FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	AI detectors for automated identification of irregular events in time-series of big data
Student:	Anna Husieva
Supervisor:	Ing. Petr Habásko
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

1. Research existing supervised and unsupervised methods for anomaly detection, with a special focus on the time-series data
2. Research the state-of-art electric vehicle battery technologies with recent development in this area
3. Optimize these methods for big data (from fleets of thousands of connected cars) and scalability on cloud environments
4. Python implementation of the developed anomaly detectors using available AI packages and demonstration on selected datasets

Literatura: Anomaly Detection Principles and Algorithms Authors: Mehrotra, Kishan G., Mohan, Chilukuri, Huang, Huaming; Practical Time Series Analysis: Prediction with Statistics and Machine Learning by Aileen Nielsen

---

Electronically approved by Ing. Karel Klouda, Ph.D. on 18 October 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

AI detectors for automated identification of irregular events in time-series of big data

Anna Husieva

Department of Applied Mathematics
Supervisor: Ing. Petr Habásko

May 9, 2022

Acknowledgements

Firstly, I would like to extend my deepest gratitude to my family, which gave me the opportunity to study at this university and was my biggest support throughout my studies. I would also like to extend my sincere thanks to my supervisor Ing. Petr Habásko, for his expertise, guidance and invaluable support, as well as his patience and willingness to help.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 9, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Husieva, Anna. *AI detectors for automated identification of irregular events in time-series of big data.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Detekce anomálií v časových řadách hraje hlavní roli při odhalování podvodů, hardwarových závad a dalších méně častých událostí, které mohou mít velký dopad, avšak je obtížné je odhalit. Tato práce si klade za cíl prozkoumat existující metody detekce anomálií. Skládá se z přehledu algoritmů se zvláštním zaměřením na autoenkodéry. Praktická část analýzy je provedena na datasetu, který obsahuje velké množství nabíjecích cyklů elektrických vozidel.

Klíčová slova Detekce anomálií, strojové učení, baterie elektrických vozidel, optimalizační metody AI, big data časových řad, Python

Abstract

Time-series anomaly detection plays a starring role in detecting fraud, hardware defects, and other infrequent events that may have great impact but are hard to find. This thesis sets the goal to research existing outlier identification methods. It consists of algorithms' overview, with a special focus on autoencoders. The practical analysis part is performed on a dataset, which contains charging cycles of electric vehicles.

Keywords Anomaly detection, machine learning, electric vehicle battery, optimization AI methods, big data of time-series, Python

Contents

List of Figures

List of Tables

Introduction

Researchers have been studying anomaly detection for more than a century. Anomalies occur in a variety of fields, including medicine, finance, cybersecurity and sociology.

In some areas, abnormal behaviour is an indicator of problematic behaviour. On the other hand, in some areas, abnormal events can also be indicators of positive outcomes. For example, if a company's share price rises unexpectedly. In other areas it may simply indicate a phenomenon that is not understood, or an unknown object or process. It is a catalyst for human inquiry and the exploration of new ideas [1]. Anomaly detection in electric vehicle battery is often connected with recognition abnormal behavior of various systems.

In time series data, an anomaly or outlier can be termed as a data point which is not following the common collective trend or seasonal or cyclic pattern of the entire data and is significantly distinct from rest of the data. By significant, most data scientists mean statistical significance, which in order words, signify that the statistical properties of the data point is not in alignment with the rest of the series [2].

There is an urgent need to track anomalies in various sectors and to continuously investigate the details and causes of these anomalies in order to prevent system failures or irreversible processes. It has become a natural expectation for data scientists to track, investigate and analyse anomalous data points and come up with ways to gain meaningful information.

Businesses at all stages, from supply chain and manufacturing to sales and marketing, need to be well-informed about these anomalies, in order to shape processes and maximise productivity and outcomes. It is therefore important to isolate and examine anomalies with time and attention, as long as there are common patterns in the data, especially time series data.

Battery management system monitoring contributes to time series data as well. Battery management systems (BMS) are electronic control circuits that monitor and regulate the charging and discharge of batteries. The battery

characteristics to be monitored include the detection of battery type, voltages, temperature, capacity, state of charge, power consumption, remaining operating time, charging cycles, and some more characteristics. Many different components contribute to the correct charging of electric cars. This is a wide field for examining anomalous behaviour.

The battery is still the most expensive component of any electric car and, if mishandled, its service life can be considerably shortened and under unfavorable conditions, it also presents a safety hazard for the car itself and its crew. It is important to ensure the right conditions for individual battery cells and thus for the entire battery, which is what BMS takes care of.

The main contribution of this thesis is an automated anomaly detection for EV battery management systems which will be further developed in an automotive company.

Aim of the Thesis

The primary aim of this thesis can be summarized as follows:

1. Analyze existing algorithms for detecting outliers.
2. Compare AI methods for applicability for anomaly detection.
3. Research the state-of-art electric vehicle battery technologies.
4. Research methods for optimizing methods for anomaly detection for big data.

Structure of the work

This bachelor thesis contains several chapters. Chapter 1 is dedicated to the introduction to the battery management system of electric vehicles. Chapter 2 provides the theoretical description of basic anomaly detection approaches. Chapter 3 describes the application of artificial neural networks for anomaly identification. It also provides basics of the machine learning. Lastly, Chapter 4 shows all of our experimental results and discussion of those results.

Chapter 1

Battery Management System

Each manufacturer makes the battery slightly differently, uses different chemicals, or makes cells with different shapes. Among the various types of batteries in the market, lithium-ions are the most efficient in electrical systems. This is due to the high energy and power density of this type as well as the wide temperature operating range, small size, long lifespan, fast recharging characteristics, and low self-discharge rate [3]. Battery Management Systems (BMSs) are essentially important for increasing the efficiency of battery state monitoring and protection from over current and voltage as well as internal and external short circuits [4]. Although there has been a lot of research and patents on BMSs and their applications in the last decade, many are still open for further research.

1.1 Main Fuctions of the BMS

Batteries of electric vehicles have to be protected from overcurrent or overvoltage during charging or discharging mode, i.e., driving on-road or connected to grid. Therefore, a battery management in these modes are important to effectively protect it and prolong its life cycle [5].

When the battery is in the discharging mode, it may be exposed to undercurrent and under-voltage. While in the charging mode, the battery may be exposed to over-current and over-voltage, and consequently, its temperature will increase rapidly [6]. Therefore, the battery protection is indispensable in BMS and plays a crucial role. In the past few years, many accidents have been witnessed and have led to life and financial losses. These issues prompted the battery manufacturers to develop solutions for temperature control and heat management that guarantee operations in the permissible and tolerable ranges of the cells and prevent from thermal runaway and internal short circuit [7].

In order to implement BMS in EV, a combination of hardware and software is always needed. With the development of the wireless charging of EVs over the sparse charging stations in the smart network, communication and

networking as one of the subsections of BMS will affect the overall battery performance [8].

It is necessary that the charging always adapts in real time, because the parameters of the battery itself change over time (oxidation occurs at the terminals, changes in the capacity of the battery cells, etc.). This requires the whole system to be intelligent.

1.2 BMS as a part of an electric vehicle

In EVs, series-connected battery cells are used to feed the electric motors and their accessories. The operating conditions of these cells are different meanwhile the charging and discharging modes of battery. Each cell might have different voltage and current from other cells, and can lead to overcharge or undercharge to some of the cells. These may cause early damage to some cells and sometimes internal short circuit due to deformation of the battery anode, cathode and separator. To solve such a problem, cell balancing is used to equalize the voltage levels of cells and energy distribution in EV [9].

Recording the voltage, current and temperature of the battery cells using sensors and data acquisition system [10], data can be generated to analyze the consumption pattern of electric vehicles and the prediction of battery's future status by using feature extraction and data-driven methods [11].

This is one of the most important functions for the BMS to tell the driver how long he can drive. However, determining the state of charge is not as simple as it may seem. In fact, it is one of the most complicated problems in BMS development.

1.3 Machine Learning Approaches in BMS

Many studies agree that battery health is affected by temperature, battery charging current, number of charging cycles and other primary factors. However, not all processes in the battery are fully known, so there are no precise methods for determining battery health. As with determining the state of charge, it is necessary to rely on approximate computer models that take into account internal resistance, conductivity, self-discharge rate, capacity, energy received during charging, temperature during use, age, number of cycles, etc [12].

Machine Learning Approaches in BMS Applications Due to the complex internal dynamic behavior of the battery and uncertain external operating conditions, is usually difficult to accurately model the battery by equivalent circuit and physical-based models that are associated with estimating the model parameters using model-based approaches. Machine learning methods which are based on mathematical and statistical concepts are considered as reliable and convenient techniques to be used in BMS. Machine learning techniques use

the battery state-of-health (SOH) data, which can be measured by advanced sensor technology [13]. Such methods extract appropriate feature information and build the degradation model to predict remaining useful life (RUL) and end of life (EOL). These techniques are able to represent degradation-intrinsic relationships and trends based on history data [14]. Although a huge number of data are needed during the training phase and the predictive model is non-transparent [15].

There are some critical issues in the battery management system, including protection of over/under voltage and over current, which are a common fault type of battery systems [16]. In charge/discharge mode, the battery undergoes irreversible chemical reactions that can affect the lithium plating and dendrite formation, especially in low temperature. In addition, the formation of dendrite due to the intercalation between anode and cathode can lead to an internal short circuit, which can affect the battery performance and safety. Ignoring this critical issue can cause catastrophic faults owing to thermal runaway. Therefore, a lot of efforts have been done on the fault diagnosis and safety management using model-based and machine learning methods for the battery protection [17].

The effectiveness of machine learning approaches using supervised learning methods is investigated in classification. The algorithms that are evaluated for diagnosing battery cells are k-nearest neighbors (k-NN), logistic regression (LR), Gaussian naive Bayes (GNB), kernel vector machine (KSVM) and neural network (NN). These linear and nonlinear techniques are proven to classify battery cells that are unbalanced and damaged.

Chapter 2

Basic Anomaly Detection Approaches

By definition, identifying an anomaly involves figuring out that a data point is "different" from others. This definition is necessarily parameterized by the data set against which the data point is compared: a person who is five feet tall may be anomalous among male college basketball players, but not among horse-riding jockeys.

In continuous spaces where all data attributes are real-valued (possibly within a bounded range), let's say a data point is "different" from others if its distance to other points is large.

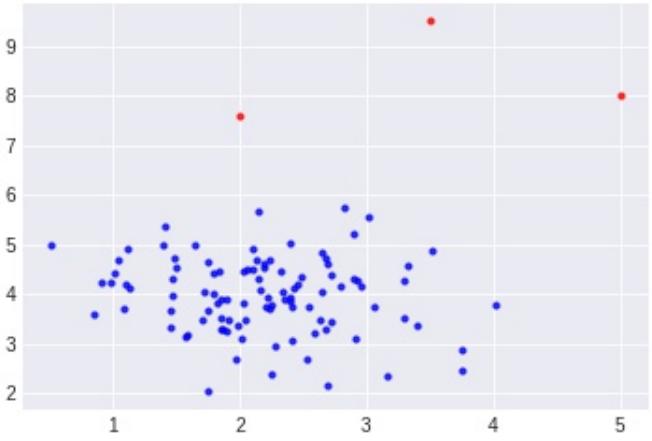

Figure 2.1: Example of distance-based outliers

However, anomaly detection algorithms differ in how this distance is evaluated. This is because no consensus exists on which sets of points are to be

used to compare distances, nor on how to evaluate the distance to a collection of points, even though most researchers agree to work with the standard and well-known definitions of the distance between two points [1].

The literature on measuring distances between time series also uses the term similarity to describe these metrics. In most cases, you can treat these terms interchangeably, namely as a way to establish which time series are more or less like one another [18].

Anomaly detection is important in time series for a few reasons: it can be helpful to remove outliers when fitting models that are not sufficiently robust to such outliers; it can be helpful to identify outliers for building a forecasting model specifically to predict the extent of such anomalous events conditional on knowing whey will happen [19].

2.1 Distance-Based Anomaly Detection Approaches

If other measures in data set $\mathcal{D}$ cannot be compared and the cross-correlation of is different, then Mahalanobis distance is the preferred measure.

Definition 2.1.1. Mahalanobis distance is defined by next equation

$$\sqrt{(p-q)^T S^{-1}(p-q)}, \tag{2.1}$$

where p, q data points in $\mathcal{D}$, S is the covariance matrix measuring the mutual correlations between dimensions for all points in the dataset $\mathcal{D}$.

Definition 2.1.2. The Minkowski distance of order l between two points $p = (p_1, ..., p_d)$ and $q = (q_1, ..., q_d) \in \mathcal{D}$, where d is a dimension of dataset $\mathcal{D}$, is defined as

$$\left(\sum_{i=1}^{d} |p_i - q_i| \right)^{\frac{1}{l}}, \tag{2.2}$$

Most often used values of l are 1 and 2; for $l = 2$ the Minkowski distance is equal to the Euclidean distance and for $l = 1$ this distance is equal to Manhattan distance.

The simplest anomaly detection algorithm evaluates every point $p \in D$ for every point in D. The sub of distances form all points can be used as the anomalousness metric. Another possible metric is the distance to the nearest neighbor.

One of the most popular indicators is the average distance to the nearest k Nearest Neighbors. Adding an additional points can significantly change the results of the calculation, so the arithmetic mean is not very robust.

2.2 Clustering-Based Anomaly Detection Approaches

Clustering can be based on similarity or distance calculations. Although these two approaches are different, the end result is often the same because the similarity measurement is strongly negatively correlated with the distance measurement.

Distance-based clustering techniques are based on the idea that points in the same cluster are separated by a relatively small distance, while points in different clusters are apart.

The similarity-based clustering approach suggests that points that are similar to each other must belong to the same cluster, as points that are closer to each other are expected to be more similar.

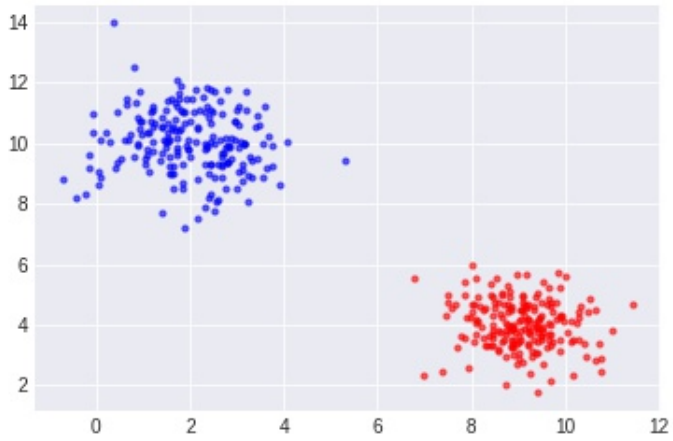

Figure 2.2: Example of two clusters

Clustering algorithms generally assume that the data resides in a continuous multidimensional space with boundaries and that a measure of similarity or distance is already selected.

This chapter describes an anomaly detection approach based on the explicit identification of clusters in a dataset.

Points that are not in the cluster are candidates for anomalies. Variations between algorithms evaluate relative anomalies at points near (but not within) the cluster and points around the cluster.

2.2.1 Nearest Neighbor Clustering

The k-Nearest Neighbor algorithm has been proposed for many types of problems. This algorithm is based on the main idea that an individual must

resemble most of its k adjacencies, not the centroid or a large set of data. The algorithm implementation can be written in this pseudo code. 1

Algorithm 1: K nearest neighbors

Input : Test data $\mathcal{D}_{test}$, train data $\mathcal{D}_{train}$ and non-negative integer value K.

```
1  foreach point_test ∈ D_test do
       /* List of distances between the point_test and points
          from D_train                                          */
2      D := ∅ foreach point_train ∈ D_train do
          /* Calculate distance between points point_test and
             point_train                                        */
3         d := dist(point_test, point_train)
4         D ← D ∪ {d}
5      end
       /* Sort D in non-descending order                        */
6      D ← sort(D)
       /* Choose the first K points from the D and put them
          into the D_K                                          */
7      D_K := D[0..K]
8      assign a class to the point_test based on the majority of classes
        present in the D_K
       /* Assign a class c to the point_text based on the
          majority of classes present in the D_K                */
9      Class(point_test) := c
10 end
```

Therefore, this approach gives more weight to the local properties of the data space, but is computationally expensive and contradicts the statistical philosophy of condensing the properties of large datasets into some numerical features. This approach can be applied to clustering tasks, as well as class labels for classification problems, when a set of distant points is initially selected to be labeled with a different cluster ID. The points are labeled the same as most of the k-nearest neighbors.

A "Region-growing" heuristic method consists of starting with a single new point at a time, labeling it with a new cluster-id, and iteratively labeling all the immediate neighbors of labeled points whose distance (to a labeled point) is less than a threshold, value of which depends on the distances between labeled points in that cluster. At the conclusion of this step, "relaxation" iterations may be carried out, labeling points based on the majority of their k immediate neighbors; these iterations may start at the boundaries of the previously identified clusters, and result in merging some clusters.

2.2.2 k-Means Clustering

This well-known algorithm iteratively computes the center of each current cluster, then updates the same after re-associating each data point with the center of the nearest (current) cluster.

Figure 2.3: K-Means data clustering

This algorithm implicitly identifies each cluster with its centroid, resulting in symmetric clusters such as for example spheres for three-dimensional data. The K-means algorithm can be described by the following pseudocode 2.

Algorithm 2: K-means

Input : dataset $\mathcal{D}$ with data points d.
Initialization: Set the number of clusters k. Initialize k centroids $[\mu_1, \mu_2, ..., \mu_k]$ randomly

while *stopping criterion has not met* **do**
  foreach $d \in \mathcal{D}$ **do**
    `// Calculate distances between point` d `and centroids`
    foreach $i \in [1..k]$ **do**
      $\omega_i := dist(d, \mu_i)$
    end
    `// Find the nearest centroid`
    $m_{min} := m_j$ such that $(w_j == \min_{i \in [1..k]}(w_i))$
    `// Add` d `to the cluster of the nearest centroid`
    $D_{\mu_{min}} \leftarrow D_{\mu_{min}} \cup \{d\}$
  end
  `// Recalculate the position of centroids`
  $\mu_i \leftarrow \mu_{D_i}$
end

Although this is probably the most widely used clustering algorithm, there are many problems for which the k-means clustering approach is unsatisfactory. The first difficulty with this algorithm is the final result depends on the initial choice of the starting points.

An additional difficulty with this algorithm is the determination of the number of clusters (k). One guideline is that the ratio of the intra-cluster distance to the intercluster distance should be small. Some implementations start with a small value of k and work their way up as long as there is significant and measurable progress, e.g., in terms of the above ratio or the *silhouette* measure s:

$$s = 1 - \frac{(distance\ to\ own\ centroid)}{(distance\ to\ next\ nearest\ centroid)}. \tag{2.3}$$

In the "elbow" method, the number of clusters are chosen at the point where the incremental improvement is small.

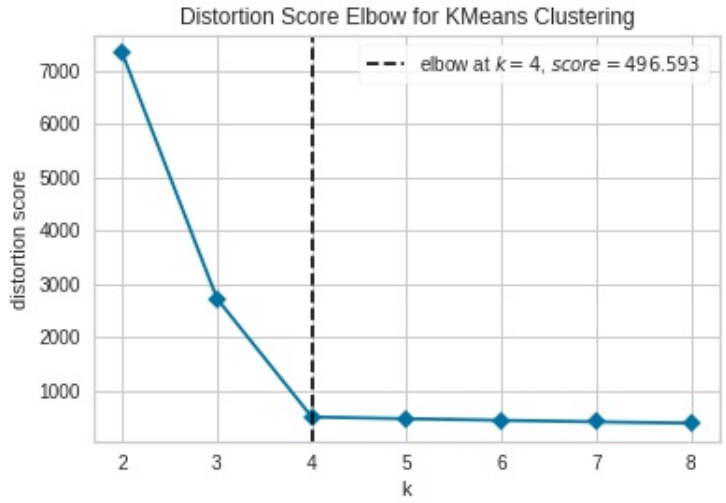

Figure 2.4: Determining the number of clusters using Elbow method

Another approach is based on rate-distortion theory, with an analytical expression being used to determine the amount of data compression that can be achieved. Unfortunately, these latter approaches are computationally expensive and thus impractical for large sets of data.

2.2.3 Agglomerative Clustering

This approach, described in the Agglomerative or Hierarchical clustering algorithm, starts with many sub-clusters, each containing a single element, and successively merges sub-clusters to form larger clusters. Two clusters are candidates for merging if they are closest to each other, for example, based on the distance between the centroids of the clusters. The process may be terminated when the number or size of clusters considered to be is satisfied, or until the next merge results in a single cluster. The result is a tree representation of the objects, called a dendrogram.

Definition 2.2.1. A dendrogram is a diagram that shows the hierarchical relationship between objects.

This algorithm does not need an externally defined parameter for the number of clusters (k), and is deterministic, i.e. giving the same result for the same input data, unlike the k-means algorithm. But this algorithm requires more computational effort and still might need some outside decision to determine how deep we need to traverse each path from the root before declaring a node as representing a genuine cluster.

The merge process is binary by default. At each step, the two nodes are merged. For some problems, a tree with a branching coefficient greater than

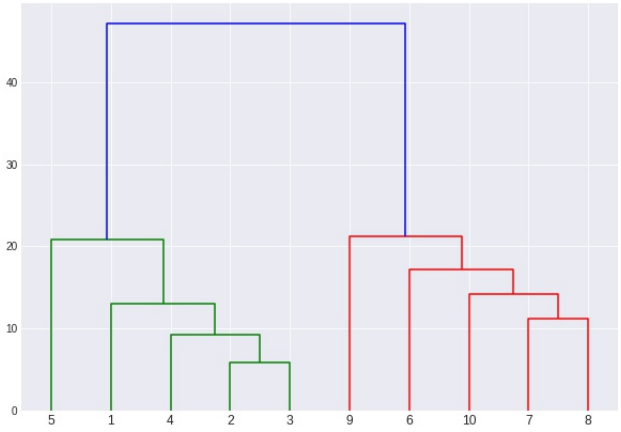

Figure 2.5: Example of a dendrogram

2 may represent the data better. By defining an algorithm variant, multiple nodes can be merged in one iteration.

2.2.4 Density-Based Agglomerative Clustering

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm helps to discover arbitrarily shaped clusters from noisy datasets.

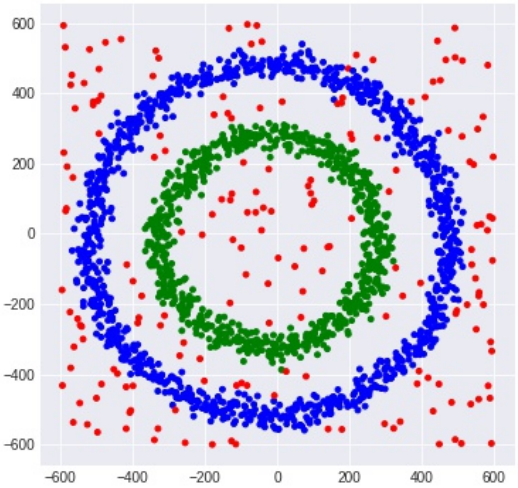

Figure 2.6: Example of DBSCAN data clustering

DBSCAN algorithm observes two parameters a distance threshold and a minimum number of points, based on these parameters points are being grouped together. One can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers. This algorithm establishes accessibility of data points.

Definition 2.2.2. A data point is reachable from another if it lies within a particular predefined distance from it.

Definition 2.2.3. A Core point is a point that has at least m points within distance n from itself.

Definition 2.2.4. A Border point is a point that has at least one Core point at a distance n.

Definition 2.2.5. A Noise point is a point that has less than m points within distance n from itself.

The algorithm picks up a point from the dataset until all points have been processed. If chosen point has at least predefined number of neighbors within a predefined radius, then if can consider the point and all the neigbors with the radius to be associated with a same cluster. Then, the neighborhood calculation is being repeated for each adjacent point with the radius and that's how the clusters are being expanded 3.

Algorithm 3: DBSCAN

Input : Database $\mathcal{D}$, a radius ϵ, density threshold $minPts$ and a distance function $dist$.

Initialization: Initialize point labels $Label(point) := undefined$, for each $point$ from $\mathcal{D}$.

```
1  foreach point ∈ D do
2      if Label(point) ≠ undefined then
3          continue
4      N := {n : n ∈ D ∧ dist(point, n) ≤ ε}
5      if |N| < minPts then
6          Label(point) ← Noise
7          continue
8      c := next cluster label
9      Label(point) ← c
10     S := N \ {point}
11     foreach q ∈ D do
12         if Label(q) == Noise then
13             Label(q) ← c
14         if Label(q) ≠ undefined then
15             continue
16         N := {n : n ∈ D ∧ dist(q, n) ≤ ε}
17         Label(q) ← c
18         if |N| < minPts then
19             continue
20         S ← S ∪ N
21     end
22 end
```

2.3 Time Series Anomaly Detection Using Clusters

As mentioned earlier, some clustering algorithms allow data points to be placed outside the identified cluster. In such cases, points that do not belong to any cluster can be considered anomalous. The outcome of the approach depends on the specified threshold, such as the number of clusters and the minimum number of points required for the cluster. Some algorithms, such as DBSCAN, explicitly apply a minimum cluster size. However, it is not clear how to choose this threshold.

In addition, when choosing an anomaly detection algorithm, you need to consider whether the selected anomaly detection algorithm can be used in time series of big data. That is, it must be taken into account that the

data is generated over time and the algorithm should not be computationally intensive.

2.3.1 Proximity to Nearest Neighbor

Algorithms that rely on the distances to the cluster's centroid have an implicit assumption that clusters are "symmetric", e.g., circular in two dimensions, because two points at the same distance from a given centroid must have the same α values. However, real problems are often characterized by asymmetric clusters, and these asymmetries could not be fixed or removed by any straightforward linear or nonlinear transformations of the data. Thus, the distance between the point and its nearest neighbor then gives a more useful indicator, whether the data point is anomalous. One of such algorithm is k-Nearest Neighbor Clustering that has been described above 1. When detecting outliers, kNN-based methods only focus on whether the NN number reaches the threshold but do not care the changes of NN number with time. However, there may be some cases where the NN number is normal but the distribution is significantly uneven during a time period. This kind of anomaly cannot be detected by only counting NN, so that we need time sensitive anomaly detection algorithm [20].

2.3.2 Proximity to Other Points

A partitioning algorithm, such as the k-means clustering, places every point in some cluster and every such cluster meets the size constraints; i.e., the size of each cluster is $\geq$ the size of the threshold. But some points in the cluster may be so far from all the others points from the cluster and may be identified as anomalous.

Since calculating the sum of distances between any two points in a cluster is computationally expensive, a surrogate is to compute the distance of each point from the cluster's centroid. This approach can also be applied with non-partitioning algorithms, i.e., when some points do not have to belong to any cluster. For this approach data points should be transformed to a one-dimensional scale, e.g., defining a function α such that $\alpha(p) \in \mathbb{R}$ measures the anomalousness of $p \in \mathcal{D}$. We then define

$$\alpha(p) = min_j d(p, \mu_j), \tag{2.4}$$

where $d(p, \mu_j)$ is the distance between data point and μ_j and μ_j is the centroid of cluster C_j, i.e.,

$$\mu_j = \sum_{p_j \epsilon C_j} \frac{p_j}{|C_j|}, \tag{2.5}$$

If $\alpha(p)$ is 'large' p_j is considered to be an anomaly. This approach can be applied to time series data for identifying contextual outliers.

In order to make this approach work well, the number of clusters k must be correctly chosen; k should not be neither very large, nor very small. If k is large, to some extent the difficulty may be overcome if a cluster size threshold is applied to the result. In case k is very small and the intra-cluster distances are too large and if a small increase of the k can significantly decrease the intra-cluster distances, then such an increase should be permitted.

2.3.3 Distances from Multiple Points

If the algorithm considers distances to multiple data points or multiple clusters the results obtained tend to be more robust and susceptible to the "noise" of the process that generated the data. This approach is being used in anomaly detection algorithms based on the density 2.2.4. If the distance value chosen is too low, a significant part of the data will not be clustered. It will be considered outliers because it doesn't satisfy the number of points to create a dense region. From the other point of view, if the value has been chosen too high, clusters will be merged and the majority of objects will be a part of the same cluster. The distance should be chosen based on distances between points in the dataset, but generally small distances are preferable. The DBSCAN algorithm should be used to find associations and interconnections in the data that are hard to find manually, but that can be relevant and be used to find patterns and predict trends in time series.

From the consideration of computational effort, of course, it is best to choose a small set of values over which averages (or medians) are computed. This leads to the common pragmatic choice of $k = 3$ (*or* 5) used by many practitioners, who reason that the additional advantage gained by using larger values of k is not worth the additional computational effort.

In problems where the computational effort considerations are not critical, however, the choice of k requires greater attention.

2.3.4 Section summary

A large number of anomaly detection algorithms discussed in this chapter are based on clustering the given data; such an algorithm declares anomalies to be the data points from the outside of the clusters, or are near the boundaries of clusters. Although many practitioners restrict themselves to one or two popular algorithms, there are many cases when the other algorithms discussed here should have been preferred, e.g., when the clusters in a dataset are not symmetric, or if densities vary across different regions of data space.

Many clustering algorithms for anomaly detection require the number of clusters to be predefined. Even though there are many techniques of estimating the number of clusters, but when we are dealing with time series data, it is not feasible to dynamically estimate the number of clusters for each series. Summarizing all pros and cons of algorithms that have been described above,

density-based anomaly detection algorithm like DBSCAN can be picked up as the most suitable for identifying anomalies in time series of electric vehicle. DBSCAN does not group all data points to a cluster, thus, outliers are not being associated with any cluster.

Chapter 3

Artificial Neural Network Approaches

3.1 Artificial Neural Networks

Building a machine or autonomous mechanism endowed with intelligence is an ancient dream of researchers from the diverse areas of sciences and engineering. Although the first articles about Artificial Neural Networks (ANN) were published more than 50 years ago, this subject began to be deeply researched on the early 90s, and still have an enormous research potential. The applications involving systems considered intelligent cover a wide range, including: Analysis of images acquired from artificial satellites, speech and writing pattern classification, face recognition with computer vision, control of high-speed trains, stocks forecasting on financial market, anomaly identification on medical images, automatic identification of credit profiles for clients of financial institutions, control of electronic devices and appliances, such as washing machines, microwave ovens, freezers, coffee machines, frying machines, video cameras, and so on [21].

The reason, why artificial neural networks is such a good tool for solving diverse problems, is their ability to map nonlinear systems and learn the underlying behaviors and internal dependencies just from data received from such systems. Deep learning for time series is a relatively new endeavor, but it's a promising one. Because deep learning is a highly flexible technique, it can be advantageous for time series analysis. Most promisingly, it offers the possibility of modeling highly complex and nonlinear temporal behavior without having to guess at functional forms, which could potentially be a game changer for nonstatistical forecasting techniques. There are many relevant features related to artificial neural networks applications. Internal parameters of the network(usually synaptic weights) are being adjusted by examining a series of successive examples(patterns, samples, or measurements) related to the behavior of the process, this enabling the learning by experience. Learn-

ing methods allow networks to extract existing relationships between various variables in an application. Knowledge of the behavior of specific processes learned by neural networks is stored at each of multiple synapses between artificial neurons, improving the robustness of the architecture in case of some neurons are lost. Depending on the details of your application, most neural architectures can be easily prototyped in hardware or software, since some fundamental mathematical operations is enough to acquire results.

3.1.1 Potential Application Areas

Deep learning for time series is a relatively new endeavor, but it's a promising one. Because deep learning is a highly flexible technique, it can be advantageous for time series analysis. Most promisingly, it offers the possibility of modeling highly complex and nonlinear temporal behavior without having to guess at functional forms, which could potentially be a game changer for nonstatistical forecasting techniques [18].

Artificial neural networks are computational models inspired by the nervous system of living beings. They have the ability to acquire and maintain knowledge (information based) and can be defined as a set of processing units, represented by artificial neurons, interlinked by a lot of interconnections (artificial synapses), implemented by vectors and matrices of synaptic weights [22].

Artificial neural networks can be used to solve several engineering and science problems. The potential application areas can be split into several categories.

The first category is functions approximation of universal curve fitting. The objective is to map the functional relationship between variables (usually real numbers) of a particular system from a known set of meaningful values. These applications are as diverse as possible, and often involve mapping processes that are difficult to model using traditional methods.

The second one is called process control. This category consists of identifying control actions capable of meeting quality, efficiency, and security requirements. Among the multiple available applications, neural controllers are of particular interest to robotics, airplanes, elevators, appliances, satellites, and so on.

The next category is a pattern recognition or pattern classification. The goal is to associate a given input pattern (sample) with one of the known classes, e.g. image, speech and writing recognition. In this case thus, the problem's outcome has a discrete and known set of possible values.

Artificial neural networks also can be used for data clustering. In this case, the purpose is to detect and identify similarities and particularities of the several input patterns, and group them together. Some examples, to cite a few, are applications involving automatic class identification and data mining.

Artificial neural networks can be also used to build prediction systems. The goal is to determine future values of a particular process by observation of previous samples. Among the known applications, it is possible to find systems for time series prediction, stock market projection, weather forecast, and so on.

The last but not least category is associative memory. The task is to restore a pattern even when its inner elements are uncertain or inaccurate. Some examples include image processing, signal transmission, written character identification, and so forth.

3.1.2 Artificial Neuron

The artificial neural network structures were developed from known models of biological nervous systems and the human brain itself. The computational components or processing units, called artificial neurons, are simplified models of biological neurons.

The artificial neurons used in artificial neural networks are nonlinear, usually providing continuous outputs, and performing simple functions, such as gathering signals available on their inputs, assembling them according to their operational functions, and producing a response considering their innate activation functions [21].

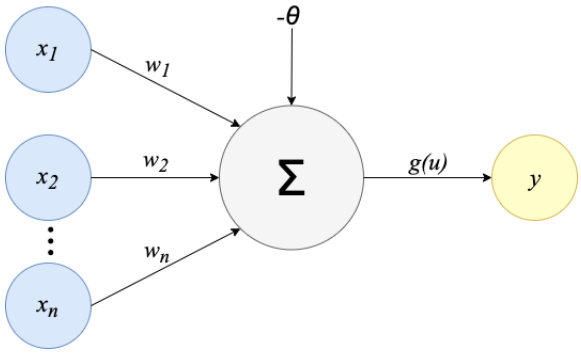

Figure 3.1: The artificial neuron structure

Artificial neuron is composed of seven basic elements.

Definition 3.1.1. Input signals $(x_1, x_2, ..., x_n)$ are the signals or samples coming from the external environment and representing the values assumed by the variables of a particular application. The input signals are usually normalized in order to enhance the computational efficiency of learning algorithms.

Definition 3.1.2. Synaptic weights $(w_1, w_2, ..., w_n)$ are the values used to weight each one of the input variables, which enables the quantification of their relevance with respect to the functionality of the neuron.

Definition 3.1.3. Linear aggregator ($\sum$) gathers all input signals weighted by the synaptic weights to produce an activation voltage.

Definition 3.1.4. Activation threshold or bias (θ) is a variable used to specify the proper threshold that the result produced by the linear aggregator should have to generate a trigger value toward the neuron output.

Definition 3.1.5. Activation potential (u) is the result produced by the difference between the linear aggregator and the activation threshold. If this value is positive, i.e. if $u \geq \theta$, then the neuron produces an excitatory potential; otherwise, it will be inhibitory.

Definition 3.1.6. Activation function (g) whose goal is limiting the neuron output within a reasonable range of values, assumed by its own functional image.

Definition 3.1.7. Output signal (y) consists on the final value produced by the neuron given a particular set of input signals, and can also be used as input for other sequentially interconnected neuron.

The weighing carried out by the synaptic junctions of the network are implemented on the artificial neuron as a set of synaptic weights $\{w_1, w_2, ..., w_n\}$. Analogously, the relevance of each of the $\{x_i\}$ neuron inputs is calculated by multiplying them by their corresponding synaptic weight $\{w_i\}$, thus weighting all the external information arriving to the neuron. Therefore, it is possible to verify that the output of the artificial cellular body, denoted by u, is the weighted sum of its inputs.

The two following expressions synthesize the result produced by the artificial neuron proposed by McCulloch and Pitts [23]:

$$u = \sum_{i-1}^{n} \omega_i \cdot x_i - \theta, \tag{3.1}$$

$$y = g(u). \tag{3.2}$$

Thus, the artificial neuron operation can be summarized by the following steps:

1. Present a set of values to the neuron, representing the input variables.
2. Multiply each input of the neuron to its corresponding synaptic weight.
3. Obtain the activation potential produced by the weighted sum of the input signals and subtract the activation threshold.
4. Applying a proper activation function to limit the neuron output.

5. Compile the output by employing the neural activation function in the activation potential.

Activation functions are a critical part of the design of a neural network. The choice of activation function in the hidden layer will control how well the network model learns the training dataset. The choice of activation function in the output layer will define the type of predictions the model can make. As such, a careful choice of activation function must be made for each deep learning neural network project [24].

The activation functions can be categorized into two fundamental groups, partially differentiable functions, and fully differentiable functions, when considering their complete definition domains.

3.1.2.1 Partially Differentiable Activation Functions

Partially differentiable activation functions are functions with points whose first order derivatives are nonexisting. The three main functions of this category are the following: step function 3.1.8, bipolar step function 3.1.9, and symmetric ramp function 3.1.10.

Definition 3.1.8. Results produced by the binary sted function assume a unitary positive value when the activation potential of the neuron is above zero. Otherwise, the result is null. This function is defined by next equation:

$$g(u) = \begin{cases} 1, if\ u \geq 0, \\ 0, if\ u < 0. \end{cases} \tag{3.3}$$

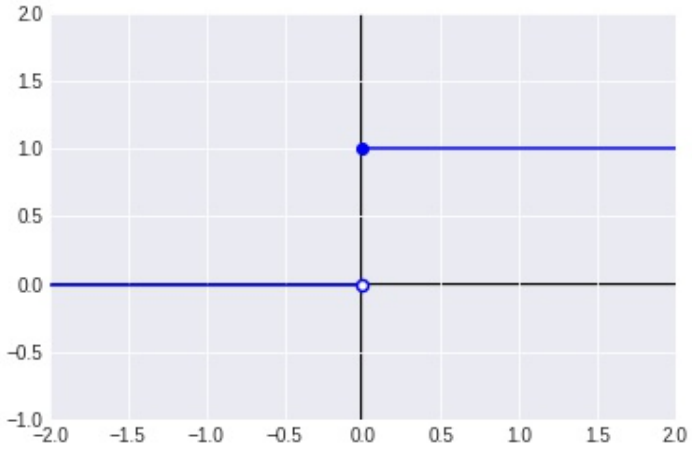

Figure 3.2: Binary step activation function

Definition 3.1.9. Signal function is described as follows:

$$g(u) = \begin{cases} 1, if\ u > 0, \\ 0, if\ u = 0, \\ -1, if\ u < 0. \end{cases} \tag{3.4}$$

The result produced by Bipolar step function or Signal function will assume unitary positive values when the neuron activation potential is above zero; null value, if the potential is null; and negative unitary values otherwise.

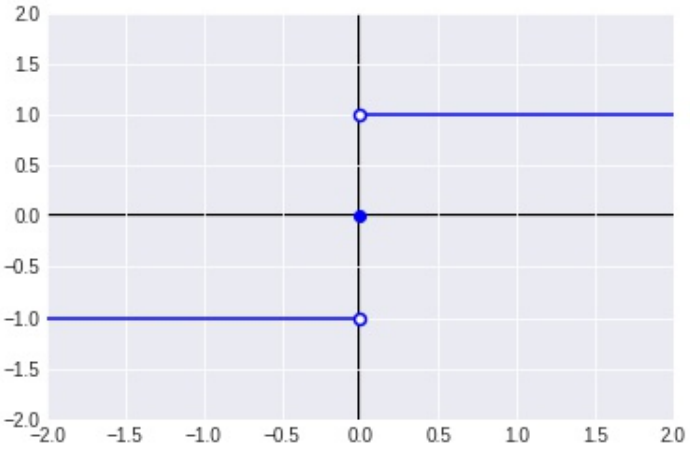

Figure 3.3: Bipolar step activation function

Definition 3.1.10. The mathematical notation of the Symmetric ramp is:

$$g(u) = \begin{cases} a, if\ u > a, \\ 0, if\ -a \leq u \leq a, \\ -a, if\ u < a. \end{cases} \tag{3.5}$$

Symmetric ramp functions return the value of the potential itself, if the value is within the interval $[-a, a]$, and limits the value otherwise.

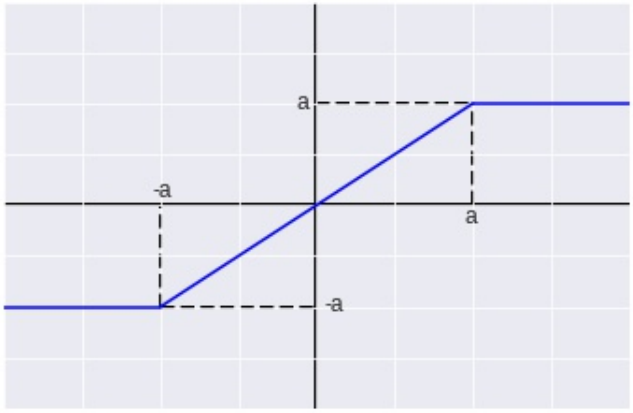

Figure 3.4: Symmetric ramp activation function

Definition 3.1.11. The mathematical notation of the Rectified Linear Unit (ReLU) is:

$$g(u) = \begin{cases} 0, if\ u < 0, \\ u, if\ u \geq 0. \end{cases} \tag{3.6}$$

ReLU is the most commonly used activation function in neural networks. It's cheap to compute because there's no complicated math. As a result, the model may take less time to train or run. Since ReLU is zero for all negative inputs, there is a chance that a certain unit will not trigger.

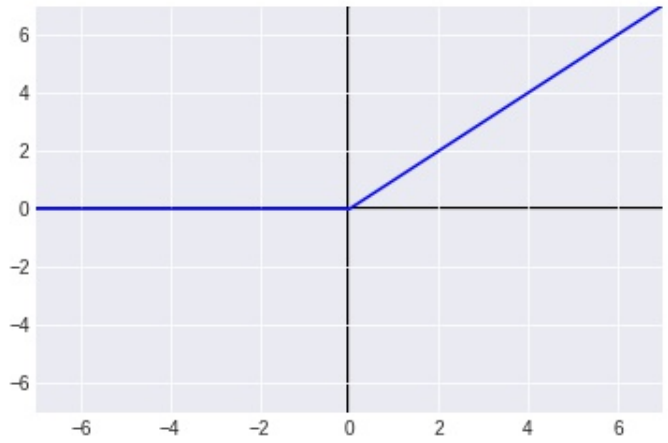

Figure 3.5: Rectified Linear Unit (ReLU) activation function

3.1.2.2 Fully Differentiable Activation Functions

Fully differentiable activation functions are those whose first order derivatives exist for all points of their definition domain. The four main functions of this category, which can be employed on artificial neural networks, are the sigmoid function 3.1.12, hyperbolic tangent 3.1.13, Gaussian function 3.1.14 and linear function 3.1.15.

Definition 3.1.12. Sigmoid function mathematical expression is given by:

$$g(u) = \frac{1}{1 + e^{-u}}, \tag{3.7}$$

The result produced by this functions is always a real number within a range $[0, 1]$.

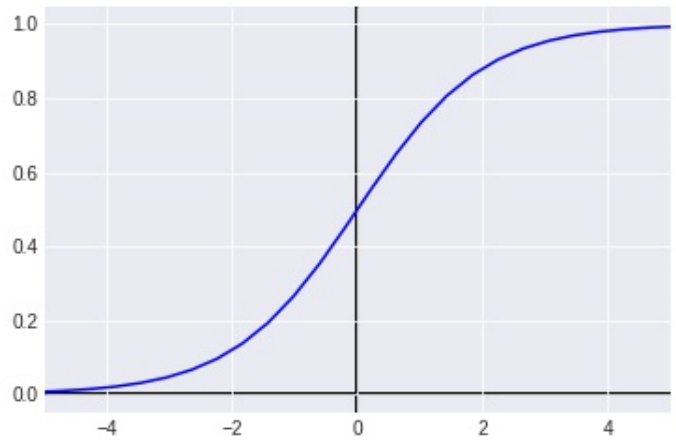

Figure 3.6: Sigmoid activation function

Definition 3.1.13. Hyperbolic tangent function is expressed with the following mathematical expression:

$$g(u) = \frac{1 - e^{-\beta \cdot u}}{1 + e^{-\beta \cdot u}}, \tag{3.8}$$

where β is associated with the slope of the hyperbolic tangent function in its inflection point. The Hyperbolic tangent function, unlike the logistic function, will output real values within an interval $[-1, 1]$.

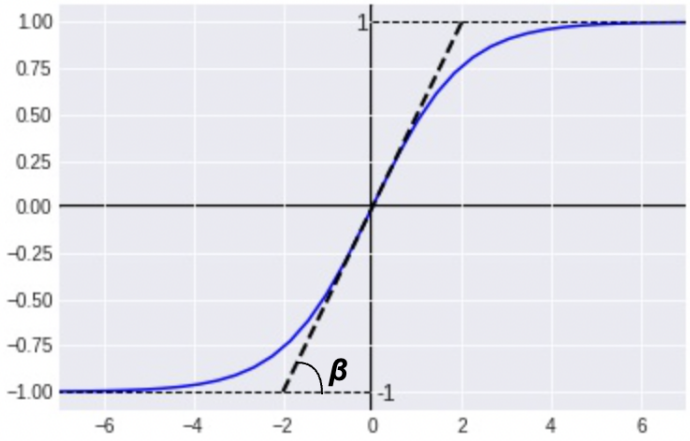

Figure 3.7: Hyperbolic tangent activation function

Definition 3.1.14. Gaussian activation function is given by:

$$g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}}, \tag{3.9}$$

where c is the parameter that defines the center of the Gaussian function and $\{\sigma\}$ denotes the associated standard deviation, that is, how scattered (dispersed) is the curve in relation to its center. In the case of Gaussian activation functions , the neuron output will produce equal results for those activation potential values $\{u\}$ placed at the same distance from its center (average). The curve is symmetric to this center.

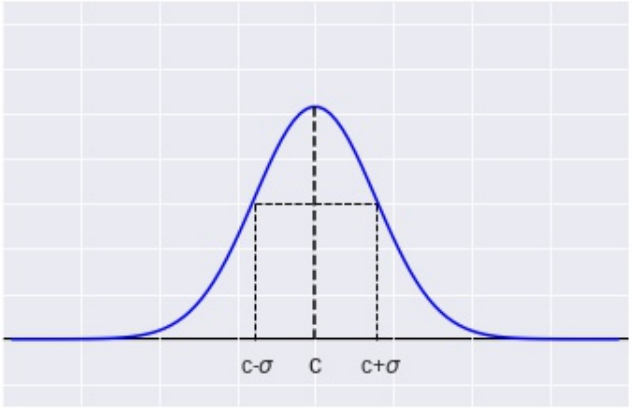

Figure 3.8: Symmetric ramp activation function

Definition 3.1.15. The linear activation function, or identity function, produces output results equal to the activation potential $\{u\}$, having its mathematical expression given by:

$$g(u) = u. \tag{3.10}$$

One application of the linear activation functions is in artificial neural networks performing universal curve fitting (function approximation), to map the behavior of the input/output variables of a particular process.

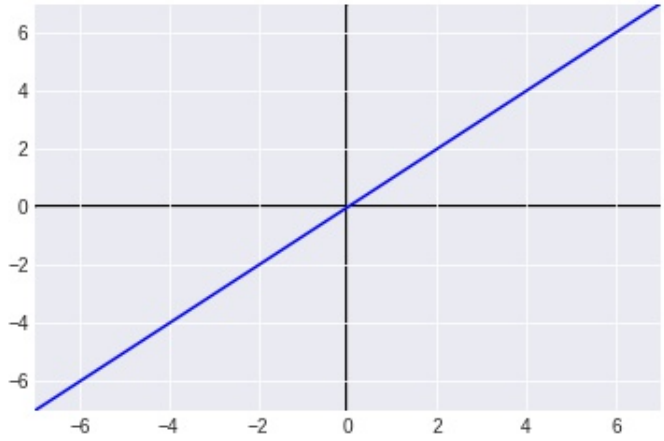

Figure 3.9: Symmetric ramp activation function

3.1.3 Artificial Neural Networks Architectures

The goal of the artificial neural network architecture is to arrange its neurons in realtion to each other. These arrangements are made essentially by directing the neurons synaptic connections.

The topology of a particular neural network within a particular architecture can be defined in terms of various possible structural configurations. That is, you can use two topologies that belong to the same architecture. The first topology consists of 10 neurons and the second topology consists of 20 neurons. Also, one may be composed of neurons having a logistic activation function, and the other may be composed of neurons having a hyperbolic tangent as an activation function.

Training on a particular architecture, on the other hand, involves applying a series of ordered steps to adjust neuron weights and thresholds. Therefore, such a tuning process, also known as a learning algorithm, aims to tune the network so that the output is close to the desired value.

In general, artificial neural network can be split into three parts: the input layer, the hidden (intermediate or invisible) layers, and the layer called the output layer.

The input layer is responsible for receiving information (data), signals, functions, or measurements from the external environment. These inputs (samples or patterns) are usually normalized within the limits created by the activation function. This normalization improves the numerical accuracy of the mathematical operations performed by the network.

The second part, called hidden layers, is composed of neurons which are responsible for extracting patterns related to the process or analyzed system.

These layers perform most of the internal processing from a network.

Because the output layer is also made up of neurons, it is responsible for generating and presenting the final network output that results from the processing performed by the neurons in the previous layer.

The main architectures of artificial neural networks are single-layer feedforward network and multilayer feedforward networks.

3.1.3.1 Single-Layer Feedforward Architectures

It is a network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers. The connection has different weights upon them. There is no feedback loop means the signal can only flow in one direction, from input to output [25]. A single-layer feedforward architecture is the simplest ANN architecture. In this type of network, there are only two layers input layer and output layer however the input layer isn't being counted due to the fact no computation is executed in this layer.

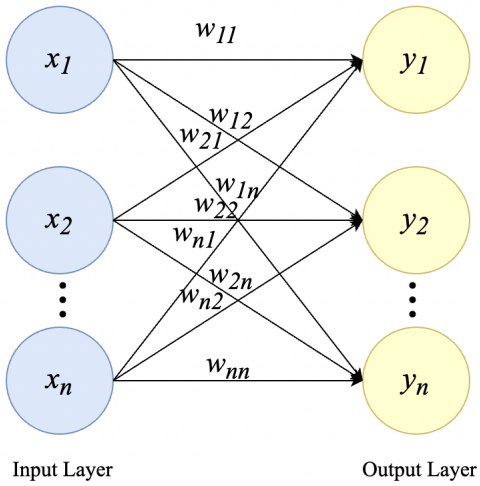

Figure 3.10: Single-Layer Feedforward Architecture

3.1.3.2 Multiple-Layer Feedforward Architectures

A multi-layer feedforward network consists of one or more hidden neural layers. They are used to solve various problems such as function approximation, pattern classification, system identification, process control, optimization, and robotics.

The main networks that use the multi-layer feedforward architecture are the multi-layer perceptron (MLP) and the radial basis function (RBF), and the learning algorithms used in the training process are respectively based on the generalized delta rule and the competitive/delta rule.

The number of neurons that make up the first hidden layer is usually different from the number of signals that make up the input layer of the network. In fact, the number of hidden layers and the set of each neuron depends on the nature and complexity of the problem mapped by the network, and the quantity and quality of data available about the problem. Nevertheless, even in a single-layer feedforward network, the set of output signals always corresponds to the number of neurons from each layer.

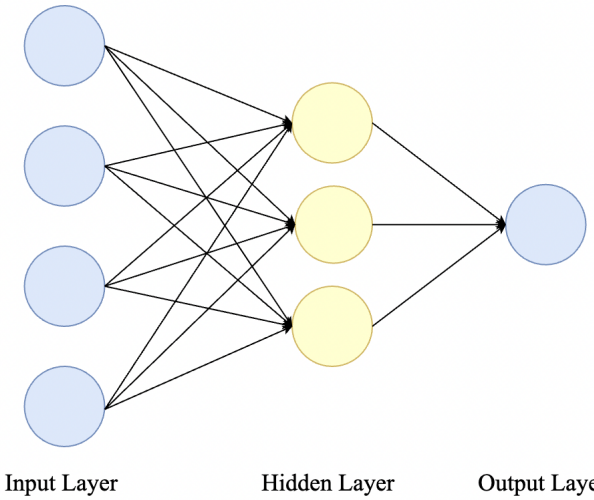

Figure 3.11: Example of Multiple-Layer Feedforward Architecture

Feed forward networks are good tests for whether there really are complex time-axis dynamics in your sequence. Not all time series really are time series in the sense of having time-axis dynamics, where earlier values have a specific relationship to later values. It can be good to fit a feed forward neural network as one baseline, apart from a simpler linear model [18].

3.1.4 Training Processes and Properties of Learning

One of the most important functions of artificial neural networks is their ability to learn from representations of patterns (templates) that describe the behavior of a system. So, once the network learns the relationship between inputs and outputs, it can generalize the solution. That is, the network can produce an output that is close to the expected output for a given input value.

In means that the training process of a neural network consists of applying the required ordinated steps for tuning the synaptic weights 3.1.2 and thresholds 3.1.4 of its neurons, in order to generalize the solutions produced by its outputs.

Definition 3.1.16. The calculation of the output from a neural network by propagating an input signal through each layer until the output layer outputs its values is called forward-propagation.

In forward-propagation phase predictions are generated during training process. These predictions need to be corrected by backpropagation.

Definition 3.1.17. The learning method adopted for training multi-layer feedforward networks is called backpropagation.

The backpropagation algorithm can be applied to multi-layer feedforward networks. This is a supervised learning algorithm that continuously adjusts the weights of connected neurons to reduce the difference between the output signal and the target output.

In this method, the deviation (error) between the output value of the output layer and the expected value is propagated from the output layer back to the previous layer.

Definition 3.1.18. Each iteration when the whole dataset go forward and backward through the neural network is called epoch.

The count of epochs is a hyperparameter that defines the number of iterations which consist of forward phase and backward phase.

3.1.4.1 Supervised Learning

A supervised learning strategy is to achieve the desired result for a given set of inputs, hence each training sample consists of an input signal and a corresponding output signal.

Therefore, the application of supervised learning only depends on the availability of that corresponding output value, and it behaves as it learns with the coach what is the correct response for each sample presented for its input.

Usually, the complete set containing all available samples of the system behavior is divided into two subsets, which are called training subset and test subset. The training subset, composed of 60–90 % of random samples from the complete set, will be used essentially in the learning process. On the other hand, the test subset, which is composed of 10–40 % from the complete sample set, which will contain 30-50% of validation data, will be used to verify if the network capabilities of generalizing solutions are within acceptable levels, thus allowing the validation of a given topology. Nonetheless, when dimensioning these subsets, statistical features of the data must also be considered.

3.1.4.2 Unsupervised Learning

Unlike supervised learning, applying an unsupervised learning algorithm does not require any knowledge of the desired outcome.

Therefore, networks must self-organize by identifying clusters that represent similarities if there is an existing feature between the elements that make up the entire sample set. The learning algorithm adjusts the synaptic weights and thresholds of the network to reflect these clusters in the network itself.

Alternatively, the network designer can specify the maximum number of such possible clusters using their knowledge of the problem.

3.1.5 Autoencoders

An autoencoder is a special type of neural network that copies an input value to an output value. It does not require target variables like traditional Y, so it is classified as unsupervised learning.

The number of neurons in the hidden layer is less than the number of input layers. The hidden layer extracts important information from the input values. This condition causes the hidden layer to learn most of the data patterns and ignore "noise". Therefore, in the autoencoder model, the hidden layer dimension must be less than the input or output layer. If the number of neurons in the hidden layer is greater than the number of input layers, the neural network has too much capacity to train the data. In extreme cases, you can simply copy the input to a noisy output value without extracting important information.

Definition 3.1.19. Encoder compresses the input data into an encoded low-dimensional representation that is typically several orders of magnitude smaller than the input data.

Definition 3.1.20. Bottleneck is a middle layer that contains the compressed knowledge representations. The bottleneck exists to restrict the flow of information to the decoder from the encoder, thereby allowing only the most important information to pass through.

Definition 3.1.21. Decoder is a feedforward network with the same structure as the encoder, it decompresses the knowledge representations and reconstructs the data back from its encoded representation. The output data is then compared with a ground truth (input data). When it uncompresses, it tries to reach close to input, but the output is not the same.

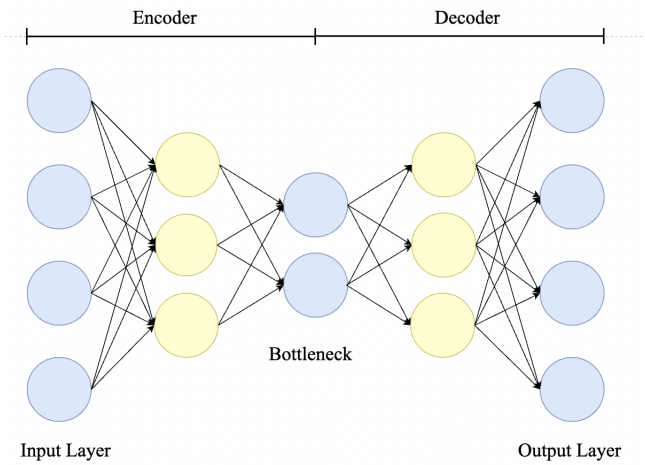

Figure 3.12: Example of Autoencoder Architecture

3.1.5.1 Applications of Autoencoders

Autoencoders can be employed to solve many kinds of problems. The most popular of them is the problem of dimensionality reduction. The information from the input has been compressed and stored in the bottleneck layer. Each node can now be treated as a variable. Thus, by removing the decoder part, an autoencoder with the bottleneck layer as output can be used for dimensionality reduction.

Once the main pattern is identified, the outliers become apparent. Many distance-based techniques (such as KNN) suffer from the curse of dimensionality when calculating the distance of each data point across feature space. It is necessary to reduce the high dimensionality. Interestingly, outliers are identified during the dimensionality reduction process. Outlier detection is a by-product of dimensionality reduction.

Another application of autoencoder is feature extraction, which is basically connected to dimensionality reduction. The encoding part of the autoencoder helps to extract hidden features presented in the input data.

Definition 3.1.22. Reconstruction error is the distance between the original data point and its projection onto a lower-dimensional subspace.

It learns with the reconstruction function that works with normal data. Thus, the reconstruction error of normal data is low, and the reconstruction error of abnormal data is high.

There are already many useful tools like Principal Component Analysis (PCA) for detecting outliers. Remember that PCA uses linear algebra to transform. In contrast, autoencoder technology with a non-linear activation function and multiple layers can perform non-linear transformations. It is

more efficient to train multiple layers using an autoencoder than to train a huge transformation using a PCA. Therefore, autocoder techniques show their benefits when data problems are complex and non-linear.

Chapter 4

Realisation

This chapter is dedicated to the practical application and comparison of researched methods. Due to conclusion from Chapter 2, the first selected model that is trained is DBSCAN 3. The second model is Autoencoder 3.1.5.

Since dataset with charging cycles of electric vehicles cannot be published, the example of training process will be done on open-source dataset. Results and evaluation of the training process will be done for both open-source dataset and the dataset which is gathered for the EV replicating charging cycles profiles of batteries.

The computation was done on the CPU Intel (R) Xeon(R) 2.20 GHz with $x86\_64$ ISO architecture. The processor supports both 32-bit and 64-bit op-modes. This is a single-core processor with two threads. Additional hardware accelerators such as GPU weren't used. The size of RAM is 13 GB.

4.1 Used tools

Python was selected as the programming language for the implementation mostly because it is the most popular language for machine learning, thus it has a lot of dedicated frameworks and libraries. Such libraries and frameworks were used:

- **Pandas:** data manipulation control
- **NumPy:** data manipulation and high-level math functions
- **Matplotlib:** data visualization
- **Seaborn:** data visualization
- **Scikit-learn:** machine learning
- **TensorFlow:** machine learning and artificial intelligence

4.2 Datasets

4.2.1 ECG Dataset

The following dataset is used as a benchmark for experiments.

The original dataset for "ECG5000" is a 20-hour long ECG downloaded from Physionet. The data was pre-processed in two steps: (1) extract each heartbeat, (2) make each heartbeat equal length using interpolation. This dataset was originally used in paper "A general framework for never-ending learning from time series streams", DAMI 29(6). After that, 5,000 heartbeats were randomly selected. The patient has severe congestive heart failure and the class values were obtained by automated annotation [26].

ECG (electrocardiogram) is a test that measures the electrical activity of the heartbeat. There are 5000 rows in the dataset. Each row corresponds to one electrocardiogram. In this experiment is used all normal cardiograms (2919) and 200 anomalous cardiograms to imbalance dataset for making data more suitable for anomaly detection. Dataset has 141 columns. One electrocardiogram contains of 140 sensor signals by time. The first column contains label of the electrocardiogram:

- $label == 1$: electrocardiogram is normal
- $label > 1$: electrocardiogram is abnormal

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	...
0	1.0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522	...
1	1.0	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680	...
2	1.0	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580	...
3	1.0	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884	...
4	1.0	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450	...

Figure 4.1: Example of data in ECG dataset

4.2.2 Charging Cycles Dataset

BMS signals (see Chapter 1) contain a significant amount of data. For analyzing charging cycles total voltage is extracted. Thus, each charging cycle will be presented as the value of voltage over time.

Synthetic charging cycles were generated and labeled automatically for this experiment to enable testing of the algorithms for anomaly detection. Data are labeled as normal (1080 cycles) and anomalous (77 cycles) charging cycles.

For data synthesis Synthetic Minority Over Sampling Technique (SMOTE) was used. For SMOTE a new synthetic sample is created by selecting a random point in feature-space along a line intersecting k randomly chosen samples of the same class [27]. SMOTE has the advantage of not creating duplicate data

points, but rather synthetic data points that differ slightly from the original data points. SMOTE is a superior oversampling option [28].

The SMOTE algorithm works like this:

1. Select a random sample from the minority group.
2. Determine the k nearest neighbours for the observations in this sample.
3. Then, using one of those neighbours, determine the vector between the current data point and the chosen neighbour.
4. The vector is multiplied by a random number between 0 and 1.
5. Add this to the current data point to get the synthetic data point.

This operation is essentially the same as moving the data point slightly in the direction of its neighbour. This ensures that your synthetic data point is not an exact replica of an existing data point, while also ensuring that it is not too dissimilar from known observations in your minority class [28].

Since each charging cycle length is different, data were normalized and unified to length of 250 data points. Here is the example of normal and anomalous charging cycle of the synthetic data 4.2. The increasing of voltage in non-anomalous charging cycles is smooth, while anomalous charging cycles have oscillations.

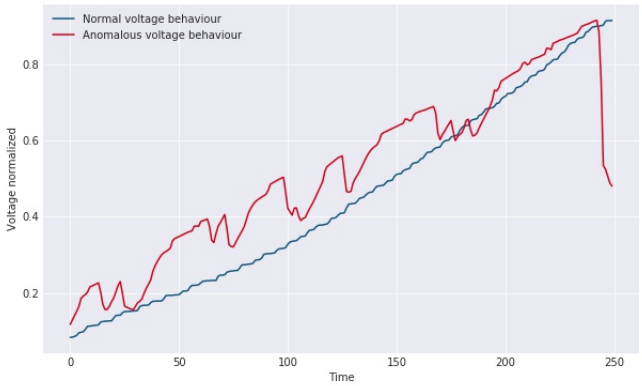

Figure 4.2: Example of normal and anomalous charging cycles

4.3 Applying Techniques

In this section, selected techniques are applied for anomaly detection, then they are compared and results are analyzed. The first step that should be done is defining metrics that is used for comparing results.

4.3.1 Introduction of Used Metrics

The main reason for training these models is to correctly recognize anomalous events. Thus, the confusion matrix and F1 score were selected as suitable metrics for the given use case.

Definition 4.3.1. Confusion matrix for binary anomaly classification is defined as follows:

		Predicted Class	
		Negative (Normal)	Positive (Anomalous)
Actual Class	Negative (Normal)	TN	FP
	Positive (Anomalous)	FN	TP

Table 4.1: Confusion matrix

Values TP, FP, TN and FN correspond to:

True Negative (TN) Actual event is normal and predicted as normal

False Positive (FP) Actual event is normal, but predicted as anomalous

False Negative (FN) Actual event is anomalous, but predicted as normal

True Positive (TP) Actual event is anomalous, and predicted as anomalous

Definition 4.3.2. The F1 score is defined as the harmonic mean of precision and recall [29]. F1 score can be describes with next equation:

$$\frac{TP}{TP + \frac{FN+FP}{2}} \tag{4.1}$$

Definition 4.3.3. The precision is described with the next equation:

$$\frac{TP}{TP + FP} \tag{4.2}$$

Definition 4.3.4. The recall is described with the following equation:

$$\frac{TP}{TP + FN} \tag{4.3}$$

4.3.2 DBSCAN

DBSCAN has two parameters that need to be defined: *min_samples*, which refers to minimum samples and *eps*, which refers to maximum distance between two samples 3.

Definition 4.3.5. The dimension of a dataset corresponds to the number of attributes/features that exist in a dataset. A dataset with a large number of attributes, generally of the order of a hundred or more, is referred to as high dimensional data [30].

The *min_samples* value cannot be determined automatically for DBSCAN. Nonetheless, there is an implicit rule that if the $dimension\ of\ a\ dataset \geq 2$, then *min_samples* should be chosen as

$$min\_samples = 2 \cdot (dimension\ of\ a\ dataset). \tag{4.4}$$

Hence, *min_samples* value for ECG dataset is set to 280.

One technique for choosing the value of *eps*, which is described in the following paper [31], is to calculate the average distance between each two point and its k-nearest neighbors, where $k = min\_samples$. The average k-distances are then plotted in ascending order. The optimal value for *eps* is located at the point of maximum curvature. In this case 4.3 the value of *eps* is around 0.7 .

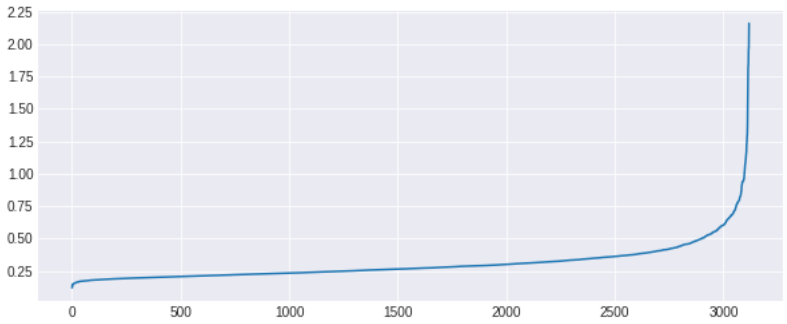

Figure 4.3: The curve of k-distances (ECG dataset)

min_samples	280
eps	0.7

Table 4.2: DBSCAN hyperparameters (ECG dataset)

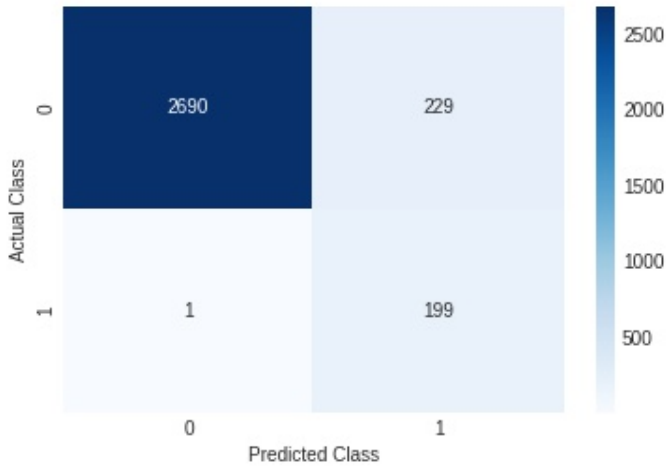

Figure 4.4: Confusion matrix for DBSCAN results (ECG dataset)

F1 score	Precision	Recall
0.63376	0.46495	0.99500

Table 4.3: DBSCAN metrics values (ECG dataset)

Looking at metrics, and confusion matrix 4.4, one can conclude that this model proved to produce accurate prediction on the ECG dataset and thus it can also be used to the charging cycles dataset.

min_samples	500
eps	0.9

Table 4.4: DBSCAN hyperparameters (Charging cycles dataset)

Results of the model trained on charging cycles data are presented in 4.5 and 4.5.

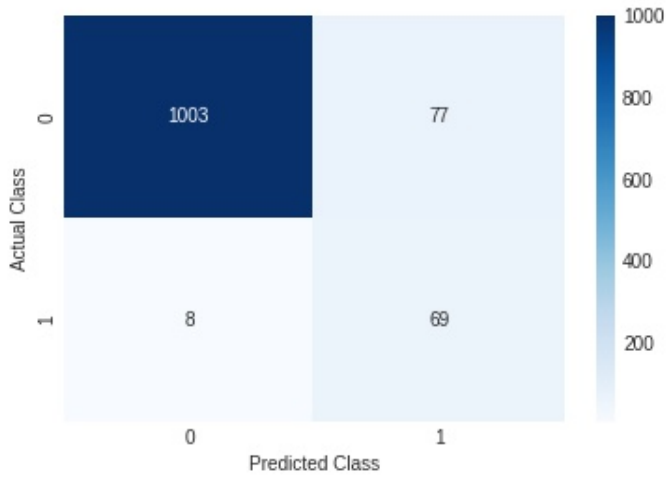

Figure 4.5: Confusion matrix for DBSCAN results (Charging cycles dataset)

F1 score	Precision	Recall
0.61883	0.47260	0.89610

Table 4.5: DBSCAN metrics values (Charging cycles dataset)

DBSCAN performs worse on the charging cycles dataset. One of the reason is that DBSCAN struggles with the curse of dimensionality. The curse of dimensionality is the problem caused by the exponential increase in volume associated with adding extra dimensions to Euclidean space.

The curse of dimensionality basically means that the error increases with the increase in the number of features. It refers to the fact that algorithms are harder to design in high dimensions and often have a running time exponential in the dimensions. A higher number of dimensions theoretically allow more information to be stored, but practically it rarely helps due to the higher possibility of noise and redundancy in the real-world data [32].

4.3.2.1 Application for Big Data

DBSCAN is one of the most efficient clustering algorithms that can efficiently identify noise. However, its complexity makes it unfeasible and not scalable for applications involving big data clustering. DBSCAN has a worst-case runtime complexity of $\mathcal{O}(n^2)$. A large part of DBSCAN's computation time is occupied by finding neighbours, which becomes the bottleneck for its performance. Moreover, as data increase, new clusters can be formed. However, trained model may identify new normal clusters as a noise. For this use case the best choice is the model, that will learn and improve prediction with the increase of the data. One of such models is neural network based Autoencoder.

4.3.3 Autoencoder

Firstly, the concrete architecture of the Autoencoder used for this application needs to be defined.

All the layers use the ReLU activation function 3.6, as it is the standard with deep neural networks. The last layer uses the sigmoid activation 3.7 because we need the outputs to be between [0, 1]. The input is also in the same range.

The number of layers and neurons in each layer was determined experimentally. The number of neurons in the last layer is equal to the dimension of a dataset, so the comparison of ground truth and prediction can be done.

# of layers	8
# of neurons in each layer	$[16, 8, 4, 2, 4, 8, 16, 140]$
Loss function	Mean Absolute Error (MAE)
Optimizer	Adam
# of epochs	15

Table 4.6: Autoencoder hyperparameters (ECG dataset)

In the section 3.1.5.1 there was discussed that the reconstruction error will be greater for anomalous data, since most of data are not anomalous. For identifying outliers the threshold for the reconstruction error has to defined. Looking at the histogram of reconstruction errors 4.6, one can mention that there are two peaks around 0.025 and 0.11. The threshold can be located in the place where the first distribution ends and the second one starts, e.g. in this case the threshold was set to 0.08.

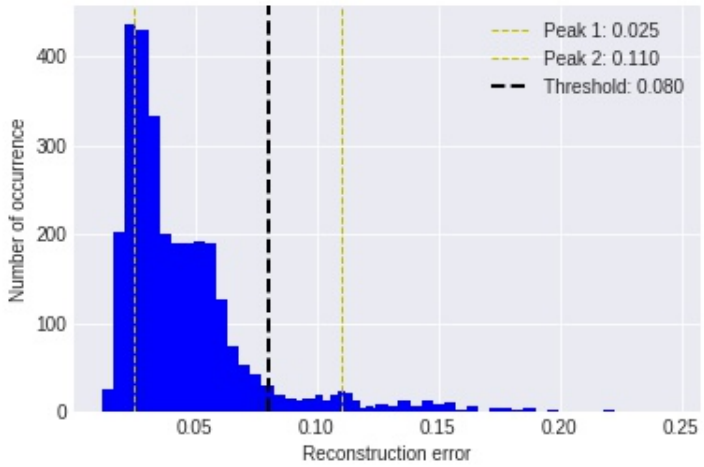

Figure 4.6: The histogram of reconstruction errors (ECG dataset)

Electrocardiograms that have the reconstruction error greater than a threshold are labeled as anomalous. In this case the confusion matrix is following 4.7:

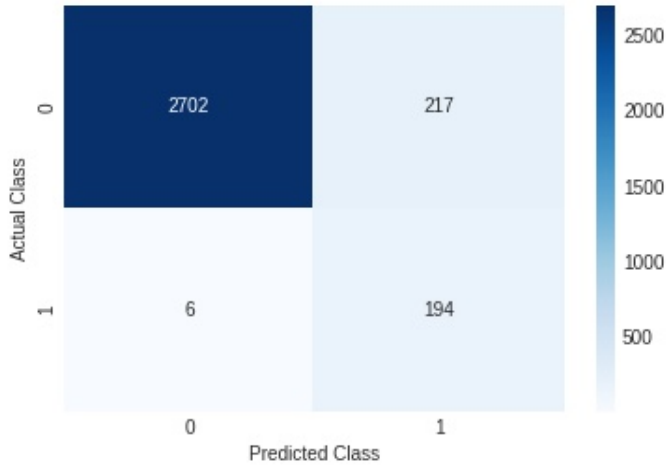

Figure 4.7: Confusion matrix for Autoencoder results (ECG dataset)

In this case metrics values are 4.9:

F1 score	Precision	Recall
0.63502	0.47202	0.97000

Table 4.7: Autoencoder metrics values (ECG dataset)

As results show, there is no big difference in prediction quality for both Autoencoder and DBSCAN, that were trained on the ECG dataset.

The model trained on charging cycles dataset has the following hyperparameters.

# of layers	8
# of neurons in each layer	$[64, 32, 16, 8, 16, 32, 64, 250]$
Loss function	Mean Absolute Error (MAE)
Optimizer	Adam
# of epochs	50

Table 4.8: Autoencoder hyperparameters (Charging cycles dataset)

There is a gap in the histogram of the reconstruction errors 4.8 around 0.028. This value is set as a threshold.

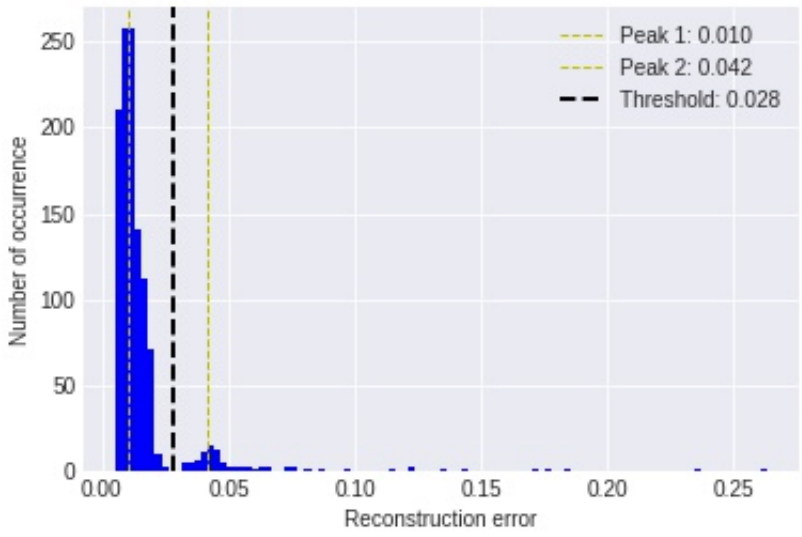

Figure 4.8: The histogram of reconstruction errors (Charging cycles dataset)

From results presented in 4.9 and 4.9 and one can conclude that the Autoencoder model performs significantly better than DBSCAN model on charging cycles dataset. This can be explained with the fact that the behaviour of charging cycles of electric vehicles can be described as non-linear. Moreover, unlike DBSCAN, the Autoencoder model does not struggle with the curse of dimensionality.

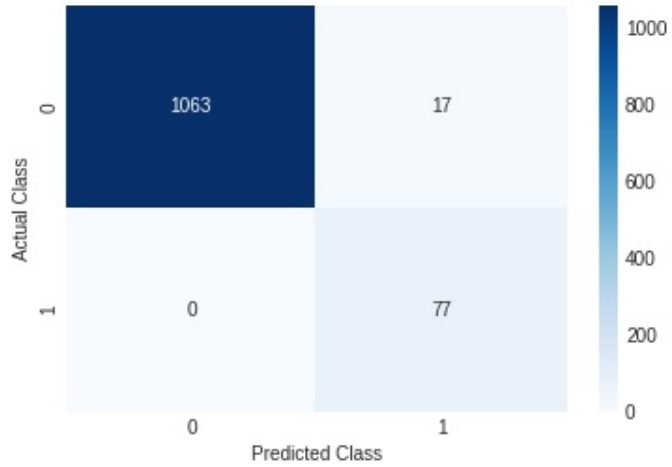

Figure 4.9: Confusion matrix for Autoencoder results (Charging cycles dataset)

F1 score	Precision	Recall
0.90058	0.81915	1.00000

Table 4.9: Autoencoder metrics values (Charging cycles dataset)

4.3.3.1 Application for Big Data

Feed forward networks are highly parallelizable, which means they are quite performant. If one can find a reasonably good feed forward model for your purposes, one can compute it very quickly [18]. The neural network based Autoencoder is suitable for gradual learning. Autoencoder can help to build a non-linear mathematical model, which is more useful for a great amount of time-series data from the real world.

4.4 Results summary

As part of this experiment the generation of synthetic data was done using SMOTE algorithm 4.2.2. For both the DBSCAN and the Autoencoder models the hyperparameter tuning was done. After training models the exploratory analysis of reconstruction errors had been done.

In the experimentation part four experiments were done with two models for anomaly detection: the DBSCAN and the Autoencoder, these models were applied to ECG and charging cycles datasets. The metrics summary for all experiments is provided in the following tables 4.10, 4.11.

	F1 score	Precision	Recall
DBSCAN	0.63376	0.46495	0.99500
Autoencoder	0.63502	0.47202	0.97000

Table 4.10: Metrics summary (ECG dataset)

	F1 score	Precision	Recall
DBSCAN	0.61883	0.47260	0.89610
Autoencoder	0.90058	0.81915	1.00000

Table 4.11: Metrics summary (Charging cycles dataset)

Autoencoder and DBSCAN models had similar results on the ECG dataset 4.4, 4.7 and 4.10, in other words none of the analyzed methods is significantly better than another.

In the experiment on charging cycles dataset the Autoencoder model demonstrated significantly better results. This method is also more suitable for big data analysis, especially for data with non-linear behaviour, this is exactly the description of charging profiles. The DBSCAN model trained on ECG

dataset had also better results, than the same model trained on charging cycles dataset. This can be explained with the fact that the DBSCAN struggles with the curse of dimensionality, while the dimensionality of the charging cycles dataset is bigger, than the dimensionality of the ECG dataset.

Conclusion

In this chapter, the summary of the thesis is provided, and results are concluded.

In Chapter 1, the background for Battery Management System was discussed.

The main focus of Chapter 2 is basic anomaly detection approaches. Advantages and disadvantages of each algorithm were described. Moreover, there were shown which algorithms can be applied to time series data. DBSCAN was selected as the algorithm for practical analysis.

In Chapter 3 the background for artificial neural networks were provided. Such special architecture of neural networks as Autoencoder was described with the special focus. The Chapter 4 was dedicated for analyzing and comparison of experiments that were done on the two different datasets for both the DBSCAN and the Autoencoder model.

Recapitulation of thesis goals

The summarized primary aims of this thesis are:

1. Analyze existing algorithms for detecting outliers.
2. Compare AI methods for applicability for anomaly detection.
3. Research the state-of-art electric vehicle battery technologies.
4. Research methods for optimizing methods for anomaly detection for big data.

The first two goals was discussed in Chapter 2 and 3. Such algorithms as K-means, DBSCAN and Autoencoder were described and compared. The applicability for anomaly detection in time series was provided. In the Chapter 1 the state-of-the-art of Battery Management System, which corresponds to

the third goal, was provided. The last goal was discussed in the Chapter 4. The aim of this goal was also the part of the implementation and can be found on the enclosed SD card.

Bibliography

[1] Mehrotra, K. G.; Mohan, C. K.; et al. *Anomaly Detection Principles and Algorithms*. Cham: Springer International Publishing AG, first edition, 2017, ISBN 978-3-319-67526-8.

[2] Bhattacharya, A. Effective Approaches for Time Series Anomaly Detection. *Towards Data Science [online]*, 2020, [cit. 2021-12-12]. Available from: `https://towardsdatascience.com/effective-approaches-for-time-series-anomaly-detection-9485b40077f1`

[3] Liu, K.; Li, K.; et al. A brief review on key technologies in the battery management system of electric vehicles. *Frontiers of Mechanical Engineering*, volume 14, no. 1, Mar. 2019: pp. 47–64, doi:https://doi.org/10.1007/s11465-018-0516-8, [cit. 2021-02-12].

[4] Rahimi-Eichi, H.; Ojha, U.; et al. Battery Management System: An Overview of Its Application in the Smart Grid and Electric Vehicles. *IEEE Industrial Electronics Magazine*, volume 7, no. 2, 2013: pp. 4–16, ISSN 1941-0115, doi:https://doi.org/10.1109/MIE.2013.2250351, [cit. 2021-11-12].

[5] Fang, H.; Wang, Y.; et al. Health-Aware and User-Involved Battery Charging Management for Electric Vehicles: Linear Quadratic Strategies. *IEEE Transactions on Control Systems Technology*, volume 25, no. 3, May 2017: pp. 911–923, ISSN 1558-0865, doi:10.1109/TCST.2016.2574761, [cit. 2021-11-12].

[6] Feng, X.; Ouyang, M.; et al. Thermal runaway mechanism of lithium ion battery for electric vehicles: A review. *Energy Storage Materials*, volume 10, 2018: pp. 246–267, ISSN 2405-8297, doi:https://doi.org/10.1016/j.ensm.2017.05.013, [cit. 2021-11-12]. Available from: `https://www.sciencedirect.com/science/article/pii/S2405829716303464`

[7] Feng, X.; Weng, C.; et al. Online internal short circuit detection for a large format lithium ion battery. *Applied Energy*, volume 161, 2016: pp. 168–180, ISSN 0306-2619, doi:https://doi.org/10.1016/j.apenergy.2015.10.019, [cit. 2021-11-12]. Available from: `https://www.sciencedirect.com/science/article/pii/S0306261915012465`

[8] Ravikiran, V.; Keshri, R. K.; et al. Efficient Wireless Charging of Batteries With Controlled Temperature and Asymmetrical Coil Coupling. In *2018 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, Dec 2018, pp. 1–5, doi:10.1109/PEDES.2018.8707441, [cit. 2021-11-12].

[9] Cao, X.; Zhong, Q.-C.; et al. Multilayer Modular Balancing Strategy for Individual Cells in a Battery Pack. *IEEE Transactions on Energy Conversion*, volume 33, no. 2, June 2018: pp. 526–536, ISSN 1558-0059, doi:10.1109/TEC.2017.2779520, [cit. 2021-11-12].

[10] Williams, R. M.; Haumann, J. R.; et al. A Battery-Operated Data-Acquisition System. *IEEE Transactions on Instrumentation and Measurement*, volume 32, no. 2, June 1983: pp. 356–360, ISSN 1557-9662, doi:https://doi.org/10.1109/TIM.1983.4315078, [cit. 2021-11-12].

[11] Dong, G.; Wei, J.; et al. Remaining dischargeable time prediction for lithium-ion batteries using unscented Kalman filter. *Journal of Power Sources*, volume 364, 2017: pp. 316–327, ISSN 0378-7753, doi:https://doi.org/10.1016/j.jpowsour.2017.08.040, [cit. 2021-11-12]. Available from: `https://www.sciencedirect.com/science/article/pii/S0378775317310650`

[12] EVEXPERT. Battery Management System [online]. November 2021, [cit. 2021-12-12]. Available from: `https://www.evexpert.eu/eshop1/knowledge-center/bms1`

[13] Ardeshiri, R. R.; Balagopal, B.; et al. Machine Learning Approaches in Battery Management Systems: State of the Art: Remaining useful life and fault detection. In *2020 2nd IEEE International Conference on Industrial Electronics for Sustainable Energy Systems (IESES)*, volume 1, 2020, ISBN 978-1-7281-4017-9, pp. 61–66, doi:https://doi.org/10.1109/IESES45645.2020.9210642.

[14] Liu, D.; Luo, Y.; et al. Lithium-ion battery remaining useful life estimation based on fusion nonlinear degradation AR model and RPF algorithm. *Neural Computing and Applications*, volume 25, no. 3, Sep 2014: pp. 557–572, ISSN 1433-3058, doi:10.1007/s00521-013-1520-x, [cit. 2021-11-12]. Available from: `https://doi.org/10.1007/s00521-013-1520-x`

[15] Razavi-Far, R.; Chakrabarti, S.; et al. Extreme learning machine based prognostics of battery life. *International Journal on Artificial Intelligence Tools*, volume 27, no. 08, 2018: p. 1850036, [cit. 2021-11-12].

[16] He, W.; Williard, N.; et al. State of charge estimation for Li-ion batteries using neural network modeling and unscented Kalman filter-based error cancellation. *International Journal of Electrical Power & Energy Systems*, volume 62, 2014: pp. 783–791, ISSN 0142-0615, doi:https://doi.org/10.1016/j.ijepes.2014.04.059, [cit. 2021-11-12]. Available from: `https://www.sciencedirect.com/science/article/pii/S0142061514002646`

[17] Hong, J.; Wang, Z.; et al. Fault prognosis of battery system based on accurate voltage abnormity prognosis using long short-term memory neural networks. *Applied Energy*, volume 251, 2019: p. 113381, ISSN 0306-2619, doi:https://doi.org/10.1016/j.apenergy.2019.113381, [cit. 2021-11-12]. Available from: `https://www.sciencedirect.com/science/article/pii/S0306261919310554`

[18] Nielsen, A. *Practical time series analysis prediction with statistics and machine learning*. O'Reilly, 2020, ISBN 978-1-492-04165-8.

[19] Aileen, N. *Practical time series analysis: Prediction with statistics and machine learning*. Sebastopol, CA: O'Reilly Media, 2019, ISBN 978-1-492-04162-7.

[20] Wu, G.; Zhao, Z.; et al. A Fast kNN-Based Approach for Time Sensitive Anomaly Detection over Data Streams. In *Computational Science – ICCS 2019*, edited by J. M. F. Rodrigues; P. J. S. Cardoso; J. Monteiro; R. Lam; V. V. Krzhizhanovskaya; M. H. Lees; J. J. Dongarra; P. M. Sloot, Cham: Springer International Publishing, 2019, ISBN 978-3-030-22741-8, pp. 59–74.

[21] Nunes da Silva, I.; Spatti, D. H.; et al. *Artificial neural networks: A practical course*. Basel, Switzerland: Springer International Publishing, first edition, 2016, ISBN 978-3-319-43162-8.

[22] Dorffner, G.; Bischof, H.; et al. *Artificial neural networks - ICANN 2001: international conference, Vienna, Austria, August 21-25, 2001 : proceedings*. Berlin: Springer, 2001;2006;, ISBN 0302-9743.

[23] Kumar, N. McCulloch Pitts neuron - deep learning building blocks. [online], Dec 2019, [cit. 2021-11-12]. Available from: `https://medium.com/hackernoon/mcculloch-pitts-neuron-deep-learning-building-blocks-7928f4e0504d`

[24] Brownlee, J. How to choose an activation function for deep learning. [online], Jan 2021, [cit. 2022-01-02]. Available from:

https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[25] tutorialspoint. Artificial neural network - building blocks. [online], [cit. 2022-01-02]. Available from: https://www.tutorialspoint.com/artificial_neural_network/artificial_neural_network_building_blocks.htm

[26] Chen, Y.; Keogh, E. [online], [cit. 2022-02-02]. Available from: https://www.timeseriesclassification.com/description.php?Dataset=ECG5000

[27] Wong, S. C.; Gatt, A.; et al. Understanding Data Augmentation for Classification: When to Warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2016, pp. 1–6, doi:10.1109/DICTA.2016.7797091, [cit. 2022-03-04].

[28] Lendave, V. How can smote technique improve the performance of weak learners? [online], Jan 2022, [cit. 2022-03-04]. Available from: https://analyticsindiamag.com/how-can-smote-technique-improve-the-performance-of-weak-learners/

[29] Korstanje, J. The F1 score. [online], Aug 2021, [cit. 2022-02-02]. Available from: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

[30] Team, G. L. Understanding curse of dimensionality. [online], Mar 2022, [cit. 2022-04-04]. Available from: https://www.mygreatlearning.com/blog/understanding-curse-of-dimensionality/

[31] Rahmah, N.; Sitanggang, I. S. Determination of Optimal Epsilon (Eps) Value on DBSCAN Algorithm to Clustering Data on Peatland Hotspots in Sumatra. *IOP Conference Series: Earth and Environmental Science*, volume 31, jan 2016: p. 012012, doi:10.1088/1755-1315/31/1/012012, [cit. 2022-04-04]. Available from: https://doi.org/10.1088/1755-1315/31/1/012012

[32] Choudhury, A. Curse of dimensionality and what beginners should do to overcome it. [online], Dec 2019, [cit. 2022-04-04]. Available from: https://analyticsindiamag.com/curse-of-dimensionality-and-what-beginners-should-do-to-overcome-it/

Appendix A

Acronyms

AI Artificial intelligence

ANN Artificial neural network

BMS Battery management systems

EV Electric vehicle

SOH State-of-health

RUL Remaining useful life

EOL End of life

ID Identification number

DBSCAN Density-Based Spatial Clustering of Applications with Noise

KNN K-nearest neighbors

PCA Principal Component Analysis

ECG Electrocardiogram

Appendix B

Contents of enclosed SD

- `readme.txt` ..... the file with SD contents description
- `notebooks` ..... the directory with Jupyter notebooks
- `thesis` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `src` ..... the directory of LaTeX source codes of the thesis