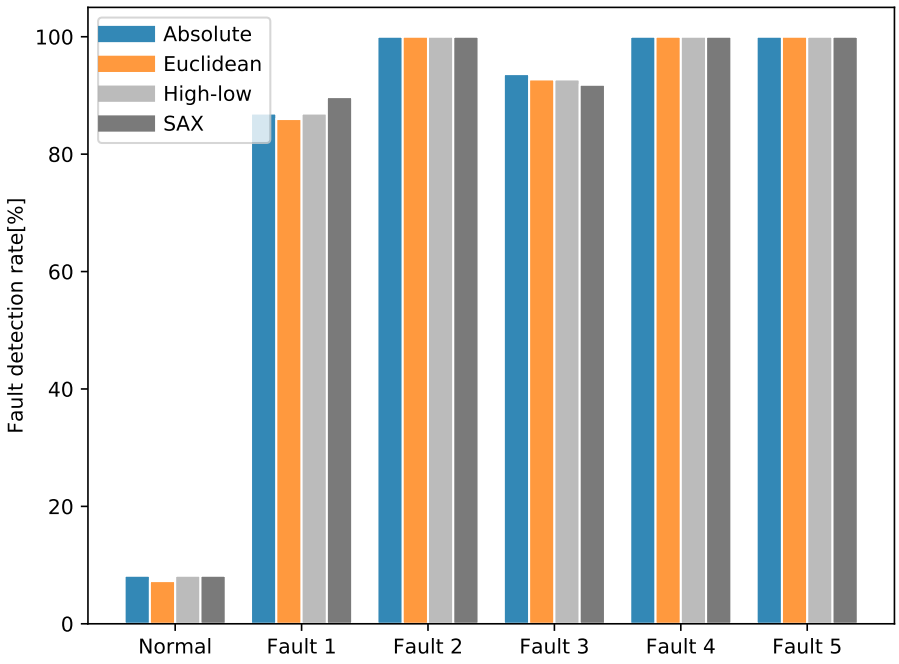



(a) Fault detection rates for San Francisco weather and office type 1 heat load.

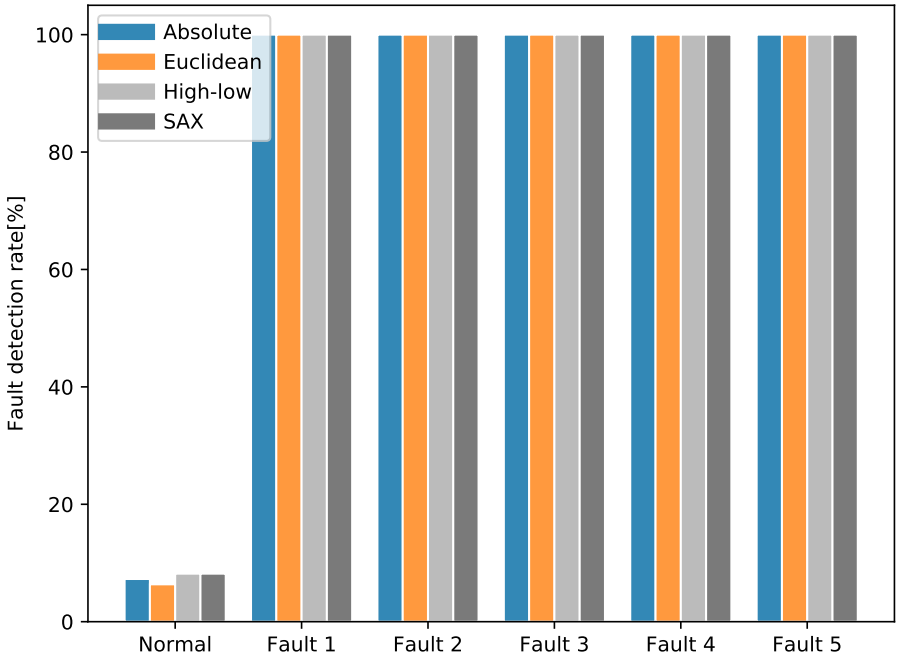



(b) Fault detection rates for San Francisco weather and office type 2 heat load.

**Figure 5.7:** Fault detection rates for San Francisco weather with 4 different distance measures using Local Outlier Factor. The fault descriptions are listed in table 5.1.

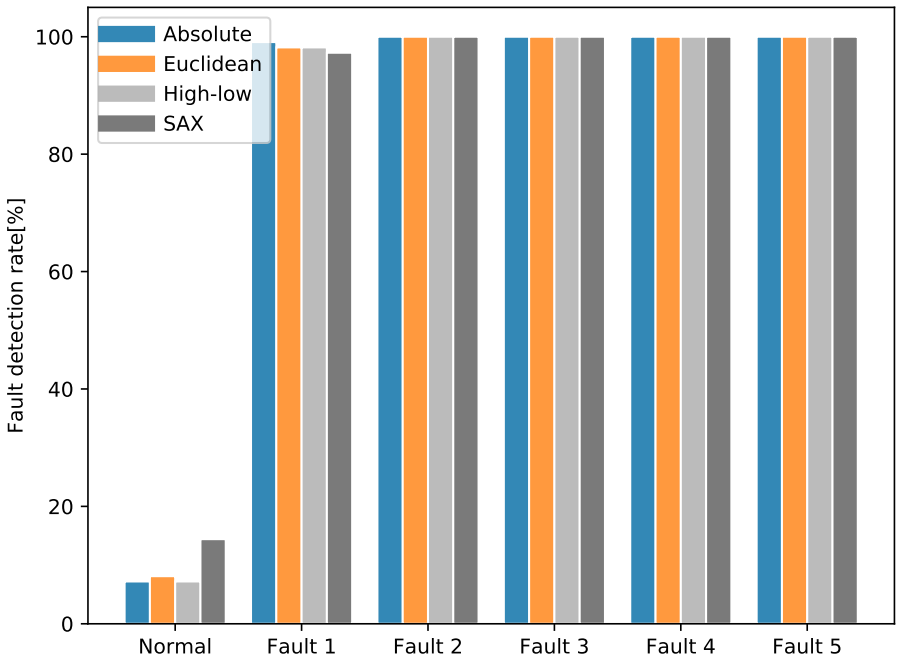



(a) Fault detection rates for Boston weather and office type 1 heat load.

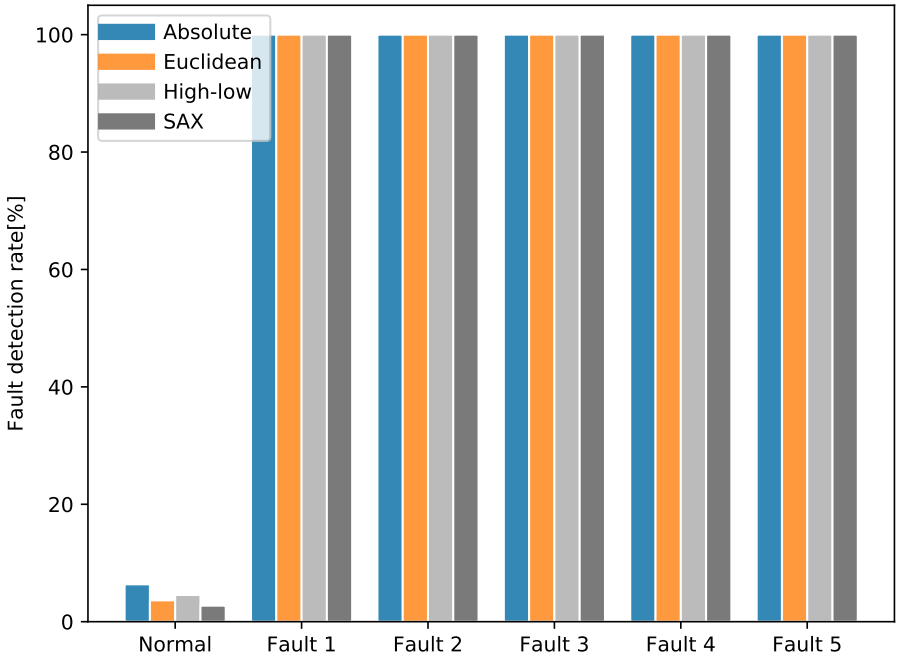



(b) Fault detection rates for Boston weather and office type 2 heat load.

**Figure 5.8:** Fault detection rates for Boston weather with 4 different distance measures using Isolation Forest. The fault descriptions are listed in table 5.1.




(a) Fault detection rates for San Francisco weather and office type 1 heat load.




(b) Fault detection rates for San Francisco weather and office type 2 heat load.

**Figure 5.9:** Fault detection rates for San Francisco weather with 4 different distance measures using Isolation Forest. The fault descriptions are listed in table 5.1.

# 6 Data-driven fault diagnosis approach using raw time-series data

Thus far, we have demonstrated how a fault detection approach can be conducted on raw manipulated variable time-series data in sections 4 and 5 for different HVAC system model types, including a rooftop unit using a CAV strategy for a single room set up and a centralized multiple room HVAC system with a VAV control strategy set up. However, we have not yet presented work on fault diagnosis. In this section, we will be dealing with HVAC fault diagnosis by assuming we have some prior knowledge of the system's faults based on historical data. In order to work on fault diagnosis, one would either have knowledge of the system model design or have access to historical data that contains information of the system. Since we are working on a data-driven approach to achieve a scalable FDD, we will stick to the latter assumption.

In the following subsections, we will present two HVAC fault diagnosis approaches. First, we will be going in an opposite direction of our previous fault detection work; that is, we will invert the training and testing datasets of the anomaly detection algorithms and try to find which fault has the highest match. Our second approach will be applying classification algorithms to our new faulty data sample and see which faulty dataset it has been classified to. All of the work will again be using raw time-series data.

## 6.1 Data source

In this section, we will be reusing the datasets we have generated by Modelica simulations from previous sections. Control data, that is, manipulated variables data, will be extracted from the result files and be used as training datasets. Historical faulty data samples are randomly drawn; also, new faulty data samples for testing will be newly generated by the Modelica models we have built in previous sections.

The set up for Modelica simulations is the same, using TMY3 weather data from NOAA and heat load data from DOE as input data for our HVAC models, and using *Modelica Standard Library*(v3.2.3) and the *Buildings Library*(Buildings 4.0.0) in OpenModelica environment(v1.11.0-64bit). Simulations are run with *JModlieca.org*(v2.1). All software packages are executed on a Intel Core i5-2400 3.10GHz CPU with 8.00GB RAM machine running Windows 10 64bit.

## 6.2 Data cleaning

Data cleaning process is basically the same; the first two hours are removed to avoid unstable and noisy data during the initialization period. Data sampling rates for simulations are set to be conformable with previously generated data. If a higher sampling rate is used for new data generation, a down sampling process would need to be carried out to make datasets compatible. For details, see sections 4.2 and 5.3.

## 6.3 A fault pattern matching approach by adjusting our fault detection method

By assuming we have datasets of formerly collected historical data, the idea here is that we can learn the patterns of faults that had occurred previously in the system. Suppose some kind of fault has existed for some period of time in the system before, and its data has been recorded in time-series form; due to the system's design, materials used, or whatever reason a fault happens again, from the results of our previous work about fault detection and with historical data at hand, we should be able to not only detect a reoccurring fault, but also identify and recognize its manipulated variable patterns. Even though the new fault may not be exactly the same (for example, faults 2 and 5 in section 4 are the same type of fault but with different severity levels), we should at least be able to guess or find similarities between their data patterns.

We think that for a HVAC system having comparable inputs, including similar weather and heat load profiles, its manipulated variable patterns are comparable as well. If a fault is introduced, it becomes a faulty system, which is still a system; the manipulated variables of the control system should reflect the inputs the same faulty way. Therefore, we can compare the patterns of manipulated variables; if a fault happened to the system before and has been recorded, we should be able to find matches in the patterns. Having this idea in mind, we find that our fault detection approach is actually some kind of pattern matching technique for normal operating conditions. If we change the training data to a historical dataset of some fault, we will be matching faulty manipulated variable patterns, since 'faulty' is considered as 'normal' to a faulty system. Hence, by changing the training datasets we use to different historical faults, we can carry out a fault diagnosis task.

Faulty testing datasets that are generated from previous sections are now used as our historical database. These datasets are then used as training data for anomaly algorithms. New testing data samples are generated and tested against our old data. We expect to see higher match rates for faults of the same type and lower rates for normal conditions and/or different fault types.

### 6.3.1 Fault diagnosis results and discussion

Our single room HVAC model used in section 4 is tested out, using San Francisco weather and an office heat load data as inputs, and the matching rates (normal operating rates) of it are shown in table 6.1, where the rows are the training datasets used and the columns are the testing datasets. We see that with the same type of system fault, there is a very high probability the system would be considered 'normal' to the faulty dataset. A similar type of fault or a fault that causes the control system to compensate with similar manipulated variable patterns would also lead to a close match. This result is also plotted as a radar chart shown in figure 6.1. A peak in the radar charts stands for a good match; the ideal case is to only have one peak; however, if multiple peaks exist, this means the testing dataset has similar manipulated variable

patterns to multiple types of faults. For example, fault 5 (purple color) testing data is both similar to the patterns of fault 2 and 5 (see table 4.3). This is expected since both of these faults are basically the same but with different severity levels. Faults 3 and 6 seem to have high matches to the normal dataset; this is due to the fact that they are minor faults, especially fault 6, for it can be considered undetectable for some set ups. This can be verified by the ROC curve and fault detection rates back in section 4.

| **LOF** | | **Test data** | | | | | |
|---|---|---|---|---|---|---|---|
| | | Fault1 | Fault2 | Fault3 | Fault4 | Fault5 | Fault6 |
| | Normal | 0.0 | 0.0 | 65.1 | 4.6 | 4.4 | 88.5 |
| | Fault1 | 98.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | Fault2 | 0.0 | 97.2 | 16.5 | 8.3 | 53.1 | 11.5 |
| **Training data** | Fault3 | 0.0 | 0.9 | 94.5 | 0.9 | 12.4 | 86.5 |
| | Fault4 | 0.0 | 0.0 | 8.3 | 95.4 | 1.8 | 7.7 |
| | Fault5 | 0.0 | 22.0 | 65.1 | 13.9 | 97.3 | 65.4 |
| | Fault6 | 0.0 | 0.0 | 89.9 | 2.8 | 12.4 | 97.1 |

| **iForest** | | **Test data** | | | | | |
|---|---|---|---|---|---|---|---|
| | | Fault1 | Fault2 | Fault3 | Fault4 | Fault5 | Fault6 |
| | Normal | 0.0 | 0.0 | 61.5 | 4.6 | 0.0 | 86.5 |
| | Fault1 | 97.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | Fault2 | 0.0 | 97.2 | 7.3 | 0.0 | 47.8 | 7.7 |
| **Training data** | Fault3 | 0.0 | 0.0 | 94.5 | 63.0 | 8.0 | 90.4 |
| | Fault4 | 0.0 | 0.0 | 30.3 | 83.3 | 0.0 | 38.5 |
| | Fault5 | 0.0 | 11.0 | 50.5 | 19.4 | 94.7 | 46.2 |
| | Fault6 | 0.0 | 0.0 | 95.4 | 67.6 | 3.5 | 97.1 |

**Table 6.1:** Matching rates for testing fault data. Values are in percentages. Single room HVAC model.

We test out our multiple room HVAC model used in section 5 the same way, using San Francisco weather and office building heat load data as inputs. Here we have generated a new faulty dataset, fault 6, for our centralized multiple room HVAC model. The faults are listed in table 6.2, matching results are listed in table 6.3, and the radar charts are shown in figure 6.2. Similarly, fault 3 and 6 are basically the same type of fault; therefore, we find that they have similar matching results. In addition, fault 6 is 'less detectable' than fault 3 in nature; it has multiple peaks in the radar charts. We can also see in the radar charts that Isolation Forest performs better than Local Outlier Factor for this case.

| Fault | Fault description |
|---|---|
| Normal | A normal operating dataset for testing purpose. |
| Fault 1 | A stuck/clogged VAV damper for a VAV box. |
| Fault 2 | A stuck/clogged hot water valve of the heating coils in one of the VAV boxes. |
| Fault 3 | A leaking air duct. Supply air leaking to the outside. |
| Fault 4 | Heating loop with a malfunctioning boiler. Hot water temperature is lower than normal. |
| Fault 5 | A malfunctioning thermostat. AHU supply air temperature is 1°C lower than setpoint. |
| Fault 6 | Same as Fault 3, but with a smaller leak. |

**Table 6.2:** Simulated testing datasets and descriptions for our centralized multiple room HVAC system model. A more detailed description list can be found in appendix C.3.

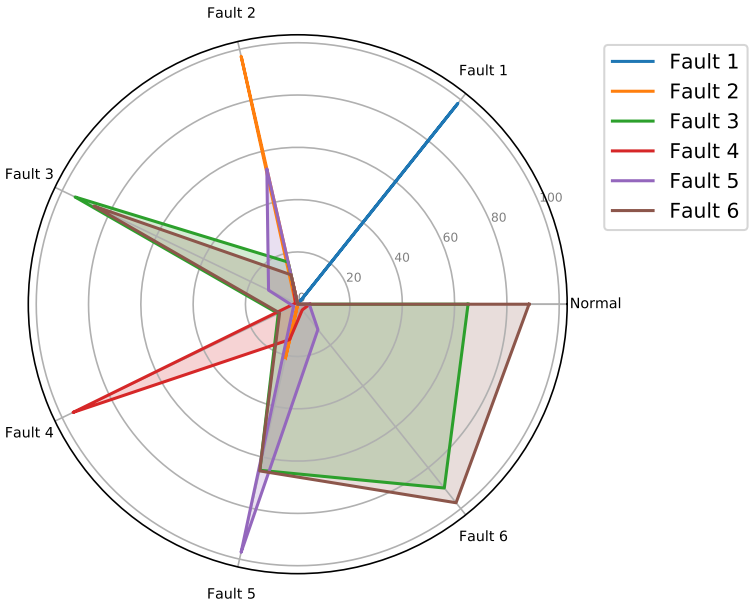
**LOF**

| | | Test data | | | | | |
|---|---|---|---|---|---|---|---|
| | | Fault 1 | Fault2 | Fault 3 | Fault 4 | Fault 5 | Fault 6 |
| | Normal | 62.7 | 0.0 | 41.8 | 0.0 | 0.0 | 93.3 |
| | Fault1 | 92.7 | 0.0 | 5.5 | 0.0 | 0.0 | 3.3 |
| | Fault2 | 0.0 | 88.7 | 0.0 | 0.0 | 0.0 | 0.0 |
| **Training data** | Fault3 | 30.0 | 0.9 | 91.8 | 0.0 | 0.0 | 76.7 |
| | Fault4 | 0.0 | 0.0 | 0.0 | 92.7 | 0.0 | 0.0 |
| | Fault5 | 0.0 | 0.0 | 0.0 | 0.0 | 90.0 | 0.0 |
| | Fault6 | 64.5 | 30.2 | 89.1 | 0.0 | 0.0 | 83.3 |

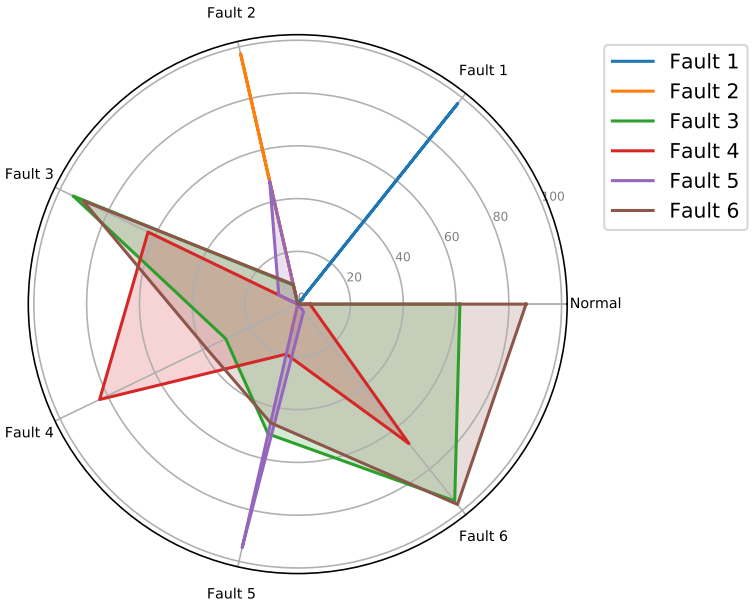
**iForest**

| | | Test data | | | | | |
|---|---|---|---|---|---|---|---|
| | | Fault 1 | Fault2 | Fault 3 | Fault 4 | Fault 5 | Fault 6 |
| | Normal | 5.5 | 0.0 | 21.8 | 0.0 | 0.0 | 36.7 |
| | Fault1 | 80.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | Fault2 | 0.0 | 84.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **Training data** | Fault3 | 1.8 | 0.0 | 85.5 | 0.0 | 0.0 | 40.0 |
| | Fault4 | 0.0 | 0.0 | 0.0 | 77.3 | 0.0 | 0.0 |
| | Fault5 | 0.0 | 0.0 | 0.0 | 0.0 | 89.1 | 0.0 |
| | Fault6 | 20.9 | 0.0 | 46.4 | 0.0 | 0.0 | 80.0 |

**Table 6.3:** Matching rates for testing fault data. Values are in percentages. Multiple room HVAC model.

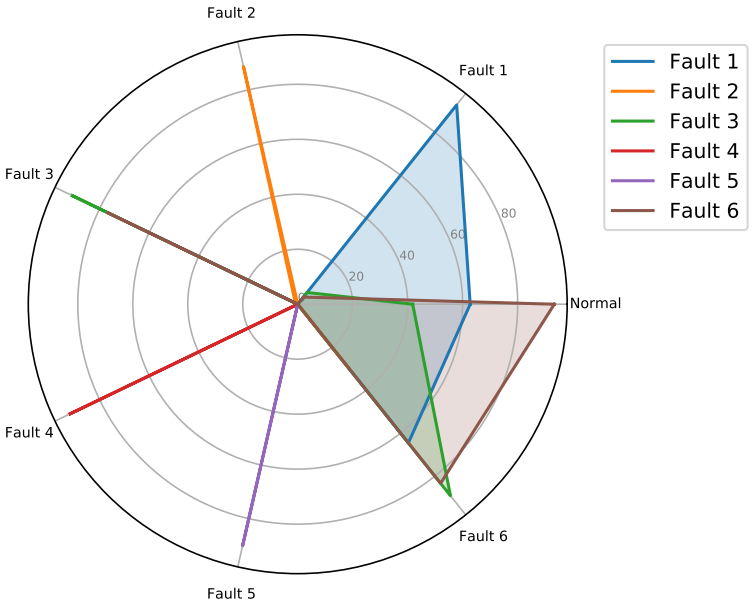



(a) Radar chart of matching rates for each testing fault using LOF.

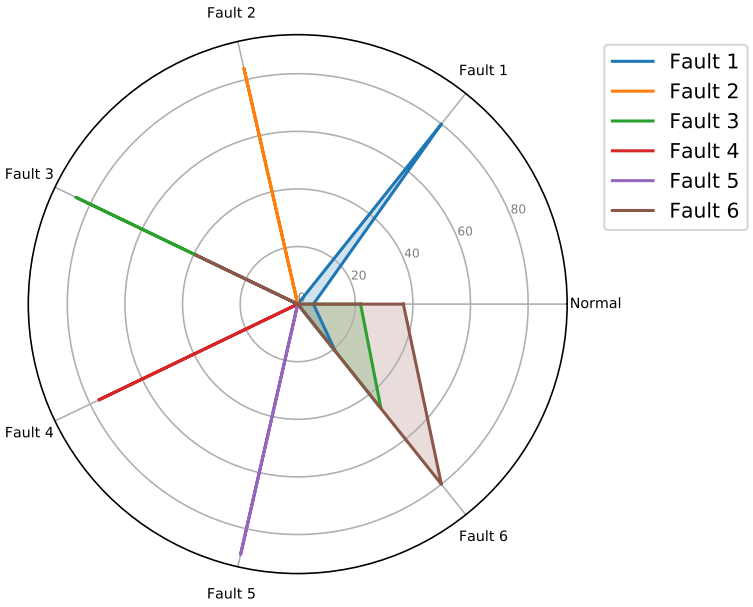



(b) Radar chart of matching rates for each testing fault using iForest.

**Figure 6.1:** Radar chart of matching rates. Values are in percentages. Single room HVAC model.




(a) Radar chart of matching rates for each testing fault using LOF.




(b) Radar chart of matching rates for each testing fault using iForest.

**Figure 6.2:** Radar chart of matching rates. Values are in percentages. Multiple room HVAC model.

The result shown here tells us that if we have some historical data of the system, these records could provide us some sort of 'guess' about what the system fault might be. Each testing dataset is tested against different training datasets individually; this means that each test is conducted independently; thus, even if we do not know all the faults beforehand, we can still try out this approach to see what the faults might be like. That is, once the HVAC system is detected as faulty, if some historical data is available, this matching process could be helpful in further fault diagnosis.

## 6.4 A classification approach for fault diagnosis

In contrast to using anomaly detection algorithms as faulty pattern matching tools in the previous section, we will work on another approach here. Since we are making the assumption that faulty data are available, this means that we have prior knowledge about the faults and our data has class (fault type) labels; a fault diagnosis task can then be considered as a classification problem (supervised learning). That is, suppose manipulated variables carry the information of a system's state, we would be able to spot differences if some fault is introduced. Also, if we have datasets of various types of faults, we could diagnose a system's state depending on how complete the datasets are. This means that all classification algorithms can be regarded as candidates.

In order to run classification algorithms on our data, we would first need to re-structure and add labels to our datasets. For each testing data sample, we find the nearest weather profiles in each dataset, then we can pull out time-series data of their corresponding manipulated variables. Next we add class labels to these sets of time-series data according to their parent dataset and stack these data for each manipulated variable. The data structure and this process are shown in figure 6.3. These stacked data will then be used as training datasets for the classifiers.

Here we will test out a few different classifiers. Which classifier should one choose is a tricky question, for every classifier has different characteristics. Generally speaking, there is no best classification algorithm; it depends on the problem itself. An algorithm may be powerful and accurate for some cases, and weak in other conditions. Take support vector machine (SVM) for example: SVM is known for its strength in handling high dimensional data very well, while it struggles to work on large datasets due to its algorithm computational cost, $\mathcal{O}(FN^3)$ [94], where $F$ is the number of features and $N$ is the number of samples.

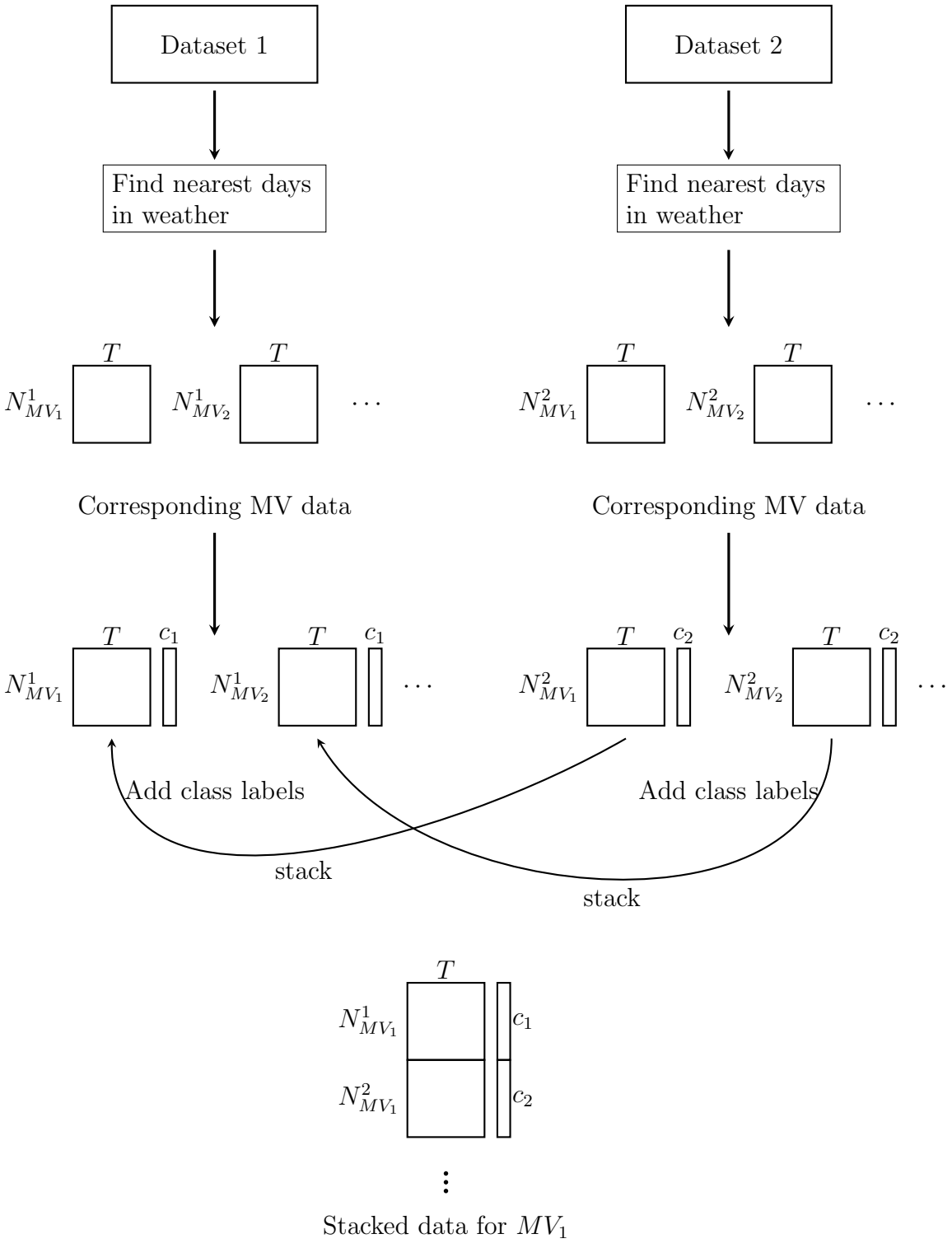



**Figure 6.3:** A diagram showing the stacked data structure.

### 6.4.1 Fault diagnosis results and discussion

Compared to our work in section 6.3, we are now assuming that all faulty datasets are revealed. The task becomes straightforward; we pull out corresponding manipulated variable data and run classification algorithms on these test data. A number of classification algorithms are selected for our test, including linear-SVM (linear-support vector machine), RBF-SVM (radial basis function-support vector machine), GPC (Gaussian process classifier), kNN (k-nearest neighbors), and SGD-logistic (logistic regression using stochastic gradient descent).

As in section 6.3, we will start with working on our single room HVAC model first. After preprocessing the data by restructuring and adding labels as in figure 6.3, we apply the classifiers to our data; the correctness rates[45] results are shown in table 6.4. We find that linear-SVM works very well in classifying our time-series data. This could be due to the fact that SVM is good at working with high dimensional data. kNN and RBF-SVM classifiers work pretty good as well; however, we do notice that all classifiers seems to perform relatively bad for fault 6. Again, if we go back and check with the results of section 4, we find that fault 6 is a very minor fault that is barely detectable. It is surprising that linear-SVM is able to classify fault 6 with such a high correctness rate.

| | | Classifier | | | | |
|---|---|---|---|---|---|---|
| | | GPC | RBF-SVM | SGD-logistic | kNN | linear-SVM |
| Fault | Fault 1 | 1.000 | 1.000 | 0.533 | 1.000 | 1.000 |
| | Fault 2 | 0.725 | 0.954 | 0.119 | 0.954 | 1.000 |
| | Fault 3 | 0.275 | 0.807 | 0.101 | 0.908 | 0.945 |
| | Fault 4 | 0.157 | 0.889 | 0.111 | 0.926 | 1.000 |
| | Fault 5 | 0.062 | 0.805 | 0.080 | 0.832 | 1.000 |
| | Fault 6 | 0.038 | 0.404 | 0.163 | 0.481 | 0.750 |

**Table 6.4:** Classification correctness rates for single room HVAC model.

The same classification process is carried out for our multiple room HVAC model. The results are shown table 6.5. Again, linear-SVM performs best for our multiple room HVAC model. kNN and RBF-SVM delivers a decent classification result. As for fault 6, it not only is a weaker version of fault 3, but also has less data samples than the rest (30 samples compared to 100+). This explains why the classification results of fault 6 are the worst among all classifiers. The poor results of fault 6 with a smaller

[45]Note that true positive rates and false negative rates are used for binary classifiers; however, our task here is a multi-class classification; we will use correctness rate to represent the ratio of number of samples classified correctly to the total number of samples.

| | | Classifier | | | | |
|---|---|---|---|---|---|---|
| | | GPC | RBF-SVM | SGD-logistic | kNN | linear-SVM |
| Fault | Fault 1 | 0.509 | 0.973 | 0.227 | 1.000 | 1.000 |
| | Fault 2 | 1.000 | 0.877 | 0.708 | 0.943 | 1.000 |
| | Fault 3 | 0.218 | 0.673 | 0.245 | 0.755 | 0.973 |
| | Fault 4 | 0.464 | 0.682 | 0.082 | 0.827 | 1.000 |
| | Fault 5 | 0.873 | 0.991 | 0.191 | 0.973 | 1.000 |
| | Fault 6 | 0.167 | 0.233 | 0.133 | 0.267 | 0.633 |

**Table 6.5:** Classification correctness rates for multiple room HVAC model.

sample size reminds us the fact that it is known in the machine learning community that more data is better than a strong classifier. Also, if one would prefer to improve the results, such as the SGD-logistic classifier, one could apply ensemble methods (e.g., bagging and boosting) to incorporate multiple classifiers.

In this section, we have demonstrated that if historical faulty datasets are recorded and available, then a fault diagnosis task can be carried out using raw manipulated variable time-series data of the control system alone.

# 7 Conclusion and future work

## 7.1 Conclusion

Unless a perfect system is developed and built, a fault detection and diagnosis task will always be needed. FDD has been considered costly and tedious but necessary. Therefore, people have started to develop FDD systems to automate these laborious and routine jobs. This has also been the case for HVAC systems as well. Various kinds of approaches have been proposed and all have their own strengths and weaknesses. In this thesis, we have proposed an approach that focuses on being scalable and economical. This idea is introduced in section 1.

In order to achieve our goal of being scalable, we have introduced the notion of focusing on control systems (section 3.3), for they are considered to be universal in most systems, including HVAC systems. Moreover, due to the nature of controllers that tend to hide faults, exploiting control data gains us the benefit of discovering hidden faults of a system, which would be a much harder task if we are only relying on sensor data. Sticking to our goal of being scalable and low cost, we have chosen to adopt machine learning and data analysis techniques using raw time-series data (section 3).

In section 2 we have introduced the use of Modelica as our source of simulated data. This choice is made due to two reasons; one is because of our limited access to real building data, and second, the ability to conduct experiments on multiple HVAC models and/or run tests on the exact same building under identical conditions. This would not be possible, or at least extremely expensive if we worked on real buildings.

Fault detection (sections 4 and 5) and diagnosis (section 6) experiments are carried out; it is demonstrated that a FDD task can be simplified significantly (according to our set up and results, the number of features is shrunk from $\sim 10^4$ to $\sim 10^1$) by focusing on control variables alone. This not only saves us a great amount of effort to work on feature selection, but also helps us avoid the daunting high-dimensionality space. We have also learned that solving a problem with an insightful approach is easier and more efficient than throwing all kinds of data analysis tools to it.

Machine learning and data analysis techniques have caught eyes in all kinds of areas. According to our experiment results, it is shown that data-driven FDD approaches have great potential. Exploring tools, such as clustering algorithms, were able to help us identify the similarities among different days in the weather datasets. Anomaly algorithms were able to learn the rules by comparing raw data without an expert to explicitly list out the rules. These techniques become more and more important as datasets grow larger.

## 7.2 Future work

Because of the limited time and computational resources, we have shown our FDD approach for two different types of HVAC system models; this could be further extended to more variants of HVAC systems in order to be more robust. Also, more datasets could be collected and tested.

We have mentioned the high cost to work on real buildings; however, it would be best if we could get our hands on a real one to test out and verify our work. Another possible future work is to test out our approach on other types of systems instead of HVAC systems. Since we have been focusing on control data, theoretically speaking, our approach should also be valid for any system that incorporates a control system in general. Although this sounds like taking an indirect path, this could provide us a cheaper way of validation and discover more potential applications of this FDD approach.

Moreover, in light of the fast growth of data science, a wide variety of data analytical tools and the development of their infrastructures, it is almost certain that our work can be improved and benefit from new algorithms and cloud services.

# Bibliography


[1] *Residential Energy Consumption Survey*. 2009. URL: https://www.eia.gov/todayinenergy/detail.php?id=9951&src=%E2%80%B9%20Consumption%20%20%20%20%20%20Residential%20Energy%20Consumption%20Survey%20(RECS)-f4.

[2] *Commercial Buildings Energy Consumption Survey*. 2017. URL: https://www.eia.gov/consumption/commercial/reports/2012/lighting/?src=%E2%80%B9%20Consumption%20%20%20Commercial%20Buildings%20Energy%20Consumption%20Survey%20(CBECS)-b1.

[3] John A. Rice. *Mathematical Statistics and Data Analysis*. 2006.

[4] Philip Haves. "Overview of Diagnostic Methods". In: *Proceedings of the workshop on diagnostics for commercial buildings: From research to practice* (1999).

[5] Tim Salsbury and Rick Diamond. "Performance Validation and Energy Analysis of HVAC Systems using Simulation". In: *Energy and Buildings* (2000).

[6] Rolf Isermann. "Process Fault Detection Based on Modeling and Estimation Methods - A Survey". In: *Automatica* (1984).

[7] *Overview of Diagnostic Methods*. URL: https://pdfs.semanticscholar.org/2cdf/fe2157dc0a6dcc9f1212dcdf8f4ed59e6c4c.pdf.

[8] James Shia and David M. Auslander. *Continuous Commissioning of Buildings*. Report. UC Berkeley, 2012.

[9] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*.

[10] *Modelica*. URL: https://www.modelica.org/.

[11] David M. Auslander. "Modelica MultiBody Library: Tutorial By Example".

[12] *OpenModelica*. URL: https://openmodelica.org/.

[13] *Jmodelica.org*. URL: http://jmodelica.org/.

[14] Ross Montgomery and Robert McDowall. *Fundamentals of HVAC Control Systems*. 2009.

[15] ASHRAE. *2013 ASHRAE Handbook - Fundamentals*. 2013.

[16] ASHRAE. *2015 ASHRAE HANDBOOK - Heating, Ventilating, and Air-Conditioning Applications*. 2015.

[17] Ronald H. Howell, William J. Coad, and Jr. Harry J. Sauer. *Principles of Heating Ventilating and Air Conditioning*. 2013.

[18] Michael Wetter. "Fan and Pump Model that has a Unique Solution for any Pressure Boundary Condition and Control Signal". In: *Proceedings of BS2013* (2013). URL: http://simulationresearch.lbl.gov/modelica/releases/latest/Resources/Images/Fluid/Movers/UsersGuide/2013-IBPSA-Wetter.pdf.

[19] Samuel R. West et al. "Automated Fault Detection and Diagnosis of HVAC Subsystems Using Statistical Machine Learning". In: *Proceedings of Building Simulation* (Jan. 2011).

[20] Massieh Najafi. "Fault Detection and Diagnosis in Building HVAC Systems". UC Berkeley, 2010.

[21] K. Verbert, R. Babuška, and B. De Schutter. "Combining knowledge and historical data for system-level fault diagnosis of HVAC systems". In: *Engineering Applications of Artificial Intelligence* (Mar. 2017).

[22] W.J.N. Turner, A. Staino, and B. Basu. "Residential HVAC fault detection using a system identification". In: *Energy and Buildings* (June 2017).

[23] Philip Michael Van Every et al. "Advanced detection of HVAC faults using unsupervised SVM novelty detection and Gaussian process models". In: *Energy and Buildings* (May 2017).

[24] Xue-Bin Yang et al. "A novel model-based fault detection method for temperature sensor using fractal correlation dimension". In: *Building and Environment* (Oct. 2010).

[25] Shengwei Wang and Fu Xiao. "AHU sensor fault diagnosis using principal component analysis method". In: *Energy and Buildings* (2004).

[26] Shun Li and JinWen. "Application of pattern matching method for detecting faults in air handling unit system". In: *Automation in Construction* (2014).

[27] Shengwei Wang, Qiang Zhou, and Fu Xiao. "A system-level fault detection and diagnosis strategy for HVAC systems involving sensor faults". In: *Energy and Buildings* (2008).

[28] Siyu Wu and J.Q. Sun. "A top-down strategy with temporal and spatial partition for fault detection and diagnosis of building HVAC systems". In: *Energy and Buildings* (2010).

[29] A.Beghi et al. "Data-driven Fault Detection and Diagnosis for HVAC water chillers". In: *Control Engineering Practice* (2015).

[30] Yimin Chen and Jin Wen. "A Whole Building Fault Detection Using Weather Based Pattern Matching and Feature Based PCA Method". In: *2017 IEEE International Conference on Big Data* (2017).

[31] Balakrishnan Narayanaswamy et al. "Data Driven Investigation of Faults in HVAC Systems with Model, Cluster and Compare (MCC)". In: *BuildSys '14 Proceedings* (2014).

[32] Rong Lily Hu. "Machine Learning to Scale Fault Detection in Smart Energy Generation and Building Systems". UC Berkeley, 2016.

[33] Achmad Widodo and Bo-Suk Yang. "Support vector machine in machine condition monitoring and fault diagnosis". In: *Mechanical Systems and Signal Processing* (2006).

[34] J. Liang and R. Du. "Model-based Fault Detection and Diagnosis of HVAC systems using Support Vector Machine method". In: *International Journal of Refrigeration* (2006).

[35] K. Choi et al. "Fault Diagnosis in HVAC Chillers". In: *IEEE Proceedings AUTOTESTCON 2004* (2004).

[36] Yang Zhao, Shengwei Wang, and Fu Xiao. "Pattern recognition-based chillers fault detection method using Support Vector Data Description". In: *Applied Energy* (2012).

[37] William Hand Allen and Ahmed Rubaai. "Fuzzy-Neuro Health Monitoring System for HVAC System Variable-Air-Volume Unit". In: *IEEE Industry Applications Society Annual Meeting* (2013).

[38] C.H. Lo et al. "Fuzzy-genetic algorithm for automatic fault detection in HVAC systems". In: *Applied Soft Computing* (2006).

[39] Paul M. Frank and Birgit K oppen-Seliger. "Fuzzy Logic and Neural Network Applications to Fault Diagnosis". In: *International Journal of Approximate Reasoning* (1997).

[40] A. Parvaresh et al. "Fault Detection and Diagnosis in HVAC System Based on Soft Computing Approach". In: *International Journal of Soft Computing and Engineering* (2012).

[41] L.F. Mendonça, J.M.G. Sá da Costa, and J.M. Sousa. "Fault detection and diagnosis using fuzzy models". In: *European Control Conference (ECC)* (2003).

[42] Hanchuan Peng, Fuhui Long, and Chris Ding. "Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy". In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* (2005).

[43] Chris Ding and Hanchuan Peng. "Minimum redundancy feature selection from microarray gene expression data". In: *Journal of Bioinformatics and Computational Biology* (2003).

[44] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. 2017.

[45] Timothy J. Ross. *Fuzzy Logic with Engineering Applications*. 2010.

[46] Avraam Tapinos and Pedro Mendes. "A Method for Comparing Multivariate Time Series with Different Dimensions". In: *PlOS ONE* (2013).

[47] Zhimin Du and Xinqiao Jin. "Detection and diagnosis for sensor fault in HVAC systems". In: *Energy Conversion and Management* (Dec. 2006).

[48] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. "Time-series clustering – A decade review". In: *Information Systems* (May 2015).

[49] *Time series clustering*. URL: `https://xuedong.github.io/static/documents/time_series.pdf`.

[50] Sangeeta Rani and Geeta Sikka. "Recent Techniques of Clustering of Time Series Data: A Survey". In: *International Journal of Computer Applications* (Aug. 2012).

[51] T.Warren Liao. "Clustering of time series data - a survey". In: *Pattern Recognition* (Nov. 2005).

[52] Anthony Bagnall and Gareth Janacek. "Clustering Time Series with Clipped Data". In: *Machine Learning* (Oct. 2004).

[53] Félix Iglesias and Wolfgang Kastner. "Analysis of Similarity Measures in Time Series Clustering for the Discovery of Building Energy Patterns". In: *Energies* (2013).

[54] Jin Shieh and Eamonn Keogh. "iSAX: Indexing and Mining Terabyte Sized Time Series". In: *ACM* (2008).

[55] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Space". In: *International Conference on Database Theory* (2001).

[56] Meinard Müller. "Dynamic Time Warping". In: *Information Retrieval for Music and Motion*. 2007.

[57] Pablo Pedregal. *Introduction to Optimization*. 2004.

[58] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 2004.

[59] Giuseppe C. Calafiore and Laurent El Ghaoui. *Optimization models*. 2014.

[60] Stan Salvador and Philip Chan. "FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space". In: *Intelligent Data Analysis* (Oct. 2007). URL: `https://goo.gl/zeL4Wm`.

[61] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Feb. 2009.

[62] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2006.

[63] Micheal I. Jordan. "An Introduction to Probabilistic Graphical Models".

[64] Dorin Comaniciu and Peter Meer. "Mean Shift: A Robust Approach toward Feature Space Analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2002).

[65] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: (Aug. 1996).

[66] Malika Bendechache, Nhien-An Le-Khac, and M-Tahar Kechadi. “Efficient Large Scale Clustering based on Data Partitioning”. In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* (Oct. 2016).

[67] Mihael Ankerst et al. “OPTICS: Ordering Points To Identify the Clustering Structure”. In: *ACM Press* (1999).

[68] Malika Bendechache, Nhien-An Le-Khac, and M-Tahar Kechadi. “Efficient Large Scale Clustering based on Data Partitioning”. In: *Data Science and Advanced Analytics* (2017).

[69] Matt Duckham, Mike Worboys Lars Kulik, and Antony Galton. “Efficient generation of simple polygons for characterizing the shape of a set of points in the plane”. In: *Pattern Recognition* (2008).

[70] Ingwer Borg and Patrick J.F. Groenen. *Modern Multidimensional Scaling - Theory and Applications*. 2005.

[71] D. M. Hawkins. *Identification of Outliers*. 1980.

[72] Peter J. Rousseeuw and Annick M. Leroy. *Robust Regression and Outlier Detection*. 1987.

[73] Varun Chandola and Vipin Kumar Arindam Banerjee. “Anomaly Detection : A Survey”. In: *ACM Computing Surveys* (2009).

[74] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. 2006.

[75] I.T. Jolliffe. *Principal Component Analysis*. 2002.

[76] J. Edward Jackson. *A User’s Guide To Principal Components*. 1991.

[77] Markus M. Breunig et al. “LOF: Identifying Density-Based Local Outliers”. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (2000).

[78] Markus M. Breunig et al. “OPTICS-OF: Identifying Local Outliers”. In: *PKDD99* (1999).

[79] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation Forest”. In: *2008 Eighth IEEE International Conference on Data Mining* (2008).

[80] Rongpeng Zhang and Tianzhen Hong. “Modeling of HVAC operational faults in building performance simulation”. In: *Applied Energy* (2016).

[81] *On the Runtime-Efficacy Trade-off of Anomaly Detection Techniques for Real-Time Streaming Data*. Oct. 2017. URL: https://arxiv.org/abs/1710.04735.

[82] Pedro Domingos. “A Few Useful Things to Know about Machine Learning”. In: *Communications of the ACM* (2012).

[83] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. “REVIEW - A Survey on Unsupervised Outlier Detection in High-Dimensional Numerical Data”. In: *Statistical Analysis and Data Mining* (2012).

[84] Rongpeng Zhang and Tianzhen Hong. “Modeling of HVAC operational faults in building performance simulation”. In: *Applied Energy* (2016).

[85] Eamonn Keogh and Chotirat Ann Ratanamahatana. “Exact indexing of dynamic time warping”. In: *Knowledge and Information Systems* (2004).

[86] Alessandro Camerra et al. “iSAX 2.0: Indexing and Mining One Billion Time Series”. In: *ACM* (2010).

[87] Eamonn Keogh, Jessica Lin, and Ada Fu. “HOT SAX: Finding the Most Unusual Time Series Subsequence: Algorithms and Applications”. In: *ACM* (2005).

[88] *Using the Receiver Operating Characteristic (ROC) curve to analyze a classification model*. 2017. URL: `http://www.math.utah.edu/~gamez/files/ROC-Curves.pdf`.

[89] Tom Fawcett. “An introduction to ROC analysis”. In: *Pattern Recognition Letters* (2006).

[90] Annette J. Dobson and Adrian G. Barnett. *An Introduction to Generalized Linear Models*. 2008.

[91] David A. Freedman. *Statistical Models: Theory and Practice*. 2008.

[92] Jorge M. Lobo, Alberto Jiménez-Valverde, and Raimundo Real. “AUC: a misleading measure of the performance of predictive distribution models”. In: *Global Ecology and Biogeography* (2008).

[93] Blaise Hanczar et al. “Small-sample precision of ROC-related estimates”. In: *Bioinformatics* (2010).

[94] Abdiansah Abdiansah and Retantyo Wardoyo. “Time Complexity Analysis of Support Vector Machines (SVM) in LibSVM”. In: *International Journal of Computer Applications* (2015).

# Appendices

## A List of acronyms

- HVAC: Heating, Ventilation, and Air Conditioning
- FDD: Fault Detection and Diagnosis
- BTU: British Thermal Unit
- OAT: Outside Air Temperature
- SAT: Supply Air Temperature
- RAT: Return Air Temperature
- EAT: Exhaust Air Temperature
- OAD Outside Air Damper
- SAD: Supply Air Damper
- RAD: Return Air Damper
- EAD: Exhaust Air Damper
- AHU: Air Handling Unit
- CAV: Constant Air Volume
- VAV: Variable Air Volume
- DAE: Differential Algebraic Equations
- ODE: Ordinary Differential Equations
- AE: Algebraic Equations
- HVAC: Heating Ventilation and Air Conditioning
- FDD: Fault Detection and Diagnosis
- SVD: Singular Value Decomposition
- PCA: Principal Component Analysis
- PLS: Partial Least Squares
- PID: Proportional-Integral-Differential controller

- SP: Setpoint
- MV: Manipulated Variable
- PV: Process Variable
- PAA: Piecewise Aggregate Approximation
- SAX: Symbolic Aggregate approXimation
- HMM: Hidden Markov Model
- ARMA: Auto Regression Moving Average
- GP: Gaussian Process
- SSM: State Space Model
- FCD: Fractal Correlation Dimension
- ANN: Artificial Neural Network
- mRMR: minimalRedundancy-Maximal-Relevance
- MDS: MultiDimensional Scaling
- LOF: Local Outlier Factor
- iForest: Isolation Forest
- DTW: Dynamic Time Warping
- DBSCAN: Density-Based Spatial Clustering of Applications with Noise
- OPTICS: Ordering Points To Identify the Clustering Structure
- SVM: Support Vector Machine
- kNN: k-Nearest Neighbors
- GPC: Gaussian process classifier
- SGD: Stochastic Gradient Descent
- RBF: Radial Basis Function
- TPR: True Positive Rate
- FPR: False Positive Rate
- ROC: Receiver Operating Characteristic

- AUROC: Area Under Receiver Operating Characteristic
- AIC: Akaike Information Criterion
- BIC: Bayesian Information Criterion
- TMY3: Typical Meteorological Year 3
- DOE: Department of Energy
- NREL: National Renewable Energy Laboratory
- NOAA: National Oceanic and Atmospheric Administration
- EIA: U.S. Energy Information Administration
- PC: Personal Computer
- CPU: Central Processing Unit
- RAM: Random-Access Memory

# B Modelica models used

Modelica models used in our work which were not shown in previous sections.

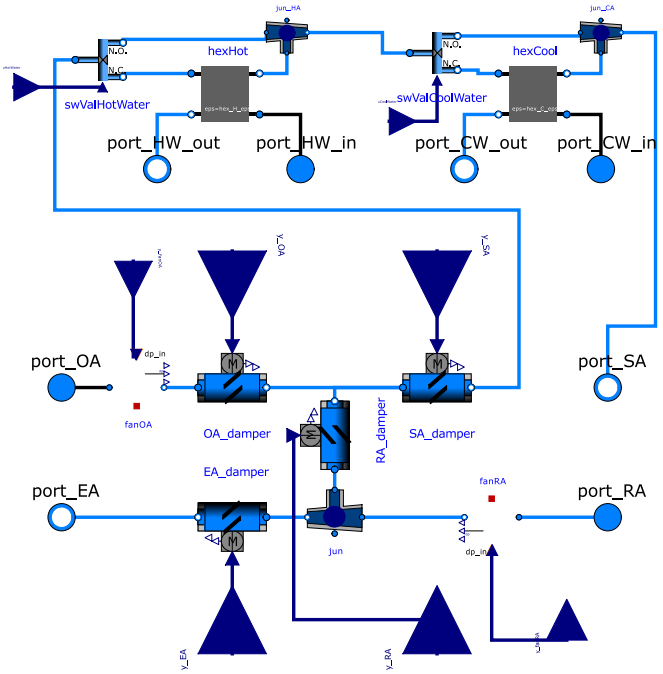



**Figure B.1:** AHU model used for our centralized multiple room HVAC system.

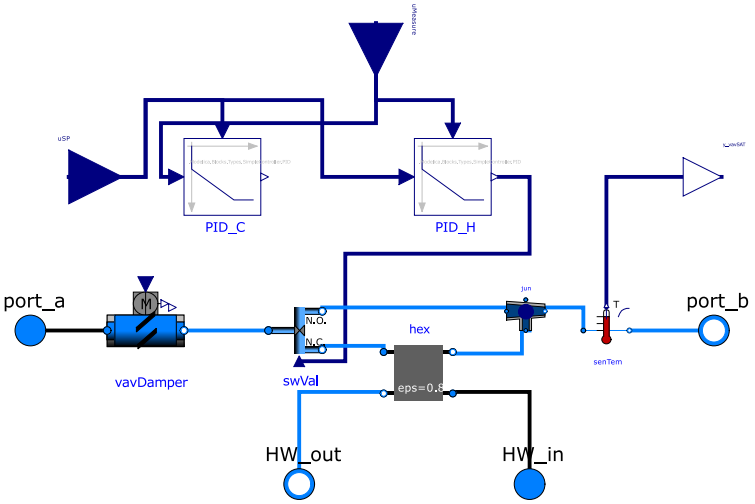



**Figure B.2:** VAV box model used for our centralized multiple room HVAC system.

# C Fault descriptions for sections 4, 5, and 6

Here we list the fault descriptions shown in tables 4.3, 5.1, and 6.2.

## C.1 Fault descriptions for section 4

Descriptions of faults for section 4 listed in table 4.3.

**Fault 1** A stuck/clogged hot water valve in the AHU. This fault is simulated by limiting the hot water valve control signal output from the control unit to a small range. Here we have the control signal set to [0.6,0.8], where the total range should be [0,1]. A 0 signal closes the hot water valve completely (no heating) while 1 will open it fully (heating at full power).

**Fault 2** A leaking air duct. This fault is assuming the supply air duct has an unsealed joint. The leaking air flow depends on the transient state of the system; however, it is around $0.4[m^3/s]$ based on one of our simulation results, that is about 30% of the total supply air flow.

**Fault 3** Cooling loop subsystem malfunctioning with a lower efficiency. In the cooling loop subsystem we used the `Carnot_TEva` type chiller. In our normal system, the chiller parameter `TSet` is set to $279.15[K]$ while in a faulty one it is set to $281.15[K]$. This simulates a higher cooling water temperature output (Still much colder than the required temperature).

**Fault 4** A malfunctioning thermostat. This is done by adding a pulse to the RAT sensor (offset 0.1[K], period 7200[s], amplitude 0.7[K]). This simulates a thermostat switch which has worn out and has a hysteresis effect.

**Fault 5** Same fault as Fault 2; however, a much minor leak. This is simulated by adding an orifice component with an effective leaking area of $0.01[m^2]$. Again, the leaking air flow is not a fixed number; however, based on simulation results it fluctuates between $0.17[m^3/s]$ to $0.2[m^3/s]$, that is around 12-15% of the total supply air flow.

**Fault 6** Same as Fault 4, but with a smaller offset. To be specific, this is done exactly the same as Fault 4, but we have changed the pulse parameter settings of the RAT sensor to (offset 0.1, period 43200, amplitude 0.1).

## C.2 Fault descriptions for section 5

Descriptions of faults for section 5 listed in table 5.1.

**Fault 1** Stuck/clogged VAV damper. This is done by replacing the VAV box with a faulty VAV box model. The faulty VAV box is made by limiting the VAV

damper position of one of the VAV boxes (there are three VAV boxes in this HVAC model) to [0.30,0.35]. Position 0 means the VAV damper is completely closed while position 1 means the VAV damper is fully open.

**Fault 2** Stuck/clogged heating water valve. This is done by replacing the VAV box with a faulty VAV box model. The faulty VAV box has a reheating hot water valve with its position limited to [0.01,0.05]. For a centralized HVAC system, the supply air is usually fixed to 13[°C]; if a higher supply air temperature is needed, the air will be reheated by the VAV box.

**Fault 3** A leaking supply air duct. This is simulated by adding an orifice with an effective area of $0.005[m^2]$ to the supply air duct. Again, the leaking air flow fluctuates, but is very close to $0.18[m^3/s]$ based on our simulation result. The total supply air flow for this model is about $4.15[m^3/s]$, so we have a leak of about 4%.

**Fault 4** Heating loop with a malfunctioning boiler. Hot water temperature is lower than normal. This is done by adjusting the floating switch setpoints of the boiler. That is, high temperature setpoint changed from 80°C to 70°C and low temperature setpoint changed from 75°C to 50°C (Our FDD work did not collect the setting parameters).

**Fault 5** A malfunctioning thermostat. This is done by adjusting the AHU setpoint from 13°C to 12°C. Note that this is not changing the temperature setpoints for individual rooms; the AHU setpoint controls the supply air temperature.

## C.3 Fault descriptions for section 6

Descriptions of faults for section 6 listed in table 6.2.

**Fault 1** Same as Fault 1 in C.2.

**Fault 2** Same as Fault 2 in C.2.

**Fault 3** Same as Fault 3 in C.2.

**Fault 4** Same as Fault 4 in C.2.

**Fault 5** Same as Fault 5 in C.2.

**Fault 6** Same as Fault 3; however, a much minor leak. A leaking supply air duct is simulated by adding an orifice with an effective area of $0.001[m^2]$ to the supply air duct. Again, the leaking air flow fluctuates, but is very close to $0.054[m^3/s]$ based on our simulation result. The total supply air flow for this model is about $4.16[m^3/s]$, so we have a leak of about 0.024%.

# D Python code for LOF, OPTICS, k-means, ... etc

## D.1 Python code for LOF

Code for Local Outlier Factor (LOF):

```
import numpy as np
class meLOF:
    def __init__(self,coords,MinPts = 10,threshold = 90,D=None):
        '''
        inputs:
            coords: data matrix of coordinates, in NF format (N by
                F)
                    Set to None if distance matrix D is provided
            MinPts: parameter of min points used for LOF, default
                is 10
            threshold: threshold percentile parameter, default is
                90, 0-100
            D: distance matrix (N by N), if not provided, will
                generate one)
        ----------------------
        Initializes meLOF
        needs numpy and scipy
        '''
        self.coords = coords
        self.MinPts = MinPts
        self.threshold = threshold

        if D is None:
            from scipy.spatial import distance
            dist = distance.minkowski
            D = self.gen_dist_mat(coords,dist)
        self.D = D

    def gen_dist_mat(self,X,dist_func=None,print_ = False):
        '''
        inputs:
            X: data matrix in NT(or NF) format
            dist_func: distance measure used, if not specified,
                will use Euclidean distance
            print_: if set to True, will print out progress during
                computations
        Outputs:
            D: distance matrix (N by N)
        ----------------------
        # Generate distance matrix (N*N)
        # uses Euclidean distance if not assigned
        # format should be NT
        # T: number of time steps in the time series
        # N: number of time series samples
        '''
        from scipy.spatial import distance
```

```
41          X = np.array(X)
42          N,T = X.shape[0],X.shape[1]
43
44          if dist_func is None:
45              dist = distance.minkowski
46          else:
47              dist = dist_func
48          # intialize distance matrix
49          D = np.zeros((N,N))
50          for i in range(N):
51              for j in range(N):
52                  if i==j: # identical
53                      D[i,j] = 0
54                  elif i > j: # distance matrix is symmetric, no need
                        to compute twice
55                      D[i,j] = D[j,i]
56                  else:
57                      D[i,j] = dist(X[i,:],X[j,:])
58              if print_: print('{}/{} finished'.format(i+1,N))
59          return(D)
60
61      def k_dist(self,D,k = 4):
62          '''
63          inputs:
64              D: distance matrix(N by N)
65              k: k-th neighbor distance, default is 4
66          '''
67          D = np.array(D)
68          N = D.shape[0]
69          # initialize k_dist vector
70          k_dist = np.zeros((N,1))
71          for i in range(N):
72              row = list(D[i,:])
73              for j in range(k):
74                  row.remove(min(row))
75              k_dist[i] = min(row)
76          return(k_dist)
77
78
79
80      def r_dist(self,D,k_distances,dist=None):
81          '''
82          inputs:
83              D: precomputed distance matrix(N by N)
84              k_distances: a list of precomputed k-distances.
85              dist: distance function
86          Outputs:
87              r_dists: reachability distance matrix (p,o), distance
                    of p(rows) from o(cols)
88          '''
89          if dist is None:
```

```
90              from scipy.spatial import distance
91              dist = distance.minkowski
92
93          N = D.shape[0]
94          r_dists = np.empty((N,N),dtype = float)
95
96          for i in range(N):
97              for j in range(N):
98                  r_dists[i,j] = np.max([k_distances[j],D[i,j]])
99          return(r_dists)
100
101
102      def nearest_neighbors(self,coords,k,D=None):
103          '''
104          inputs:
105              coords: data coordinates in NF format, ignored if
                     distance matrix D is provided
106              k: Parameter MinPts, the k-nearest neighbors
107              D: distance matrix, if not given, will use gen_dist_mat
                      to generate one
108          Outputs:
109              NN_dists: k nearest neighbors distances matrix, np.
                     array, (N by k)
110              NN: k nearest neighbors matrix, np.array, (N by k)
111                  Contains the indices of coords, NOT the coordinates
                         themselves
112          '''
113          if D is None:
114              from scipy.spatial import distance
115              dist = distance.minkowski
116              D = gen_dist_mat(coords,dist)
117          N = D.shape[0]
118          # initialize nearest neighbors
119          NN_dists = np.empty((N,k),dtype=float)
120          NN = np.empty((N,k),dtype=int)
121
122          for i in range(N):
123              # use numpy's structured array for sorting
124              dtype = [('distance',float),('index',int)]
125              structure_dist = np.empty((N,),dtype=dtype)
126              structure_dist['distance'] = D[i]
127              structure_dist['index'] = np.arange(N)
128              structure_dist = np.sort(structure_dist,order='distance
                     ')
129
130              # starts from 1 to remove itself, since the distance to
                      itself is always 0
131              NN_dists[i] = structure_dist['distance'][1:k+1]
132              NN[i] = structure_dist['index'][1:k+1]
133
134          return([NN,NN_dists])
```

```
135
136     def avg_r_dists(self,r_dists,NN):
137         '''
138         inputs:
139             r_dists: reachability distance matrix (p,o), distance
                    of p(rows) from o(cols)
140             NN: k-nearest neighbors matrix, np.array, (N by k)
141                 Contains the indices, NOT the coordinates
                        themselves
142         Outputs:
143             avg_r_dists: list of average reachable distances
144         '''
145         N = NN.shape[0] # number of samples
146         k = NN.shape[1] # number of neighbors
147         # initialize
148         avg_r_dists = np.empty((N,),dtype=float)
149         for i in range(N):
150             avg_r_dists[i] = np.mean(r_dists[i][NN[i]])
151
152         return(avg_r_dists)
153
154     def my_LOFs(self,NN,lrds):
155         '''
156         inputs:
157             NN: k-nearest neighbors matrix, np.array, (N by k)
158                 Contains the indices, NOT the coordinates
                        themselves
159             lrds: list of local reachability densities
160         Outputs:
161             LOFs: Local outlier factor scores
162         '''
163         N = NN.shape[0]
164         k = NN.shape[1]
165         # initialize
166         LOFs = np.empty((N,),dtype=float)
167         for i in range(N):
168             neighbor_lrds = lrds[NN[i]]
169             numerator = np.sum(neighbor_lrds)
170             lrd = lrds[i]
171             LOFs[i] = numerator/lrd/k
172
173         return(LOFs)
174
175
176     def labels_(self,myLOF_scores,threshold_percentile = 90):
177         '''
178         inputs:
179             myLOF_scores: LOF scores from function my_LOFs
180             threshold_percentile: threshold percentile parameter,
                    default is 90
181         Outputs:
```

```
182             myLOF_labels: 1 for inliers and -1 for outliers
183
184         ----------------------
185         LOF in sklearn, contamination sets the threshold, default
                is 0.1
186         check source code of fit_predict function:
187
188         self.threshold_ = -scoreatpercentile(
189                     -self.negative_outlier_factor_, 100. * (1. -
                        self.contamination))
190
191         '''
192         # check percentiles - set up threshold
193         my_lof_threshold = np.percentile(myLOF_scores,
                threshold_percentile)
194         myLOF_scores[myLOF_scores > my_lof_threshold]
195         # LOF labels
196         myLOF_labels = np.empty(myLOF_scores.shape,dtype=int)
197         for i,score in enumerate(myLOF_scores):
198             if score > my_lof_threshold:
199                 myLOF_labels[i] = -1
200             else:
201                 myLOF_labels[i] = 1
202
203         return(myLOF_labels)
204
205     def compute_scores_(self):
206         '''
207         outputs:
208             myLOF_scores: original LOFs
209         ----------------------
210         Uses default settings for everything
211
212         '''
213         # Generate distance matrix D
214         D = self.D
215         coords = self.coords
216         k=self.MinPts
217         k_distances = self.k_dist(D,k)
218
219         # reachability distances
220         r_dists = self.r_dist(D,k_distances)
221
222         # Nearest neighbors
223         NN, NN_dists = self.nearest_neighbors(coords,k,D)
224
225         # average reachability distances
226         r_dists_k = self.avg_r_dists(r_dists,NN)
227
228         # local reachability densities
229         lrds = 1/r_dists_k
```

```
230
231         # LOF scores
232         myLOF_scores = self.my_LOFs(NN,lrds)
233
234         return(myLOF_scores)
235
236
237     def compute_labels_(self):
238         '''
239         outputs:
240             myLOF_labels: 1 for inliers and -1 for outliers
241         ----------------------
242         Uses default settings for everything
243
244         '''
245         # LOF scores
246         myLOF_scores = self.compute_scores_()
247         # LOF labels
248         myLOF_labels = self.labels_(myLOF_scores, self.threshold)
249
250         return(myLOF_labels)
251
252
253     def scores_labels_(self):
254         '''
255         Outputs a tuple of myLOF_scores and myLOF_labels (
               myLOF_scores,myLOF_labels)
256         outputs:
257             myLOF_scores: original LOFs
258             myLOF_labels: 1 for inliers and -1 for outliers
259         ----------------------
260         Uses default settings for everything
261
262         '''
263         # LOF scores
264         myLOF_scores = self.compute_scores_()
265         # LOF labels
266         myLOF_labels = self.labels_(myLOF_scores, self.threshold)
267
268         return(myLOF_scores,myLOF_labels)
```

## D.2 Python code for OPTICS

Code for Ordering Points To Identify the Clustering Structure (OPTICS):

```
import numpy as np
class DataPoint:
    def __init__(self,index):
        self.core_dist = None # core distance
        self.r_dist = None # reachability distance
        self.processed = False # flag for processed
        # index of point, need indexing because we would be dealing
            with time-series,
        # may only have a distance matrix and don't have the
           coordinates
        self.index = index
        self.clusterID = None # cluster ID (label)

class meOPTICS:
    def __init__(self,coords,eps,eps2,MinPts,D = None,xi = 0.05):
        '''
        inputs:
            coords: data matrix of coordinates, in NF format (N by
               F)
            eps: the maximum distance (radius) to consider
            eps2: parameter epsilon prime < epsilon in OPTICS
            MinPts: parameter of min points used for LOF, default
               is 10
            D: distance matrix (N by N), if not provided, will
               generate one)
            xi: parameter for xi-steep points, 0 < xi < 1, default
               is set to 0.05
        -------------------------------------
        Initializes meOPTICS
        needs numpy and scipy
        '''
        self.coords = coords
        self.eps = eps
        self.eps2 = eps2
        self.MinPts = MinPts
        if D is None:
            from scipy.spatial import distance
            dist = distance.minkowski
            D = self.gen_dist_mat(coords,dist)
        self.D = D
        self.xi = xi
        # initialize data points
        DPs = [] # list of data points
        for i,datapoint in enumerate(coords):
            p = DataPoint(i)
            DPs.append(p)
        self.DPs = DPs
        # initialize nearest neighbors
```

```
43          self.NN, self.NN_dists = nearest_neighbors(None,MinPts,D)
44
45      def fit(self):
46          '''
47          Runs OPTICS
48          ----------------------
49          outputs:
50              order_list: returns the order_list with cluster ID/
                    label for the DataPoints(class)
51          '''
52          order_list = self.get_order()
53          order_list = self.cluster(order_list)
54          return(order_list)
55
56      def get_order(self):
57          '''
58          inputs from class attributes:
59              DPs: list of Data points(class DataPoints)
60              D: distance matrix (N by N)
61              eps: the radius parameter epsilon for OPTICS
62              MinPts: parameter of min points used for OPTICS
63          outputs:
64              order_list: list of DataPoints(class) with OPTICS'
                    ordering
65          '''
66          # initialize parameters
67          D = self.D
68          DPs = self.DPs
69          eps = self.eps
70          MinPts = self.MinPts
71          NN_dists = self.NN_dists
72          # initialize order_list
73          order_list = []
74          for p in DPs:
75              if not p.processed:
76                  neighbor_index = get_neighbors(p.index,eps,D)
77                  number_of_neighbors = neighbor_index.shape[0]
78                  p.processed = True
79                  order_list.append(p)
80                  # if core distance is not defined, means
                        reachability distance is not defined as well,
81                  # and the point is not a core object
82                  # expand the cluster order only if the point is a
                        core object
83                  if core_distance(p,NN_dists,number_of_neighbors) !=
                        None:
84                      seeds = [] # initialize orderseeds
85                      seeds = self.update_seeds(seeds,p,
                            neighbor_index,D)
86                      while len(seeds)>0: # while seeds is not empty
```

```
87                          # the smallest reachability-distance in the
                                seed-list is selected by the method
                               OrderSeeds:next().
88                          # c = seeds[0] # current object(datapoint)
                               // OrderSeeds.Next()
89                          c = seeds.pop(0) # current object(datapoint
                               ) // OrderSeeds.Next()
90                          c_neighbors = get_neighbors(c.index,eps,D)
91                          c.processed = True
92                          c.core_dist = core_distance(c,NN_dists,
                               c_neighbors.shape[0])
93                          order_list.append(c)
94
95                          if not c.core_dist is None:
96                              seeds = self.update_seeds(seeds,c,
                                   c_neighbors,D)
97
98          return(order_list)
99
100     def update_seeds(self,seeds,p,neighbors,D):
101         '''
102         inputs:
103             seeds: list of the order seeds
104             p: the datapoint
105             neighbors: list of neighbor indices
106             D: distance matrix (N by N)
107         outputs:
108             seeds: list of the updated order seeds
109         '''
110         # initialize parameters
111         NN_dists = self.NN_dists
112         DPs = self.DPs
113
114         c_dist = core_distance(p,NN_dists,neighbors.shape[0])
115
116         for o_index in neighbors:
117             o = DPs[o_index]
118
119             if not o.processed:
120                 new_r_dist = np.max([c_dist,D[p.index,o.index]])
121                 if o.r_dist is None:
122                     o.r_dist = new_r_dist
123                     seeds.append(o)
124                 else: # object already in seeds
125                     if new_r_dist < o.r_dist:
126                         o.r_dist = new_r_dist
127
128         # OrderSeeds are sorted by their reachability-distance
129         # sort seeds wrt r_dist, use structured np.array
130         # numpy sort doesn't support object type
131         dtype = [('r_dist',float),('index',int)]
```

```
132             N = len(seeds)
133             seedsArr = np.empty((N,),dtype=dtype)
134             seedsArr['index'] = np.arange(N)
135             for i,ob in enumerate(seeds):
136                 seedsArr['r_dist'][i] = seeds[i].r_dist
137             seedsArr = np.sort(seedsArr,order='r_dist')
138             # seeds sorted wrt reachability-distance
139             seeds = list(np.array(seeds)[seedsArr['index']])
140 
141             return(seeds)
142 
143         def cluster(self,order_list):
144             '''
145             Clustering by assigning a eps2 parameter, works as DBSCAN,
                    also called ExtractDBSCAN-Clustering
146             inputs:
147                 order_list: list of DataPoints(class) with OPTICS'
                        ordering
148             outputs:
149                 order_list: returns the order_list with cluster ID/
                        label for the DataPoints(class)
150             '''
151             # initialize parameters
152             eps2 = self.eps2
153             MinPts = self.MinPts
154             # clusterID = -1 # noise
155             clusterID = 0
156             for o in order_list:
157                 # assume UNDEFINED to be greater than any defined
                        distance
158                 r_dist = np.inf if o.r_dist is None else o.r_dist
159                 c_dist = np.inf if o.core_dist is None else o.core_dist
160 
161                 if r_dist > eps2:
162                     if c_dist <= eps2:
163                         clusterID += 1 # next clusterID
164                         o.clusterID = clusterID
165                     else:
166                         o.clusterID = -1 # noise
167                 else:
168                     o.clusterID = clusterID
169             return(order_list)
170 
171         def auto_cluster(self,order_list):
172             '''
173             Automatically cluster data points using steep point extract
                    cluster algorithm
174             inputs:
175                 order_list: list of DataPoints(class) with OPTICS'
                        ordering
176             outputs:
```

```
177             #order_list: returns the order_list with cluster ID/
                    label for the DataPoints(class)
178             clusters: a list of cluster areas [start,end] indices
                    wrt to order_list
179         ----------------------
180         Note: This outputs a hierachical list of clusters!
181         '''
182         # initialize parameter xi
183         xi = self.xi
184         MinPts = self.MinPts
185         # find the steep upward/downward areas
186         SUAset,SDAset = self.max_steep_area(order_list)
187         SUAset = np.array(SUAset) # row: steep upward area, col =
                start/end indices
188         SDAset = np.array(SDAset) # row: steep downward area, col =
                 start/end indices
189         # initialize
190         index = 1
191         mib_g = 0 # global max in between
192         N = len(order_list)
193         SDAlist = [] # list of the wanted steep down areas(
                SetOfSteepDownAreas in paper)
194         miblist = [] # list of steep down areas' mibs
195         clusters = [] # set of clusters
196         while index < N-1:
197             o = order_list[index]
198             # max between end of last steep upward or downward area
                    and current index
199             mib_g = np.max([mib_g , o.r_dist]) if not o.r_dist is
                    None else mib_g
200             if index in SDAset[:,0]: # start of a steep down area
                    at index
201                 # find the SDA in SDAset
202                 SDA = SDAset[SDAset[:,0]== index]
203                 # update mib values: max between end of steep down
                        region and current index
204                 for i,iSDA in enumerate(SDAlist):
205                     eoSDA = iSDA[0][1] # end of steep down area
                            index
206                     ps = order_list[eoSDA:index] # all points
                            between end of steep down area and current
                            index
207                     rs = [p.r_dist for p in ps] # reachability-
                            distances
208                     miblist[i] = np.max(rs)
209                 # Set local mib = 0
210                 miblist.append(0)
211                 # Add SDA to SDAlist
212                 SDAlist.append(SDA)
213                 # update index and mib_g
214                 index = SDA[0][1] + 1
```

```
215                     mib_g = order_list[index].r_dist # end of last
                            steep upward or downward area
216                 elif index in SUAset[:,0]: # start of a steep up area
                        at index
217                     # find the SUA in SUAset
218                     SUA = SUAset[SUAset[:,0]== index]
219                     # update mib values: max between end of steep down
                            region and current index
220                     for i,iSDA in enumerate(SDAlist):
221                         eoSDA = iSDA[0][1] # end of steep down area
                                index
222                         ps = order_list[eoSDA:index] # all points
                                between end of steep down area and current
                                index
223                         rs = [p.r_dist for p in ps] # reachability-
                                distances
224                         miblist[i] = np.max(rs)
225
226                     # update index and mib_g
227                     index = SUA[0][1] + 1
228                     mib_g = order_list[index].r_dist # end of last
                            steep upward or downward area
229                     for i,SDA in enumerate(SDAlist):
230                         # check combiniation of U and D is valid
231                         # find SDA's corresponing mib: just use
                                enumerate
232                         mib = miblist[i]
233                         # compare: "reachability-distance" of end of
                                SUA*(1-xi) >= mib
234                         eoSUA = order_list[SUA[0][1]].r_dist
235                         if eoSUA*(1-xi) >= mib:
236                             # cluster condition 4:
237                             s_D = order_list[SDA[0][0]] # s_D
238                             e_U = order_list[SUA[0][1]] # e_U
239                             e_U1 = order_list[SUA[0][1]+1] # e_U+1
240                             if s_D.r_dist*(1-xi) >= e_U1.r_dist: #
                                    cluster condition 4b
241                                 # ps = order_list[np.arange(SDA[0][0],
                                        SDA[0][1]+1)] # all points in SDA
242                                 ps = order_list[SDA[0][0]:SDA[0][1]+1]
                                        # all points in SDA
243                                 rs = np.array([p.r_dist for p in ps]) #
                                        reachability-distances of points in
                                        SDA
244                                 # rs = rs[rs > e_U1.r_dist]
245                                 s = ps[np.argmax(rs)] # point in SDA
                                        with max reachability-distance
246                                 e = e_U
247                             elif e_U1.r_dist*(1-xi) >= s_D.r_dist: #
                                    cluster condition 4c
```

```
248                                 # ps = order_list[np.arange(SUA[0][0],
                                        SUA[0][1]+1)] # all points in SUA
249                                 ps = order_list[SUA[0][0]:SUA[0][1]+1]
                                        # all points in SUA
250                                 rs = np.array([p.r_dist for p in ps]) #
                                        reachability-distances of points in
                                        SUA
251                                 s = s_D
252                                 e = ps[np.argmin(rs)] # point in SUA
                                        with min reachability-distance
253                             else: # cluster condition 4a
254                                 s = s_D
255                                 e = e_U
256                             # check cluster conditions 1,2,3a:
257                             # condition 1: s in SDA
258                             s_order_index = order_list.index(s)
259                             cond1 = SDA[0][0] <= s_order_index <= SDA
                                    [0][1]
260
261                             # condition 2: e in SUA
262                             e_order_index = order_list.index(e)
263                             cond2 = SUA[0][0] <= e_order_index <= SUA
                                    [0][1]
264
265                             # condition 3a: e-s > MinPts
266                             cond3a = e_order_index - s_order_index >
                                    MinPts
267
268                             # condition 3b is taken care by mib set up
269                             if (cond1 and cond2 and cond3a):
270                                 # clusters.append([s,e])
271                                 # store index wrt order_list
272                                 clusters.append([order_list.index(s),
                                        order_list.index(e)])
273                 else:
274                     index += 1
275         return(clusters)
276
277     def is_steep_point(self,order_list,p):
278         '''
279         inputs:
280             order_list: list of DataPoints(class) with OPTICS'
                    ordering
281             p: the datapoint
282             parameter for xi-steep points, 0 < xi < 1
283         outputs:
284             verdict: 'U' if is an upward steep point, 'D' if is a
                    downward steep point
285                     'X' if is not a steep point
286         ----------------------
```

```
287             Note: last point in order_list isn't defined upward or
                    downward
288             '''
289             # initialize parameter xi
290             xi = self.xi
291
292             N = len(order_list)
293             if p.index == order_list[-1].index:
294                 print('Last data point\'s steepness is undefined')
295                 return('X')
296
297             order_indices = [o.index for o in order_list]
298             # find p's index in order_list
299             index = order_indices.index(p.index)
300             # find next data point in order_list   # next_index = index
                    + 1
301             o = order_list[index + 1]
302
303             verdict = 'X'
304             if (not p.r_dist is None):
305                 if p.r_dist <= o.r_dist * (1-xi):
306                     verdict = 'U'
307                 elif o.r_dist <= p.r_dist * (1-xi):
308                     verdict = 'D'
309
310             return(verdict)
311
312         def is_steep_area(self,order_list,s,e):
313             '''
314             Checks the first 3 conditions:
315             1. s and e are steep upward/downward points
316             2. reachability-distances are monotonically increasing/
                    decreasing
317             3. doesn't contain more than MinPts of consecutive non-
                    steep points
318             ----------------------
319             inputs:
320                 order_list: list of DataPoints(class) with OPTICS'
                        ordering
321                 s: starting data point in area
322                 e: ending data point in area
323             outputs:
324                 verdict: 'UA' if is an upward steep area, 'DA' if is a
                        downward steep area
325                       'XA' if is not a steep point
326             '''
327             # initialize paramters
328             MinPts = self.MinPts
329             is_steep_point = self.is_steep_point
330             # find start, end indices in order_list
331             order_indices = [o.index for o in order_list]
```

```
332             start_index = order_indices.index(s.index)
333             end_index = order_indices.index(e.index)
334             # check upward area
335             cond1 =  is_steep_point(order_list,s) == 'U' and
                    is_steep_point(order_list,e) == 'U'
336             cond2 = True
337             count = 0 # counter for condition 3: number of consecutive
                    points that are not steep upward
338             for i in range(start_index,end_index):
339                 cond2 = cond2 and (order_list[i].r_dist <= order_list[i
                        +1].r_dist)
340                 if is_steep_point(order_list,order_list[i]) != 'U':
                        count += 1
341             cond3 = ( count < MinPts )
342             upward = cond1 and cond2 and cond3
343             # check downward area
344             cond1 =  is_steep_point(order_list,s) == 'D' and
                    is_steep_point(order_list,e) == 'D'
345             cond2 = True
346             count = 0 # counter for condition 3: number of consecutive
                    points that are not steep downward
347             for i in range(start_index,end_index):
348                 cond2 = cond2 and (order_list[i].r_dist >= order_list[i
                        +1].r_dist)
349                 if is_steep_point(order_list,order_list[i]) != 'D':
                        count += 1
350             cond3 = ( count < MinPts )
351             downward = cond1 and cond2 and cond3
352
353             # assign verdict
354             if upward:
355                 verdict = 'UA'
356             elif downward:
357                 verdict = 'DA'
358             else:
359                 verdict = 'XA'
360             return(verdict)
361
362         def max_steep_area(self,order_list):
363             '''
364             Checks the 4th condition: steep area is maximal
365             ----------------------
366             inputs:
367                 order_list: list of DataPoints(class) with OPTICS'
                        ordering
368                 # s: starting data point in area
369                 # e: ending data point in area
370             outputs:
371                 [USA,DSA]:
372                 - USA: list of upward steep areas
373                 - DSA: list of downward steep areas
```

```
374                 areas are defined by [starting index, ending index
                        ], indices wrt to order_list, not coords
375         '''
376         # initialize paramters
377         MinPts = self.MinPts
378         is_steep_point = self.is_steep_point
379         is_steep_area = self.is_steep_area
380 
381         # Get list of steep points
382         steep_points = [is_steep_point(order_list,order_list[i])
                for i in range(len(order_list)-1)]
383         steep_points = np.array(steep_points)
384         upward_index = np.arange(steep_points.shape[0])[
                steep_points == 'U']
385         downward_index = np.arange(steep_points.shape[0])[
                steep_points == 'D']
386         # find all upward areas
387         USA = []
388         for i,ui in enumerate(upward_index):
389             for j,uj in enumerate(upward_index[i:]):
390                 if (is_steep_area(order_list,order_list[ui],
                        order_list[uj]) == 'UA'): USA.append([ui,uj])
391         # include only max areas
392         i = 0
393         while i < len(USA):
394             flag = False
395             ui = USA[i]
396             for j,uj in enumerate(USA):
397                 if ((uj[0] < ui[0] and uj[1] >= ui[1]) or (uj[0] <=
                        ui[0] and uj[1] > ui[1])):
398                     USA.pop(i)
399                     flag = True
400                     break
401             if not flag: i += 1
402         # find all downward areas
403         DSA = []
404         for i,ui in enumerate(downward_index):
405             for j,uj in enumerate(downward_index[i:]):
406                 if (is_steep_area(order_list,order_list[ui],
                        order_list[uj]) == 'DA'): DSA.append([ui,uj])
407         # include only max areas
408         i = 0
409         while i < len(DSA):
410             flag = False
411             ui = DSA[i]
412             for j,uj in enumerate(DSA):
413                 if ((uj[0] < ui[0] and uj[1] >= ui[1]) or (uj[0] <=
                        ui[0] and uj[1] > ui[1])):
414                     DSA.pop(i)
415                     flag = True
416                     break
```

```
417                 if not flag: i += 1
418             return(USA,DSA)
419
420         def gen_dist_mat(self,X,dist_func=None,print_ = False):
421             '''
422             inputs:
423                 X: data matrix in NT(or NF) format
424                 dist_func: distance measure used, if not specified,
                        will use Euclidean distance
425                 print_: if set to True, will print out progress during
                        computations
426             Outputs:
427                 D: distance matrix (N by N)
428             ----------------------
429             # Generate distance matrix (N*N)
430             # uses Euclidean distance if not assigned
431             # format should be NT
432             # T: number of time steps in the time series
433             # N: number of time series samples
434             '''
435             from scipy.spatial import distance
436             X = np.array(X)
437             N,T = X.shape[0],X.shape[1]
438
439             if dist_func is None:
440                 dist = distance.minkowski
441             else:
442                 dist = dist_func
443             # intialize distance matrix
444             D = np.zeros((N,N))
445             for i in range(N):
446                 for j in range(N):
447                     if i==j: # identical
448                         D[i,j] = 0
449                     elif i > j: # distance matrix is symmetric, no need
                            to compute twice
450                         D[i,j] = D[j,i]
451                     else:
452                         D[i,j] = dist(X[i,:],X[j,:])
453                 if print_: print('{}/{} finished'.format(i+1,N))
454             return(D)
455
456         def nearest_neighbors(self,coords,k,D=None):
457             '''
458             inputs:
459                 coords: data coordinates in NF format, ignored if
                        distance matrix D is provided
460                 k: Parameter MinPts, the k-nearest neighbors
461                 D: distance matrix, if not given, will use gen_dist_mat
                        to generate one
462             Outputs:
```

```
463                 NN_dists: k nearest neighbors distances matrix, np.
                        array, (N by k)
464                 NN: k nearest neighbors matrix, np.array, (N by k)
465                     Contains the indices of coords, NOT the coordinates
                            themselves
466             '''
467             if D is None:
468                 from scipy.spatial import distance
469                 dist = distance.minkowski
470                 D = gen_dist_mat(coords,dist)
471             N = D.shape[0]
472             # initialize nearest neighbors
473             NN_dists = np.empty((N,k),dtype=float)
474             NN = np.empty((N,k),dtype=int)
475
476             for i in range(N):
477                 # use numpy's structured array for sorting
478                 dtype = [('distance',float),('index',int)]
479                 structure_dist = np.empty((N,),dtype=dtype)
480                 structure_dist['distance'] = D[i]
481                 structure_dist['index'] = np.arange(N)
482                 structure_dist = np.sort(structure_dist,order='distance
                        ')
483                 # starts from 1 to remove itself, since the distance to
                        itself is always 0
484                 NN_dists[i] = structure_dist['distance'][1:k+1]
485                 NN[i] = structure_dist['index'][1:k+1]
486             return([NN,NN_dists])
487
488         def get_neighbors(self,p_index,eps,D):
489             '''
490             inputs:
491                 p_index: data point index
492                 eps: the maximum distance (radius) to consider
493                 D: distance matrix
494             Outputs:
495                 neighbor_index: a list of neighboring point indices
                        with distance < eps
496             '''
497             eps = self.eps
498             D = self.D
499             N = D.shape[0]
500             # initialize
501             neighbor_index = []
502             for i in range(N):
503                 if p_index != i: # exclude itself
504                     if D[p_index,i] < eps:
505                         neighbor_index.append(i)
506
507             neighbor_index = np.array(neighbor_index)
508             return(neighbor_index)
```

## D.3 Python code for k-means clustering

Code for k-means clustering:

```
import numpy as np
from scipy.spatial import distance
import copy
## class
class DataPoint:
    def __init__(self,index,isCenter=False):
        self.coord = None # coordinates, could be None if only
            given a distance matrix
        self.index = index # the order numbering from given data
        self.label = None # cluster assignment
        self.isCenter = isCenter # a boolean that indicates if this
            data point is a cluster center

class KMeans:
    def __init__(self,X,k,dist_func=distance.minkowski,initSeeds=
        None,tol=1e-5,max_iter=100):
        '''
        Inputs:
            X: Data matrix, np.array(n_samples*n_dimensions)
            k: number of clusters, integer
            dist_func: distance measure function, default is
                distance.minkowski
            initSeeds: assigning the initial cluster centers np.
                array(k * n_dimensions)
                        if set to None, then random generates
                            centers
            tol: tolerance for convergence, if centers difference <
                tol, then is considered as converged
            max_iter: max iterations, integer
        '''
        # initialize
        self.X = X
        self.k = int(k)
        self.dist = dist_func
        self.initSeeds = initSeeds
        self.max_iter = max_iter
        self.labels = np.zeros((X.shape[0],))
        self.centers = []
        self.tol = tol

        if not initSeeds==None and k != initSeeds.shape[0]:
            print('number of clusters does not match with initSeeds
                !')
        elif self.k < 2:
            print('should have at least 2 clusters!')

        # initialize data points
        DPs = [] # list of data points
```

```
        for i,datapoint in enumerate(X):
            p = DataPoint(i)
            p.coord = X[i]
            DPs.append(p)
        self.DPs = DPs

    def setSeeds(self):
        '''
        Set up initial centers
        '''
        # initialize
        DPs = self.DPs
        k = self.k
        initSeeds = self.initSeeds

        centers = []
        for i in range(k):
            c = DataPoint(index=i,isCenter=True)
            c.label = int(i)
            centers.append(c)

        if initSeeds == None: # generate starting points
            N = len(DPs) # number of data points
            indices = np.random.randint(10,size = k)
            for i in range(k):
                centers[i].coord = DPs[indices[i]].coord
        else: # use initSeeds
            for i in range(k):
                centers[i].coord = initSeeds[i]
        self.centers = centers

    def cluster_one_step(self):
        '''
        Runs only one step
        ----------
        '''
        # one iteration:
        self.assign_labels()
        self.update_centers()

    def cluster(self):
        '''
        Runs the algorithm
        ----------
        '''
        # initialize
        max_iter = self.max_iter
        # initialize centers
        self.setSeeds()
        # start iterations:
        for i in range(max_iter):
```

```
92              pre_centers = copy.deepcopy(self.centers) # save
                    centers of previous step
93              self.assign_labels()
94              self.update_centers()
95              # if converged, then stop
96              if self.converged(pre_centers):
97                  print('converged at {} iteration'.format(i+1))
98                  break
99
100     def converged(self,pre_centers):
101         '''
102         check if converged
103         ----------
104         Inputs:
105             pre_centers: centers of previous step
106         Outputs:
107             returns True if converged
108         '''
109         # initialize
110         centers = self.centers
111         tol = self.tol
112         dist = self.dist
113         N = len(centers)
114         # calculate center changes
115         sum = 0
116         for i,c in enumerate(centers):
117             pc = pre_centers[i]
118             d = dist(c.coord,pc.coord)
119             sum += d
120
121         if sum <= tol:
122             return(True)
123         else:
124             return(False)
125
126     def assign_labels(self):
127         '''
128         Assignment step
129         '''
130         # initialize
131         DPs = self.DPs
132         centers = self.centers
133         dist = self.dist
134
135         for p in DPs:
136             dists = [] # list of distances to centers
137             for c in centers:
138                 d = dist(p.coord,c.coord)
139                 dists.append(d)
140             dists = np.array(dists)
141             index = np.argmin(dists) # find the closest center
```

```
142                 p.label = centers[index].label # assign cluster label
143
144             self.DPs = DPs # update
145
146         def update_centers(self):
147             '''
148             Update step
149             '''
150             # initialize
151             DPs = self.DPs
152             centers = self.centers
153             # dist = self.dist
154             clusters = {} # dictionary of lists, key= cluster label,
                    val = list of cluster data points(coords)
155             # list out data points according to their labels
156             for p in DPs:
157                 if not p.label in clusters.keys(): # initialize cluster
158                     clusters[p.label] = [p.coord]
159                 else:
160                     clusters[p.label].append(p.coord)
161             # update centers
162             for i,c in enumerate(centers):
163                 if i in clusters.keys(): # check if cluster has members
164                     coords = np.array(clusters[i])
165                     c.coord = np.mean(coords,axis=0)
166             self.centers = centers
167
168         def labels_(self):
169             '''
170             returns a np.array(n_samples,) of cluster labels
171             '''
172             # initialize
173             DPs = self.DPs
174             labels = []
175             for p in DPs:
176                 labels.append(p.label)
177             labels = np.array(labels)
178
179             return(labels)
180
181         def centers_(self):
182             '''
183             returns the center coordinates in np.array(n_centers *
                    n_dimensions) format
184             '''
185             # initialize
186             centers = self.centers
187             coord_list = []
188             for c in centers:
189                 coord_list.append(c.coord)
190             return(np.array(coord_list))
```

## D.4 Python code for randomly generated 2D data

Code for randomly generated example data:

```
import numpy as np

# set random seed
np.random.seed(0)

# example data
n_points_per_cluster = 120 # 300, number of data points in each
    cluster

C1 = [-5, -2] + .8 * np.random.randn(n_points_per_cluster, 2)
C2 = [4, -1] + .1 * np.random.randn(n_points_per_cluster, 2)
C3 = [1, -2] + .2 * np.random.randn(n_points_per_cluster, 2)
C4 = [-2, 3] + .3 * np.random.randn(n_points_per_cluster, 2)
C5 = [3, -2] + 1.6 * np.random.randn(n_points_per_cluster, 2)
C6 = [5, 6] + 2 * np.random.randn(n_points_per_cluster, 2)
coords = np.vstack((C1, C2, C3, C4, C5, C6))
```

# E Tabulated result data

| Distance measure | Anomaly algorithm | Experiment | Absolute | | Euclidean | | High-low | | SAX | |
|---|---|---|---|---|---|---|---|---|---|---|
| Rates | | | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate |
| | IsolationForest | 126_HVAC4a_Weather_Boston+SmallOffice_Weekday_random | 24.3 | 75.7 | 25.2 | 74.8 | 22.4 | 77.6 | 27.1 | 72.9 |
| | IsolationForest | 122_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 123_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault3 | 100.0 | 1.0 | 99.0 | 1.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 124_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault5 | 35.6 | 64.4 | 36.6 | 63.4 | 37.6 | 62.4 | 37.6 | 62.4 |
| | IsolationForest | 125_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault6 | 91.2 | 8.8 | 91.8 | 8.2 | 93.7 | 6.3 | 90.6 | 9.4 |
| | IsolationForest | 128_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault3_orifice | 86.9 | 13.1 | 87.9 | 12.1 | 79.4 | 20.6 | 86.0 | 14.0 |
| | IsolationForest | 129_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault6_2 | 22.2 | 77.8 | 22.2 | 77.8 | 25.0 | 75.0 | 23.1 | 76.9 |
| | IsolationForest | 131_HVAC4a_Boston+SmallOffice_Workday_random | 27.2 | 72.8 | 25.2 | 74.8 | 24.3 | 75.7 | 29.1 | 70.9 |
| | IsolationForest | 133_HVAC4a_Boston+SmallOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 134_HVAC4a_Boston+SmallOffice_Workday_Fault3 | 100.0 | 0.0 | 99.1 | 0.9 | 99.1 | 0.9 | 99.1 | 0.9 |
| | IsolationForest | 135_HVAC4a_Boston+SmallOffice_Workday_Fault5 | 47.7 | 52.3 | 46.7 | 53.3 | 45.8 | 54.2 | 46.7 | 53.3 |
| | IsolationForest | 136_HVAC4a_Boston+SmallOffice_Workday_Fault6 | 94.5 | 5.5 | 93.6 | 6.4 | 96.3 | 3.7 | 95.4 | 4.6 |
| | IsolationForest | 138_HVAC4a_Boston+SmallOffice_Workday_Fault3_orifice | 86.5 | 13.5 | 87.5 | 12.5 | 85.6 | 14.4 | 86.5 | 13.5 |
| | IsolationForest | 139_HVAC4a_Boston+SmallOffice_Workday_Fault6_2 | 29.9 | 70.1 | 30.8 | 69.2 | 25.2 | 74.8 | 29.9 | 70.1 |
| | IsolationForest | 141_HVAC4a_SF+SmallOffice_Workday_random | 10.3 | 89.7 | 8.4 | 91.6 | 13.1 | 86.9 | 19.6 | 80.4 |
| | IsolationForest | 143_HVAC4a_SF+SmallOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 144_HVAC4a_SF+SmallOffice_Workday_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 145_HVAC4a_SF+SmallOffice_Workday_Fault5 | 86.2 | 13.8 | 86.2 | 13.8 | 83.5 | 16.5 | 82.6 | 17.4 |
| | IsolationForest | 146_HVAC4a_SF+SmallOffice_Workday_Fault6 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 148_HVAC4a_SF+SmallOffice_Workday_Fault3_orifice | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 149_HVAC4a_SF+SmallOffice_Workday_Fault6_2 | 53.8 | 46.2 | 56.7 | 43.3 | 53.8 | 46.2 | 59.6 | 40.4 |
| | IsolationForest | 151_HVAC4a_SF+LargeOffice_Workday_random | 5.7 | 94.3 | 6.6 | 93.4 | 6.6 | 93.4 | 9.0 | 91.0 |
| | IsolationForest | 153_HVAC4a_SF+LargeOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 154_HVAC4a_SF+LargeOffice_Workday_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 155_HVAC4a_SF+LargeOffice_Workday_Fault5 | 36.1 | 63.9 | 30.6 | 69.4 | 36.1 | 63.9 | 27.8 | 72.2 |
| | IsolationForest | 156_HVAC4a_SF+LargeOffice_Workday_Fault6 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 158_HVAC4a_SF+LargeOffice_Workday_Fault3_orifice | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 159_HVAC4a_SF+LargeOffice_Workday_Fault6_2 | 40.0 | 60.0 | 30.5 | 69.5 | 36.2 | 63.8 | 52.4 | 47.6 |
| | IsolationForest | 161_HVAC4a_Boston+LargeOffice_Workday_random | 18.3 | 81.7 | 19.3 | 80.7 | 18.3 | 81.7 | 19.3 | 80.7 |
| | IsolationForest | 163_HVAC4a_Boston+LargeOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 164_HVAC4a_Boston+LargeOffice_Workday_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | IsolationForest | 165_HVAC4a_Boston+LargeOffice_Workday_Fault5 | 24.8 | 75.2 | 25.7 | 74.3 | 23.9 | 76.1 | 23.9 | 76.1 |
| | IsolationForest | 166_HVAC4a_Boston+LargeOffice_Workday_Fault6 | 96.4 | 3.6 | 95.5 | 4.5 | 94.5 | 5.5 | 91.8 | 8.2 |
| | IsolationForest | 168_HVAC4a_Boston+LargeOffice_Workday_Fault3_orifice | 96.5 | 3.5 | 94.7 | 5.3 | 93.8 | 6.2 | 94.7 | 5.3 |
| | IsolationForest | 169_HVAC4a_Boston+LargeOffice_Workday_Fault6_2 | 21.5 | 78.5 | 21.5 | 78.5 | 20.6 | 79.4 | 21.5 | 78.5 |
| | LOF | 126_HVAC4a_Weather_Boston+SmallOffice_Weekday_random | 21.5 | 78.5 | 21.5 | 78.5 | 19.6 | 80.4 | 27.1 | 72.9 |
| | LOF | 122_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | LOF | 123_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault3 | 96.0 | 4.0 | 96.0 | 4.0 | 96.0 | 4.0 | 96.0 | 4.0 |
| | LOF | 124_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault5 | 42.6 | 57.4 | 42.6 | 57.4 | 39.6 | 60.4 | 43.6 | 56.4 |
| | LOF | 125_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault6 | 94.3 | 5.7 | 94.3 | 5.7 | 97.5 | 2.5 | 95.0 | 5.0 |
| | LOF | 128_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault3_orifice | 79.4 | 20.6 | 79.4 | 20.6 | 79.4 | 20.6 | 80.4 | 19.6 |
| | LOF | 129_HVAC4a_Weather_Boston+SmallOffice_Weekday_Fault6_2 | 25.9 | 74.1 | 25.9 | 74.1 | 24.1 | 75.9 | 24.1 | 75.9 |
| | LOF | 131_HVAC4a_Boston+SmallOffice_Workday_random | 31.1 | 68.9 | 31.1 | 68.9 | 26.2 | 73.8 | 31.1 | 68.9 |
| | LOF | 133_HVAC4a_Boston+SmallOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | LOF | 134_HVAC4a_Boston+SmallOffice_Workday_Fault3 | 96.3 | 3.7 | 96.3 | 3.7 | 94.4 | 5.6 | 96.3 | 3.7 |
| | LOF | 135_HVAC4a_Boston+SmallOffice_Workday_Fault5 | 49.5 | 50.5 | 49.5 | 50.5 | 44.9 | 55.1 | 52.3 | 47.7 |
| | LOF | 136_HVAC4a_Boston+SmallOffice_Workday_Fault6 | 97.2 | 2.8 | 97.2 | 2.8 | 96.3 | 3.7 | 99.1 | 0.9 |
| | LOF | 138_HVAC4a_Boston+SmallOffice_Workday_Fault3_orifice | 82.7 | 17.3 | 82.7 | 17.3 | 79.8 | 20.2 | 85.6 | 14.4 |
| | LOF | 139_HVAC4a_Boston+SmallOffice_Workday_Fault6_2 | 30.8 | 69.2 | 29.0 | 71.0 | 26.2 | 73.8 | 31.8 | 68.2 |
| | LOF | 141_HVAC4a_SF+SmallOffice_Workday_random | 10.3 | 89.7 | 9.3 | 90.7 | 9.3 | 90.7 | 20.6 | 79.4 |
| | LOF | 143_HVAC4a_SF+SmallOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | LOF | 144_HVAC4a_SF+SmallOffice_Workday_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| | LOF | 145_HVAC4a_SF+SmallOffice_Workday_Fault5 | 85.3 | 14.7 | 84.4 | 15.6 | 80.7 | 19.3 | 82.6 | 17.4 |
| | LOF | 146_HVAC4a_SF+SmallOffice_Workday_Fault6 | 99.1 | 0.9 | 99.1 | 0.9 | 100.0 | 0.0 | 99.1 | 0.9 |
| | LOF | 148_HVAC4a_SF+SmallOffice_Workday_Fault3_orifice | 99.1 | 0.9 | 98.2 | 1.8 | 100.0 | 0.0 | 98.2 | 1.8 |
| | LOF | 149_HVAC4a_SF+SmallOffice_Workday_Fault6_2 | 41.3 | 58.7 | 41.3 | 58.7 | 48.1 | 51.9 | 44.2 | 55.8 |
| | LOF | 151_HVAC4a_SF+LargeOffice_Workday_random | 3.3 | 96.7 | 3.3 | 96.7 | 11.5 | 88.5 | 7.4 | 92.6 |
| | LOF | 153_HVAC4a_SF+LargeOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| LOF | 154_HVAC4a_SF+LargeOffice_Workday_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 0.0 |
| LOF | 155_HVAC4a_SF+LargeOffice_Workday_Fault5 | 30.6 | 69.4 | 30.6 | 69.4 | 31.5 | 68.5 | 29.6 | 70.4 | 0.0 |
| LOF | 156_HVAC4a_SF+LargeOffice_Workday_Fault6 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 0.0 |
| LOF | 158_HVAC4a_SF+LargeOffice_Workday_Fault3_orifice | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 0.0 |
| LOF | 159_HVAC4a_SF+LargeOffice_Workday_Fault6_2 | 16.2 | 83.8 | 17.1 | 82.9 | 21.9 | 78.1 | 24.8 | 75.2 | 0.0 |
| LOF | 161_HVAC4a_Boston+LargeOffice_Workday_random | 19.3 | 80.7 | 19.3 | 80.7 | 14.7 | 85.3 | 20.2 | 79.8 | 0.0 |
| LOF | 163_HVAC4a_Boston+LargeOffice_Workday_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 0.0 |
| LOF | 164_HVAC4a_Boston+LargeOffice_Workday_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 0.0 |
| LOF | 165_HVAC4a_Boston+LargeOffice_Workday_Fault5 | 23.9 | 76.1 | 23.9 | 76.1 | 22.0 | 78.0 | 22.0 | 78.0 | 0.0 |
| LOF | 166_HVAC4a_Boston+LargeOffice_Workday_Fault6 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 0.0 |
| LOF | 168_HVAC4a_Boston+LargeOffice_Workday_Fault3_orifice | 95.6 | 4.4 | 96.5 | 3.5 | 93.8 | 6.2 | 93.8 | 6.2 | 0.0 |
| LOF | 169_HVAC4a_Boston+LargeOffice_Workday_Fault6_2 | 16.8 | 83.2 | 17.8 | 82.2 | 15.9 | 84.1 | 20.6 | 79.4 | 0.0 |

| Distance measure | | Absolute | | Euclidean | | High-low | | SAX | |
|---|---|---|---|---|---|---|---|---|---|
| Rates | | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate |
| Anomaly algorithm | Experiment | | | | | | | | |
| IsolationForest | 201_HVAC6a_3room_Boston+SmallOffice_Workday_random | 8.2 | 91.8 | 7.3 | 92.7 | 8.2 | 91.8 | 8.2 | 91.8 |
| IsolationForest | 202_HVAC6a_3R_Boston+SmallOffice_Fault1 | 86.9 | 13.1 | 86.0 | 14.0 | 86.9 | 13.1 | 89.7 | 10.3 |
| IsolationForest | 203_HVAC6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 204_HVAC6a_3R_Boston+SmallOffice_Fault3 | 93.6 | 6.4 | 92.7 | 7.3 | 92.7 | 7.3 | 91.8 | 8.2 |
| IsolationForest | 205_HVAC6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 206_HVAC6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 211_HVAC6a_3room_Boston+LargeOffice_Workday_random | 7.3 | 92.7 | 6.4 | 93.6 | 8.2 | 91.8 | 8.2 | 91.8 |
| IsolationForest | 212_HVAC6a_3room_Boston+LargeOffice_Fault1 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 213_HVAC6a_3room_Boston+LargeOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 214_HVAC6a_3room_Boston+LargeOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 215_HVAC6a_3room_Boston+LargeOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 216_HVAC6a_3room_Boston+LargeOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 221_HVAC6a_3room_SF+SmallOffice_Workday_random | 7.2 | 92.8 | 8.1 | 91.9 | 7.2 | 92.8 | 14.4 | 85.6 |
| IsolationForest | 222_HVAC6a_3room_SF+SmallOffice_Fault1 | 99.1 | 0.9 | 98.2 | 1.8 | 98.2 | 1.8 | 97.3 | 2.7 |
| IsolationForest | 223_HVAC6a_3room_SF+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 224_HVAC6a_3room_SF+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 225_HVAC6a_3room_SF+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 226_HVAC6a_3room_SF+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 231_HVAC6a_3room_SF+LargeOffice_Workday_random | 6.4 | 93.6 | 3.6 | 96.4 | 4.5 | 95.5 | 2.7 | 97.3 |
| IsolationForest | 232_HVAC6a_3room_SF+LargeOffice_Fault1 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 233_HVAC6a_3room_SF+LargeOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 234_HVAC6a_3room_SF+LargeOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 235_HVAC6a_3room_SF+LargeOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 236_HVAC6a_3room_SF+LargeOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 201_HVAC6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 9.1 | 90.9 | 10.0 | 90.0 | 10.0 | 90.0 |
| LOF | 202_HVAC6a_3R_Boston+SmallOffice_Fault1 | 63.6 | 36.4 | 62.6 | 37.4 | 60.7 | 39.3 | 65.4 | 34.6 |
| LOF | 203_HVAC6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 99.1 | 0.9 | 100.0 | 0.0 | 99.1 | 0.9 |
| LOF | 204_HVAC6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 62.7 | 37.3 | 64.5 | 35.5 | 67.3 | 32.7 |
| LOF | 205_HVAC6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 206_HVAC6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 211_HVAC6a_3room_Boston+LargeOffice_Workday_random | 9.1 | 90.9 | 8.2 | 91.8 | 10.9 | 89.1 | 10.9 | 89.1 |
| LOF | 212_HVAC6a_3room_Boston+LargeOffice_Fault1 | 98.3 | 1.7 | 98.3 | 1.7 | 98.3 | 1.7 | 99.2 | 0.8 |
| LOF | 213_HVAC6a_3room_Boston+LargeOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 214_HVAC6a_3room_Boston+LargeOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 215_HVAC6a_3room_Boston+LargeOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 216_HVAC6a_3room_Boston+LargeOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 221_HVAC6a_3room_SF+SmallOffice_Workday_random | 8.1 | 91.9 | 7.2 | 92.8 | 2.7 | 97.3 | 10.8 | 89.2 |
| LOF | 222_HVAC6a_3room_SF+SmallOffice_Fault1 | 52.7 | 47.3 | 61.8 | 38.2 | 50.0 | 50.0 | 58.2 | 41.8 |
| LOF | 223_HVAC6a_3room_SF+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 224_HVAC6a_3room_SF+SmallOffice_Fault3 | 97.3 | 2.7 | 94.5 | 5.5 | 100.0 | 0.0 | 92.7 | 7.3 |
| LOF | 225_HVAC6a_3room_SF+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 226_HVAC6a_3room_SF+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 231_HVAC6a_3room_SF+LargeOffice_Workday_random | 8.2 | 91.8 | 5.5 | 94.5 | 1.8 | 98.2 | 4.5 | 95.5 |
| LOF | 232_HVAC6a_3room_SF+LargeOffice_Fault1 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 233_HVAC6a_3room_SF+LargeOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 234_HVAC6a_3room_SF+LargeOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 235_HVAC6a_3room_SF+LargeOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 236_HVAC6a_3room_SF+LargeOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 | 100.0 | 0.0 |

# F Tabulated result data for parameter selection

| Distance measure | | | | Euclidean | Euclidean | High-low | High-low |
|---|---|---|---|---|---|---|---|
| Rates | | | | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate |
| Anomaly algorithm | Contamination | Experiment | | | | | |
| IsolationForest | 0.001 | 130_HVACv4a_Boston+SmallOffice_Workday | | 13.29639889 | 86.70360111 | 11.63434903 | 88.36565097 |
| IsolationForest | 0.001 | 131_HVACv4a_Boston+SmallOffice_Workday_random | | 12.62135922 | 87.37864078 | 8.737864078 | 91.26213592 |
| IsolationForest | 0.001 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | | 100 | 0 | 100 | 0 |
| IsolationForest | 0.001 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | | 95.3271028 | 4.672897196 | 95.3271028 | 4.672897196 |
| IsolationForest | 0.001 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | | 32.71028037 | 67.28971963 | 36.44859813 | 63.55140187 |
| IsolationForest | 0.001 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | | 69.72477064 | 30.27522936 | 72.47706422 | 27.52293578 |
| IsolationForest | 0.001 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | | 70.19230769 | 29.80769231 | 68.26923077 | 31.73076923 |
| IsolationForest | 0.001 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | | 10.28037383 | 89.71962617 | 12.14953271 | 87.85046729 |
| IsolationForest | 0.002 | 130_HVACv4a_Boston+SmallOffice_Workday | | 12.18836565 | 87.81163435 | 11.91135734 | 88.08864266 |
| IsolationForest | 0.002 | 131_HVACv4a_Boston+SmallOffice_Workday_random | | 11.65048544 | 88.34951456 | 9.708737864 | 90.29126214 |
| IsolationForest | 0.002 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | | 100 | 0 | 100 | 0 |
| IsolationForest | 0.002 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | | 94.39252336 | 5.607476636 | 94.39252336 | 5.607476636 |
| IsolationForest | 0.002 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | | 32.71028037 | 67.28971963 | 35.51401869 | 64.48598131 |
| IsolationForest | 0.002 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | | 71.55963303 | 28.44036697 | 72.47706422 | 27.52293578 |
| IsolationForest | 0.002 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | | 71.15384615 | 28.84615385 | 70.19230769 | 29.80769231 |
| IsolationForest | 0.002 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | | 11.21495327 | 88.78504673 | 9.345794393 | 90.65420561 |
| IsolationForest | 0.003 | 130_HVACv4a_Boston+SmallOffice_Workday | | 13.29639889 | 86.70360111 | 11.35734072 | 88.64265928 |
| IsolationForest | 0.003 | 131_HVACv4a_Boston+SmallOffice_Workday_random | | 12.62135922 | 87.37864078 | 10.67961165 | 89.32038835 |
| IsolationForest | 0.003 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | | 100 | 0 | 100 | 0 |
| IsolationForest | 0.003 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | | 93.45794393 | 6.542056075 | 94.39252336 | 5.607476636 |
| IsolationForest | 0.003 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | | 35.51401869 | 64.48598131 | 34.57943925 | 65.42056075 |
| IsolationForest | 0.003 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | | 70.64220183 | 29.35779817 | 70.64220183 | 29.35779817 |
| IsolationForest | 0.003 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | | 73.07692308 | 26.92307692 | 67.30769231 | 32.69230769 |
| IsolationForest | 0.003 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | | 10.28037383 | 89.71962617 | 12.14953271 | 87.85046729 |
| IsolationForest | 0.004 | 130_HVACv4a_Boston+SmallOffice_Workday | | 13.29639889 | 86.70360111 | 11.63434903 | 88.36565097 |
| IsolationForest | 0.004 | 131_HVACv4a_Boston+SmallOffice_Workday_random | | 13.59223301 | 86.40776699 | 10.67961165 | 89.32038835 |
| IsolationForest | 0.004 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | | 100 | 0 | 100 | 0 |
| IsolationForest | 0.004 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | | 94.39252336 | 5.607476636 | 94.39252336 | 5.607476636 |
| IsolationForest | 0.004 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | | 33.64485981 | 66.35514019 | 35.51401869 | 64.48598131 |
| IsolationForest | 0.004 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | | 75.2293578 | 24.7706422 | 71.55963303 | 28.44036697 |
| IsolationForest | 0.004 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | | 72.11538462 | 27.88461538 | 67.30769231 | 32.69230769 |
| IsolationForest | 0.004 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | | 10.28037383 | 89.71962617 | 12.14953271 | 87.85046729 |
| IsolationForest | 0.005 | 130_HVACv4a_Boston+SmallOffice_Workday | | 11.63434903 | 88.36565097 | 11.91135734 | 88.08864266 |
| IsolationForest | 0.005 | 131_HVACv4a_Boston+SmallOffice_Workday_random | | 12.62135922 | 87.37864078 | 10.67961165 | 89.32038835 |
| IsolationForest | 0.005 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | | 100 | 0 | 100 | 0 |
| IsolationForest | 0.005 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | | 93.45794393 | 6.542056075 | 95.3271028 | 4.672897196 |
| IsolationForest | 0.005 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | | 33.64485981 | 66.35514019 | 36.44859813 | 63.55140187 |
| IsolationForest | 0.005 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | | 70.64220183 | 29.35779817 | 69.72477064 | 30.27522936 |
| IsolationForest | 0.005 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | | 72.11538462 | 27.88461538 | 68.26923077 | 31.73076923 |

| | | | | | | |
|---|---|---|---|---|---|---|
| IsolationForest | 0.005 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 11.21495327 | 88.78504673 | 13.08411215 | 86.91588785 |
| IsolationForest | 0.006 | 130_HVACv4a_Boston+SmallOffice_Workday | 11.91135734 | 88.08864266 | 12.74238227 | 87.25761773 |
| IsolationForest | 0.006 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 11.65048544 | 88.34951456 | 10.67961165 | 89.32038835 |
| IsolationForest | 0.006 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.006 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 93.45794393 | 6.542056075 | 96.26168224 | 3.738317757 |
| IsolationForest | 0.006 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 36.44859813 | 63.55140187 | 37.38317757 | 62.61682243 |
| IsolationForest | 0.006 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 67.88990826 | 32.11009174 | 73.39449541 | 26.60550459 |
| IsolationForest | 0.006 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 73.07692308 | 26.92307692 | 69.23076923 | 30.76923077 |
| IsolationForest | 0.006 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 10.28037383 | 89.71962617 | 13.08411215 | 86.91588785 |
| IsolationForest | 0.007 | 130_HVACv4a_Boston+SmallOffice_Workday | 11.08033241 | 88.91966759 | 11.08033241 | 88.91966759 |
| IsolationForest | 0.007 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 15.53398058 | 84.46601942 | 11.65048544 | 88.34951456 |
| IsolationForest | 0.007 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.007 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 92.52336449 | 7.476635514 | 94.39252336 | 5.607476636 |
| IsolationForest | 0.007 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 33.64485981 | 66.35514019 | 34.57943925 | 65.42056075 |
| IsolationForest | 0.007 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 73.39449541 | 26.60550459 | 73.39449541 | 26.60550459 |
| IsolationForest | 0.007 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 72.11538462 | 27.88461538 | 67.30769231 | 32.69230769 |
| IsolationForest | 0.007 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 12.14953271 | 87.85046729 | 13.08411215 | 86.91588785 |
| IsolationForest | 0.008 | 130_HVACv4a_Boston+SmallOffice_Workday | 12.74238227 | 87.25761773 | 13.01939058 | 86.98060942 |
| IsolationForest | 0.008 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 13.59223301 | 86.40776699 | 9.708737864 | 90.29126214 |
| IsolationForest | 0.008 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.008 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 93.45794393 | 6.542056075 | 93.45794393 | 6.542056075 |
| IsolationForest | 0.008 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 35.51401869 | 64.48598131 | 35.51401869 | 64.48598131 |
| IsolationForest | 0.008 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 72.47706422 | 27.52293578 | 72.47706422 | 27.52293578 |
| IsolationForest | 0.008 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 69.23076923 | 30.76923077 | 68.26923077 | 31.73076923 |
| IsolationForest | 0.008 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 11.21495327 | 88.78504673 | 13.08411215 | 86.91588785 |
| IsolationForest | 0.009 | 130_HVACv4a_Boston+SmallOffice_Workday | 14.12742382 | 85.87257618 | 11.35734072 | 88.64265928 |
| IsolationForest | 0.009 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 11.65048544 | 88.34951456 | 10.67961165 | 89.32038835 |
| IsolationForest | 0.009 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.009 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 94.39252336 | 5.607476636 | 94.39252336 | 5.607476636 |
| IsolationForest | 0.009 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 36.44859813 | 63.55140187 |
| IsolationForest | 0.009 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 75.2293578 | 24.7706422 | 74.31192661 | 25.68807339 |
| IsolationForest | 0.009 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 72.11538462 | 27.88461538 | 67.30769231 | 32.69230769 |
| IsolationForest | 0.009 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 11.21495327 | 88.78504673 | 12.14953271 | 87.85046729 |
| IsolationForest | 0.01 | 130_HVACv4a_Boston+SmallOffice_Workday | 12.18836565 | 87.81163435 | 12.18836565 | 87.81163435 |
| IsolationForest | 0.01 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 12.62135922 | 87.37864078 | 8.737864078 | 91.26213592 |
| IsolationForest | 0.01 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.01 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 92.52336449 | 7.476635514 | 95.32710280 | 4.672897196 |
| IsolationForest | 0.01 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 33.64485981 | 66.35514019 | 36.44859813 | 63.55140187 |
| IsolationForest | 0.01 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 68.80733945 | 31.19266055 | 73.39449541 | 26.60550459 |
| IsolationForest | 0.01 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 70.19230769 | 29.80769231 | 68.26923077 | 31.73076923 |
| IsolationForest | 0.01 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 12.14953271 | 87.85046729 | 11.21495327 | 88.78504673 |
| IsolationForest | 0.025 | 130_HVACv4a_Boston+SmallOffice_Workday | 12.74238227 | 87.25761773 | 12.46537396 | 87.53462604 |

| | | | | | | |
|---|---|---|---|---|---|---|
| IsolationForest | 0.025 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 13.59223301 | 86.40776699 | 9.708737864 | 90.29126214 |
| IsolationForest | 0.025 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.025 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 94.39252336 | 5.607476636 | 94.39252336 | 5.607476636 |
| IsolationForest | 0.025 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 31.77570093 | 68.22429907 | 38.31775701 | 61.68224299 |
| IsolationForest | 0.025 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 73.39449541 | 26.60550459 | 69.72477064 | 30.27522936 |
| IsolationForest | 0.025 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 70.19230769 | 29.80769231 | 69.23076923 | 30.76923077 |
| IsolationForest | 0.025 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 10.28037383 | 89.71962617 | 14.01869159 | 85.98130841 |
| IsolationForest | 0.05 | 130_HVACv4a_Boston+SmallOffice_Workday | 21.32963989 | 78.67036011 | 20.77562327 | 79.22437673 |
| IsolationForest | 0.05 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 21.3592233 | 78.6407767 | 18.44660194 | 81.55339806 |
| IsolationForest | 0.05 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.05 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 98.13084112 | 1.869158879 | 99.06542056 | 0.934579439 |
| IsolationForest | 0.05 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 40.18691589 | 59.81308411 | 42.05607477 | 57.94392523 |
| IsolationForest | 0.05 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 87.1559633 | 12.8440367 | 86.23853211 | 13.76146789 |
| IsolationForest | 0.05 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 80.76923077 | 19.23076923 | 75.96153846 | 24.03846154 |
| IsolationForest | 0.05 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 22.42990654 | 77.57009346 | 18.69158879 | 81.30841121 |
| IsolationForest | 0.075 | 130_HVACv4a_Boston+SmallOffice_Workday | 21.05263158 | 78.94736842 | 20.22160665 | 79.77839335 |
| IsolationForest | 0.075 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 19.41747573 | 80.58252427 | 17.47572816 | 82.52427184 |
| IsolationForest | 0.075 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.075 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 99.06542056 | 0.934579439 | 98.13084112 | 1.869158879 |
| IsolationForest | 0.075 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 42.99065421 | 57.00934579 | 41.12149533 | 58.87850467 |
| IsolationForest | 0.075 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 88.99082569 | 11.00917431 | 86.23853211 | 13.76146789 |
| IsolationForest | 0.075 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 79.80769231 | 20.19230769 | 79.80769231 | 20.19230769 |
| IsolationForest | 0.075 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 19.62616822 | 80.37383178 | 19.62616822 | 80.37383178 |
| IsolationForest | 0.1 | 130_HVACv4a_Boston+SmallOffice_Workday | 30.19390582 | 69.80609418 | 27.1468144 | 72.8531856 |
| IsolationForest | 0.1 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 28.15533981 | 71.84466019 | 25.24271845 | 74.75728155 |
| IsolationForest | 0.1 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.1 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 99.06542056 | 0.934579439 | 99.06542056 | 0.934579439 |
| IsolationForest | 0.1 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 49.53271028 | 50.46728972 | 47.6635514 | 52.3364486 |
| IsolationForest | 0.1 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 95.41284404 | 4.587155963 | 95.41284404 | 4.587155963 |
| IsolationForest | 0.1 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 87.5 | 12.5 | 84.61538462 | 15.38461538 |
| IsolationForest | 0.1 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 28.03738318 | 71.96261682 | 23.36448598 | 76.63551402 |
| IsolationForest | 0.125 | 130_HVACv4a_Boston+SmallOffice_Workday | 28.80886427 | 71.19113573 | 27.1468144 | 72.8531856 |
| IsolationForest | 0.125 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 28.15533981 | 71.84466019 | 26.21359223 | 73.78640777 |
| IsolationForest | 0.125 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.125 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 100 | 0 | 98.13084112 | 1.869158879 |
| IsolationForest | 0.125 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 47.6635514 | 52.3364486 | 44.85981308 | 55.14018692 |
| IsolationForest | 0.125 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 94.49541284 | 5.504587156 | 92.66055046 | 7.339449541 |
| IsolationForest | 0.125 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 86.53846154 | 13.46153846 | 85.57692308 | 14.42307692 |
| IsolationForest | 0.125 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 30.8411215 | 69.1588785 | 25.23364486 | 74.76635514 |
| IsolationForest | 0.15 | 130_HVACv4a_Boston+SmallOffice_Workday | 36.28808864 | 63.71191136 | 32.68698061 | 67.31301939 |
| IsolationForest | 0.15 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 32.03883495 | 67.96116505 | 33.00970874 | 66.99029126 |
| IsolationForest | 0.15 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| IsolationForest | 0.15 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.15 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 51.40186916 | 48.59813084 | 51.40186916 | 48.59813084 |
| IsolationForest | 0.15 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 97.24770642 | 2.752293578 | 97.24770642 | 2.752293578 |
| IsolationForest | 0.15 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 90.38461538 | 9.615384615 | 90.38461538 | 9.615384615 |
| IsolationForest | 0.15 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 33.64485981 | 66.35514019 | 33.64485981 | 66.35514019 |
| IsolationForest | 0.175 | 130_HVACv4a_Boston+SmallOffice_Workday | 36.56509695 | 63.43490305 | 32.68698061 | 67.31301939 |
| IsolationForest | 0.175 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 32.03883495 | 67.96116505 | 33.00970874 | 66.99029126 |
| IsolationForest | 0.175 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.175 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.175 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 52.3364486 | 47.6635514 | 52.3364486 | 47.6635514 |
| IsolationForest | 0.175 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 97.24770642 | 2.752293578 | 95.41284404 | 4.587155963 |
| IsolationForest | 0.175 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 89.42307692 | 10.57692308 | 91.34615385 | 8.653846154 |
| IsolationForest | 0.175 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 35.51401869 | 64.48598131 | 35.51401869 | 64.48598131 |
| IsolationForest | 0.2 | 130_HVACv4a_Boston+SmallOffice_Workday | 43.2132964 | 56.7867036 | 40.16620499 | 59.83379501 |
| IsolationForest | 0.2 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 37.86407767 | 62.13592233 | 34.95145631 | 65.04854369 |
| IsolationForest | 0.2 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.2 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 100 | 0 | 100 | 0 |
| IsolationForest | 0.2 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 55.14018692 | 44.85981308 | 55.14018692 | 44.85981308 |
| IsolationForest | 0.2 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 97.24770642 | 2.752293578 | 98.16513761 | 1.834862385 |
| IsolationForest | 0.2 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 94.23076923 | 5.769230769 | 94.23076923 | 5.769230769 |
| IsolationForest | 0.2 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 42.99065421 | 57.00934579 | 41.12149533 | 58.87850467 |
| LOF | 0.001 | 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.001 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.001 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.001 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.001 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.001 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.001 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.001 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.002 | 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.002 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.002 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.002 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.002 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.002 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.002 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.002 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.003 | 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.003 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.003 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.003 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.003 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |

| | | | | | |
|---|---|---|---|---|---|
| LOF | 0.003 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.003 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.003 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.004 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.004 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.004 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.004 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.004 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.004 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.004 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.004 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.005 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.005 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.005 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.005 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.005 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.005 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.005 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.005 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.006 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.006 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.006 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.006 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.006 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.006 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.006 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.006 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.007 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.007 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.007 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.007 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.007 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.007 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.007 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.007 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.008 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.008 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.008 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.008 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.008 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.008 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.008 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |

| | | | | | | |
|---|---|---|---|---|---|---|
| LOF | 0.008 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.009 | 130_HVACv4a_Boston+SmallOffice_Workday | 13.01939058 | 86.98060942 | 9.418282548 | 90.58171745 |
| LOF | 0.009 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.009 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.009 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.009 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.009 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.009 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.009 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.01 | 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.01 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.01 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.01 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.01 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.01 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.01 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.01 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.025 | 130_HVACv4a_Boston+SmallOffice_Workday | 13.29639889 | 86.70360111 | 9.418282548 | 90.58171745 |
| LOF | 0.025 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 20.38834951 | 79.61165049 | 14.5631068 | 85.4368932 |
| LOF | 0.025 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.025 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 90.65420561 | 9.345794393 | 86.91588785 | 13.08411215 |
| LOF | 0.025 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 34.57943925 | 65.42056075 | 35.51401869 | 64.48598131 |
| LOF | 0.025 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 93.57798165 | 6.422018349 | 92.66055046 | 7.339449541 |
| LOF | 0.025 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 75.96153846 | 24.03846154 | 72.11538462 | 27.88461538 |
| LOF | 0.025 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 14.95327103 | 85.04672897 | 14.01869159 | 85.98130841 |
| LOF | 0.05 | 130_HVACv4a_Boston+SmallOffice_Workday | 18.55955679 | 81.44044321 | 14.95844875 | 85.04155125 |
| LOF | 0.05 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 25.24271845 | 74.75728155 | 22.33009709 | 77.66990291 |
| LOF | 0.05 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.05 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 95.3271028 | 4.672897196 | 93.45794393 | 6.542056075 |
| LOF | 0.05 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 42.05607477 | 57.94392523 | 40.18691589 | 59.81308411 |
| LOF | 0.05 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 95.41284404 | 4.587155963 | 96.33027523 | 3.669724771 |
| LOF | 0.05 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 80.76923077 | 19.23076923 | 75.96153846 | 24.03846154 |
| LOF | 0.05 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 23.36448598 | 76.63551402 | 21.4953271 | 78.5046729 |
| LOF | 0.075 | 130_HVACv4a_Boston+SmallOffice_Workday | 18.55955679 | 81.44044321 | 14.95844875 | 85.04155125 |
| LOF | 0.075 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 25.24271845 | 74.75728155 | 22.33009709 | 77.66990291 |
| LOF | 0.075 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.075 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 95.3271028 | 4.672897196 | 93.45794393 | 6.542056075 |
| LOF | 0.075 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 42.05607477 | 57.94392523 | 40.18691589 | 59.81308411 |
| LOF | 0.075 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 95.41284404 | 4.587155963 | 96.33027523 | 3.669724771 |
| LOF | 0.075 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 80.76923077 | 19.23076923 | 75.96153846 | 24.03846154 |
| LOF | 0.075 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 23.36448598 | 76.63551402 | 21.4953271 | 78.5046729 |
| LOF | 0.1 | 130_HVACv4a_Boston+SmallOffice_Workday | 23.54570637 | 76.45429363 | 19.11357341 | 80.88642659 |

| | | | | | | |
|---|---|---|---|---|---|---|
| LOF | 0.1 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 31.06796117 | 68.93203883 | 26.21359223 | 73.78640777 |
| LOF | 0.1 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.1 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 96.26168224 | 3.738317757 | 94.39252336 | 5.607476636 |
| LOF | 0.1 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 49.53271028 | 50.46728972 | 44.85981308 | 55.14018692 |
| LOF | 0.1 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 97.24770642 | 2.752293578 | 96.33027523 | 3.669724771 |
| LOF | 0.1 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 82.69230769 | 17.30769231 | 79.80769231 | 20.19230769 |
| LOF | 0.1 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 28.97196262 | 71.02803738 | 26.1682243 | 73.8317757 |
| LOF | 0.125 | 130_HVACv4a_Boston+SmallOffice_Workday | 23.54570637 | 76.45429363 | 19.11357341 | 80.88642659 |
| LOF | 0.125 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 31.06796117 | 68.93203883 | 26.21359223 | 73.78640777 |
| LOF | 0.125 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.125 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 96.26168224 | 3.738317757 | 94.39252336 | 5.607476636 |
| LOF | 0.125 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 49.53271028 | 50.46728972 | 44.85981308 | 55.14018692 |
| LOF | 0.125 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 97.24770642 | 2.752293578 | 96.33027523 | 3.669724771 |
| LOF | 0.125 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 82.69230769 | 17.30769231 | 79.80769231 | 20.19230769 |
| LOF | 0.125 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 28.97196262 | 71.02803738 | 26.1682243 | 73.8317757 |
| LOF | 0.15 | 130_HVACv4a_Boston+SmallOffice_Workday | 29.6398892 | 70.3601108 | 24.65373961 | 75.34626039 |
| LOF | 0.15 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 35.9223301 | 64.0776699 | 29.12621359 | 70.87378641 |
| LOF | 0.15 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.15 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 98.13084112 | 1.869158879 | 98.13084112 | 1.869158879 |
| LOF | 0.15 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 55.14018692 | 44.85981308 | 52.3364486 | 47.6635514 |
| LOF | 0.15 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 100 | 0 | 100 | 0 |
| LOF | 0.15 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 83.65384615 | 16.34615385 | 80.76923077 | 19.23076923 |
| LOF | 0.15 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 30.8411215 | 69.1588785 | 33.64485981 | 66.35514019 |
| LOF | 0.175 | 130_HVACv4a_Boston+SmallOffice_Workday | 29.6398892 | 70.3601108 | 24.93074792 | 75.06925208 |
| LOF | 0.175 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 35.9223301 | 64.0776699 | 29.12621359 | 70.87378641 |
| LOF | 0.175 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.175 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 98.13084112 | 1.869158879 | 98.13084112 | 1.869158879 |
| LOF | 0.175 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 55.14018692 | 44.85981308 | 52.3364486 | 47.6635514 |
| LOF | 0.175 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 100 | 0 | 100 | 0 |
| LOF | 0.175 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 83.65384615 | 16.34615385 | 80.76923077 | 19.23076923 |
| LOF | 0.175 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 30.8411215 | 69.1588785 | 33.64485981 | 66.35514019 |
| LOF | 0.2 | 130_HVACv4a_Boston+SmallOffice_Workday | 33.51800554 | 66.48199446 | 28.80886427 | 71.19113573 |
| LOF | 0.2 | 131_HVACv4a_Boston+SmallOffice_Workday_random | 39.80582524 | 60.19417476 | 36.89320388 | 63.10679612 |
| LOF | 0.2 | 133_HVACv4a_Boston+SmallOffice_Workday_Fault2 | 100 | 0 | 100 | 0 |
| LOF | 0.2 | 134_HVACv4a_Boston+SmallOffice_Workday_Fault3 | 99.06542056 | 0.934579439 | 99.06542056 | 0.934579439 |
| LOF | 0.2 | 135_HVACv4a_Boston+SmallOffice_Workday_Fault5 | 59.81308411 | 40.18691589 | 57.00934579 | 42.99065421 |
| LOF | 0.2 | 136_HVACv4a_Boston+SmallOffice_Workday_Fault6 | 100 | 0 | 100 | 0 |
| LOF | 0.2 | 138_HVACv4a_Boston+SmallOffice_Workday_Fault3_orifice | 88.46153846 | 11.53846154 | 83.65384615 | 16.34615385 |
| LOF | 0.2 | 139_HVACv4a_Boston+SmallOffice_Workday_Fault6_2 | 35.51401869 | 64.48598131 | 35.51401869 | 64.48598131 |

| Distance measure | | | Euclidean | Euclidean | High-low | High-low |
|---|---|---|---|---|---|---|
| Rates | | | Fault detection rate | Normal operation rate | Fault detection rate | Normal operation rate |
| Anomaly algorithm | Contamination | Experiment | | | | |
| IsolationForest | 0.001 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 7.3 | 92.7 | 6.4 | 93.6 |
| IsolationForest | 0.001 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.0 | 14.0 | 86.9 | 13.1 |
| IsolationForest | 0.001 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.001 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 92.7 | 7.3 | 92.7 | 7.3 |
| IsolationForest | 0.001 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.001 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.002 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 6.4 | 93.6 | 8.2 | 91.8 |
| IsolationForest | 0.002 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.0 | 14.0 | 86.0 | 14.0 |
| IsolationForest | 0.002 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.002 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 93.6 | 6.4 | 91.8 | 8.2 |
| IsolationForest | 0.002 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.002 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.003 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 5.5 | 94.5 |
| IsolationForest | 0.003 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.0 | 14.0 | 86.0 | 14.0 |
| IsolationForest | 0.003 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.003 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 91.8 | 8.2 | 90.9 | 9.1 |
| IsolationForest | 0.003 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.003 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.004 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 7.3 | 92.7 | 8.2 | 91.8 |
| IsolationForest | 0.004 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.9 | 13.1 | 86.0 | 14.0 |
| IsolationForest | 0.004 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.004 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 92.7 | 7.3 | 93.6 | 6.4 |
| IsolationForest | 0.004 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.004 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.005 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 7.3 | 92.7 | 11.8 | 88.2 |
| IsolationForest | 0.005 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 87.9 | 12.1 | 86.0 | 14.0 |
| IsolationForest | 0.005 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.005 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 92.7 | 7.3 | 94.5 | 5.5 |
| IsolationForest | 0.005 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.005 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.006 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 7.3 | 92.7 | 6.4 | 93.6 |
| IsolationForest | 0.006 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.0 | 14.0 | 86.9 | 13.1 |
| IsolationForest | 0.006 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.006 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 90.0 | 10.0 | 92.7 | 7.3 |
| IsolationForest | 0.006 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.006 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.007 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 6.4 | 93.6 | 6.4 | 93.6 |
| IsolationForest | 0.007 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 87.9 | 12.1 | 86.0 | 14.0 |
| IsolationForest | 0.007 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| IsolationForest | 0.007 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 93.6 | 6.4 | 92.7 | 7.3 |
| IsolationForest | 0.007 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.007 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.008 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 8.2 | 91.8 |
| IsolationForest | 0.008 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.9 | 13.1 | 85.0 | 15.0 |
| IsolationForest | 0.008 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.008 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 92.7 | 7.3 | 95.5 | 4.5 |
| IsolationForest | 0.008 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.008 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.009 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 6.4 | 93.6 | 6.4 | 93.6 |
| IsolationForest | 0.009 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.9 | 13.1 | 86.0 | 14.0 |
| IsolationForest | 0.009 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.009 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 93.6 | 6.4 | 92.7 | 7.3 |
| IsolationForest | 0.009 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.009 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.01 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 5.5 | 94.5 |
| IsolationForest | 0.01 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.0 | 14.0 | 85.0 | 15.0 |
| IsolationForest | 0.01 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.01 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 93.6 | 6.4 | 92.7 | 7.3 |
| IsolationForest | 0.01 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.01 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.02 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 8.2 | 91.8 |
| IsolationForest | 0.02 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 86.9 | 13.1 | 85.0 | 15.0 |
| IsolationForest | 0.02 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.02 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 93.6 | 6.4 | 93.6 | 6.4 |
| IsolationForest | 0.02 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.02 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.04 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 15.5 | 84.5 | 13.6 | 86.4 |
| IsolationForest | 0.04 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 91.6 | 8.4 | 87.9 | 12.1 |
| IsolationForest | 0.04 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.04 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 96.4 | 3.6 | 97.3 | 2.7 |
| IsolationForest | 0.04 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.04 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.06 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 16.4 | 83.6 | 16.4 | 83.6 |
| IsolationForest | 0.06 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 89.7 | 10.3 | 88.8 | 11.2 |
| IsolationForest | 0.06 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.06 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 95.5 | 4.5 | 97.3 | 2.7 |
| IsolationForest | 0.06 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.06 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.08 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 25.5 | 74.5 | 20.0 | 80.0 |
| IsolationForest | 0.08 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 92.5 | 7.5 | 92.5 | 7.5 |
| IsolationForest | 0.08 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| IsolationForest | 0.08 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 97.3 | 2.7 | 99.1 | 0.9 |
| IsolationForest | 0.08 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.08 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.1 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 31.8 | 68.2 | 26.4 | 73.6 |
| IsolationForest | 0.1 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 92.5 | 7.5 | 93.5 | 6.5 |
| IsolationForest | 0.1 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.1 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.1 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.1 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.2 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 48.2 | 51.8 | 47.3 | 52.7 |
| IsolationForest | 0.2 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 93.5 | 6.5 | 93.5 | 6.5 |
| IsolationForest | 0.2 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.2 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.2 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.2 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.3 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 59.1 | 40.9 | 60.0 | 40.0 |
| IsolationForest | 0.3 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 97.2 | 2.8 | 96.3 | 3.7 |
| IsolationForest | 0.3 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.3 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.3 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.3 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.4 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 70.0 | 30.0 | 71.8 | 28.2 |
| IsolationForest | 0.4 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 98.1 | 1.9 | 98.1 | 1.9 |
| IsolationForest | 0.4 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.4 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.4 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.4 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.5 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 82.7 | 17.3 | 83.6 | 16.4 |
| IsolationForest | 0.5 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.5 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.5 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.5 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| IsolationForest | 0.5 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.001 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.001 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.001 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.001 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.001 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.001 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.002 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.002 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.002 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| LOF | 0.002 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.002 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.002 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.003 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.003 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.003 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.003 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.003 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.003 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.004 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.004 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.004 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.004 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.004 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.004 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.005 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.005 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.005 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.005 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.005 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.005 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.006 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.006 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.006 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.006 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.006 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.006 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.007 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.007 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.007 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.007 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.007 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.007 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.008 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.008 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.008 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.008 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.008 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.008 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.009 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.009 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.009 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| LOF | 0.009 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.009 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.009 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.01 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.01 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.01 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.01 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.01 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.01 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.02 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 9.1 | 90.9 | 10.0 | 90.0 |
| LOF | 0.02 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 62.6 | 37.4 | 60.7 | 39.3 |
| LOF | 0.02 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 99.1 | 0.9 | 100.0 | 0.0 |
| LOF | 0.02 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 62.7 | 37.3 | 64.5 | 35.5 |
| LOF | 0.02 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.02 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.04 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 14.5 | 85.5 | 18.2 | 81.8 |
| LOF | 0.04 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 76.6 | 23.4 | 78.5 | 21.5 |
| LOF | 0.04 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.04 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 79.1 | 20.9 | 80.0 | 20.0 |
| LOF | 0.04 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.04 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.06 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 14.5 | 85.5 | 18.2 | 81.8 |
| LOF | 0.06 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 76.6 | 23.4 | 78.5 | 21.5 |
| LOF | 0.06 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.06 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 79.1 | 20.9 | 80.0 | 20.0 |
| LOF | 0.06 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.06 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.08 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 27.3 | 72.7 | 22.7 | 77.3 |
| LOF | 0.08 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 84.1 | 15.9 | 87.9 | 12.1 |
| LOF | 0.08 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.08 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 90.9 | 9.1 | 88.2 | 11.8 |
| LOF | 0.08 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.08 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.1 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 34.5 | 65.5 | 30.0 | 70.0 |
| LOF | 0.1 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 91.6 | 8.4 | 91.6 | 8.4 |
| LOF | 0.1 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.1 | 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 95.5 | 4.5 | 96.4 | 3.6 |
| LOF | 0.1 | 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.1 | 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.2 | 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 59.1 | 40.9 | 59.1 | 40.9 |
| LOF | 0.2 | 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 97.2 | 2.8 | 97.2 | 2.8 |
| LOF | 0.2 | 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |

| | | | | | |
|---|---|---|---|---|---|
| LOF | 0.2 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 97.3 | 2.7 | 98.2 | 1.8 |
| LOF | 0.2 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.2 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.3 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 75.5 | 24.5 | 74.5 | 25.5 |
| LOF | 0.3 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 99.1 | 0.9 | 99.1 | 0.9 |
| LOF | 0.3 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.3 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 99.1 | 0.9 | 98.2 | 1.8 |
| LOF | 0.3 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.3 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.4 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 88.2 | 11.8 | 87.3 | 12.7 |
| LOF | 0.4 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.4 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.4 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 99.1 | 0.9 |
| LOF | 0.4 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.4 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.5 201_HVACv6a_3room_Boston+SmallOffice_Workday_random | 93.6 | 6.4 | 94.5 | 5.5 |
| LOF | 0.5 202_HVACv6a_3R_Boston+SmallOffice_Fault1 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.5 203_HVACv6a_3R_Boston+SmallOffice_Fault2 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.5 204_HVACv6a_3R_Boston+SmallOffice_Fault3 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.5 205_HVACv6a_3R_Boston+SmallOffice_Fault4 | 100.0 | 0.0 | 100.0 | 0.0 |
| LOF | 0.5 206_HVACv6a_3R_Boston+SmallOffice_Fault5 | 100.0 | 0.0 | 100.0 | 0.0 |