Raphael Benjamin Rauter

Comparison of Outlier Detection Algorithms in NILM Datasets

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

Studium: Informationstechnik

Schwerpunkt: Smart Grids

Universität Klagenfurt

Begutachter:

Univ.–Ass. Dipl.-Ing. Christoph Klemenjak
Institut für Vernetzte und Eingebettete Systeme

Klagenfurt, Februar 2021

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich

- die eingereichte wissenschaftliche Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe,
- die während des Arbeitsvorganges von dritter Seite erfahrene Unterstützung, einschließlich signifikanter Betreuungshinweise, vollständig offengelegt habe,
- die Inhalte, die ich aus Werken Dritter oder eigenen Werken wortwörtlich oder sinngemäß übernommen habe, in geeigneter Form gekennzeichnet und den Ursprung der Information durch möglichst exakte Quellenangaben (z.B. in Fußnoten) ersichtlich gemacht habe,
- die eingereichte wissenschaftliche Arbeit bisher weder im Inland noch im Ausland einer Prüfungsbehörde vorgelegt habe und
- bei der Weitergabe jedes Exemplars (z.B. in gebundener, gedruckter oder digitaler Form) der wissenschaftlichen Arbeit sicherstelle, dass diese mit der eingereichten digitalen Version übereinstimmt.

Mir ist bekannt, dass die digitale Version der eingereichten wissenschaftlichen Arbeit zur Plagiatskontrolle herangezogen wird.

Ich bin mir bewusst, dass eine tatsachenwidrige Erklärung rechtliche Folgen haben wird.

Vorname Nachname e. h.

Ort, Monat Jahr

Feldkirchen, Februar 2021

Abstract

The current state of the art makes it possible to store ever-larger amounts of data. It is becoming increasingly important to recognize whether the data obtained adds value or is erroneous. Outlier detection is already being used in a wide variety of use cases, such as detecting insider trading or insurance fraud.
In this work, a comparison of two widely used algorithms, Isolation Forest and Local Outlier Factor, on NILM datasets is made. The term NILM or Non-Intrusive Load Monitoring refers to the periodic acquisition of changes in currents and voltages at household and application levels. The REFIT dataset, which includes power values from 20 households in the UK, is used as the dataset for comparison. For this dataset, the power values of the entire household as well as of a wide variety of household appliances, such as computers, dishwashers, televisions, etc. were recorded at periodic intervals over a period of two years. The comparison of the algorithms is further refined by filtering the values of the data set. The rolling median and the hampel filter were chosen as filters. For both, the neighboring points per data point are used to detect and filter out deviating points. This results in the following test constellations: Isolation Forest and Local Outlier Factor with no, rolling median or hampel filtering.
This thesis is divided into a total of five chapters. The first chapter gives a general introduction to outlier detection. In the second chapter, all terms used and important definitions are examined in more detail. Furthermore, the basics and implementations of both algorithms and filter types are described. The experimental results are then presented in the third chapter. Here, the measured values of 6 household appliances and all 20 households in a half-year period are used as the data basis. For each of the household appliances, the performance of the Isolation Forest is compared in relation to the results of the Local outlier factor. At the household level, the results are considered when the data of the two algorithms are prefiltered. In the fourth chapter, a discussion is made in order to put the obtained results into relation. Finally, a conclusion of this work is made in the fifth chapter.

Zusammenfassung

Der heutige Stand der Technik ermöglicht es, immer größere Datenmengen zu speichern. Es wird immer wichtiger zu erkennen, ob erfasste Daten einen Mehrwert bieten oder fehlerhaft sind. Outlier detection wird bereits bei einer Vielzahl von Anwendungsfällen eingesetzt, z. B. zur Erkennung von Insiderhandel oder Versicherungsbetrug.
In dieser Arbeit wird ein Vergleich von zwei weit verbreiteten Algorithmen, Isolation Forest und Local Outlier Factor, auf NILM-Datensätzen durchgeführt. Der Begriff NILM beziehungsweise Non-Intrusive Load Monitoring bezieht sich auf die periodische Erfassung von Strom- und Spannungsänderungen auf Haushalts- und Applikationsebene. Als Vergleichsdatensatz wird der REFIT-Datensatz verwendet, der Leistungswerte von 20 Haushalten in Großbritannien enthält. Für diesen Datensatz wurden in periodischen Abständen über einen Zeitraum von zwei Jahren die Leistungswerte des gesamten Haushalts sowie einer Vielzahl von Haushaltsgeräten wie Computer, Geschirrspüler, Fernseher etc. aufgezeichnet. Der Vergleich der Algorithmen wird durch die Filterung der Daten des Datensatzes weiter verfeinert. Als Filter wurden der Rolling-Median und der Hampel-Filter gewählt. Bei beiden werden die benachbarten Punkte für jeden Datenpunkt verwendet, um Abweichungen zu erkennen und herauszufiltern. Daraus ergeben sich die folgenden Testkonstellationen: Isolation Forest und Local Outlier Factor mit keinem Filter, Rolling Median oder Hampel-Filter.
Die vorliegende Arbeit gliedert sich in insgesamt vier Kapitel. Das erste Kapitel gibt eine allgemeine Einführung in die Thematik. Im zweiten Kapitel werden alle verwendeten Begriffe und wichtige Definitionen näher erklärt. Weiters werden die Grundlagen und Implementierungen der beiden Algorithmen und der Filtertypen beschrieben. Die experimentellen Ergebnisse werden dann im dritten Kapitel vorgestellt. Als Datenbasis werden hier die Messwerte von 4 Haushaltsgeräten und allen 20 Haushalten des REFIT Datensatzes eines Halbjahres verwendet. Für jedes der Haushaltsgeräte wird die Performance des Isolation Forest im Verhältnis zu den Ergebnissen des Local Outlier Factor Algorithmus verglichen. Auf Haushaltsebene werden die Ergebnisse bei Anwenden eines Filters auf die Datensätze betrachtet. Im vierten Kapitel werden verschiedene Paper vorgestellt, die einen Vergleich von Algorithmen anstellen um die erhaltenen Ergebnisse in Relation zu setzen. Schlussendlich wird im letzten Kapitel ein Fazit dieser Arbeit gezogen.

Acknowledgments

I would like to thank my supervisor Univ. Ass. Dipl.-Ing. Christoph Klemenjak for the support during my work as well as the Alpen-Adria Universität Klagenfurt in particular the Institute for Networked and Embedded Systems for the provision of a workspace and the access to a university server, for running my simulations.

List of Acronyms

W	**W**att
IF	**I**solation **F**orest
ML	**M**achine **L**earning
AAU	**A**lpen-**A**dria **U**niversität
NES	**N**etworked and **E**mbedded **S**ystems
LOF	**L**ocal **O**utlier **F**actor
NILM	**N**on-**I**ntrusive **L**oad **M**onitoring
NILMTK	**N**on-**I**ntrusive **L**oad **M**onitoring **T**oolkit

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 Outlier Detection

The current state of technology makes it possible to store ever-larger amounts of data. It is becoming more and more important to recognize whether the acquired data brings added value or is faulty.

The detection of outliers is a technical field that is becoming increasingly important. Due to the fact that data acquisition is becoming easier and easier, ever-larger data sets are being processed. However, it is usually necessary to determine whether data is useful for further decisions. For example, credit card companies need to determine whether a transaction has been carried out by the credit card owner [24].
Another use case lies in Non-Intrusive Load Monitoring. Here the power consumption of various household appliances is measured and stored in a defined cycle. Further use cases for outlier detection are Network Intrusion Detection, Insurance Claim Fraud Detection, Insider Trading Detection, or Industrial Damage Detection. The outlier detection methods in all these applications aim to detect points that deviate from the expected behavior. These values can either be errors in the data sets and therefore be corrected, or they can be points of interest that can provide useful information and therefore be investigated further like in Credit Card Fraud Detection where the credit card company would be interested in why exactly the data is considered an outlier.

1.1.2 Non-Intrusive Load Monitoring

Due to increasing digitalization, energy demand is rising not only in industry but also in households, and the topic of energy efficiency is therefore coming more and more into focus. An important fact that underlines this is[8, p. 1]:

"According to the statistics from the U.S. Department of Energy and the European Union Energy Commission, global energy consumption is likely to increase in the next decade, with residential and commercial buildings raising their aggregate figure to 20%–40% of the total yearly consumption."

In order to cope with the increasing energy demand, it is becoming more important that on the one hand the customers are sensitized to deal with their energy consumption and to reduce it if possible, and on the other hand for the energy suppliers to be able to better evaluate the required amount of energy in order to be able to react better to short-term changes in consumption. One approach to this is the use of NILM.

Non-intrusive load monitoring tries to determine which household appliances are in a house and at which times these appliances are switched on. For this purpose, the power values of the entire household are recorded at periodic intervals. By applying various algorithms, it is tried to determine the devices. Since in Europe as well as in the USA an ever-larger number of so-called Smart meters is installed, ever-larger data sets become available, by which this topic becomes even more important. The following figure shows an example of power recordings where the power of a house was measured over a period of about one hour. It can be clearly seen when a device is switched on or off. Based on the device specifications and performance characteristics, it is then possible to identify which devices lead to which change.

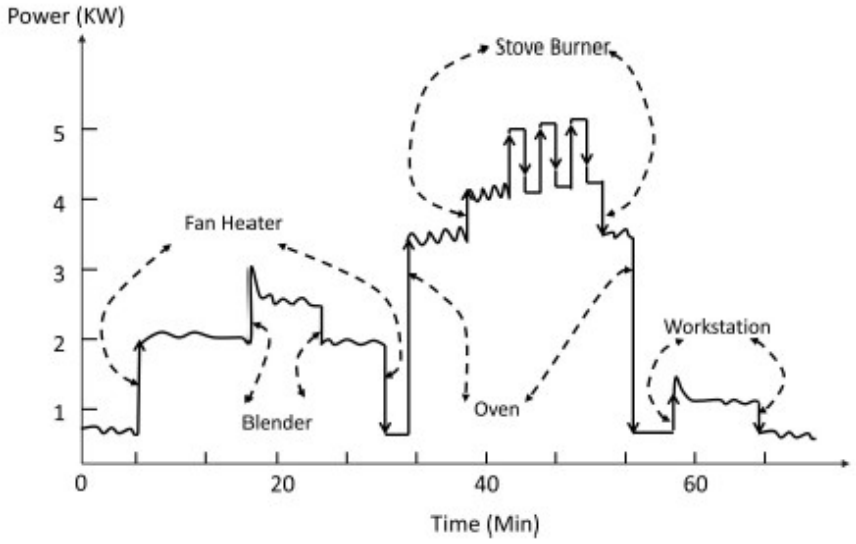

Figure 1.1: NILM power data example [28, Fig 1b]

Appliances are typically divided into three categories based on their power con-

sumption characteristics [14, p. 2-3]:

- **On/Off appliances**
  This category includes, for example, appliances such as toasters that only have the two states on and off. These appliances consume a precisely defined amount of power when switched on.
- **Multi-state appliances**
  These appliances have several operating states with different power inputs, such as washing machines (heating water, pumping water out, ...).
- **Infinite-state appliances**
  This category includes units that have an infinite number of states and therefore no fixed power consumption.

The following graph shows how the power consumption of these different appliances differs over time.

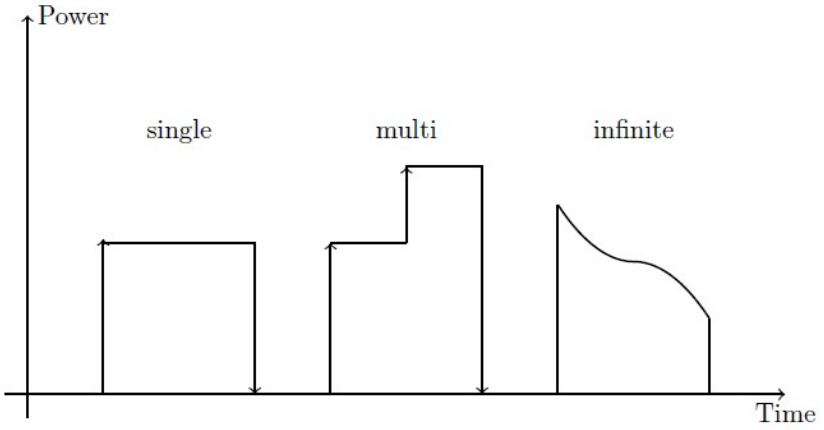

Figure 1.2: Power consumption characteristics [14, Fig 2]

As with all measurements, erroneous data can be recorded. These errors in the data records can also be referred to as data noise. Data noise can be defined as unexpected or unaccounted anomalies that may occur in a data set being analyzed by an algorithm. [17, p. 2] The possible causes of such errors are numerous, for example missing values, faulty measurement due to broken sensors, or malfunctions in the measurement recording. Therefore, this work aims to apply different outlier detection algorithms on these datasets and evaluate the potential benefits.

1.2 Structure of the Thesis

This thesis is divided into 5 chapters. The first chapter will give an introduction to the topics of Outlier detection and NILM. In the second chapter then the individual

aspects of the methods and/or algorithms are more near described. In the third chapter, the results obtained by simulation will be discussed. In order to be able to compare the results better, an overview of already accomplished comparisons of Outlier detection algorithms is made in the fourth chapter. In the last chapter, a conclusion of this thesis is drawn and possible further in-depth studies are discussed.

Chapter 2

Background

2.1 Novelty or Outlier?

In many use cases today, it is important to know whether new data sets obtained have the same distribution as existing data or whether they show an anomaly. In this context, a distinction must be made between the terms outlier and novelty.

The term novelty is used for outliers if training data that is not contaminated without any outliers is used to determine whether new data sets show the same distribution or can be seen as outliers.
The term outlier describes data points in training data that are very different from the others and therefore must be considered an anomaly.

2.2 Outlier Detection

One of the best definitions to describe the concept of outliers was provided by [11, p. 2]:

"An observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism."

Outliers can have two different meanings. They can either be unwanted, so they are basically erroneous values in datasets, which are then removed from the dataset in further processing or are corrected. Another significance of outliers is, that the interest lies in the outliers themselves, and therefore further investigations of a point is done.

2.2.1 Classification of Outlier Detection Methods

In this subsection, it is described in more detail in which categories the outlier detection methods can be classified. They can be distinguished by different aspects like described in the following Figure2.1.

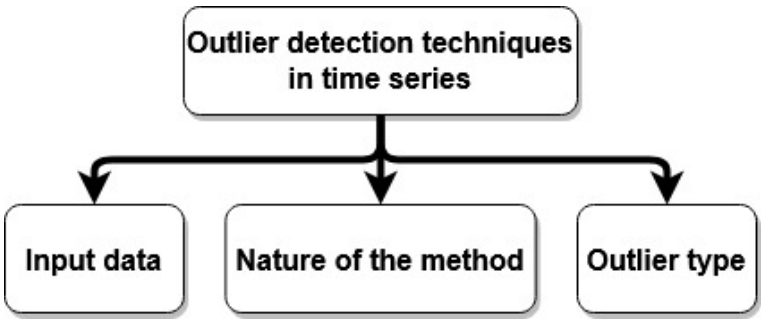

Figure 2.1: A classification of outlier detection techniques

2.2.1.1 Input Data

The input data can be divided into two types, univariate time series, and multivariate time series. A univariate time series can be specified by the following equations: [5, p. 3]

A univariate time series

$$X = \{x_t\}_{t \in T} \tag{2.1}$$

is an ordered set of real-valued observations, where each observation is recorded at a specific time

$$t \in T \subseteq \mathbb{Z}^+. \tag{2.2}$$

The difference between multivariate and univariate time series lies in the fact that a multivariate time series is defined as an ordered set of k-dimensional vectors, each of which is recorded at a specific time t that consists of k real-valued observations. Since this paper deals only with univariate time series, the mathematical background of multivariate time series is omitted, but more information can be found in [5, p. 3].

2.2.1.2 Nature of the Method

The second type of classification of detection methods is the nature of the method. The methods are used either for detection in univariate or multivariate time series. It is important to note that univariate methods can be used for univariate and multivariate input data since in the case of multivariate data a single time-dependent variable can be considered. However, multivariate detection methods can only be used for multivariate time series.

2.2.1.3 Outlier Type

The methods for outlier detection can be further distinguished according to the type of outlier. There are

- Point outliers
- Subsequence outliers
- Outlier time series

An important point in the context of outlier types is that outliers depend on the context. This means that if an entire time series is used as a data basis, the detected outliers are global. If only a part or section (time window) of the time series is used, the outliers are called local. These can also be called outliers within their neighborhood. It must be noted that all global outliers are also local, but not all local outliers are also global outliers.[5, p. 4] Therefore, for further analysis it must always be considered whether the entire data set was used for the outlier detection or only a section of it.

The first category of outliers is point outliers. A point outlier is a point (time/value pair) in the time series that behaves unusually in a specific time instant when compared either to the other values in the time series (global outlier) or to its neighboring points (local outlier). Point outliers can be univariate or multivariate depending on whether they affect one or more time-dependent variables.

In Figure 2.2, a point outlier in a univariate time series can be seen. It is obvious that point O1 behaves differently than the other points in the time series.

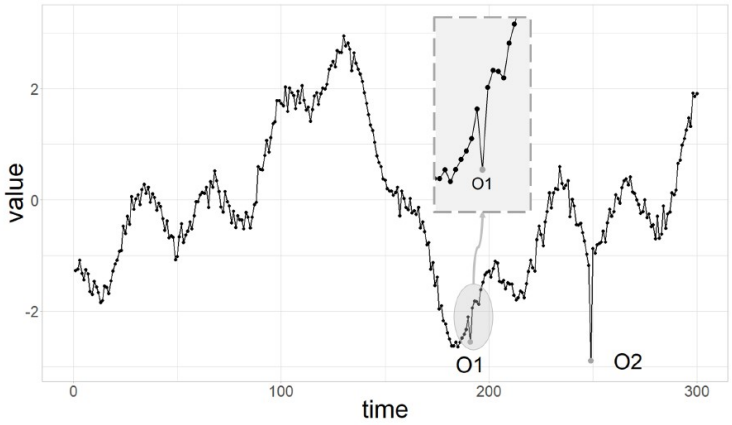

Figure 2.2: Point outlier in univariate time series [5, Fig. 3a]

A subsequence outlier is present if a series of points in the time series show abnormal behavior. Furthermore, it is not necessary that the points alone would be considered as outliers. With this type of outlier, it is again possible that they

are local or global. Subsequence outliers can also be univariate or multivariate depending on the input data.

Figure 2.3 shows an example for a subsequence outlier in a univariate time series. The series of points O1 and O2 differ significantly from the course of the series.

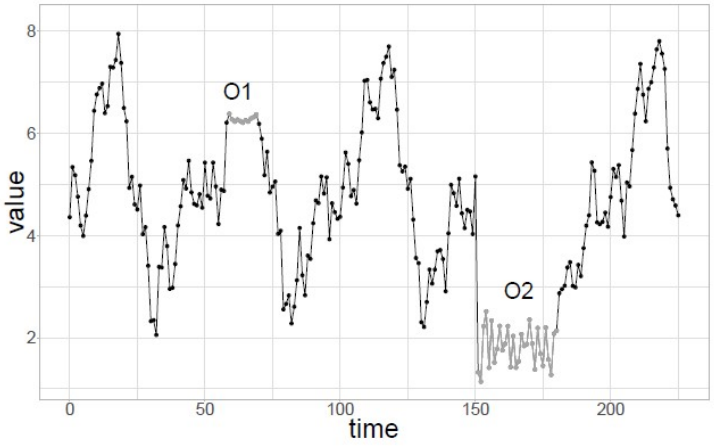

Figure 2.3: Subsequence outlier in univariate time series [5, Fig. 4a]

The third type of outliers are entire time series. If in multivariate time series, one variable shows a very different behavior to the other variables of the time series, the entire series can be considered an outlier. An example for a time series outlier in the case of multivariate input data can be seen in the following Figure 2.4, where the fourth variable shows a significantly different behavior to the other three variables.

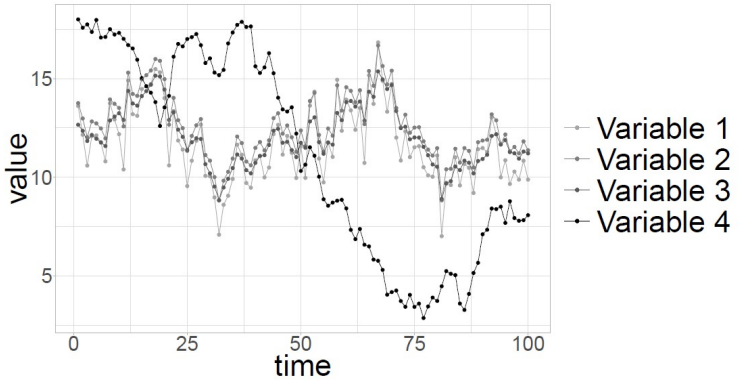

Figure 2.4: Outlier time series in multivariate input data [5, Fig. 5]

2.2.2 Point Outliers

In this paper, the focus lies on the detection of point outliers in univariate time series since it is the most common and therefore a great number of algorithms are

available. As outlier detection is a very large scientific field, the scope of this paper would otherwise be exceeded. The techniques for point outlier detection can be divided into the following three aspects:

- Model-based
- Density-based
- Histogramming

A widely used definition for the concept of point outliers is [5, p. 6], that a point can be called a point outlier if it deviates significantly from its expected value. Applying this concept to time series, a point in the time series at time t can be considered an outlier if the distance from the point's value to its expected one is greater than a predefined threshold τ. This can be represented by the following equation [5, p. 6]:

$$|x_t - \hat{x_t}| > \tau \tag{2.3}$$

x_t ... observed value, $\hat{x_t}$ ... expected value., τ ... threshold

Outlier detection methods based on this concept are called model-based methods and are the most common for point outliers. A further breakdown of these methods can be made based on the determination of the expected value $\hat{x_t}$. If only past data is used to determine the expected value, the methods are called prediction model-based. However, if past, current, and future data is used, they are called estimation model-based.

While model-based detection methods only consider single points and their deviation from the expected, density-based methods deal with neighboring points for single points. Points are considered to be outliers if they have less than a defined number τ of adjacent points. This concept can be represented mathematically with the following equation (4) [5, p. 9]. A point is an outlier if it has less than a predefined numberτ of neighboring points. A point is called a neighboring point if it lies within a distance R of the point under consideration. This method uses one section of the data at a time and then looks at the points before and after the point being analyzed.

$$x_t \text{ is an outlier} \iff |(x \in X | d(x, x_t) \leq R)| < \tau \tag{2.4}$$

d ... Euclidean distance, x_t ... analyzed data point at time t, X ... set of data points, $R \in \mathbb{R}+$

According to this equation, a point can be called an outlier if $\tau_p + \tau_s < \tau$. τ_p and τ_s are the number of preceding and succeeding neighbors. [5, p. 9]

This equation is illustrated very well with the following Figure2.5. Here a so-called sliding window was used, in which several points are summarized. This can be applied to streaming time series, for example, to determine whether new values are normal or deviate from the previous ones. Due to the sliding window, historical data is included in the consideration.
In this example a window from $t = 3$ to $t = 13$ is used. A point is selected for which it is to be checked whether this is an outlier and then the distance R is applied to this point. If there are several points in this range, this point is normal, otherwise, it is an outlier.
For points S4 and R10 it can be seen that they have several neighboring points whereas point O13 has no neighboring points and would therefore be recognized as an outlier.

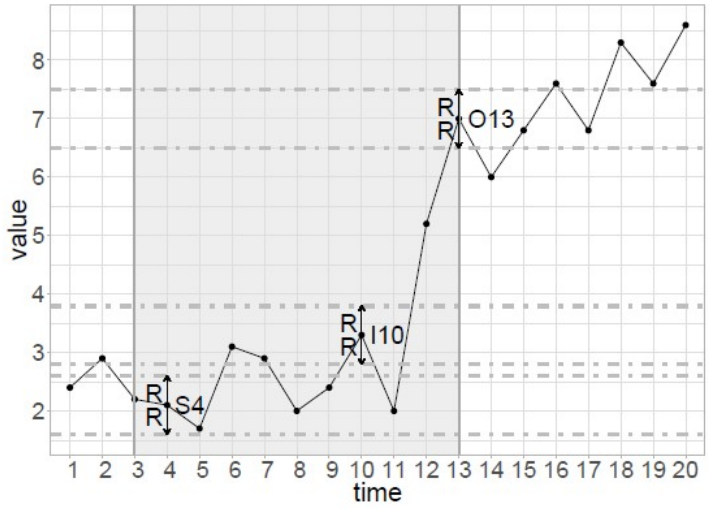

Figure 2.5: Density-based outliers within a sliding window [5, Fig. 9a]

An important point here is that this depends very much on the window, if the window would be moved further, the point O13 would have several neighbors and would not be an outlier.

The last aspect of point outlier detection is histogramming [10]. Here the data is divided into so-called buckets. These buckets contain points of a certain value range. An attempt is then made to find out which points would have to be removed from the data set to obtain a histogram with lower error, these points would then be considered as outliers.

These techniques have been considered for univariate time series, but it is important to note that all these presented methods are used for multivariate time series as well. Here, a single time series/variable of multivariate data would be considered.

2.3 Outlier Detection Algorithms

2.3.1 Isolation Forest

In this section, an algorithm that is based on the concepts introduced in the previous section is presented. After introducing the most important basics of outlier detection in the previous section, an algorithm based on these concepts is presented in this section. The Isolation Forest algorithm is an unsupervised learning technique that was first introduced in [16]. This algorithm can be related to the model-based techniques.

The difference between unsupervised and supervised learning can be outlined as follows [9, p. 1]: Unsupervised learning examines how systems can learn to represent input patterns in a way that reflects the static structure of the collected set of input patterns. In supervised learning, there are no explicit target outputs associated with each input.

The Isolation Forest Algorithm creates multiple isolation trees and then determines if a point is an outlier by calculating the path lengths. The isolation trees are based on the concept of decision trees.

2.3.2 Decision Trees

A decision tree is used to represent decisions in a tree-like model. The structure of a decision tree can be seen in the following figure 2.6.

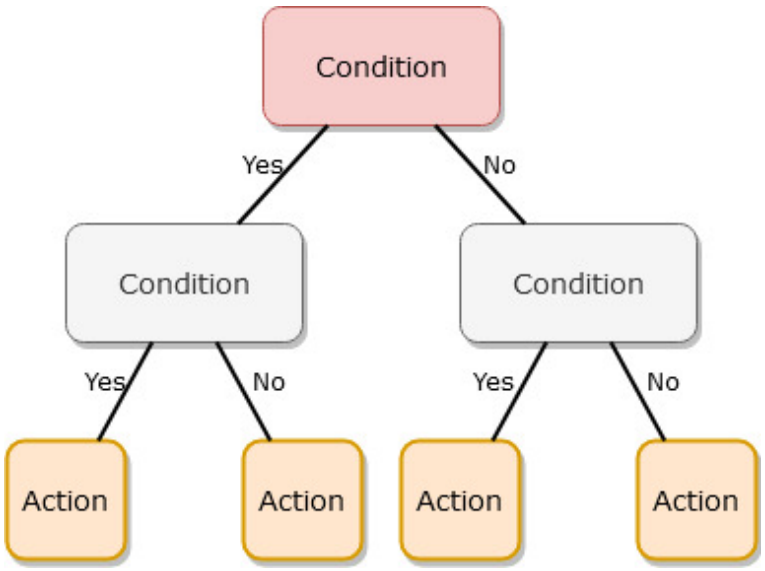

Figure 2.6: Example of a decision tree

A decision tree is started with the topmost node, also called the root node, which is shown in red in this example. Starting from this node there are two possible options, in this case, Yes or No. If further decisions are made for a node, the nodes following it are called child nodes or internal nodes. If no further refinements can be made, the tree is finished. These nodes where no more decisions are made or which have no more child nodes are called external nodes or terminating nodes.

2.3.3 Creation of Isolation Trees

The first step of the algorithm is to create the isolation trees using the training data. The training data is either a part of the total available data set or historical data can also be used for this purpose. The isolation trees are created by starting at the root node. In the case of isolation trees, this would be the entire training data. First, a feature of the data set is randomly selected and then a value between the maximum and minimum of the selected feature is randomly chosen. Based on this value the data set is then divided into two parts. In this case, these would be the child nodes of the root node. This process is repeated until either all instances of the data set are isolated or a certain height of the tree is reached. The phrase instance refers to a point respectively value in the data. This process of partitioning is repeated several times until a desired number of trees is obtained. The height of the tree and the total number of trees is determined by the user and according to [16], the best performance is achieved with around 100 trees.

Anomalies are often more susceptible to isolation if partitioning is done randomly. This can be illustrated with the two Figures 2.7, 2.8 below. Comparing the two figures it becomes clear that much more partitioning is needed to isolate

the normal point x_i than for the outlier x_o. This is the concept on which outlier detection in the Isolation Forest algorithm is based.

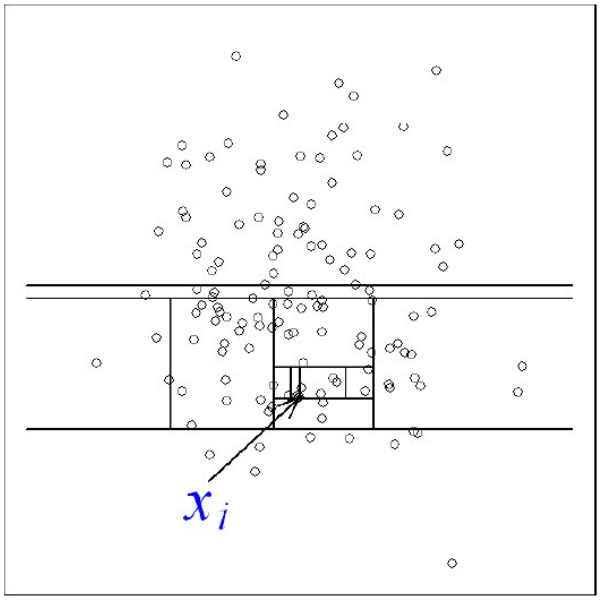

Figure 2.7: Isolating normal point x_i [16, Fig. 1a]

2.3.4 Application of the Isolation Trees

After the creation of the isolation trees with the given training data, these trees are then applied to the data to be tested. Each test instance (data point) is applied to all trees to determine an anomaly score that indicates whether the value is an outlier or not. For each test point, it is started at the root node of a tree and the tree is traversed until a terminating node is reached. Each node in the tree contains a certain amount of the dataset. The number of edges from the root node to the terminating node is then determined. If a point is very easy to isolate, there are not many branches in the tree, so the number of edges is very small. However, if a point is difficult to isolate, a large number of nodes are needed to reach the terminating node. This process is repeated for each tree of the forest.

This process is illustrated by the following example. The data set for the example consists of the points contained in the graph in Figure 2.9.

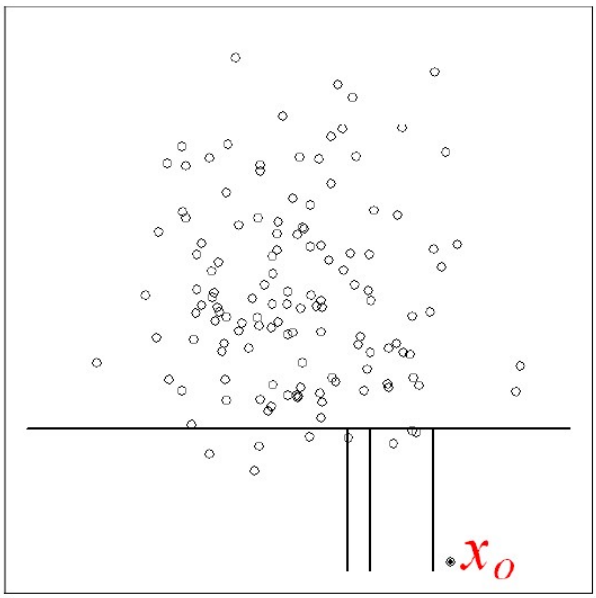

Figure 2.8: Isolating anomalous point x_o [16, Fig. 1b]

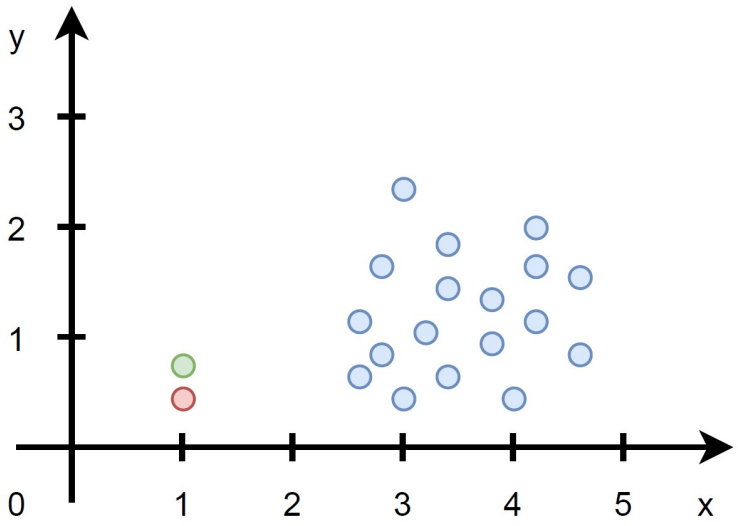

Figure 2.9: Example dataset to illustrate the application of the Isolation Forest

The first step is the creation of the trees which is shown in Figure 2.10. This is done several times to obtain the so-called forest. For one tree, the creation is shown in the following graph. All points are used as a starting point, then a feature is randomly selected, in this case, the x or y-axis, next a value between the minimum and maximum value of the points on the selected axis is then randomly selected. The partitioning of the data is then done on the respective axis at the value. This is repeated until either all points are isolated or a certain number of partitionings

have taken place. In this example, an outlier is shown in red. It can be seen that only a few partitionings are necessary for the isolation. For normal values, which are shown in blue, several steps are necessary.

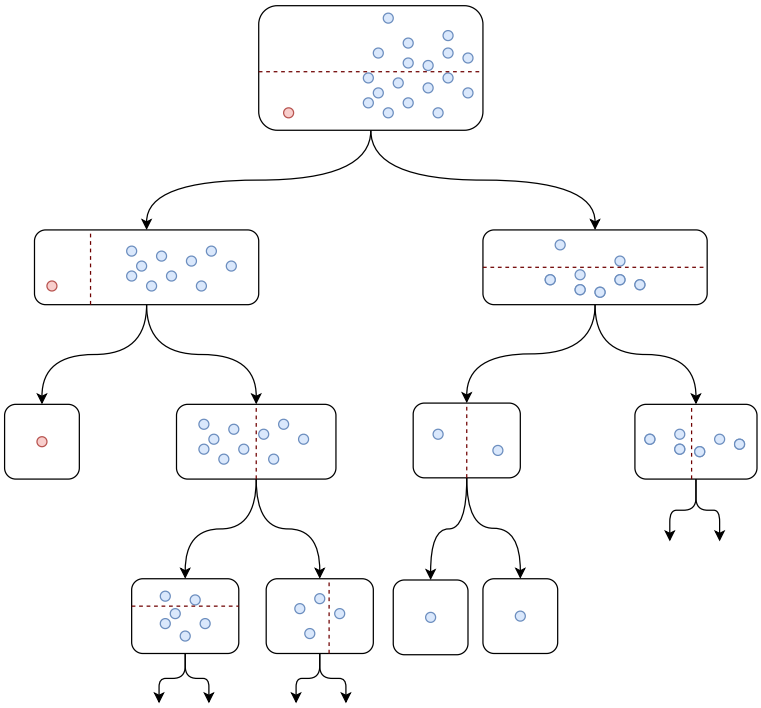

Figure 2.10: Creation of the trees for the example dataset shown in 2.9

After the trees are created, they can be applied to the test data. This is shown in Figure 2.11. The test point is displayed in green in the figure with the example dataset. The root node is considered as the starting point, then the value of the test point is used to determine which path to follow, i.e. which node is considered next. Since the test point is very close to the outlier, the path length is therefore the number of edges is small, therefore this point would be considered as an outlier. A normal point that would be inside the blue point cloud would have a much larger path length.

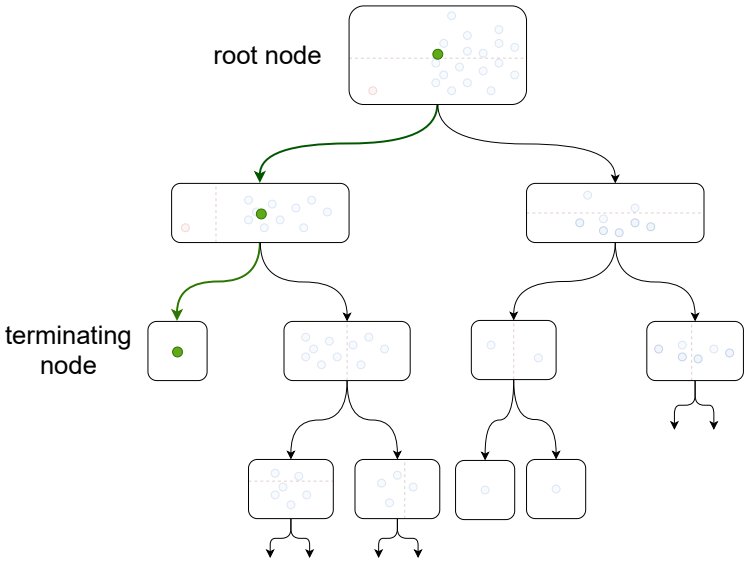

Figure 2.11: Application of the created trees for the green dot from 2.9 - outlier

As with other outlier detection methods, an anomaly score is required for decision making. In the case of the Isolation Forest, the anomaly score s for the instance x is defined as [16]:

$$s(x,h) = 2^{\frac{-E(h(x))}{c(n)}} \tag{2.5}$$

where h(x) is determined by running an instance x through an isolation tree and then determining the number of edges from the root node to the terminating node of the instance x, the so-called path length. The number of external nodes is given by n and E(h(x)) is the average path length determined by applying the instance to all isolation trees. c(n) is the average path length of the unsuccessful search in a Binary Search Tree.
So, it can be simplified that the anomaly score is determined by the path length from the root node to the terminating node in which the instance is placed and since it is an ensemble, the average of all the path lengths is taken.

The resulting anomaly score is in the range of $0 < s \leq 1$ and the following statements can be made based on it:

- s close to 1 $\Rightarrow$ anomaly
- s much smaller than 0.5 $\Rightarrow$ normal point
- s close to 0.5 $\Rightarrow$ if all scores are around 0.5, the entire sample does not seem to have clearly distinct anomalies

2.3.5 Local Outlier Factor

The local outlier factor (LOF) algorithm is a density-based algorithm and is also unsupervised like the isolation forest algorithm. The idea behind this algorithm is to compare for each point in the dataset the density of the point with the densities of nearby points. If a point has a lower density than the nearby points, it can be considered an outlier.
The calculation of a LOF factor for a specific point p can be defined as [6]:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts(p)}} \frac{lrd_{MinPs}(o)}{lrd_{MinPs}(p)}}{|N_{MinPts}(p)|} \tag{2.6}$$

The hereby calculated LOF factor indicates the degree to which this point can be regarded as an outlier. This factor is the average of the ratio of the local reachability density of the point p and its close neighbors. The local reachability density of a point can be calculated as follows [6]:

$$lrd_{MinPts}(p) = 1/(\frac{\sum_{o \in N_{MinPts(p)}} reach - dist_{MinPts}(p,o)}{|N_{MinPts}(p)|}) \tag{2.7}$$

The reachability density needed for the local reachability density is defined as:

Let k be a real number. The reachability distance of object p with respect to object o is defined as [6]

$$reach - dist_k(p,o) = max\{k - distance(o), d(p,o)\} \tag{2.8}$$

The calculation of the local outlier factor depends mainly on the choice of the parameter MinPts. This parameter defines the number of neighboring points that are included in the calculation of the outlier factor of a point. Since the result of the calculation of the LOF does not show monotonous behavior with different MinPts values, it is advised according to [6] choose a value range for this parameter. After determining the upper and lower limits of the MinPts parameter, the LOF for each value should be calculated for the data to be examined and then ranked according to the results for the LOF.

2.4 k-fold Cross-Validation

Cross-validation is a technique used in machine learning to determine the performance of a classifier. Classifiers often do not work well with new data with which the model was not trained. Therefore cross-validation is used to ensure that the

trained model delivers the expected performance on unknown data. [1]

The so-called k-fold cross validation has only one parameter k, which indicates in how many parts the existing data sets are divided to. If a specific value is defined for the parameter k, such as k = 10, it can also be called 10-fold cross-validation. According to [1], the choice of the parameter k is very dependent on the data. For the further experiments in this work, 5-fold cross-validation is used. [2]
The procedure for applying the cross-validation is as follows:

1. Dividing the dataset into k parts.
2. For each part:
   (a) Using one part as the test dataset.
   (b) Fitting the machine learning model on the rest of the data.
   (c) Evaluating the model with the test dataset.
3. Calculating the performance of the model with the sample of evaluation scores.

2.5 Filtering Techniques

The process of data filtering is used to increase the quality of data because data can get values that occurred due to errors in the measurement chain. For the comparison of the algorithms in the next chapter, the filtering methods described in more detail below will be used to determine if there is any added value in correcting values that do not match the other values in the slightest.

2.5.1 Median Filter

The median filter is often used in the removal of noise from an image or signal and belongs to the non-linear filter techniques. This filter passes the data value by value and replaces each with the median of the neighboring points, which can also be called a window. This filtering technique can be used for both one-dimensional and multidimensional data sets and has as a parameter only the number of neighboring points that are included for the median calculation.

The following graph shows the application of a median and a moving average filter to data contaminated with noise, where it can be seen that the median filter offers better performance than the moving average, which uses the average of the neighboring points instead of the median. In the graph, (a) shows the original

[1] https://scikit-learn.org/stable/modules/cross_validation.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

data with n = 2000, (b) 25% of data points corrupted, some random noise added, (c) Moving average filter applied, window size k = 25 and (d) Median filter applied, window size k = 25. In all figures, the shaded area represents the original data.

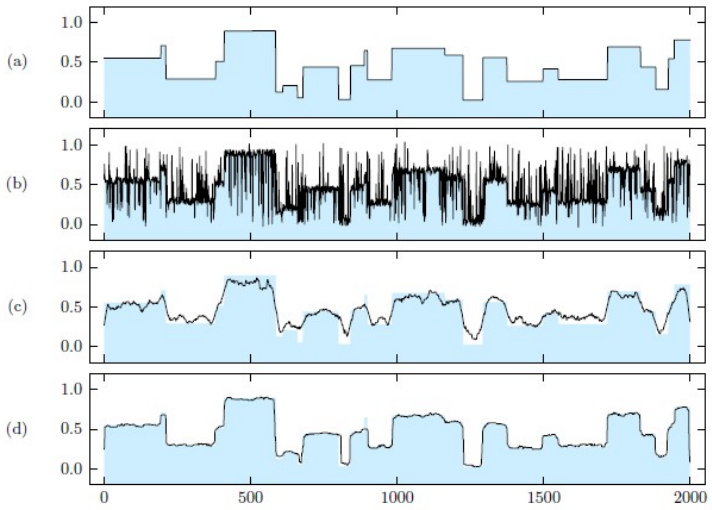

Figure 2.12: Application of a median filter on contaminated data [25, Fig 1]

2.5.2 Hampel Filter

The hampel filter uses the same approach as the median filter since a so-called sliding window is applied to the dataset. With the Hampel filter, the median and standard deviation of the points included by the window is calculated. This standard deviation is then always recalculated when the sliding window is applied. If a value should deviate from a certain number of standard deviations, it is replaced with the median. The hampel filter has as parameters the size of the window and the number of standard deviations from which a value is considered erroneous and replaced. In 2.13 it can be seen that filtering data using a Hampel filter is very suitable for removing large outliers. In this example, a hampel filter is used to remove outliers from EMG signal. EMG or electromyography signals show the electrical activities of skeletal muscles of the human face. The Python implementation for the experiments was inspired by the article[3].

[3] `https://towardsdatascience.com/outlier-detection-with-hampel-filter-85ddf523c73d`

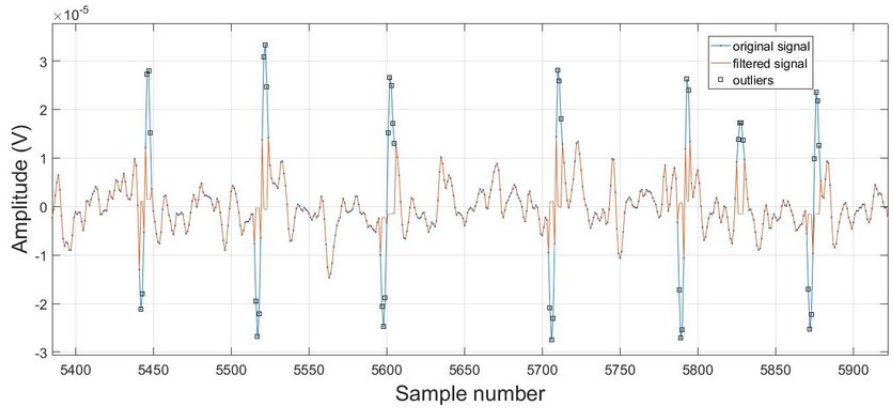

Figure 2.13: Application of a hampel filter as outlier detection method on electromyography signals [4, Fig 6]

Chapter 3

Outliers in Datasets

In this chapter, a comparison of the previously described algorithms, Isolation Forest and Local Outlier Factor, is performed. For this comparison, a widely used data set is used.

3.1 Data set

The following 4 data sets which are open to access were created within the REFIT project [19].

1. REFIT Electrical Load Measurements - includes cleaned electrical consumption data in Watts for 20 households at aggregate and appliance level, timestamped and sampled at 8-second intervals.

2. UK Homeowner Survey: Perceptions of Smart Home Benefits and Risks - dataset of a national survey of 1,054 homes conducted to measure perceptions of smart homes

3. REFIT Smart Home Interviews - contains qualitative data collected using semi-structured interviews and a structured survey at four-time points during the REFIT field trial of smart home technologies which involved 20 households

4. REFIT Smart Home dataset - includes building survey data, gas consumption, internal air temperature, local climate data, and other sensor measurements for the 20 REFIT homes.

The "REFIT Electrical Load Measurements" was used to compare the algorithms. Since data is not available for all appliances for each home in this dataset and a large number of measurements are available at the appliance level, the following appliances were selected for the calculations: dishwasher, television, computer and, washing machine

Furthermore, the electrical consumption per house is considered which is referred to as aggregate in the further course. Since the outlier detection is very resource-intensive, not the entire data sets were used, but only the period of one half-year, namely from 09.10.2013 to 08.04.2014.

3.2 Comparison Isolation Forest and Local Outlier Factor

The following section shows the results obtained for each appliance. Each table contains the absolute number as well as the relative number of outliers detected by both algorithms with the respective filter technology. Furthermore, they contain the relative number of outliers, which have a power value above 25 W.
The plots show the histograms of the detected outliers for IF and LOF. In the histograms for IF & LOF, the results for the Isolation Forest and Local Outlier Factor algorithm were examined for values jointly considered as outliers and the included table shows the total number of similarly detected outliers and to which number the results match for both algorithms.

The results were obtained by using the following setup for all experiments:

- 5-fold cross-validation
- No filter, rolling median and hampel filter
- Time period of the REFIT data: 2013-10-09 - 2014-04-08

The widely used NILMTK module [2] was used to process the NILM data sets. For the algorithms presented in the previous section, the module Scikit-learn [20] was used. This module provides the implementations of the Isolation Forest and Local Outlier Factor according to their corresponding papers.

Table 3.1 is intended to show an overview of the average data per house for all four applications, both for no filtering, rolling median and hampel filtering.

	Absolute		Relative		Rel > 25W	
	IF	LOF	IF	LOF	IF	LOF
no filter						
Television	272885	51	0.26	0.00	0.19	0.00
Computer	200142	54	0.19	0.00	0.12	0.00
Dish Washer	103870	124	0.08	0.00	0.06	0.00
Washing Machine	57809	424	0.05	0.00	0.03	0.00
rolling median						
Television	282721	44	0.26	0.00	0.19	0.00
Computer	211037	77	0.19	0.00	0.12	0.00
Dish Washer	86164	251	0.07	0.00	0.05	0.00
Washing Machine	57814	577	0.05	0.00	0.04	0.00
hampel						
Television	248295	47	0.25	0.00	0.19	0.00
Computer	205449	48	0.19	0.00	0.12	0.00
Dish Washer	85895	143	0.07	0.00	0.05	0.00
Washing Machine	59317	486	0.05	0.00	0.03	0.00

Table 3.1: Outliers for all applications and filter techniques

3.2.1 Dishwasher

	Absolute		Relative		Rel > 25W	
	IF	LOF	IF	LOF	IF	LOF
House 1	19316	166	0.01	0.00	0.01	0.00
House 2	63135	149	0.09	0.00	0.08	0.00
House 3	159188	146	0.09	0.00	0.04	0.00
House 5	220356	63	0.12	0.00	0.10	0.00
House 6	22270	149	0.02	0.00	0.02	0.00
House 7	144528	108	0.10	0.00	0.07	0.00
House 9	45635	163	0.05	0.00	0.05	0.00
House 10	685244	144	0.45	0.00	0.34	0.00
House 13	8364	75	0.02	0.00	0.02	0.00
House 14	3464	40	0.00	0.00	0.00	0.00
House 15	36193	137	0.05	0.00	0.02	0.00
House 17	11024	146	0.03	0.00	0.02	0.00
House 19	10282	148	0.04	0.00	0.01	0.00
House 20	25177	107	0.06	0.00	0.05	0.00

Table 3.2: Outliers for the dishwashers - no filtering.

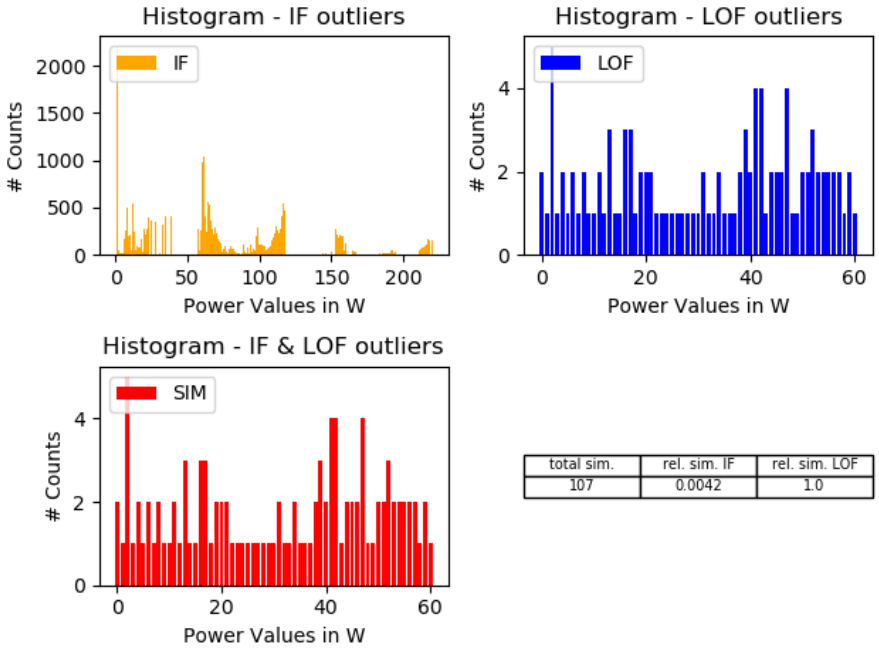

Figure 3.1: Histogram dishwasher outlier - House 20 - no filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	19225	249	0.01	0.00	0.01	0.00
House 2	63143	212	0.09	0.00	0.08	0.00
House 3	159167	391	0.09	0.00	0.04	0.00
House 5	213143	360	0.11	0.00	0.10	0.00
House 6	22287	188	0.02	0.00	0.02	0.00
House 7	131151	327	0.09	0.00	0.07	0.00
House 9	45741	286	0.05	0.00	0.05	0.00
House 10	459398	286	0.30	0.00	0.21	0.00
House 13	7979	210	0.02	0.00	0.02	0.00
House 14	3441	62	0.00	0.00	0.00	0.00
House 15	36178	217	0.05	0.00	0.02	0.00
House 17	11043	246	0.03	0.00	0.02	0.00
House 19	10283	220	0.04	0.00	0.01	0.00
House 20	24121	261	0.06	0.00	0.05	0.00

Table 3.3: Outliers for the dishwashers - rolling median filtering.

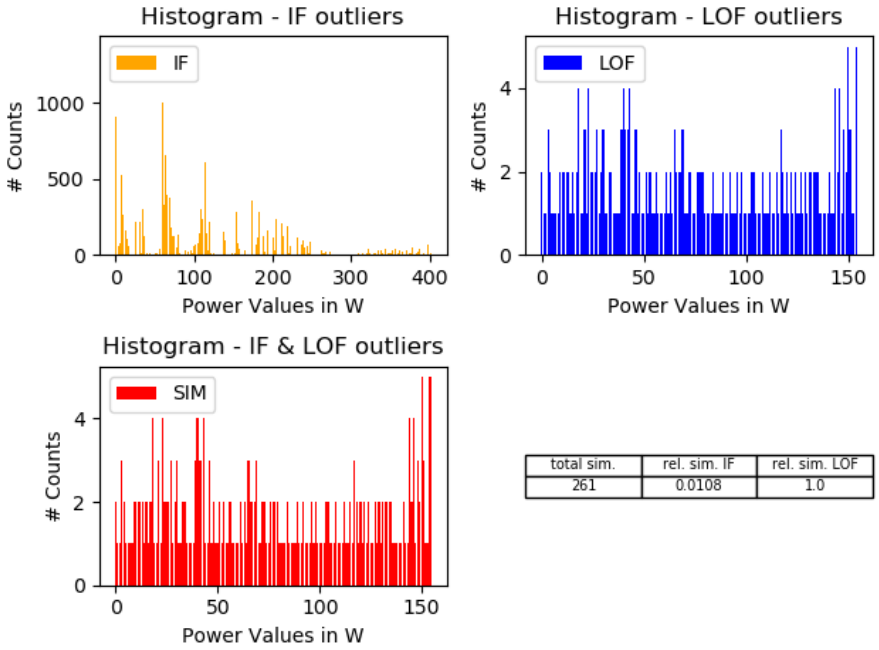

Figure 3.2: Histogram dishwasher outlier - House 20 - rolling median filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	19217	177	0.01	0.00	0.01	0.00
House 2	63023	206	0.09	0.00	0.08	0.00
House 3	159163	277	0.09	0.00	0.04	0.00
House 5	211996	210	0.11	0.00	0.10	0.00
House 6	22201	1.0	0.02	0.00	0.02	0.00
House 7	129968	259	0.09	0.00	0.07	0.00
House 9	45598	183	0.05	0.00	0.05	0.00
House 10	459028	224	0.30	0.00	0.21	0.00
House 13	7904	96	0.02	0.00	0.02	0.00
House 14	3367	30	0.00	0.00	0.00	0.00
House 15	36167	1.0	0.05	0.00	0.02	0.00
House 17	11016	152	0.03	0.00	0.02	0.00
House 19	10279	1.0	0.04	0.00	0.01	0.00
House 20	23608	187	0.05	0.00	0.05	0.00

Table 3.4: Outliers for the dishwashers - hampel filtering.

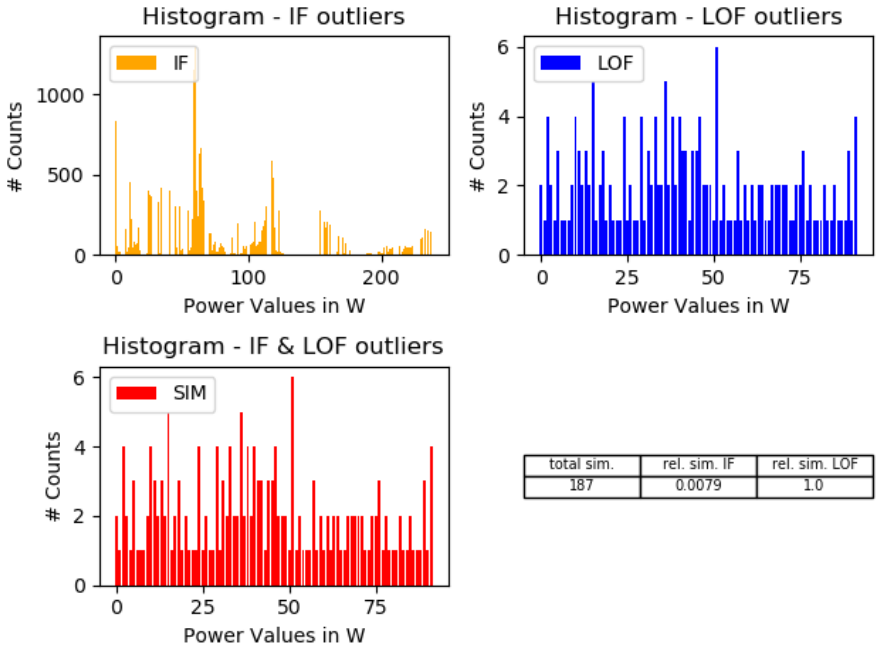

Figure 3.3: Histogram dishwasher outlier - House 20 - hampel filter

3.2.2 Television

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	277689	22	0.16	0.00	0.14	0.00
House 2	78769	30	0.11	0.00	0.10	0.00
House 3	545025	41	0.30	0.00	0.30	0.00
House 4	830730	69	0.48	0.00	0.44	0.00
House 5	285959	51	0.15	0.00	0.15	0.00
House 6	432866	77	0.33	0.00	0.25	0.00
House 7	332403	32	0.23	0.00	0.11	0.00
House 8	367926	77	0.24	0.00	0.15	0.00
House 9	291192	55	0.31	0.00	0.12	0.00
House 10	519585	46	0.34	0.00	0.33	0.00
House 13	14449	55	0.03	0.00	0.03	0.00
House 14	158943	37	0.17	0.00	0.17	0.00
House 15	191777	112	0.27	0.00	0.27	0.00
House 16	98028	28	0.22	0.00	0.08	0.00
House 17	174275	72	0.40	0.00	0.29	0.00
House 18	125669	37	0.30	0.00	0.18	0.00
House 19	92321	52	0.36	0.00	0.13	0.00
House 20	94324	28	0.22	0.00	0.18	0.00

Table 3.5: Outliers for the televisions - no filtering.

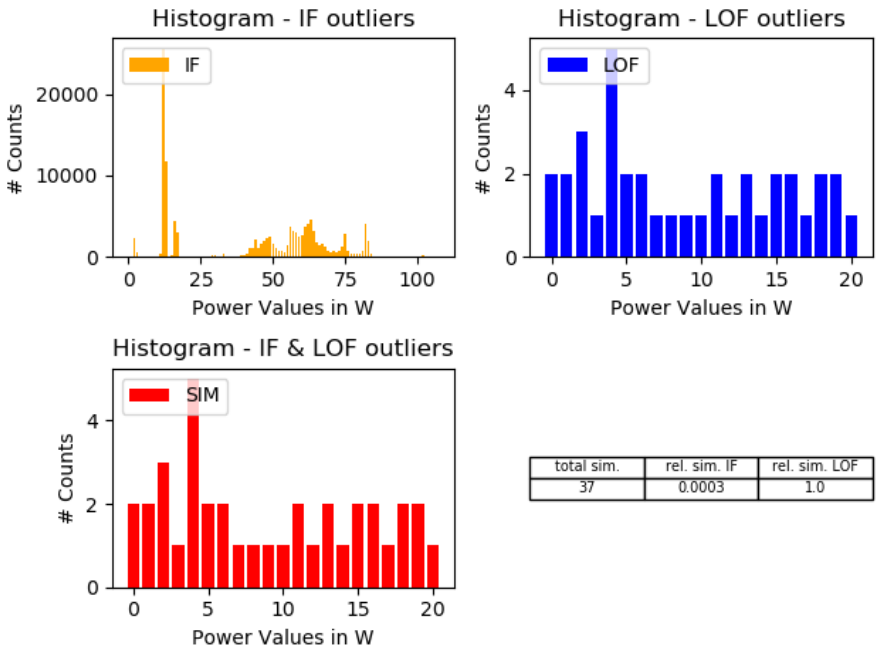

Figure 3.4: Histogram television outlier - House 18 - no filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	278082	24	0.16	0.00	0.14	0.00
House 2	78975	40	0.11	0.00	0.10	0.00
House 3	587724	80	0.32	0.00	0.32	0.00
House 4	819890	55	0.47	0.00	0.44	0.00
House 5	290212	41	0.16	0.00	0.15	0.00
House 6	495797	60	0.38	0.00	0.25	0.00
House 7	380362	13	0.26	0.00	0.14	0.00
House 8	345038	1.0	0.23	0.00	0.15	0.00
House 9	279387	73	0.30	0.00	0.12	0.00
House 10	615428	37	0.41	0.00	0.36	0.00
House 13	14448	93	0.03	0.00	0.03	0.00
House 14	149641	44	0.16	0.00	0.16	0.00
House 15	164802	1.0	0.23	0.00	0.23	0.00
House 16	124595	36	0.28	0.00	0.07	0.00
House 17	173333	58	0.40	0.00	0.29	0.00
House 18	122843	45	0.29	0.00	0.18	0.00
House 19	73993	67	0.29	0.00	0.13	0.00
House 20	94426	27	0.22	0.00	0.18	0.00

Table 3.6: Outliers for the televisions - rolling median filtering.

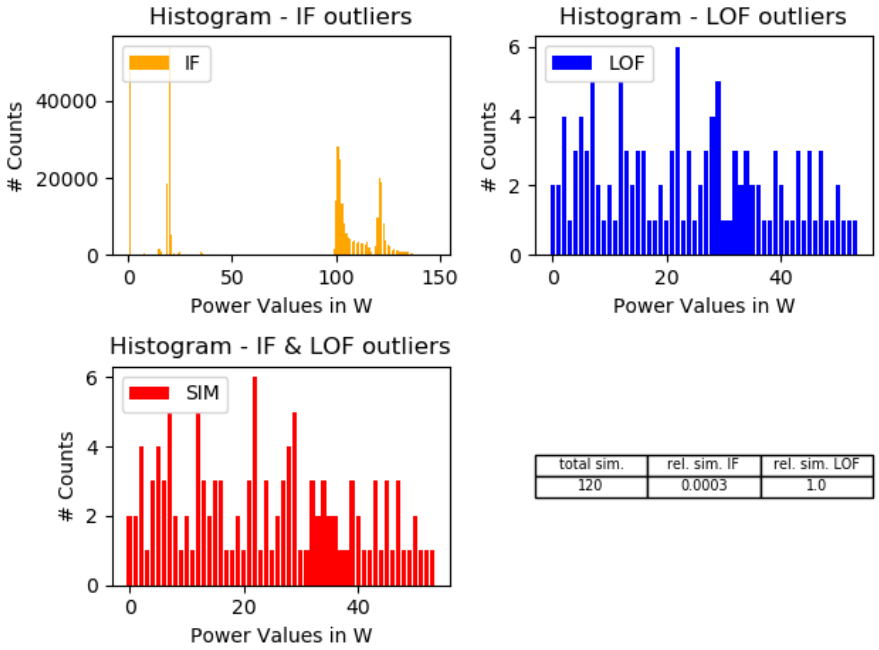

Figure 3.5: Histogram television outlier - House 18 - rolling median filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	277520	16	0.16	0.00	0.14	0.00
House 2	78737	36	0.11	0.00	0.10	0.00
House 3	437864	40	0.24	0.00	0.24	0.00
House 4	839268	53	0.48	0.00	0.45	0.00
House 5	281608	36	0.15	0.00	0.15	0.00
House 6	463687	49	0.35	0.00	0.25	0.00
House 7	3681.0	18	0.25	0.00	0.13	0.00
House 8	343866	56	0.23	0.00	0.15	0.00
House 9	268151	49	0.29	0.00	0.12	0.00
House 10	531948	37	0.35	0.00	0.31	0.00
House 13	14448	67	0.03	0.00	0.03	0.00
House 14	149655	35	0.16	0.00	0.16	0.00
House 15	191983	129	0.27	0.00	0.27	0.00
House 16	125792	27	0.28	0.00	0.07	0.00
House 17	171264	61	0.39	0.00	0.29	0.00
House 18	122691	54	0.29	0.00	0.18	0.00
House 19	72824	55	0.29	0.00	0.13	0.00
House 20	94322	25	0.22	0.00	0.18	0.00

Table 3.7: Outliers for the televisions - hampel filtering.

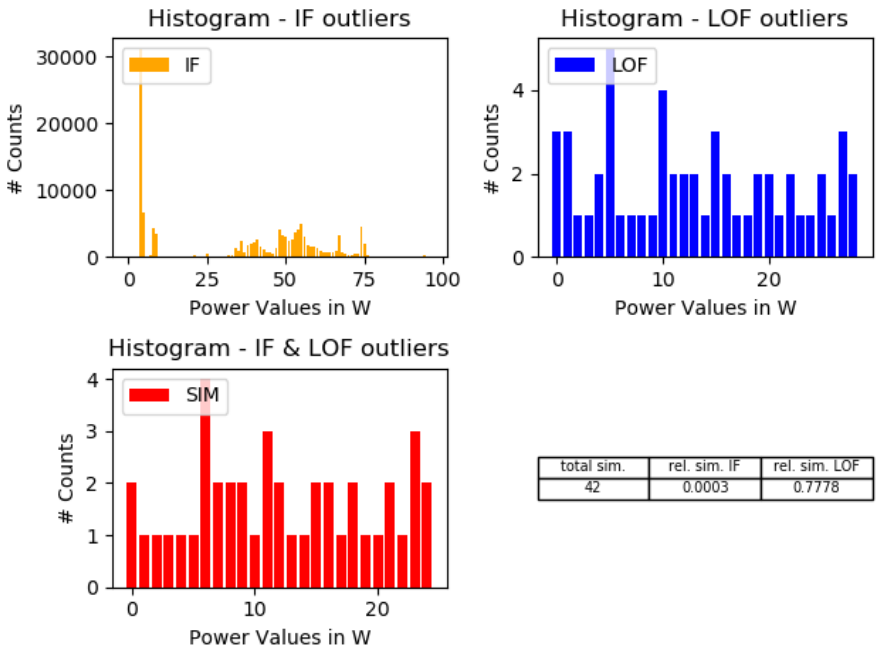

Figure 3.6: Histogram television outlier - House 18 - hampel filter

3.2.3 Computer

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	116512	22	0.07	0.00	0.03	0.00
House 5	551213	50	0.30	0.00	0.16	0.00
House 6	577632	91	0.44	0.00	0.28	0.00
House 8	262870	82	0.17	0.00	0.04	0.00
House 12	4351	112	0.01	0.00	0.01	0.00
House 14	75228	27	0.08	0.00	0.01	0.00
House 15	196877	44	0.28	0.00	0.28	0.00
House 16	116432	33	0.26	0.00	0.21	0.00
House 17	100142	52	0.23	0.00	0.14	0.00
House 19	160.0	30	0.06	0.00	0.06	0.00

Table 3.8: Outliers for the computers - no filtering.

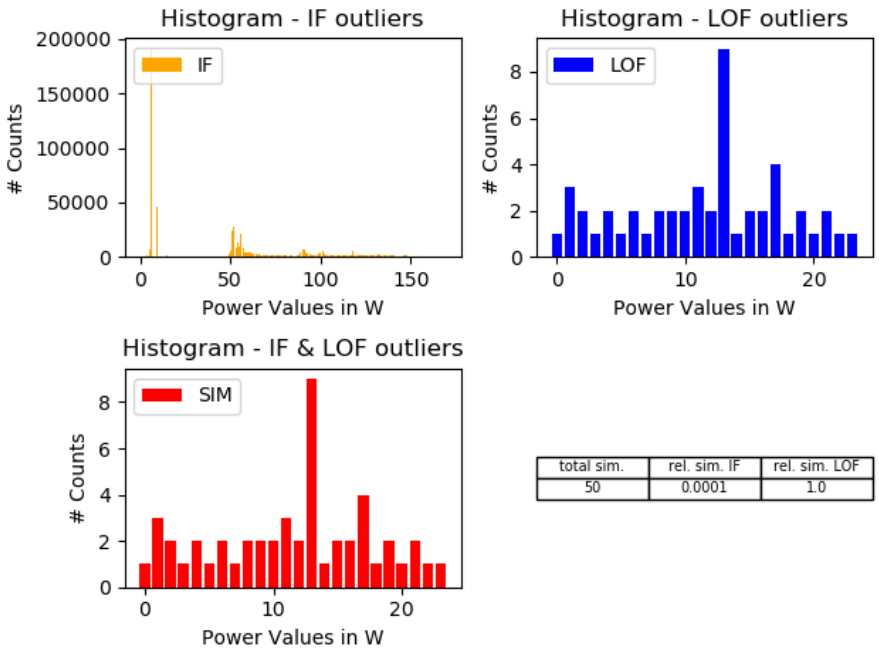

Figure 3.7: Histogram computer outlier - House 15 - no filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	118949	19	0.07	0.00	0.03	0.00
House 5	644999	66	0.35	0.00	0.16	0.00
House 6	634131	119	0.48	0.00	0.30	0.00
House 8	228404	67	0.15	0.00	0.04	0.00
House 12	4514	267	0.01	0.00	0.01	0.00
House 14	75254	16	0.08	0.00	0.01	0.00
House 15	181859	37	0.26	0.00	0.26	0.00
House 16	104189	39	0.23	0.00	0.22	0.00
House 17	101971	77	0.23	0.00	0.14	0.00
House 19	16103	60	0.06	0.00	0.06	0.00

Table 3.9: Outliers for the computers - rolling median filtering.

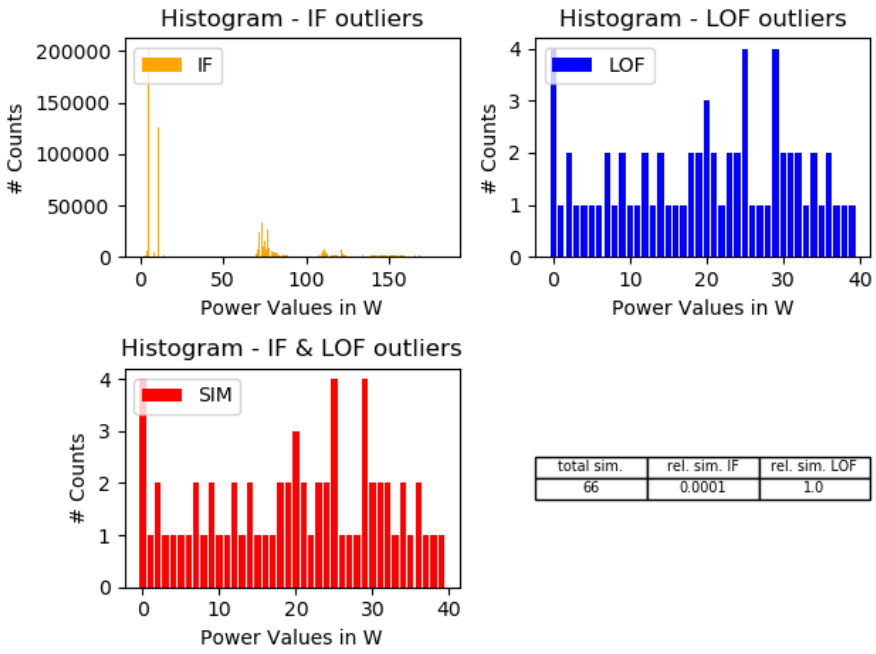

Figure 3.8: Histogram television outlier - House 15 - rolling median filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 1	119576	24	0.07	0.00	0.03	0.00
House 5	641618	32	0.34	0.00	0.16	0.00
House 6	598446	70	0.46	0.00	0.28	0.00
House 8	236178	70	0.16	0.00	0.04	0.00
House 12	4253	116	0.01	0.00	0.01	0.00
House 14	75225	19	0.08	0.00	0.01	0.00
House 15	181744	33	0.26	0.00	0.26	0.00
House 16	97860	23	0.22	0.00	0.21	0.00
House 17	99433	58	0.23	0.00	0.14	0.00
House 19	160.0	40	0.06	0.00	0.06	0.00

Table 3.10: Outliers for the computers - hampel filtering.

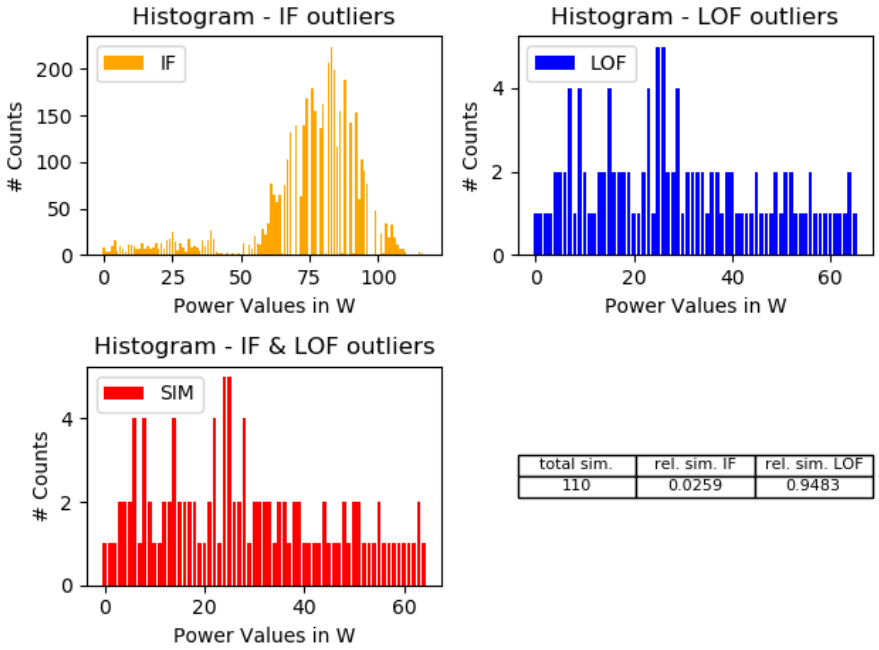

Figure 3.9: Histogram computer outlier - House 15 - hampel filter

3.2.4 Washing Machine

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 2	38488	308	0.05	0.00	0.04	0.00
House 3	102355	366	0.06	0.00	0.04	0.00
House 4	20401	336	0.01	0.00	0.01	0.00
House 5	151399	365	0.08	0.00	0.04	0.00
House 6	36806	464	0.03	0.00	0.02	0.00
House 7	158450	460	0.11	0.00	0.07	0.00
House 10	120014	316	0.08	0.00	0.06	0.00
House 13	35320	439	0.08	0.00	0.05	0.00
House 14	32725	630	0.04	0.00	0.02	0.00
House 15	51877	702	0.07	0.00	0.04	0.00
House 16	8632	438	0.02	0.00	0.01	0.00
House 18	26917	445	0.06	0.00	0.03	0.00
House 19	6129	319	0.02	0.00	0.02	0.00
House 20	19818	342	0.05	0.00	0.03	0.00

Table 3.11: Outliers for the washing machines - no filtering.

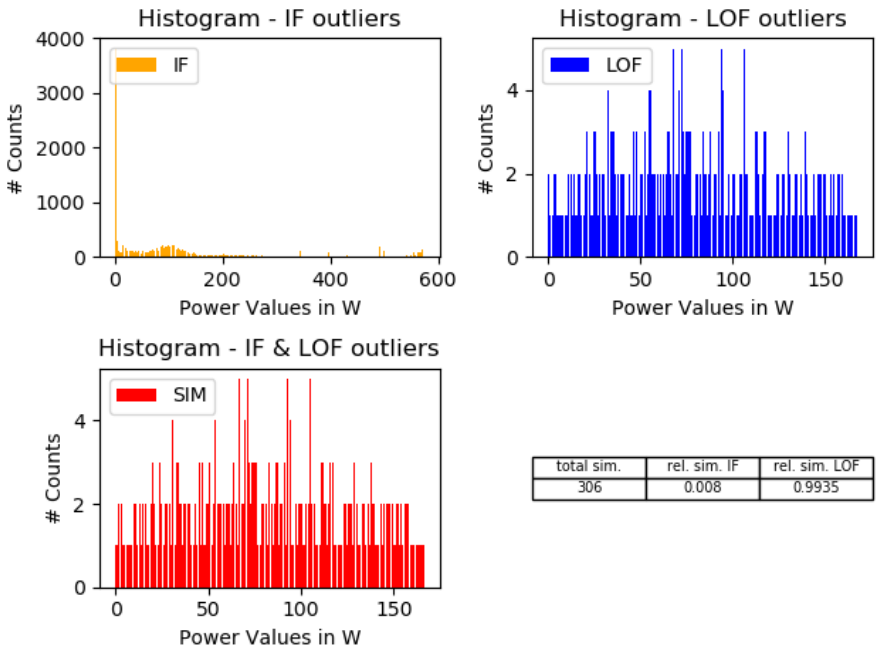

Figure 3.10: Histogram washing machine outlier - House 12 - no filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 2	38483	431	0.05	0.00	0.04	0.00
House 3	101662	648	0.06	0.00	0.05	0.00
House 4	20716	566	0.01	0.00	0.01	0.00
House 5	153875	724	0.08	0.00	0.04	0.00
House 6	36948	669	0.03	0.00	0.02	0.00
House 7	178855	666	0.12	0.00	0.08	0.00
House 10	119964	605	0.08	0.00	0.07	0.00
House 13	39934	628	0.09	0.00	0.06	0.00
House 14	32712	804	0.04	0.00	0.03	0.00
House 15	51838	730	0.07	0.00	0.04	0.00
House 16	8543	434	0.02	0.00	0.01	0.00
House 18	180.0	361	0.04	0.00	0.03	0.00
House 19	6116	320	0.02	0.00	0.02	0.00
House 20	19571	494	0.05	0.00	0.03	0.00

Table 3.12: Outliers for the washing machines - rolling median filtering.

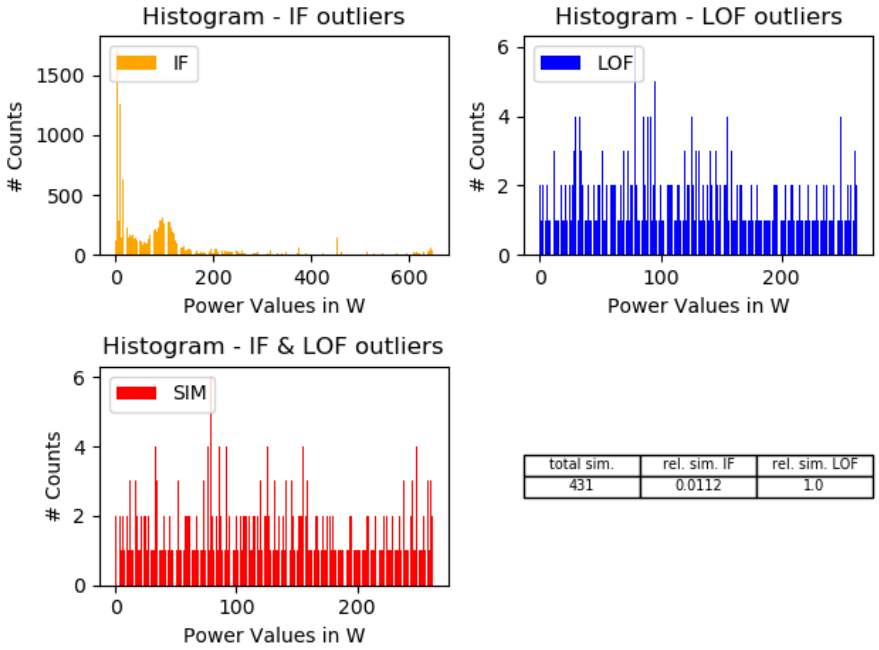

Figure 3.11: Histogram washing machine outlier - House 12 - rolling median filter

	Absolute		**Relative**		**Rel > 25W**	
	IF	LOF	IF	LOF	IF	LOF
House 2	38463	370	0.05	0.00	0.04	0.00
House 3	102019	649	0.06	0.00	0.04	0.00
House 4	20984	358	0.01	0.00	0.01	0.00
House 5	152891	415	0.08	0.00	0.04	0.00
House 6	37201	525	0.03	0.00	0.02	0.00
House 7	181795	637	0.12	0.00	0.07	0.00
House 10	119797	464	0.08	0.00	0.06	0.00
House 13	41572	438	0.09	0.00	0.05	0.00
House 14	32652	718	0.04	0.00	0.02	0.00
House 15	51951	738	0.07	0.00	0.04	0.00
House 16	8440	413	0.02	0.00	0.01	0.00
House 18	17069	390	0.04	0.00	0.03	0.00
House 19	6116	328	0.02	0.00	0.02	0.00
House 20	19490	360	0.04	0.00	0.03	0.00

Table 3.13: Outliers for the washing machines - hampel filtering.

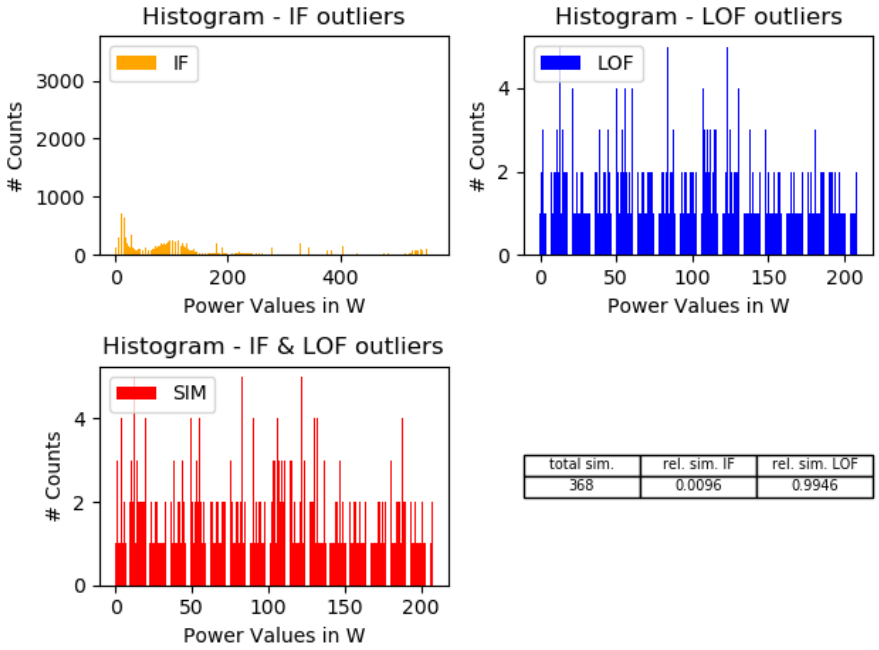

Figure 3.12: Histogram washing machine outlier - House 12 - hampel filter

3.2.5 Discussion

By analyzing the experimental results, it is clear that there are no serious deviations across all applications. For all devices, it is evident that the Isolation Forest algorithm considers more points as outliers as the Local Outlier Factor. This becomes even clearer when looking at the relative numbers. Both for total values and only for values above 25W, the number of values considered outliers is on average in the single-digit percentage range, while for LOF the number is vanishingly small. Concerning filtering, no clear statement can be made, as the number of outliers was similar with no filtering as well as with rolling median and hampel filtering. In most cases it seems that some outliers are filtered out, but not to a significant extent. This occurs for both IF and the LOF algorithm.

The plots clearly show that the agreement rate between Isolation Forest and Local Outlier Factor is mostly above 95%. The agreement rate in this context describes the percentage by which the IF and LOF consider the same points to be outliers. From the consistently high figure of over 95% it can be deduced that both algorithms agree on most points, but that the Isolation Forest algorithm detects many times more outliers.

Since this experiment deals with unlabeled data, which means that it is not known whether the values considered as outliers are really erroneous, no real statement can be made as to whether the IF or the LOF algorithm is more correct

with the results. This also applies to the filter techniques.

3.3 Comparison Filter Techniques

The following figure 3.13 illustrates how filtering affects the data. The data shown here are the for the aggregate data of House 1 from 09.12.2013 and show a time window of one hour.

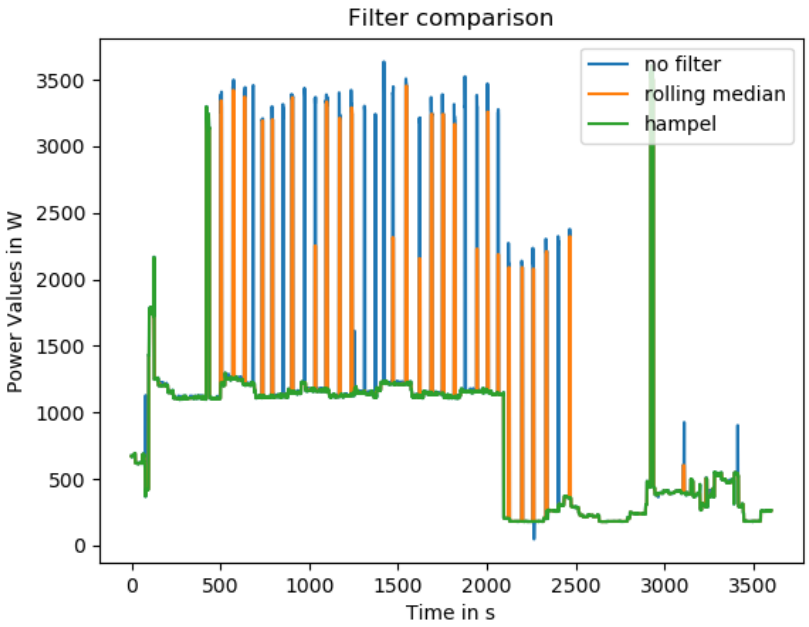

Figure 3.13: Comparison of filtering techniques for the aggregate data of House 1 - 09.12.2013

The following table shows the absolute numbers of outliers detected by the algorithms. A distinction is made whether no filtering, a rolling median, or a hampel filter was applied. This data is intended to show a comparison of whether filtering the data makes a difference in the outliers detected. The plots show the distributions of the outliers with the different filter types.

	Isolation Forest			Local Outlier		
	None	Median	Hampel	None	Median	Hampel
House 1	264694	255583	257275	3882	3789	3958
House 2	83099	82884	80225	3773	3876	3673
House 3	380625	380209	396221	4792	4667	4725
House 4	270149	273727	260862	3559	3598	3646
House 5	418869	418230	408261	4148	4052	4251
House 6	152138	142148	150925	4071	4083	4000
House 7	282760	272161	278041	3844	3644	3715
House 8	276079	285919	281387	5536	5351	5602
House 9	186047	186480	186893	5524	5521	5447
House 10	282654	294771	294995	3939	3698	3948
House 12	55802	54604	54611	3041	3057	2992
House 13	68546	70156	66021	4478	4097	4284
House 14	131826	129520	139759	3006	2826	2815
House 15	138314	130529	137771	4010	3909	4070
House 16	54294	58975	62646	2273	2002	2063
House 17	68842	70324	68568	2612	2543	2601
House 18	59073	60383	59826	2069	1977	1992
House 19	27453	29071	27037	1939	1852	1848
House 20	108459	109304	106373	2896	2614	2743

Table 3.14: Comparison of Filter Techniques.

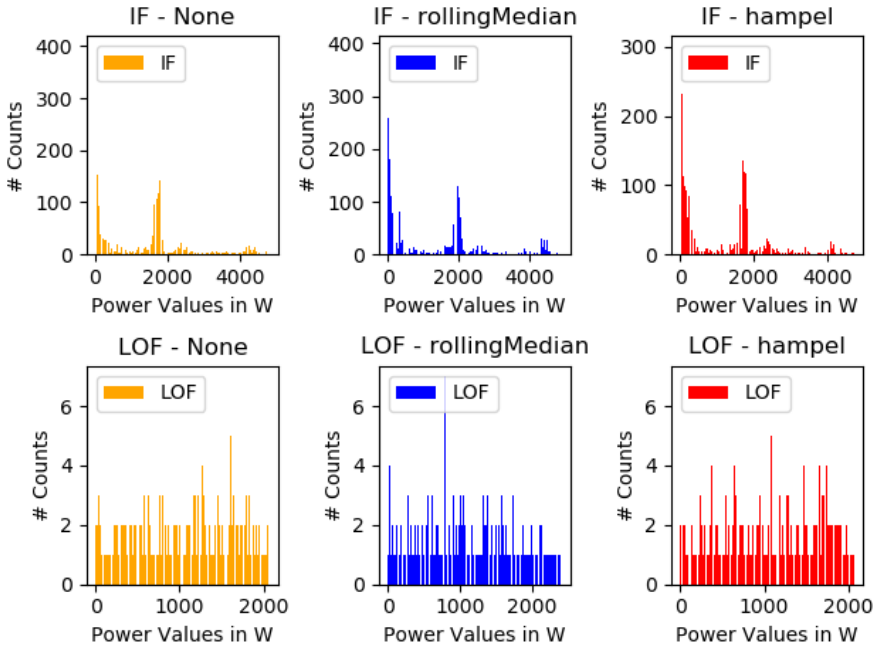

Figure 3.14: Histogram comparison - REFIT2

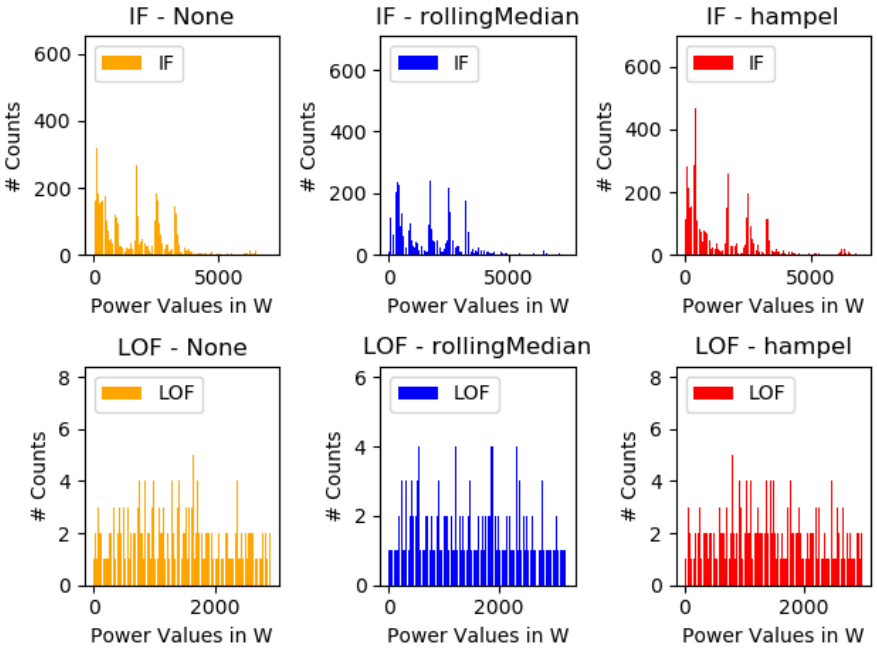

Figure 3.15: Histogram comparison - REFIT8

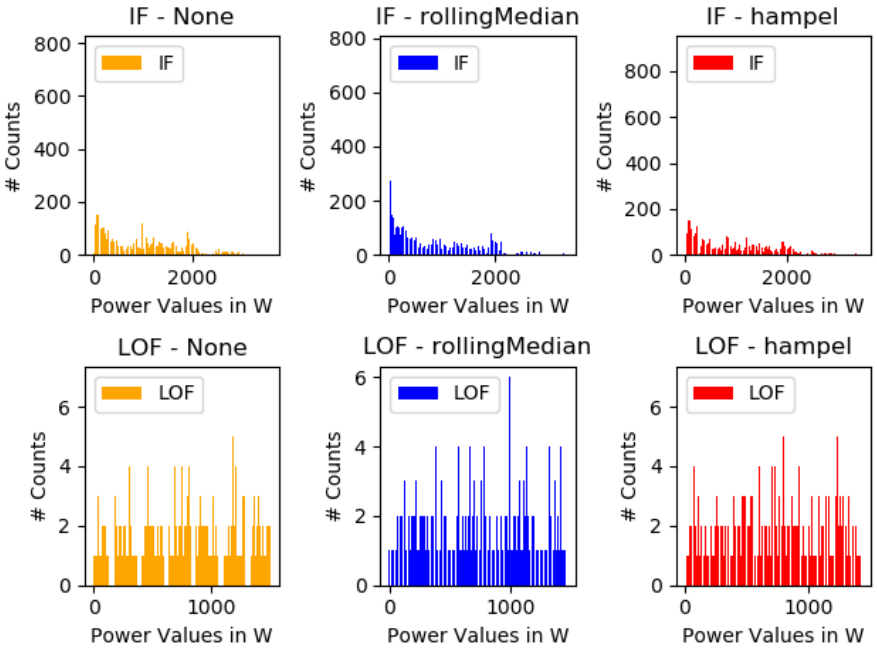

Figure 3.16: Histogram comparison - REFIT20

3.3.1 Discussion

When looking at the results obtained for the total values of the individual houses, it is evident that the number of outliers increases through the application of the filters. However, as with the applications, it must be taken into account that this is unlabelled data, which means that no real statements can be made about how many outliers there are in the data sets in general.

The plots provide an overview of the distributions of the outliers. Here it can be seen that the underlying distribution of the outliers does not change when filtering, but only the frequencies of the outliers increase.

Chapter 4

Discussion

Outlier detection is a broad scientific field. It can be seen in [7] that research on possible benefits has already been done in a variety of applications. A small excerpt of these is Network Intrusion Detection, Insurance Claim Fraud Detection, Insider Trading Detection, or Industrial Damage Detection. As the possibilities for data acquisition is increasing the use of anomaly detection in data sets will become more and more widespread. Algorithms that have become widely accepted in recent years, such as Isolation Forest, are already being improved specifically for applications, as can be seen in [18] where the Isolation Forest has been specially adapted for intrusion detection.
Since there are not yet many available tests for outlier detection on the REFIT dataset, in order to be able to evaluate the results of the previous chapter, some performance evaluations of the two algorithms on non-NILM datasets will be cited in this chapter and some papers dealing with energy losses in households are discussed.

4.1 Credit Card Fraud Detection

As more and more payments are made online by credit card, it is even more interesting for banks and credit card companies to recognize credit card fraud. In [12] both the Isolation Forest and the Local Outlier Factor algorithm were used to detect fraud in transactions. To test how well fraud can be detected, a data set of credit card transactions from September 2013 in Europe was used. In this dataset 284,808 transactions are recorded, of which 0.172% can be considered as fraud transactions. An accuracy of 71% was achieved by applying the isolation forest algorithm to this data and by applying the local outlier factor, an accuracy of 97% could be achieved which means that about 97% of the fraud transactions were correctly recognized.
Also in [26] a comparison of IF and LOF was performed for credit card fraud detection. For the comparison the Kaggle dataset was used which is the same as in [12]. Therefore, the results are comparable to the previous example and again the Isolation Forest outperformed the Local Outlier Factor.

4.2 Isolation Forest Performance

In [16] an experiment was conducted to study the behavior of the newly introduced Isolation Forest in terms of AUC and processing time. The term AUC (Area under curve) is a widely used term in machine learning and is used for method comparisons. The range of AUC values is between 0 and 1, where a value of 0 means that 100% of the predictions were wrong and 1 that 100% of the predictions were correct. The isolation forest algorithm was compared with LOF [15] (Local outlier factor), RF [23] (Random forests), and ORCA [3] (distance-based algorithm). A total of 11 publicly available data sets were used for the experiment. The datasets, such as the Http (KDDCUP99) which contains data for network intrusion detection, are described in more detail in [16]. In the following Figure 4.1 [16] all 4 algorithms are compared using the AUC.

	AUC			
	iForest	ORCA	LOF	RF
Http (KDDCUP99)	**1.00**	0.36	NA	NA
ForestCover	**0.88**	0.83	NA	NA
Mulcross	**0.97**	0.33	NA	NA
Smtp (KDDCUP99)	**0.88**	0.80	NA	NA
Shuttle	**1.00**	0.60	0.55	NA
Mammography	**0.86**	0.77	0.67	NA
Annthyroid	**0.82**	0.68	0.72	NA
Satellite	**0.71**	0.65	0.52	NA
Pima	0.67	**0.71**	0.49	0.65
Breastw	**0.99**	0.98	0.37	0.97
Arrhythmia	**0.80**	0.78	0.73	0.60
Ionosphere	0.85	**0.92**	0.89	0.85

Figure 4.1: Comparison of LOF, IF, RF and ORCA for different datasets [16, Tab. 3]

As can be seen in the table, the Isolation Forest was the best in a general comparison with the other three algorithms for almost all data sets. In the direct comparison with the Local Outlier Factor, the LOF was better than the IF in only one case, and then only with a marginal difference. In all other data sets, the IF was much better. Another advantage of the IF highlighted in [16] is the significant runtime difference between IF and LOF, where the maximum runtime of the IF was 15.58 seconds while the maximum runtime of the LOF was 14647 seconds which is due to the high computation complexity of the LOF.

4.3 Meteorological Outliers Detection

In [27] the IF and LOF algorithm was applied to meteorological data. Meteorological data of the period from 2008 to 20017 were used as the data set. These data, which include the hourly and accumulated temperature, are from the Wushaoling Meteorological Station in China. The authors applied both algorithms to the data to filter out days where there were errors in the data or problems with the measurement. As shown in the tests, both algorithms gave the same results, i.e. the same days were considered as outliers.

4.4 Performance Evaluation Conclusion

As can be seen in all these experiments, the performance of both algorithms is very dependent on the outlier detection data. An important point to note, however, is that the Isolation Forest algorithm has a significant advantage in terms of runtime.

4.5 Household Energy Losses

As described in [22], faults in household appliances, such as software program errors, increase building energy consumption by between 4% and 18%, in [13] it is mentioned that due to poorly maintained equipment, wear and tear,... about 15% to 30% of energy is wasted and in [21] that this proportion can be up to 20%. From the statements in the three papers mentioned above, the assumption can be made, that a single-digit percentage of the data from the REFIT dataset can be considered as outliers.

Chapter 5

Conclusions

This work addresses the application of two algorithms, Isolation Forest and Local Outlier Factor to NILM datasets. These two algorithms are widely used in the context of outlier detection, but so far they have only been applied to other use cases respectively data. The aim of this thesis was on the one hand to check how large the proportion of the data is which is considered an outlier by the algorithms and whether there are similarities in the results between IF and LOF, and on the other hand, how the results change after a filter has been applied. For further evaluations of the test results, it is important to note that unlabeled data was used in the experiments, which means that it is not known how much erroneous data are actually present in the REFIT data set. When applying the two algorithms to the datasets, independent of the results obtained, the important finding was made that the runtime differences between the two algorithms were significant. While the Isolation Forest had an average runtime of a few minutes, the runtime for the Local Outlier Factor was in the range of hours. While this is not as crucial in experiments, it can make a very big difference when using one of the algorithms in practical applications. Regarding the obtained results, it can be said that for all the tested applications the IF and LOF always have an agreement rate higher than 90%. This shows that the two algorithms consider the same data points as outliers. However, the difference became apparent when comparing the results for the absolute and relative numbers. In the case of IF, the relative numbers were mostly in the low double digits with peaks as high as 50%. In the case of LOF, however, the number of outliers detected was always too small to be noticeable at all in the relative numbers. The column with power values above 25W was included in the table to disregard data points that only occurred due to background noise when the device was switched off. It can be seen from the tables that single-digit percentages were obtained on average with the IF algorithm and also no noticeable results for the LOF. The comparison of the filtering techniques showed that that the application of a filter has no significant influence on the outlier detection. Often the number of detected outliers was lower with filtering than without, but it also happened that more outliers were detected with rolling median and hampel filtering than without

filtering. This applies to both IF and LOF. In conclusion, the Isolation Forest algorithm has proven to be far more feasible for outlier detection in NILM datasets, and based on the numbers for energy losses in households presented in the previous chapter, it can be assumed that the results obtained by the IF can be considered more realistic.

Future work could be to use the two algorithms to remove outliers from datasets and then compare them to see if better results can be achieved using NILM algorithms or to use labeled data to verify that the results obtained from these experiments actually detect the outliers in the data sets.

Bibliography

[1] D. Anguita, L. Ghelardoni, A. Ghio, L. Oneto, and S. Ridella. The'k'in k-fold cross validation. In *ESANN*, 2012.

[2] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava. Nilmtk: an open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th international conference on Future energy systems*, pages 265–276, 2014.

[3] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 29–38, 2003.

[4] S. Bhowmik, B. Jelfs, S. P. Arjunan, and D. K. Kumar. Outlier removal in facial surface electromyography through hampel filtering technique. In *2017 IEEE Life Sciences Conference (LSC)*, pages 258–261. IEEE, 2017.

[5] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano. A review on outlier/anomaly detection in time series data. *arXiv preprint arXiv:2002.04236*, 2020.

[6] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[8] F. Corno and F. Razzak. Intelligent energy optimization for user intelligible goals in smart home environments. *IEEE transactions on Smart Grid*, 3(4):2128–2135, 2012.

[9] P. Dayan, M. Sahani, and G. Deback. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, pages 857–859, 1999.

[10] M. Goldstein and A. Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pages 59–63, 2012.

[11] D. M. Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[12] H. John and S. Naaz. Credit card fraud detection using local outlier factor and isolation forest. *Int. J. Comput. Sci. Eng.*, 7:1060–1064, 2019.

[13] S. Katipamula and M. R. Brambley. Methods for fault detection, diagnostics, and prognostics for building systems—a review, part i. *Hvac&R Research*, 11(1):3–25, 2005.

[14] C. Klemenjak and P. Goldsborough. Non-intrusive load monitoring: A review and outlook. *arXiv preprint arXiv:1610.01191*, 2016.

[15] E. M. Knox and R. T. Ng. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the international conference on very large data bases*, pages 392–403. Citeseer, 1998.

[16] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[17] S. Makonin and F. Popowich. Nonintrusive load monitoring (nilm) performance evaluation. *Energy Efficiency*, 8(4):809–814, 2015.

[18] P.-F. Marteau, S. Soheily-Khah, and N. Béchet. Hybrid isolation forest-application to intrusion detection. *arXiv preprint arXiv:1705.03800*, 2017.

[19] D. Murray, L. Stankovic, and V. Stankovic. An electrical load measurements dataset of united kingdom households from a two-year longitudinal study. *Scientific data*, 4(1):1–12, 2017.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[21] H. Rashid, N. Batra, and P. Singh. Rimor: Towards identifying anomalous appliances in buildings. In *Proceedings of the 5th Conference on Systems for Built Environments*, pages 33–42, 2018.

[22] K. W. Roth, D. Westphalen, M. Y. Feng, P. Llana, and L. Quartararo. Energy impact of commercial building controls and performance diagnostics. *TIAX (http://apps1. eere. energy. gov/buildings/publications/pdfs/corporate/pnnl-15149_ market_ assessment. pdf, 2005)*, 2005.

[23] T. Shi and S. Horvath. Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics*, 15(1):118–138, 2006.

[24] K. Singh and S. Upadhyaya. Outlier detection: applications and techniques. *International Journal of Computer Science Issues (IJCSI)*, 9(1):307, 2012.

[25] J. Suomela. Median filtering is equivalent to sorting. *arXiv preprint arXiv:1406.1717*, 2014.

[26] V. Vijayakumar, N. S. Divya, P. Sarojini, and K. Sonika. Isolation forest and local outlier factor for credit card fraud detection system.

[27] F. Xue, W. Zheng, M. Zhang, Q. Wu, Z. Yang, and X. Ai. Meteorlogical outliers detection based on artificial intelligence. In *IOP Conference Series: Earth and Environmental Science*, volume 474, page 032039. IOP Publishing, 2020.

[28] A. Zoha, A. Gluhak, M. A. Imran, and S. Rajasegarar. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors*, 12(12):16838–16866, 2012.

Appendices

Appendix A

Code

The code used for the experiments can be seen below.

```
import numpy as np
import datetime
import sys
import os
import json
import pandas as pd
import matplotlib.pyplot as plt
from nilmtk.dataset import DataSet
from collections import Counter
from joblib import parallel_backend
from sklearn.model_selection import KFold
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM

## LOAD DATA FROM DATASET ##

def loadApplicationDataFromDataset(application, settings):
    data = DataSet(settings["h5File"])
    df = None

    window = settings["window"]
    if settings["window"] != None:
        data.set_window(window[0],window[1])
    try:
        elecMeter = data.buildings[application[0]].elec
        df = elecMeter[application[1]].power_series_all_data
```

```
            ()
    except:
        print(str(e))
        return None

    if df is None:
      return None

    if df.isnull().values.any():
        df.fillna(method='ffill')

    if settings["filterData"] == "rollingMedian":
        df = df.rolling(window=10,
                        center=True,
                        min_periods=1).median()
    elif settings["filterData"] == "hampel":
        df = hampel_filter(df, 10)

    power_values = np.arange(int(max(df))+1).reshape(-1, 1)
    x = df.astype('int').values.reshape(-1, 1)

    return {"power_val": power_values, "x": x}

def loadAggregateDataFromDataset(application, settings):
    data = DataSet(settings["h5File"])
    df = None

    window = settings["window"]
    if settings["window"] != None:
        data.set_window(window[0],window[1])
    try:
        elecMeter = data.buildings[application[0]].elec
        df = elecMeter.mains().power_series_all_data()
    except Exception as e:
        print(str(e))
        return None

    if df is None:
      return None

    if df.isnull().values.any():
        df.fillna(method='ffill')

```

```
70      if settings["filterData"] == "rollingMedian":
71          df = df.rolling(window=10,
72                          center=True,
73                          min_periods=1).median()
74      elif settings["filterData"] == "hampel":
75          df = hampelFilter(df, 10)
76
77      power_values = np.arange(int(max(df))+1).reshape(-1, 1)
78      x = df.astype('int').values.reshape(-1, 1)
79
80      return {"power_val": power_values, "x": x}
81
82  ## HAMPEL FILTER ##
83
84  def hampelFilter(df, windowSize, nSig=3):
85
86      k = 1.4826 # Gaussian distribution scale factor
87      dfFiltered = df.copy()
88
89      MAD = lambda x: np.median(np.abs(x - np.median(x)))
90      rollMedian = df.rolling(window=2*windowSize,
91                              center=True).median()
92      rollMAD = k * df.rolling(window=2*windowSize,
93                               center=True).apply(MAD, raw=
                                     False)
94      diff = np.abs(df - rollMedian)
95
96      idx = list(np.argwhere(diff > (nSig * rollMAD)).flatten
            ())
97      dfFiltered[idx] = rollMedian[idx]
98
99      return dfFiltered
100
101
102 ## ISOLATION FORESTS ###
103
104 def calcIsoForest(x_train, x_test):
105     try:
106         clf = IsolationForest(random_state=0,
107                               n_jobs=6).fit(x_train)
108         outliers = clf.predict(x_test)
109     except:
110         return None
```

```
111     return outliers
112
113 ## Local Outlier Factor
114
115 def calcLOF(x_train, x_test):
116     try:
117         clf = LocalOutlierFactor(n_neighbors=2,
118                                  n_jobs=6)
119         outliers = clf.fit_predict(x_test)
120     except:
121         return None
122     return outliers
123
124 def storeResults(settings, metrics):
125     if os.path.exists(settings["fileName"]):
126         fileMode = 'a'
127     else:
128         fileMode = 'w'
129
130     with open(settings["fileName"], fileMode) as outfile:
131         if settings["jsonIndent"]:
132             json.dump([settings, metrics],
133                       outfile,
134                       indent=1)
135         else:
136             json.dump([settings, metrics], outfile)
137         outfile.write("\n")
138
139 ## METRICS ###
140
141 def calcMetrics(results, settings):
142     metrics = {}
143
144     if "lof" in settings["algo"]:
145         metrics["lof"] = {}
146         metrics["lof"]["absApplOut"] = np.count_nonzero(
            results["outLOFTest"] == -1)
147         metrics["lof"]["relApplOut"] = np.count_nonzero(
            results["outLOFTest"] == -1)/len(results["
            outLOFTest"])
148         lofPowerOutlier = (results["testData"][(np.where(
            results["outLOFTest"] == -1))])[:,0]
149         keys_values = dict(Counter(lofPowerOutlier)).items()
```

```
150         metrics["lof"]["applOutOccur"] = {str(key): str(
                value) for key, value in keys_values}
151         metrics["lof"]["applOutlier"] = (np.where(results["
                outLOFTest"] == -1))[0].tolist()
152 
153         metrics["lof"]["absPowOut"] = np.count_nonzero(
                results["outLOFPower"] == -1)
154         metrics["lof"]["relPowOut"] = np.count_nonzero(
                results["outLOFPower"] == -1)/len(results["
                outLOFPower"])
155         metrics["lof"]["powerOutVal"] = (results["powerData"
                ][(np.where(results["outLOFPower"] == -1))])
                [:,0].tolist()
156         metrics["lof"]["powerOutlier"] = (np.where(results["
                outLOFPower"] == -1))[0].tolist()
157 
158     if "if" in settings["algo"]:
159         metrics["if"]  = {}
160         metrics["if"]["absApplOut"] = np.count_nonzero(
                results["outIsoForTest"] == -1)
161         metrics["if"]["relApplOut"] = np.count_nonzero(
                results["outIsoForTest"] == -1)/len(results["
                outIsoForTest"])
162         ifPowerOutlier = (results["testData"][(np.where(
                results["outIsoForTest"] == -1))])[:,0]
163         keys_values = dict(Counter(ifPowerOutlier)).items()
164         metrics["if"]["applOutOccur"] = {str(key): str(value
                ) for key, value in keys_values}
165         metrics["if"]["applOutlier"] = (np.where(results["
                outIsoForTest"] == -1))[0].tolist()
166 
167         metrics["if"]["absPowOut"] = np.count_nonzero(
                results["outIsoForPower"] == -1)
168         metrics["if"]["relPowOut"] = np.count_nonzero(
                results["outIsoForPower"] == -1)/len(results["
                outIsoForPower"])
169         metrics["if"]["powerOutVal"] = (results["powerData"
                ][(np.where(results["outIsoForPower"] == -1))])
                [:,0].tolist()
170         metrics["if"]["powerOutlier"] = (np.where(results["
                outIsoForPower"] == -1))[0].tolist()
171 
172     return metrics
```

```
173
174
175 def calcOutliers(data, settings):
176     results = {}
177     kf = KFold(n_splits = settings["kFoldSplits"])
178
179     for train_i, test_i in kf.split(data["x"]):
180         testStr = str(test_i[0])+"␣-␣" +str(test_i[-1])
181         settings["testDataStr"] = testStr
182
183         trainStr = str(train_i[0])+"␣-␣"+str(train_i[-1])
184         settings["trainDataStr"] = trainStr
185         results["powerData"] = data["power_val"]
186
187         x_train, x_test  = data["x"][train_i], data["x"][
              test_i]
188         results["testData"] = x_test
189         results["trainData"] = x_train
190         if "if" in settings["algo"]:
191             outliers = calcIsoForest(x_train, x_test)
192             if outliers is None:
193                 return
194             results["outIsoForTest"] = outliers
195
196             outliers = calcIsoForest(x_train, data["
                  power_val"])
197             if outliers is None:
198                 return
199             results["outIsoForPower"] = outliers
200
201         if "lof" in settings["algo"]:
202             outliers = calcLOF(x_train, x_test)
203             if outliers is None:
204                 return
205             results["outLOFTest"] = outliers
206
207             outliers = calcLOF(x_train, data["power_val"])
208             if outliers is None:
209                 return
210             results["outLOFPower"] = outliers
211
212         metrics = calcMetrics(results, settings)
213         storeResults(settings, metrics)
```

```
214
215 ## GET ALL APPLICATIONS ###
216
217 def getApplicationsFromDataset(settings, applicationTypes=
    None):
218     applications = []
219
220     data = DataSet(settings["h5File"])
221
222     for building in data.buildings:
223             for appl in data.buildings[building].elec.
                appliances:
224                 if (building, appl.type["type"]) not in
                    applications:
225                     if applicationTypes is None:
226                         applications.append((building,
227                                              appl.type["type"
                                                 ]))
228                     else:
229                         if appl.type["type"] in
                            applicationTypes:
230                             applications.append((building,
231                                                  appl.type["
                                                     type"]))
232     return applications
233
234 ## GET AGGREGATES ###
235
236 def getAggregateFromDataset(settings, applicationTypes=None)
    :
237     applications = []
238
239     data = DataSet(settings["h5File"])
240
241     for building in data.buildings:
242         if (building, None) not in applications:
243             applications.append((building, None))
244
245     return applications
246
247 ## MAIN ###
248
249 if __name__ == "__main__":
```

```
250
251     ## sys.argv usage - python outlierMain.py
252     #                          h5File
253     #                          applications
254     #                          windowStart
255     #                          windowEnd
256     ## applications - ["kettle", "computer"] or
257     #                          "aggregate" if aggregate data
258     #                          should be used
259     ## windowStart - "2013-09-08" or None if whole dataset
260     #                          should be used
261
262     if len(sys.argv) != 5:
263         print("Not enough arguments")
264         sys.exit()
265
266     settings = {}
267     files = [sys.argv[1]]
268
269     filters = [None, "rollingMedian", "hampel"]
270     applicationTypes = sys.argv[2].split(",")
271
272     for filterType in filters: # None, rollingMedian or
            hampel
273         for h5File in files:
274             ### SETTINGS ###
275             settings["kFoldSplits"] = 5
276
277             if sys.argv[3] == None:
278                 settings["window"]       = None
279             else:
280                 settings["window"]       = (sys.argv[3], sys.
                        argv[4])
281
282             settings["filterData"]  = filterType
283             settings["algo"]        = "if|lof"
284             settings["jsonIndent"]  = False # increases
                    readability
285             ### /SETTINGS ###
286
287             ## check if directory with h5Files exist else
                    path passed from user
288             if os.path.exists("h5Files"):
```

```
289             settings["h5File"]      = "h5Files/" +
                    h5File
290         else:
291             settings["h5File"]      = h5File
292
293         ## create directory for results
294         if not os.path.exists("simuResults"):
295             os.mkdir("simuResults")
296
297         time = datetime.datetime.now().strftime("_%d%m%
                Y_%H%M%S.json")
298         settings["fileName"]    = "simuResults/simRes_"
                + h5File + time
299
300         if applicationTypes == "aggregate":
301             applications = getAggregateFromDataset(
                    settings)
302         else:
303             applications = getApplicationsFromDataset(
                    settings, applicationTypes)
304
305         for appl in applications:
306             if applicationTypes == "aggregate":
307                 data = loadAggregateDataFromDataset(appl
                        , settings)
308             else:
309                 data = loadApplicationDataFromDataset(
                        appl, settings)
310
311             if data is not None:
312                 settings["application"] = appl
313                 settings["maxPowVal"]   = int(data["
                        power_val"][-1])
314                 calcOutliers(data, settings)
```

Appendix B

GitHub Repository

A repository containing the simulation script, a file to create the conda environment and the simulation results of this work can be found at the following link:

https://github.com/rauterRaphael/comparisonNILMOutlierDetection