FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER’S THESIS

Title:	Online Anomaly Detection in Time-Series
Student:	Bc. Tomáš Pajurek
Supervisor:	Ing. Tomáš Borovička
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2018/19

Instructions

Anomaly detection methods aim to identify unexpected deviations or novelties in various kinds of data. Online methods for time-series narrow this field to detecting anomalies in time-ordered data and adds the capability to adapt and continuously learn as the new data come. Variety of methods, ranging from simple statistical methods to neural networks are used for anomaly detection. However, not every method can be used for online detection in time-series.

1) Review and theoretically describe state of the art methods for anomaly detection in time series data with special emphasis on online methods for continuous time-series.
2) Use or implement at least three methods and experimentally compare their performance on various data sets. Avoid implementing anew those methods that can be easily taken over from available implementations.
3) Propose directions for further improvements of reviewed methods.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 9, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Online Anomaly Detection in Time-Series

Bc. Tomáš Pajurek

Department of Theoretical Computer Science
Supervisor: Ing. Tomáš Borovička

May 8, 2018

Acknowledgements

I would like to thank all fellow members of the Data Science Laboratory at CTU FIT who were writing their theses at the same time as me for their scientific as well as emotional support. Also, I would like to express my gratitude to the Ing. Tomáš Borovička for the mentorship and valuable advice and Datamole, s.r.o. for providing computational infrastructure for the experiments.

Declaration

In Prague on May 8, 2018 . . . .

Citation of this thesis

Pajurek, Tomáš. *Online Anomaly Detection in Time-Series.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

Abstrakt

Metody pro online detekci anomálií jsou navrženy pro odhalování anomalií ve spojitém proudu dat namísto ve statickém datasetu. Tyto metody jsou schopné se adaptovat na změny v charakteristice datového proudu, který může v čase nastávat (concept drift).

Tato práce analyzuje čtyři metody vhodné pro online detekci anomálií v časových řadách (klouzavý průměr, local outlier factor, isolation forest, hierarchical temporal memory) a několik metod detekce concept driftu včetně některých nových přístupů. Je navrženo obecné schéma, které umožňuje kombinovat různé metody pro detekci anomálií a concept driftu. Pro všechny analyzované metody jsou provedeny experimenty na pěti realných datasetech a jednom umělém. Během experimentů byly zkoumány vlastnosti jednotlivých metod a porovnáván jejich výkon s ostatními metodami.

Výsledky experimentů ukazují, že žádná metoda není lepší než ostatní na všech datasetech z hlediska F_1 skóre upraveného pro úlohu detekce anomalií (harmonický průměr specificity a míry falešné pozitivních detekcí) a AUC. Ve většině případů bylo nalezeno optimální nastavení methody s F_1 skóre $> 85\%$ a AUC $> 90\%$.

Klíčová slova detekce anomálií, online, streaming, časové řady, klouzavý průměr, local outlier factor, isolation forest, hierarchical temporal memory

Abstract

Methods for online anomaly detection are designed to reveal anomalies in a continuous stream of data rather than in a static dataset. These methods are able to adapt to the changes of underlying characteristics of the stream that might occur in time (concept drift).

This thesis reviews four methods suitable for online anomaly detection in time-series (moving average, local outlier factor, isolation forest, hierarchical temporal memory) and several concept drift detection methods including some novel approaches. A general framework that allows to orthogonally combine various anomaly detection methods and concept drift detection methods is proposed. Experiments were executed for all reviewed methods on five real-world datasets and one artificial dataset. During the experiments, the properties of individual methods were examined as well as their performance compared to the other methods.

Results of the experiments show that none of the methods is superior to the others on all datasets in terms of F_1 score adapted for anomaly detection (harmonic mean of recall and false positive rate) and AUC. In the majority of cases, an optimal method settings with F_1 score $> 85\%$ and AUC $> 90\%$ was found.

Keywords anomaly detection, online, streaming, time-series, moving average, local outlier factor, isolation forest, hierarchical temporal memory

Contents

List of Figures

List of Tables

Introduction

Anomaly detection is a problem that has been researched within many research areas and application domains. Statisticians identified and studied this problem already in the 19th-century. Probably the first effort in this area was made by F. Y. Edgeworth in 1887 [1] who proposed three statistical hypotheses for revealing items in a set of integers that were generated by a different process than the majority of the others. In these days, anomaly detection plays an important role in the critical systems e.g. for healthcare and computer security as well as in various industrial and business applications.

In healthcare, anomaly detection is used for example in order to identify possibly dangerous abnormal growths such as tumours in the outputs of magnetic resonance imaging (MRI). Another example can be the analysis of the electrocardiogram (ECG) data. In this case, properly used anomaly detection can identify deviations from a normal heartbeat and alert the medical personnel about the possible impending problem before it gets more serious. In the field of computer security, anomaly detection has been successfully used for investigating security-related incidents as well as for intrusion detection systems (IDS) to identify breaches in real-time. A well-known example of anomaly detection deployed in the industrial environment is predictive maintenance. In this context, the maintenance is understood as a continuous effort to keep some industrial systems working correctly and to minimize downtime. The predictive maintenance approaches use data from sensors attached to the machines and strive to identify anomalies. When the anomaly is found, it might signalize a change in the device condition and increase a probability that the machine or some of its components will soon fail. In the area of retail banking or telecommunications, anomaly detection can be used to minimize financial damages caused by frauds or organized scams. An example can be the modelling of the users' normal payment behaviour to be able to identify anomalies in this behaviour if the e.g. credit card is stolen.

Anomaly detection is a very general problem that has many specific subareas. Most of the introduced examples are built on a data that are ordered in some manner, typically by the time of a measurement. Data in such form is called a time-series. Also, the great deal of anomaly detection methods used in the examples is expected to identify anomalies in the continuous stream of data and rather than on a static set of data. This is so-called online anomaly detection.

The goals of the thesis

At first, this thesis strives to establish a solid theoretical foundation that is used for the ongoing description of several anomaly detection methods. As mentioned in the very beginning, the authors of the literature concerned with the anomaly detection come from various domains and their approaches and terminology is very heterogeneous and inconsistent. An output of this part is an assembly of various relevant point of views found in literature in a form of a consistent theoretical framework.

The main part of the thesis focuses on the analysis of several specific anomaly detection methods. Some of these methods are directly designed for the online anomaly detection in time-series, but many of them are the adaptations of more general approaches. The ultimate motivation for laying the theoretical framework is the ability to precisely describe the analyzed methods.

Organization of the thesis

There are three fundamental chapters. The first chapter (*Theoretical Framework*) is building up the theoretical framework of an online anomaly detection in time-series. The second chapter (*Methods*) defines a generic online anomaly detection system and then, several specific methods are analyzed and compared in the context of this system. The last, third, chapter (*Experiments*) proposes the ways how to quantitatively measure and compare the performance of online anomaly detectors. The design of the experiments is established. Finally, a significant number of experiments is executed for each of the analyzed methods on real-world datasets. The thesis is concluded by the discussion of places for possible improvement that were identified during the work.

Chapter 1

Theoretical framework for online anomaly detection in time-series

In this chapter, theoretical background needed for subsequent analysis of anomaly detection methods and experiments will be established. The chapter starts with introduction to the domain of general anomaly detection and continues with narrowing this domain to anomaly detection in time-series using online learning.

This chapter strives to be very formal to avoid possible misconceptions during further description and analysis of specific methods. Next chapters are less formal and refer to this chapter for necessary definitions.

1.1 General anomaly detection

An *anomaly* is defined as "something different, abnormal, peculiar, or not easily classified" [2]. In the context of data science, these anomalies are represented as unexpected patterns in various kinds of data (see Figure 1.1 for example). *Anomaly detection* is a field concerned with identifying such patterns. Often, it is understood as another class of machine learning problems, among e.g. classification, regression or clustering problems.

Anomalies can occur for several reasons. For example, the data arriving from sensors of a flying aircraft changes in case of a fault scenario [3]. From the statistical point of view, the underlying distribution of the incoming data changes (e.g. temperature of the engine rises) or the incoming data are mixed with data from different distribution (e.g. engine component that went loose starts making unusual noise).

In mathematical terms, an *anomaly detector* is a morphism $\mathcal{D}$ with the domain of input data $\mathbb{I}$ (see Section 1.1.1.1 below for definition) and the codomain of detection output $\mathbb{D}$ (see Section 1.1.1.2). The detection output is used to decide whether an instance of input data is an anomaly or not:

$$\begin{gathered}\mathcal{D} : \mathbb{I} \to \mathbb{D} \\ \text{where } \mathbb{I} \text{ is the domain of input data} \\ \text{and } \mathbb{D} \text{ is a domain of detection output.}\end{gathered} \tag{1.1}$$

The decision whether an instance of input data is an anomaly or not is widely dependent on the application domain, observer and other external factors [4]. Therefore only very general definition of an anomaly placed earlier in this section is provided instead of a rigorous mathematical definition.

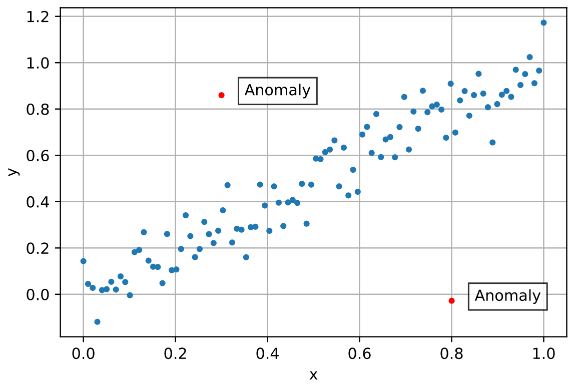


Figure 1.1: Example data generated using function $f(x) = x + 0.08 \cdot \mathcal{N}(0, 1)$. Several anomalies were introduced by generating some points using modified function $a(x) = f(x) + 0.64 \cdot \mathcal{N}(0, 1)$. This example represents situation when the normal data are mixed with data from different distribution as described in Section 1.1.

Terminology inconsistencies In literature, the term anomaly detection is often used interchangeably with *outlier detection* or *novelty detection*. This inconsistency in naming is caused by a different background of researchers studying this area [5]. Some authors make a slight distinction between an anomaly and a novelty. This distinction is based on fact that novelties are usually incorporated into the normal model after detection and unlike anomalies or outliers, are not considered once detected [4].

In the rest of this thesis, the term *anomaly detection* is used exclusively.

Relation to noise and noise removal Another related terms are *noise* and *noise removal*. However, the distinction between noise and the previous group of terms (anomaly, outlier, novelty) is very clear. In this context, noise is understood as an error in underlying data caused by the e.g. low accuracy of an industrial sensor or mistakes in human-crafted data. On the other side, anomaly “may arise from the natural variation within the population or process” [6]. Noise removal is a discipline focusing on removing the noise from data while preserving as much valuable information as possible.

Noise and noise removal are considered to be out of the scope of this thesis.

1.1.1 Taxonomy

In this section, the definitions and concepts will be stated that are essential for the further analysis of detection methods.

1.1.1.1 Input data

In the context of this thesis, the *input data* (*input dataset*) is defined as the set $\mathbb{X}$ containing n *data points* (*instances*) $\mathbf{x}_j$. Therefore $\mathbb{X} = \{\mathbf{x}_j | j \in \{1, 2, 3, \ldots, n\}\} \in \mathbb{I}$. The data points are inputs for morphism $\mathcal{D}$ defined in 1.1. Each data point $\mathbf{x}_j$ is a d-dimensional vector of scalar values. Domain of these scalar values and by extension the definition of domain $\mathbb{I}$ will be discussed later in Section 1.1.1.3. The n and d are natural numbers. We say that the dataset is *high dimensional* if $n \ll d$ [7].

Dimensions of the data points are referred to as attributes (also known as *features* or *columns*). The value of an attribute a of vector $\mathbf{x}_j$ is notated as $\mathbf{x}_j[a]$.

In terms of linear algebra, the input dataset can be defined as a $n \times d$ matrix $X^{n,d}$ containing the data point vectors as rows:

$$X^{n,d} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{1,1} & \mathbf{x}_{1,2} & \cdots & \mathbf{x}_{1,d} \\ \mathbf{x}_{2,1} & \mathbf{x}_{2,2} & \cdots & \mathbf{x}_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{n,1} & \mathbf{x}_{n,2} & \cdots & \mathbf{x}_{n,d} \end{pmatrix}. \tag{1.2}$$

Data points can be also associated with so called *label(s)*. Label $\mathbf{l}_i$ for the vector $\mathbf{x}_j$ is an e-dimensional vector of scalar values where $e \in \mathbb{N}$ is from space $\mathbb{L}$:

$$X^{n,d+e} = \begin{pmatrix} \mathbf{x}_1^T & \mathbf{l}_1^T \\ \mathbf{x}_2^T & \mathbf{l}_2^T \\ \vdots & \vdots \\ \mathbf{x}_n^T & \mathbf{l}_n^T \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{1,1} & \cdots & \mathbf{x}_{1,d} & \mathbf{l}_{1,1} & \cdots & \mathbf{l}_{1,e} \\ \mathbf{x}_{2,1} & \cdots & \mathbf{x}_{2,d} & \mathbf{l}_{2,1} & \cdots & \mathbf{l}_{2,e} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{n,1} & \cdots & \mathbf{x}_{n,d} & \mathbf{l}_{n,1} & \cdots & \mathbf{l}_{n,e} \end{pmatrix}. \tag{1.3}$$

In the context of anomaly detection, the labels carry information, whether the related data points are considered anomalous or not.

In the real-world datasets, the attribute values or labels might be missing for some data points. In such cases, the input dataset must be preprocessed

[1] in order to comply with definitions 1.2 and 1.3 or the anomaly detection methods must be able to handle such incomplete data points (see 1.1.1.5 for description of such methods).

Univariate vs. Multivariate The input dataset is said to be *univariate* if each data point $\mathbf{x}_j$ is a single-component vector, resp. *multivariate* if each $\mathbf{x}_j$ is a vector with more than one component:

$$\begin{gathered}\text{an input dataset is univariate, resp. multivariate}\\ \Leftrightarrow\\ \forall j \in \{1,2,\ldots,n\} : \mathbf{x}_j \text{ is a } d\text{-dimensional vector} \wedge d = 1, \text{ resp. } d > 1.\end{gathered}$$

Categorical vs. Continuous The attribute of input data is said to be *categorical* or *symbolic* if the related components of each $\mathbf{x}_j$ belong to a finite set $\mathbb{C}$ of categories or symbols. Sometimes, this kind of input data is inaccurately called *discrete*:

$$\begin{gathered}\text{an attribute } a \text{ is categorical}\\ \Leftrightarrow\\ \forall j \in \{1,2,\ldots,n\} : \mathbf{x}_j[a] \in \mathbb{C} \text{ where } \mathbb{C} = \{1,2,3,\ldots,c\} \text{ and } c \in \mathbb{N}, c < \infty.\end{gathered}$$

In real-world data, the set $\mathbb{C}$ is frequently just an encoding for some properties.

On the other hand, for the *continuous* attribute, the related components of all x_i are real numbers:

$$\begin{gathered}\text{an attribute } a \text{ is continuous}\\ \Leftrightarrow\\ \forall j \in \{1,2,\ldots,n\} : \mathbf{x}_j[a] \in \mathbb{R}.\end{gathered}$$

In many cases, the data points can consist of a mix of categorical and continuous attributes.

Dependencies between data points In the trivial case, there are no dependencies between data points. Otherwise, three major categories of dependencies are recognized [4].

Sequences are input data containing data points with a defined linear order. Especially significant are sequences with temporal meaning. Each data point

[1]Typically, the incomplete data points are removed from the dataset or missing values are filled with some suitable surrogate values.

describes some system in a given point in time. Such input data are called *time-series.*

In *Spatial* dataset, each data point is related to its neighboring points. This implies that there must be some relation, like distance, defined over the data points.

Data points of a *graph* input data represent vertices of a graph that are connected with edges. These edges can be defined by attributes that reference other data point(s).

Combination of these categories is also possible. For example, *spatio-temporal* dataset which contains a time-based sequence of data points with spatial properties.

1.1.1.2 Output of an anomaly detector

Anomaly detection methods can be categorized by the type of their output. It can be either the *score* or the *label.*

The *scoring* anomaly detector yields a scalar real value, typically normalized to interval $\langle 0, 1\rangle$. If the anomaly detector works as expected, this value should correlate with the confidence of whether the analyzed data point is an anomaly or not:

$$\begin{gathered}\mathcal{D} : \mathbb{I} \to \mathbb{R} \text{ for scoring detector,}\\ \mathcal{D} : \mathbb{I} \to \langle 0, 1\rangle \text{ for scoring detector with normalization.}\end{gathered}$$

The *labelling* anomaly detector yields only a binary value from $\{0, 1\}$ telling whether the analyzed data point is an anomaly or not:

$$\mathcal{D} : \mathbb{I} \to \{0 \mapsto \text{normal}, 1 \mapsto \text{anomaly}\}.$$

The detection output has a significant impact on the application. In a situation when the scoring detector is available and the label is required, the score must be converted to the label. It can be done by simple thresholding or some more sophisticated approaches combining the score with other external information (like a certainty measure of the analyzed data point) might be used. In either way, additional effort is required. On the other hand, the labelling detector is not able to produce any score which could be used for e.g. finding the top k anomalies with the highest confidence.

1.1.1.3 Domain $\mathbb{I}$ of input data and domain $\mathbb{D}$ of detection output

With definitions of input dataset (section 1.1.1.1) and detection output (section 1.1.1.2), the definition of anomaly detector 1.1 and domains $\mathbb{I}, \mathbb{D}$ can be refined as:

$$\mathcal{D} : \mathbb{I} \to \mathbb{D} \equiv \begin{cases} \mathcal{D} : \mathbb{R}^d \to \mathbb{D} \text{ for continuous attributes } (\mathbb{I} = \mathbb{R}^d), \\ \mathcal{D} : \mathbb{C}_1 \times \mathbb{C}_2 \times \cdots \times \mathbb{C}_d \to \mathbb{D} \text{ for categorical attr.,} \\ \mathcal{D} : \mathbb{R}^{d-m} \times \mathbb{C}_1 \times \cdots \times \mathbb{C}_m \to D \text{ for mixed attr.,} \end{cases} \tag{1.4}$$

where $\mathbb{D} = \{0, 1\}$, resp. $\mathbb{D} = \mathbb{R}$.

Then, an example of a morphism (function) 1.4 for continuous attributes and binary output can be written as follows:

$$\begin{gathered} \mathcal{D}((21.4, 15.6, 61.3)^T) = 0, \\ \mathcal{D}((23.4, 15.5, 14.3)^T) = 0, \\ \mathcal{D}((55.5, 0.3, 0.2)^T) = 1, \\ \mathcal{D}((12.0, 85.1, 12.3)^T) = 0. \end{gathered}$$

1.1.1.4 Types of anomalies

Chandola, Banerjee and Kumar [4] define three following types of anomalies:

Point anomalies The simplest type of anomalies. Each data point can be analyzed by the anomaly detector without considering any other data points in the input dataset.

Contextual anomalies For detecting a contextual anomaly, anomaly detector has to consider the context in which the data point occurred. In practice, this type of anomaly can occur especially in sequence or spatial (see Section 1.1.1.1) datasets.

Figure 1.2 illustrates the difference between a point anomaly and a contextual anomaly. Point anomalies in this figure are data points that have higher or lower values than the majority of other points. Therefore, they can be labelled as anomalies right away. On the other hand, the figure contains also contextual anomalies that have values in expected range but are anomalous with regard to placement of other points with similar value.

Collective anomalies Collective anomalies are special in that the only sets of data points can be labelled as collective anomalies (not individual data points). The data points in anomalous set themselves can be normal, but together, they represent an anomaly.

1.1.1.5 Learning and availability of labels

In general case, anomaly detection methods require initial phase called *learning* before actual detections are made. The output of the learning phase is

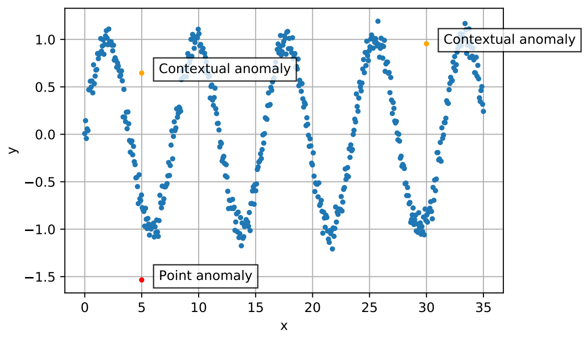


Figure 1.2: Example data generated using function $f(x) = \sin(0.8 \cdot x) + 0.08 \cdot \mathcal{N}(0, 1)$. Several anomalies were introduced by generating some points by modified functions $a_1(x) = 2 \cdot \sin(0.8 \cdot x) + 0.08 \cdot \mathcal{N}(0, 1)$ and $a_2(x) = \sin(0.8 \cdot x + 3) + 0.08 \cdot \mathcal{N}(0, 1)$. This example represents a situation when there are data points that can be considered as anomalies only in specific context.

called a *model* (this model is a realization of the anomaly detector $\mathcal{D}$ defined in 1.4). During the learning phase, the parameters of the underlying detection model are set. The dataset used for learning phase is known as *training dataset*.

The methods can be categorized depending on the presence of labels in the training dataset as described in the following paragraphs.

Unsupervised anomaly detection In this approach, labels are not considered. The methods based on unsupervised learning strive to detect the anomaly without explicit knowledge whether the learning data points are actual anomalies or not. Identification of anomalies is based on their deviation from the other (normal) data points [8]. These methods make an assumption that [4] normal data points are far more frequent than anomalous instances. The primary focus of this thesis lies in the unsupervised methods.

Assumption for unsupervised anomaly detection:

$$|\{\mathbf{x}_j | j \in \{1, \ldots, n\} \wedge \mathbf{x}_j \text{ is anom.}\}| \ll |\{\mathbf{x}_j | j \in \{1, \ldots, n\} \wedge \mathbf{x}_j \text{ is norm.}\}|.$$

Supervised anomaly detection Supervised methods require having a label for each data point in the training dataset. They use this information to learn differences between normal and anomalous instances using *discriminative* or *generative* algorithms [9]. The supervised approach in the context of anomaly detection can be understood as a significantly imbalanced binary classification.

Semi-supervised anomaly detection Semi-supervised methods are able to work with partially labelled data [9]. In case of anomaly detection, there can be labels only for e.g. part of anomalous instances or part of normal instances.

In reality, obtaining labels is usually problematic and requires an effort of human domain experts. Therefore, unsupervised methods are superior to supervised methods in this context. Getting labels for anomalous instances is especially problematic [4] and often only labels for normal instances are present if any. Semi-supervised methods are suitable candidates for this kind of tasks.

1.2 Online anomaly detection in time-series

So far, the general case of anomaly detection has been discussed. From now on, the focus will be placed on online anomaly detection in time-series. This section starts with the definition of a time-series, continues with the definition of online learning and is concluded with the discussion of the concept drift and related problems. The topic of engineering challenges connected to the online anomaly detection is also discussed, however, only marginally.

It should be noted that a lot of research and methods exist on the topic of general anomaly detection. However, the amount of literature and research on specifically online methods for time-series is significantly smaller.

1.2.1 Time-Series

Time-series is a special case of input dataset $\mathbb{X}$ that can be classified as a sequence. All previously defined properties and categorizations hold.

Data points in a time-series are ordered by one of their continuous attributes. Let this attribute be called *temporal attribute* t [2]. In the following text, the value of the temporal attribute for data point $\mathbf{x}_j$ will be referred to as t_j:

$$\mathbf{x}_j[t] = t_j.$$

Until now, the actual order of data points $\mathbf{x}_j$ was irrelevant. From now on, the data points $\mathbf{x}_j$ are considered to be ordered by their values of attribute t.

To simplify the ongoing notation, several simplification are made without loss of generality:

[2]The temporal attribute usually represents time.

- In reality, the time and therefore the ordering can be given by multiple attributes. However, only a single attribute is considered to be temporal in this thesis[3].
- Order of values of t corresponds to order of its indices (1.5).
- Data point $\mathbf{x}_j$, resp. label $\mathbf{l}_j$ can be referred to as $\mathbf{x}_{t_j}$, resp. $\mathbf{l}_{t_j}$ (1.6).
- The value of the temporal attribute is the first component of the data point vectors (1.7).

$$\begin{aligned}
&\forall j \in \{1, \ldots, n\} : \\
&j < n \Rightarrow t_j < t_{j+1}, && (1.5)\\
&\mathbf{x}_j = \mathbf{x}_{t_j}, \text{ resp. } \mathbf{l}_j = \mathbf{l}_{t_j}, && (1.6)\\
&\mathbf{x}_{j,1} = t_j. && (1.7)
\end{aligned}$$

By incorporating the temporal attribute to the definition 1.2 and applying the simplification given by 1.5, 1.6, 1.7 the following matrix is obtained:

$$X^{n,d} = \begin{pmatrix} \mathbf{x}_1^T = \mathbf{x}_{t_1}^T \\ \mathbf{x}_2^T = \mathbf{x}_{t_2}^T \\ \vdots \\ \mathbf{x}_n^T = \mathbf{x}_{t_n}^T \end{pmatrix} = \begin{pmatrix} t_1 = \mathbf{x}_{1,1} & \mathbf{x}_{1,2} & \cdots & \mathbf{x}_{1,d} \\ t_2 = \mathbf{x}_{2,1} & \mathbf{x}_{2,2} & \cdots & \mathbf{x}_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ t_n = \mathbf{x}_{n,1} & \mathbf{x}_{n,2} & \cdots & \mathbf{x}_{n,d} \end{pmatrix}. \tag{1.8}$$

A time-series is called *equidistant* if the values of temporal attribute are equally spaced by a time step Δ:

$$\forall j \in \{1, \ldots, n\} : j < n - 1 \Rightarrow t_{j+1} - t_j = t_{j+2} - t_{j+1} = \Delta,$$
$$\forall j \in \{1, \ldots, n\} : j > 1 \Rightarrow t_j = t_1 + (j - 1)\Delta.$$

This thesis focuses mainly on time-series with continuous attributes. Many comprehensive research papers and methods exist specifically on the topic of anomaly detection in symbolic time-series [3]. Some methods for symbolic series might be used also for continuous series however many of them, especially those based on string matching, are not applicable.

In the following text, a time-series is considered to be equidistant, with continuous attributes and without labels, unless otherwise specified.

[3]Multiple attributes can be collapsed to a single one and the ordering function is adapted to handle the collapsed attribute. Possibly, the collapsed attributes can be copied as normal attributes if necessary.

1.2.1.1 Windowing

Many anomaly detection methods require the input dataset to be preprocessed using a technique called *windowing* [3].

Let the function $\mathcal{W}_{\sigma,\tau}$ be the *windowing function* morphing the matrix X from 1.8 to the vector of q matrices $w = (W_1, W_2, ..., W_q)$. Each component of the vector w is a submatrix of X containing all d attributes (columns) but only σ data points (rows):

$$\mathcal{W}_{\sigma,\tau} : \mathbb{R}^{n,d} \rightarrow \underbrace{\mathbb{R}^{\sigma,d} \times \mathbb{R}^{\sigma,d} \times \cdots \times \mathbb{R}^{\sigma,d}}_{\text{q}}. \tag{1.9}$$

Size σ and *step* τ are parameters of $\mathcal{W}$ and can be chosen arbitrarily with only restriction given by the following conditions:

$$\begin{gathered}\sigma, \tau, q \in \mathbb{N},\\ \sigma \leq n,\\ q = \left\lceil \frac{n - (\sigma - 1)}{\tau} \right\rceil.\end{gathered}$$

Components of w can be defined as:

$$\forall k \in \{1, 2, \ldots, q\} :$$

$$W_k^{\sigma,d} = \begin{pmatrix} t_{\mathcal{H}(k)} & \mathbf{x}_{\mathcal{H}(k),2} & \cdots & \mathbf{x}_{\mathcal{H}(k),d} \\ t_{\mathcal{H}(k)+1} & \mathbf{x}_{\mathcal{H}(k)+1,2} & \cdots & \mathbf{x}_{\mathcal{H}(k)+1,d} \\ \vdots & \vdots & \ddots & \vdots \\ t_{\mathcal{H}(k)+(\sigma-1)} & \mathbf{x}_{\mathcal{H}(k)+(\sigma-1),2} & \cdots & \mathbf{x}_{\mathcal{H}(k)+(\sigma-1),d} \end{pmatrix} \tag{1.10}$$

$$\text{where } \mathcal{H}(j) : \mathbb{N} \rightarrow \mathbb{N} \coloneqq (j-1) \cdot \tau + 1.$$

Simple example of application of windowing function on 8×3 matrix follows:

$$\mathcal{W}_{\sigma=3,\tau=2} \left(\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \\ 7 & 7 & 7 \\ 8 & 8 & 8 \end{pmatrix} \right) = \left(\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}, \begin{pmatrix} 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \end{pmatrix}, \begin{pmatrix} 5 & 5 & 5 \\ 6 & 6 & 6 \\ 7 & 7 & 7 \end{pmatrix} \right). \tag{1.11}$$

Note that in some cases, up to $(\sigma - 1)$ last data points can be left out. In example 1.11 , it is the row $\begin{pmatrix} 8 & 8 & 8 \end{pmatrix}$. This issue must be resolved specifically by each detection method – e.g. the unused data points are ignored or the last window is constructed as incomplete or suitable dummy values are supplied to enable construction of complete last window.

1.2.2 Online learning

In this section, the scope of this thesis will be finalized by putting specific constraints on the learning phase of the detection methods. In section 1.1.1.5, different types of learning process were described based on the (un)availability of labels in the input dataset. However, the learning process can be categorized by another aspect and it is whether the detection method learns on a static dataset or a streaming dataset.

Learning on the *static* dataset is a traditional [10] way of learning for anomaly detection algorithms as well as other machine-learning algorithms. It is assumed that all necessary information can be extracted from the static training dataset before making actual detections. It means that the dataset must be available all at once and contain enough information. This approach is not suitable for situations where the complete training dataset is not available beforehand. In reality, there are many use cases (medical, IT security, agriculture) [10] where there is a little to none data in the beginning but new data arrives over time (*streaming* dataset). The detection methods must be able to start making detections as soon as possible with very little initial knowledge and adapt this knowledge as new data are available [4]. This process is called *incremental* or *online* learning.

The properties of online anomaly detection stated above are summed up and extended by Polikar [10] and Ahmad [11] who define ideal properties of real-world online anomaly detection algorithm as follows:

- The algorithm must learn continuously without storing all the previous data points and detections.
- The algorithm must incorporate information about all previous data points into the current detection.
- The algorithm must run in an automatic manner without requirements for manual parameter tweaking.
- The algorithm must adapt to dynamic environments and non-stationarity of the underlying statistics of the stream of data points.

[4]This property of online learning can be extended to cover not only data points incoming one at a time but also to the data points incoming in *batches*. Batch is the set of new data points that are presented to the detector at the same time.

- The algorithm must make anomaly detection as soon as possible. In terms of section 1.1.1.4, this applies to contextual and collective anomalies.
- The algorithm should minimize false positives and false negatives.
- The algorithm should minimize the time for making the detection.

Anytime algorithms Another concept worth mentioning while discussing online learning is an *anytime algorithm*. Similarly to online methods, anytime algorithms are able to give a result at any given moment and the results should be improving over time. However, the context and usage are different. Anytime algorithms primarily focus on trading shorter execution times for lower accuracy in a controllable manner while keeping properties like measurable quality, interruptibility or resumability [12], [8]. Anytime algorithms represent a much broader field of study than online anomaly detection and are considered to be out of the scope of this thesis.

1.3 Concept drift

The concept drift is a change of the underlying process generating the data points that happens over time. It is a phenomenon that largely influences the design of anomaly detection methods. For example, the models generated by non-online supervised algorithms have no way how to adapt to the changes and the concept drift leads to continual degradation of their results. At best, the results of these models can be continuously validated and, in case that the degradation reaches some threshold, the model is discarded and a new model is trained. On the other hand, online unsupervised algorithms have all the necessary prerequisites to continuously adapt the model to the concept drift.

Let the concept $\mathcal{C}_t$ at the time t be the joint probability distribution $P_t(\mathrm{X}_t, \mathrm{L}_t)$ where X_t and L_t are random vectors [5]. Input data points with the value of temporal attribute lesser or equal to t, resp. associated labels are realizations of X_t, resp. L_t. Concept drift occurs when $\mathcal{C}_t \neq \mathcal{C}_u$. Let the $\mathcal{R}_{t,u}$ be the *concept drift indicator* defined as:

$$\mathcal{R}_{t,u} = \begin{cases} 0 & \text{if } \mathcal{C}_t = \mathcal{C}_u, \\ 1 & \text{if } \mathcal{C}_t \neq \mathcal{C}_u. \end{cases}$$

The difference between two concepts can be quantified by binary function $\mathcal{M}$ which is applied to two concepts and returns a real number. Various measures such as *total drift magnitude* or *marginal drift magnitude* exists [13] but will not be further discussed in this thesis.

[5] multivariate random variables

Various methods for handling the concept drift (adaptation) will be studied in the following text.

In the next two sections, the pair of problems related to concept drift and online learning will be briefly introduced. These problems have been studied especially in the context of neural networks architecture, however, the ideas are considered to be general enough to apply to other algorithms capable of online (incremental) learning.

1.3.1 Stability-Plasticity dilemma

The *plasticity* is the ability of the learning algorithm to incorporate new, previously unseen patterns in data into the detection model. The *stability* is a property that enables the detection model not to discard the knowledge about previously seen patterns while processing other irrelevant or very frequent patterns. According to the Carpenter and Grossberg, "The properties of plasticity and stability are intimately related. An adequate system must be able to adaptively switch between its stable and plastic modes." [14], [15]. This tradeoff between stability and plasticity that must be made during the design of incremental machine learning algorithms is referred to as the *stability-plasticity dilemma*.

1.3.2 Catastrophic forgetting

The *catastrophic forgetting* or *catastrophic inference* is an infamous property of neural networks which manifests itself during sequential learning of a network. The neural network discards all previously gained knowledge or majority of it upon receiving new training input [16]. This problem can be understood as an instance of the stability-plasticity dilemma (low stability, high plasticity) and the methods for overcoming this problem represent a widely studied area of neural network research.

1.4 Engineering point of view

This thesis is more focused on the theoretical than the practical aspects. However, there are very interesting engineering problems related to the topic of online anomaly detection. Therefore, this section is included to peek inside the challenges associated with architecture and implementation of real anomaly detection systems.

Combination of all the required properties stated in Section 1.2.2 imposes not only theoretical challenges but engineering challenges as well. The architecture of underlying computer systems (*detection systems* or more generally *stream processing systems*) running the anomaly detection algorithms is subject to significant research effort in areas of distributed systems and software engineering.

Scenarios involving multiple independent real-time data streams originating in a large number of *source systems* (Internet of things (IoT), industrial sensors, e-commerce, social media etc.) require advanced computer systems with the following properties:

Availability, reliability, low latency In critical applications like healthcare, the reliable detection of anomalies is fundamental. The detection systems have to make sure that each data point is considered and none of them is lost or omitted (*at-least-one delivery guarantee*). Sometimes the delivery guarantee is even more strict and it is also necessary to ensure that no data point is delivered more than once. This so-called *exactly once delivery guarantee*, often required e.g. in the finance industry, is in general very hard to achieve in distributed systems. In e-commerce, on the other side, the requirements on delivery are lower, however, minimization of latency is extra important because it has been experimentally shown that the latency correlates with a decrease in revenues [17].

In reality, the requirements for low-latency or availability and reliability are inherently contradictory due to the *CAP theorem* [18]. Therefore, tradeoffs like sacrificing reliability in favor of low-latency or the other way around must be made in architecture and implementation of stream processing systems.

Also, the source systems (especially IoT devices) have often very limited capabilities. These devices cannot effort to wait long periods to complete sending a data point nor can they buffer the data points in case that stream processing system is unavailable. Therefore, any outage of the stream processing systems results in a loss of data.

High throughput, parallelization, distribution, and scalability To handle multiple large data streams, the underlying computer systems must be able to parallelize or distribute the workload to more than one physical machine (*compute instances*). This distribution requirement represents a challenge for design and implementation of anomaly detection algorithms. The algorithm must be able to efficiently exchange the gained knowledge about data points between compute instances.

The load of the detection system may also vary over time. The data stream can grow, which requires more compute instances (*scale out*). On the other side there might be some periods (e.g. night hours) of low activity of the source systems and in this situation, it is not feasible to run the same number of machines as at peak hours (the number compute instances must be *scaled down*). Because of the reasons above, the detection systems must be able to dynamically scale out and down.

Chapter 2

Methods of online anomaly detection in time-series

This chapter focuses on the theoretical description of several anomaly detection methods. At the beginning of the chapter, the general online anomaly detection system is introduced. Then, the properties of the specific methods are analyzed in the context of this system.

2.1 General online anomaly detection system

In this section, a general system for online anomaly detection will be described. Properties of the specific methods in the following text will be analyzed in terms of this system.

The detection process begins with feeding the newly incoming data point into the anomaly detector which then decides whether the data point is anomalous or not. This detection output is returned to the observer and also fed into another component called *concept drift detector*. The concept drift detector strives to reveal that a concept drift occurred. When this happens, the anomaly detector is discarded and relearned for the new concept by a learning algorithm. See Figure 2.1 for diagram of such system.

Some anomaly detectors are able to continuously adapt themselves [11] to the occurring concept drift. For such detectors, the detection of concept drift and relearning is irrelevant. The anomaly detector with such capability is considered superior to those that require relearning. Without discarding and relearning, much more information is preserved for making the subsequent detections. Also, from the practical point of view, the relearning can be time-demanding and requires to pause the detections.

The anomaly detector, the concept drift detector and the relearning algorithm all have the capability to access the last l ingested data points and use them for their internal operations. The anomaly detectors can use these data points to gain surplus information while making detections. The concept drift detector can combine these data points with detections of the anomaly detector to detect the drift. See the Section 2.1.1 for further discussion of the topic of the concept drift detection. The relearning algorithm uses the last data points for learning the new detector.

The process described above is repeated for each incoming data point. Before yielding the very first detection, all components of the system might need to be initialized. This is especially important for the anomaly detector which needs to gain some knowledge about the analyzed data. This initialization is usually done by providing a finite, relatively small, number of learning data points for learning [6].

In practice, the initialization phase can be omitted, but the first few detections should be considered to be highly unreliable.

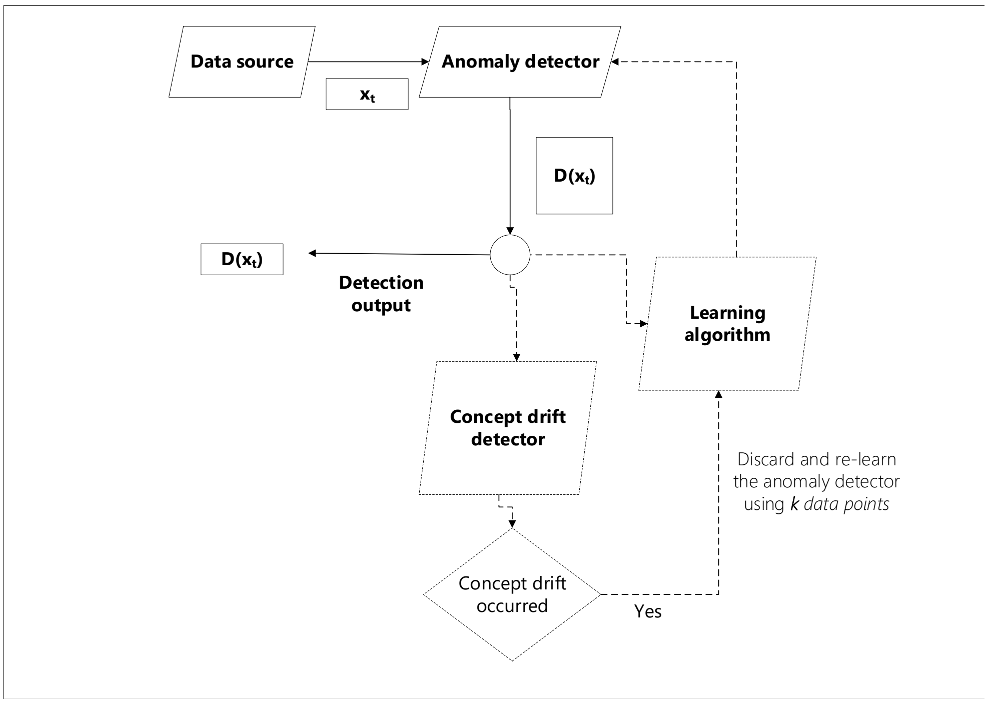


Figure 2.1: Diagram depicting the components of general online anomaly detection system. Components and relations with dashed lines are optional.

2.1.1 Detecting the concept drift

In literature, two basic methods for detecting concept drift in unsupervised online anomaly detection have been suggested. The third method combining multiple approaches is proposed. To the best of author's knowledge, this third

[6]For (semi-)supervised detectors, these data has to be labelled.

method has not been published yet. One of the methods detects the drift using solely information from data points while the other methods use the output of the anomaly detector. All of the methods require setting of one or more parameters.

2.1.1.1 Ratio of anomalies in the last l data points

This method keeps track of detections for the last l data points and an associated ratio R_A between anomalous and normal instances. The ratio can be defined as:

$$R_A = \frac{|\{\textbf{anomalous} \text{ data points from the last } l \text{ data points}\}|}{l}.$$

The concept drift is detected if $R_A > \varrho$ where ϱ is a parameter. This method is clearly suitable for a labelling anomaly detector and can be also adapted to handle data from a scoring anomaly detector. The validity of this method has been verified e.g. in [19].

Setting the correct value of the parameter ϱ might be problematic. Instead, this thesis proposes to use the complementary cumulative distribution function (CDF) of a normal distribution with mean equal to the expected ratio of anomalies in a time-series and variance equal to 1. Given that $F_X(x) = P(X \leq x)$ is the CDF of such distribution X and its complement $\bar{F}_X(x) = P(X > x) = 1 - F_X(x)$, the concept drift is detected if:

$$\bar{F}_X(R_A) < \tau,$$

where τ is a threshold probability given by some parameter. The intuition is that the drift is detected, if the value R_A of the ratio is improbably high.

2.1.1.2 Distribution of the last l data points

In this method, a static empirical distribution $\mathcal{S}$ is obtained from some set of initial data points. Another empirical distribution $\mathcal{D}$ is dynamic and is calculated from a window of the last l data points.

Next step is to perform a comparison of the distributions $\mathcal{S}$ and $\mathcal{D}$. Statistical tests like Kolmogorov–Smirnov test (K-S test) can be used for comparing the distributions. In case of the K-S test, the concept drift is detected if the null hypothesis (the distributions are the same) can be rejected on a level α where the α is a parameter. If the concept drift occurs, new static distribution $\mathcal{S}$ is obtained from the last l data points. This method is described e.g. in [20].

The beneficial property of this method is that no assumptions are made about the anomaly detector.

2.1.1.3 Distribution of anomaly scores for the last l data points

This thesis proposes to use the idea of comparing two empirical distributions as in 2.1.1.2, however, keep both distributions dynamic. Another change to the original idea is that the anomaly scores of the data points are used for obtaining the distributions instead of the data points themselves. This decision has its origin in [11] where the authors used a similar technique for the anomaly detection (not the concept drift detection).

The first distribution $\mathcal{L}$ is obtained from the anomaly scores of all last l data points. The second distribution $\mathcal{M}$ is obtained from anomaly scores of only from last m data points where $l \gg m$.

The distributions of anomaly scores in $\mathcal{L}$ and $\mathcal{M}$ can be compared to detect a concept drift. For example, the distributions can be compared by the well-known K-S test as in the previous method. The concept drift is again detected if the null hypothesis of the K-S test can be rejected on the level α which is a parameter.

Technically, only the anomaly labels can be used, but it would mean that the compared distributions have only 2 unique value. Therefore, this approach requires anomaly scores to make sense which might be problematic for some anomaly detection methods.

This method could be also extended by incorporating the idea of the previous method. Instead of comparing solely the distributions of anomaly scores, the multivariate distribution of both anomaly scores and data points could be used. However, other statistical test instead of K-S test must be used because K-S test does not handle multivariate distributions.

2.2 Anomaly detection methods

In this section, four specific methods for online anomaly detection will be introduced.

2.2.1 Moving averages

The moving averages are one of the simplest methods for online anomaly detection. In this thesis, two types of moving averages will be recognized: simple moving average (SMA) and weighted moving average (WMA).

The essential idea is to keep the average $A_{t,a}(l)$ of the last l values of some continuous attribute a of the incoming data points in each point in time t. The value of the attribute a of the next data point $\mathbf{x}_{t+1}$ is compared to the average $A_{t,a}(l)$. If the difference between these values exceeds a limit given by some parameter τ, the data point $\mathbf{x}_{t+1}$ is marked as anomalous. Alternatively, the normalized difference can be used directly as the anomaly score.

This approach can be straightforwardly extended to work with windows of size s and step 1 instead of individual data points. For each data point $\mathbf{x}_i$, the window W_i is created containing s data points and with the $\mathbf{x}_i$ as the last data point. Then the arithmetic mean from all values of the attribute a in the window is calculated and used. To keep the ongoing text clear, the value $v_{t,a}$ for data point $\mathbf{x}_i$ will be defined as:

$$v_{t,a} = \begin{cases} \mathbf{x}_{t,a} & \text{for individual data points,} \\ \frac{1}{s} \sum_{x_k \in W_t} \mathbf{x}_{k,a} & \text{for windows of size } s. \end{cases} \tag{2.1}$$

The generic moving average can be defined as:

$$A_{t,a}(l) = \frac{1}{\sum_{i=0}^{l-1} \kappa(i)} \sum_{i=0}^{l-1} \kappa(i) v_{t-i,a}, \tag{2.2}$$

where the $\kappa : \mathbb{N} \to \mathbb{R}$ is a scaling function. The SMA and WMA are the specializations of this generic formula. Their description can be found in the following subsections.

The simplest condition for anomaly detection can be defined as:

$$\left| \frac{A_{t,a}(l) - v_{t+1,a}}{A_{t,a}(l)} \right| \begin{cases} \notin (-\tau, \tau) & \implies \mathbf{x}_{t+1} \text{ is anomalous,} \\ \in (-\tau, \tau) & \implies \mathbf{x}_{t+1} \text{ is normal.} \end{cases}$$

A more advanced condition can make an assumption that the data points are normally distributed with the mean $A_{t,a}(l)$ and the variance $V_{t,a}(l)$ defined as

$$V_{t,a}(l) = \frac{1}{\sum_{i=0}^{l-1} \kappa(i)} \sum_{i=0}^{l-1} \kappa(i)(v_{t-i,a} - A_{t,a}(l))^2.$$

Then, the anomaly label of the value $v_{t-i,a}$ can be obtained as follows:

$$v_{t+1,a} \begin{cases} \notin \left(Q\left(\frac{\tau}{2}\right), Q\left(1 - \frac{\tau}{2}\right) \right) & \implies \mathbf{x}_{t+1} \text{ is anomalous,} \\ \in \left(Q\left(\frac{\tau}{2}\right), Q\left(1 - \frac{\tau}{2}\right) \right) & \implies \mathbf{x}_{t+1} \text{ is normal,} \end{cases} \tag{2.3}$$

where Q is the quantile function of the normal distribution $\mathcal{N}(A_{t,a}(l), \sqrt{V_{t,a}(l)})$.

This approach can be directly modified to use other, less popular, statistics like moving median or moving variance. Different kind of means (such geometric or harmonic) could be also used.

As stated above, this approach is very simple. However, for point anomalies, it is usable and in context of this thesis, it is considered as a baseline for comparison with other, more advanced methods.

2.2.1.1 Simple Moving Average (SMA)

Simple Moving Average is defined as the arithmetic mean over the last l values. The values are not scaled:

$$A_{t,a}^{SMA}(l) = \frac{\sum_{i=0}^{l-1} v_{t-i,a}}{l}.$$

For SMA, the κ is defined as $\kappa(p) = 1$.

2.2.1.2 Weighted Moving Average (WMA)

Weighted Moving Average is an arithmetic mean over the last l values scaled by their position in time. This scaling is defined by the function $\kappa(p)$ which is non-constant in this case.

The range of options for specific $\kappa(p)$ is very wide. In this thesis, three instances are defined and described.

Linearly Weighted Moving Average The newer values have a larger impact than the older ones. The difference in impact is given by a linear function. The formula for this average is as stated follows:

$$A_{t,a}^{LWMA}(l) = \frac{\sum_{i=0}^{l-1}(l-i)v_{t-i,a}}{\frac{l(1+l)}{2}}.$$

For LWMA, the κ is defined as $\kappa(p) = l - p$.

Exponentially Weighted Moving Average The exponentially weighted average is similar to the linearly weighted average but the values are scaled exponentially. The rate of decreasing of the impact for older values is given by a parameter $\alpha < 1$. The lesser the α, the lesser significance is assigned to the older values (the information is discarded faster):

$$A_{t,a}^{EWMA}(l) = \frac{\sum_{i=0}^{l-1} \alpha^i \cdot v_{t-i,a}}{\sum_{i=0}^{l-1} \alpha^i}.$$

For EWMA, the κ is defined as $\kappa(p) = \alpha^p$.

Non-monotonically Weighted Moving Average The previous two weighted averages favour the new data points before the older data points. Sometimes, this behaviour is unwanted because it can cause too much false anomaly detections in case that there are some lonely outliers in the data that should not be considered as an anomaly. This problem can be solved by

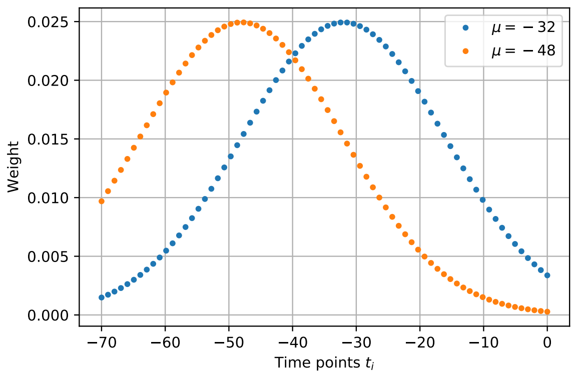


Figure 2.2: The plot showing two sets of weights for 64 data points. The x-axis represents time. The weights are based on a normal distribution with $\sigma = 256$ and various values of μ. Both sets of weights do not favor older or newer values in some simple manner. The blue weights ($\mu = -32$) are balanced between the older and newer points while the the orange weights ($\mu = -48$) are slightly biased towards older points.

not giving the largest weight to the newer points. A good example of a scaling function achieving this goal is a probability density function of a normal distribution $\mathcal{N}(\mu, \sigma)$ with appropriately chosen parameters:

$$\kappa(p) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(p-\mu)^2}{2\sigma^2}}.$$

In this case, especially important is the value of parameter μ because it directly controls the difference between the impact of older and newer data points. The Figure 2.2 illustrates the influence of the parameter μ on the distribution of scaling weights.

2.2.1.3 Concept drift detection and relearning

The moving average can be used both with and without the concept drift detector. Any concept drift detection approach from 2.1.1 can be used. The solution with concept drift detection requires some initialization on some finite set of data points and subsequent relearning on some finite set of last data points in case that the drift is detected.

However, in practice, the moving averages can be easily used without concept drift detection. Given the triviality of computation, the anomaly detector can use last data points to recalculate a value of the average each time a new data point arrives. In terms of the general anomaly detection system, this approach essentially boils down to very frequent relearning without concept drift detection.

2.2.1.4 Additional remarks

The principles of anomaly detection with moving average can be easily generalized to other statistical functions. It possible to use median, other kinds of means (harmonic, geometric), maximum, minimum, variance or even higher moments. Theoretically, any function morphing the windows to comparable scalar values can be used. The same goes also for the morphing the windows to scalar values at definition 2.1.

2.2.2 k-Nearest Neighbours (k-NN)

The k-nearest neighbours (k-NN) is a very well known algorithm for distance-based classification and regression. It is also known to be highly sensitive to the curse of dimensionality [21], [7]. This section describes the usage of k-NN for anomaly detection in time-series.

In anomaly detection terms, the k-NN is able to detect point anomalies in data with continuous attributes. More formally, the k-NN can work on any metric space (it requires a metric to be defined between all data points). Unlike e.g. moving averages, it also naturally handles multivariate data points.

Using the k-NN, the anomalies are searched for in a global manner [22]. Unlike Local Outlier Factor (LOF) method discussed below, the k-NN does not consider the local properties of analyzed data points. This means that if there is more than one natural grouping of data points in the analyzed dataset, the k-NN cannot distinguish between the groups and treat them together as one. Such behaviour does not take advantage of all available information and can result in insufficient performance. The more detailed discussion of these matters can be found in Section 2.2.3 – Local Outlier Factor. On the other hand, the LOF is more computationally demanding compared to k-NN. This is why both approaches are worth studying.

In the following paragraphs, three approaches to the anomaly detection using k-NN are described. They differ in various aspects but all of them operate on the last l data points in the analyzed time-series. All following approaches can be extended to work also with windows instead of individual data points, similarly as in Section 2.2.1.

To simplify following notation, function δ, scalar k and a set $\mathbb{L}_t$ will be defined as:

$$\begin{gathered}\delta : \mathbb{I} \times \mathbb{I} \to \langle 0, \infty) \text{ (distance)},\\ k \in \mathbb{N} \text{ (number of neighbours)},\\ \mathbb{L}_t = \{\mathbf{x}_i | i \in \mathbb{N} \wedge i > t - l \wedge i \leq t\},\\ \text{(last } l \text{ data points in a time-series at time } t\text{)}.\end{gathered}$$

2.2.2.1 Distance to the k-nearest neighbour

Given a parameter k and a metric δ, the anomaly score for the data point $\mathbf{x}_{t+1}$ can be defined as

$$\delta(\mathbf{x}_{t+1}, \mathbf{x}_k),$$

where $\mathbf{x}_k$ is the k-th data point from the set $\mathbb{L}_t$ ordered by value of $\delta(\mathbf{x}_{t+1}, \mathbf{x}_i)$ ascendingly [23]. To obtain the anomaly label, the distance can be compared against a threshold $\theta \in \mathbb{R}^{0,+}$. Then, the data point $\mathbf{x}_{t+1}$ can be labelled as anomalous if

$$\delta(\mathbf{x}_{t+1}, \mathbf{x}_k) > \theta.$$

2.2.2.2 Number of neighbours within a given distance

Given a parameter θ_1 and a metric δ, the anomaly score for data point $\mathbf{x}_{t+1}$ can be defined as the negative cardinality of the following set [24]:

$$\{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{L}_t \land \delta(\mathbf{x}_{t+1}, \mathbf{x}_i) \leq \theta_1\}.$$

The meaning of the cardinality can be rephrased as a number of data points from $\mathbb{L}_t$ within a given distance θ_1 from $\mathbf{x}_{t+1}$. The anomaly label can be obtained by comparing the cardinality to some parameter θ_2.

2.2.2.3 Average distance to the k-nearest neighbours

Given a parameter k and a metric δ, the anomaly score for data point $\mathbf{x}_{t+1}$ can be defined as [25].

$$|\mathbb{N}_k(\mathbf{x}_{t+1})|^{-1} \sum_{\mathbf{x}_i \in \mathbb{N}_k(\mathbf{x}_{t+1})} \delta(\mathbf{x}_i, \mathbf{x}_{t+1}),$$

where the neighborhood $\mathbb{N}_k(\mathbf{x}_{t+1})$ is defined as a set containing k data points from $\mathbb{L}_t$ with the lowest δ-distance from $\mathbf{x}_{t+1}$. As in previous two cases, the anomaly label can be obtained by comparison with some parameter θ.

2.2.2.4 Concept drift detection and relearning

No matter what specific method is used, the situation is similar to moving averages. No concept drift detection is required because the k-NN based methods do not need to keep some complex internal model. They support online updates just by adding points and removing the old ones from the analyzed set.

The analyzed set is not expected to be big, but if the computational performance of detections is an issue, the space-partitioning data structures like K-D trees [26] can be used to boost the detection speed. However, the usage of such structures must be carefully considered. For example, the additions and removals of points in the K-D trees are not constant-time operations. Also, the trees must be kept balanced after accepting updates which incur additional overhead.

When using the space-partitioning structures, it might be worth to employ some concept drift detection and avoid the overhead caused by the updates by making the detections based on a static set of data points. When a concept drift occurs, this static set is simply refreshed based on the last data points.

2.2.2.5 Additional remarks

The approaches based on pure k-NN will not be further pursued. Following effort is rather focused on description of more advanced method – Local Outlier Factor. However, the both methods share many concepts and therefore the information provided in this section will be used further.

2.2.3 Local Outlier Factor (LOF)

The Local Outlier Factor has been designed in 2010 to take an advantage of local properties of data points that the other similar approaches like k-NN (2.2.2) ignore. In the Figure 2.3 and its caption, there is an example of points in a 2-D space for which the k-NN based methods fail to detect all anomalies. The LOF incorporate the local properties of the data points by examining their neighborhood. Then, the anomaly score based on the (dis)similarity to the respective neighbourhoods is yielded.

The LOF has been proposed for general anomaly detection problems (not specifically to time-series). Therefore, the following definitions do not take the time-series into an account either. However, the extension for the time-series is trivial and will be described at the end of the section.

To fully describe the LOF, two supporting functions called reachability distance and local reachability density must be introduced. The *reachability distance* is a special distance function between two data points with smoothing effect. Original authors use this augmented distance to get rid of statistical fluctuations in distance between points that are too close to each other:

$$rd_k(\mathbf{p}, \mathbf{q}) = \max(\delta_k(\mathbf{q}), \delta(\mathbf{p}, \mathbf{q})) \text{ (reachability distance for } \mathbf{p} \text{ w.r.t. } \mathbf{q}\text{)},$$

where $\delta(\mathbf{p}, \mathbf{q})$ is an ordinary distance metric function (as in k-NN) and $\delta_k(\mathbf{p})$ is a distance of $\mathbf{p}$ from its k-th nearest neighbor based on the distance function δ.

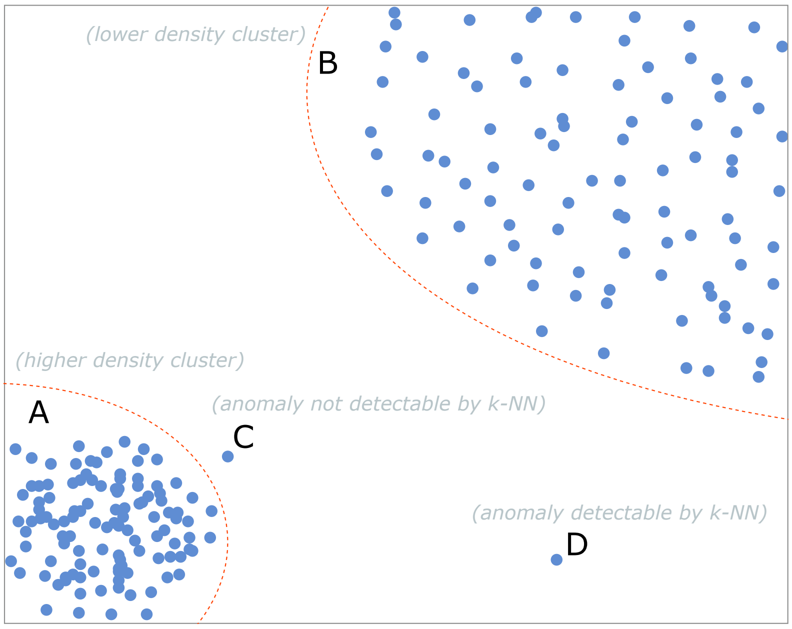


Figure 2.3: The figure shows artificially generated points in the further unspecified 2-D space. There are two clusters of points A and B. Also, there are two anomalies C and D. The anomaly D could be easily detected by k-NN based methods because it has significantly larger distance from the nearest neighbour than all other points. On the other hand, the anomaly C is hard to detect for k-NN because its distance to the nearest neighbor is similar as for points in cluster B. However, the C is clearly an anomaly with regard to the cluster A. This figure has only illustrative character and is based on an example in the original paper [22] where a more formal description can be found.

A side effect of this augmentation is, however, that the resulting function rd_k is not a metric because of its asymmetry.

Based on the reachability distance, the *local reachability density* for $\mathbf{p}$ can be defined as an inverse average of reachability distances for all neighbours of $\mathbf{p}$:

$$lrd_k(\mathbf{p}) = \left(\frac{\sum\limits_{\mathbf{q} \in \mathbb{N}_k(\mathbf{p})} rd_k(\mathbf{p}, \mathbf{q})}{|\mathbb{N}_k(\mathbf{p})|} \right)^{-1} \quad \text{(local reachability density of } \mathbf{p}\text{)},$$

where the $\mathbb{N}_k(\mathbf{p})$ is neighborhood set containing k-nearest neighbours of $\mathbf{p}$.

With these definitions, the Local Outlier Factor (LOF) for a data point $\mathbf{p}$ is equal to the average of ratios of local reachability densities w.r.t. to all neighbours of $\mathbf{p}$:

$$LOF_k(\mathbf{p}) = \frac{\sum\limits_{\mathbf{q} \in \mathbb{N}_k(\mathbf{p})} \frac{lrd_k(\mathbf{q})}{lrd_k(\mathbf{p})}}{|\mathbb{N}_k(\mathbf{p})|} \text{ (local outlier factor of } \mathbf{p}\text{).}$$

Intuitively, it can be said that LOF of a data point is low if the local reachability density of neighbouring data points is similar.

By default, the LOF returns an anomaly score. Conversion of the score to the anomaly label can be done by thresholding the LOF score. However, the LOF score is unbounded and therefore setting of the threshold is problematic. One way to obtain this threshold, implemented in the Scikit-Learn library [27] [7], is to calculate the LOF score for all training data points and then set the threshold τ equal to the $(100 \cdot \varkappa)$-th percentile of the calculated LOF scores for all data points (linearly interpolated if an exact value is not available). The $\varkappa$ is some bounded parameter from $\langle 0, 1 \rangle$ [8]. Then, the data point $\mathbf{x}_{t+1}$ is considered anomalous if

$$LOF_k(\mathbf{x}_{t+1}) \geq \tau. \tag{2.4}$$

The LOF can be used also on time-series. In that case, all following definitions hold but the neighborhood $\mathbb{N}_k(\mathbf{x}_{t+1})$ for calculating the $LOF_k(\mathbf{x}_{t+1})$ which is obtained from only the l last data points $\mathbf{x}_{t-l+1}, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t$.

The topic of concept drift detection and relearning for LOF will not be discussed because the same arguments as for k-NN based methods (section 2.2.3) apply.

2.2.4 Isolation forests

Isolation forests are an ensemble method specifically designed for general anomaly detection by Liu and Zhou [28]. Later, the isolation forests have been successfully used also for an online anomaly detection in sequence data [19].

At the beginning of this section, the fundamental ideas of isolation forests and relation to the other tree-based methods are introduced. Then, the formal definition of the anomaly detection process using the isolation forests is presented. At last, the properties related to the concept drift and relearning are discussed.

The isolation forests are based on an idea of the explicit isolation of anomalous data points. This approach is fundamentally different to a great number of methods that models the profile of normal instances instead of the profile of anomalous instances.

[7]`https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/neighbors/lof.py#L195`

[8]In the Scikit-Learn library, the parameter $\varkappa$ is called the *contamination*.

The core idea is to perform recursive partitioning of the data point space until all data points are isolated in their own single partition (subspace). This partitioning is done independently multiple times (the forest (ensemble) consists of multiple trees) and the assumption is made that the anomalous data points will be in average isolated significantly earlier than the normal data points. This assumptions, similarly to other detection methods, heavily relies on the minor representation of the anomalies among the data points and on the significant deviations in the values of (some) attributes.

The partitioning procedure for each tree is done in a stochastic manner and each partitioning step has two phases. First, a split attribute a is selected at random from all available attributes. Then, a random value from a proper range is uniformly sampled. This randomly sampled value (split point) is then used to split the data point space in the dimension of a. This partitioning is done recursively. Therefore, each partitioning step is applied on a subspace of the original data point space. This partitioning scheme can be very conveniently represented by the binary search tree (BST) data structure. The example of such partitioning can be found in the Figure 2.4.

The process of forest training and detecting an anomaly will be formally described later. Only a brief intuition is presented at this moment: multiple trees (partitionings) are created from the training dataset. Then, the newly incoming data point is evaluated by measuring the distance of the leaf that the data point would belong into from the root of the tree (height). More specifically, an average height across all trees in the forest is considered. Smaller height means more anomalous data point.

The isolation forest consists of several so-called *isolation trees* [9]. Each isolation (binary) tree represents a single instance of partitioning of the data point space as indicated in the previous paragraph. This composition of multiple isolation trees forms ensemble model called the *isolation forest.* For each tested data point, the isolation forest yields an anomaly score which is a combination of the outputs of all trees in the forest. Therefore, the isolation forest can be considered as a *bagging* ensemble model. The ensemble nature of the isolation forests enables the existence of so-called *expert trees* that can be narrowly specialized for different kinds of anomalies.

Unlike k-NN or LOF, no distance or density measure is required to be defined and calculated which also implies the lower computational costs. On the first sight, the isolation trees seem to be similar to classic decision trees used for classification or regression. However, the more detailed analysis shows that only common element is the usage of the binary search tree data structure. There is a lot of research regarding the usage of classic decision trees for online anomaly detection. The Hoeffding trees [29] are a good example. However, these methods are supervised and need both anomalous and normal labels. Therefore, they are not studied further in this thesis. In the end of this

[9]100 trees have been used in the original paper

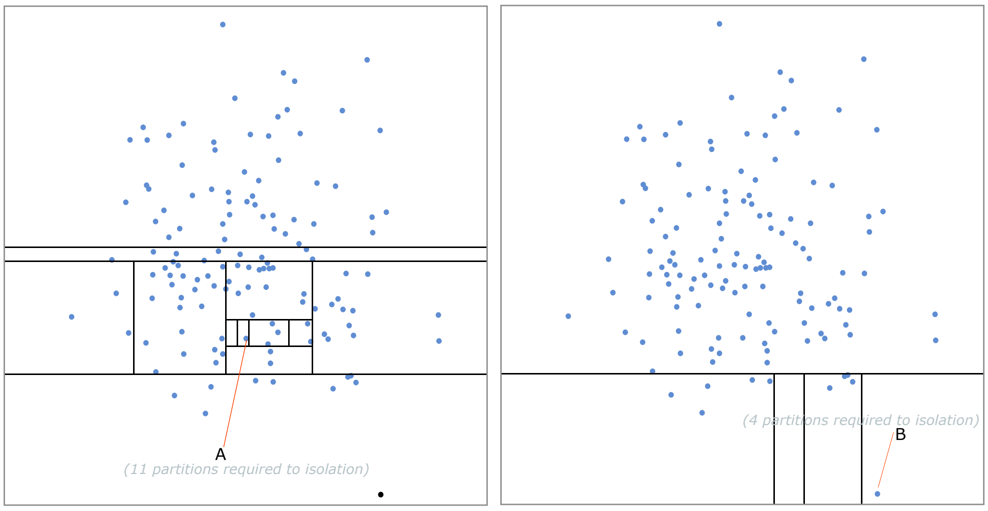


Figure 2.4: This figure based on [28] shows artificially generated points in the further unspecified 2-D space. There is a single clearly visible cluster of normal instances with several anomalous instances around its borders. Isolation of a normal point (A) from within the core of the cluster took 11 recursive random partitions. On the other hand, isolating the anomalous point (B) took only 4 such partitions.

section (2.2.4.2), more such examples can be found.

The formal definition of an isolation tree follows. The isolation tree T is a *proper tree* graph with recursive structure. Each node can have zero, resp. two descendant nodes. Such nodes are called *external*, resp. *internal*. Every node has exactly one predecessor node except the single *root node* that has no predecessor node. If a node N is internal it contains a pair of properties (q_N, p_N). The q_N is a *split attribute* and the p_N is split value with domain given by q_N. This recursive tree structure with properties unambiguously defines the partitioning of the data point space.

The height $h_T(N)$ of the node N in the tree T is defined as a number of edges from the root node to the node N. During the training, the tree is grown until all data points are isolated (in terms of the induced space partitioning) or the growing can be reduced by limiting the maximum value of h_T for each node (the node N is not further divided if the $h_T(N)$ reaches some threshold. See details in subsection 2.2.4.1.). When the tree T is constructed, each newly incoming data point can be unambiguously assigned to one of the external nodes. The *path length* $h_T(\mathbf{x}_i)$ for data point $\mathbf{x}_i$ is equal to the $h_T(M)$ where M is the node ingesting the $\mathbf{x}_i$.

The anomaly score $s_F(\mathbf{x}_i, n)$ for data point $\mathbf{x}_i$ in a forest F consisting of t trees $T_1, T_2, \ldots, T_t$ built from n data points is defined as:

$$s_F(\mathbf{x}_i, n) = 2^{-\frac{\frac{1}{t}\sum_{k=1}^{t} h_{T_k}(\mathbf{x}_i)}{c(n)}},$$

where the $c(n)$ is a normalization function in n defined as:

$$c(i) = 2H(i-1) - \frac{2(i-1)}{i},$$

where the $H(i)$ is the i-th harmonic number that can be approximated using the Euler–Mascheroni constant as $H(i) \approx ln(i) + 0.5772156649$. The value of $c(n)$ is based on the analysis of properties of classic Binary Search Trees (BST) [30] [10]. This definition of anomaly score intuitively means that the lower $h_T(\mathbf{x}_i)$ is, the data point $\mathbf{x}_i$ is more anomalous.

Conversion of the anomaly score to the anomaly label is not discussed in the original paper. During, the ongoing experiments the very same approach as for LOF (2.4) will be used (calculating the anomaly score for all training data points, obtaining a threshold based on a value given by some percentile given by a bounded parameter and then comparing the anomaly score of the new data points to this threshold). This approach is also implemented in the Scikit-Learn library [11].

2.2.4.1 Training

In this subsection, the training procedure indicated above is explained in more detail. The isolation forest is grown by constructing individual isolation trees. Each tree is built independently to the others in the following way.

1. Randomly select ψ *(subsampling size)* data points from the training dataset into the sampled training dataset $\mathbb{S}$.

2. Build an isolation tree from $\mathbb{S}$:

   a) Randomly select an attribute from the set of available attributes.

   b) Randomly select the attribute's split point (value of the attribute).

   c) Divide the $\mathbb{S}$ into two subsets $\mathbb{S}_l$ and $\mathbb{S}_r$ by comparing the relevant attributes of the data points with the split point ($\leq$, $>$).

   d) Repeat this procedure recursively for both $\mathbb{S}_l$ and $\mathbb{S}_r$. The selection of the split point (its minimal and maximal value) is now constrained by the current position in the recursion tree.

[10] It is an equivalent of the average length of unsuccessful search in the BST.

[11] `https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/ensemble/iforest.py#L203`

   e) This recursive process is stopped when the set can be split no more or if the height of the recursion tree reaches the value $\log_2(\psi)$ (average expected tree height). The reasoning behind this constraint is that the data points with the height larger than the average are not interesting for the isolation tree.

3. Repeat this procedure including the sampling of $\mathbb{S}$ t-times to obtain t isolation trees.

Authors of the original paper experimentally concluded that a good baseline parameter settings are $\psi = 256$ and $t = 100$. Larger subsampling size or larger number of trees do not increase the detection performance by a significant amount. On the contrary, larger values increase computational complexity.

A modification of this training process is proposed: The step 2b could incorporate potential knowledge of the distribution of the attribute's values. Therefore, the split point could be sampled from other distribution than the uniform one.

2.2.4.2 Concept drift and relearning

A straightforward way how to use the isolation forests for anomaly detection on streaming data with expected concept drift has been studied by [19]. Given that isolation forests yield an anomaly score for each tested point, any concept drift detector can be used together with relearning on the last l data points.

An interesting property of the isolation forests that has been experimentally verified by the original authors is that the absence of anomalous instances during the training phase does not influence the performance. This property allows to safely re-train the forest at any point on the last l data points.

An incremental approach to incorporating knowledge into the isolation forests is not known. There is a lot of research regarding the incremental learning of the classic decision trees (Hoeffding tree or VFDT [29], ID4 [31], ID5R [32]). Some of these methods are even able to deal with the concept drift (CVFDT [33]). However, all of them heavily rely on the supervised learning approach and therefore they were not further examined in this thesis.

2.2.5 Hierarchical Temporal Memory (HTM)

This section explores a method that originated in the domain of computational neuroscience and it is based on a special type of neural network specifically designed for learning and prediction of sequence data. The essence of this network is tightly coupled with the morphology of the *neocortex* which is an important part of the mammalian brain. For humans, it provides capabilities like speech understanding, visual object recognition or planning. This method, called Hierarchical Temporal Memory (HTM), strives to resemble the structure and functionality of the neocortex. Despite the very complex function of the neocortex, its structure is quite uniform and clear. Thus, the HTM utilizes the idea of combining the functionality of a large number of relatively simple components in order to get very complex capabilities, similarly as better known perceptron-based (deep) neural networks. However, the architecture of HTM and common artificial neural networks (ANN) is different. Unlike perceptron-based ANNs, the time is a crucial concept for the HTM. Also, the HTM stores a lot of information about encountered patterns and sequences [34] which is also unusual for the common ANNs.

The origins of HTM theory can be traced to 2004 [35]. In 2005, a company named Numenta [12] has been founded by the original authors with the aim to transform the theoretical framework of HTM into a software system. Since then, there has been a series of papers by the original authors as well as Numenta researchers: [34] (2011), [36] (2015), [37], [38], [39], [40] (2016). The latest publication from 2017 deals specifically with using the HTM for online anomaly detection [11].

Given the very complex neuroscience background required for the proper understanding of the HTM, more detailed formal description of the HTM is omitted in this thesis. Only the very basic concepts that are necessary for ongoing experiments are described.

2.2.5.1 Principle of HTM-based anomaly detection and sparse distributed encoding

From a simplified point of view, the HTM network accepts a sequence of special binary vectors as its input. Vectors in this input sequence is passed through the network one by one . After each passed vector, a prediction of the following vector is produced. In HTM-based anomaly detection, the predicted vector and the real vector from the next step are compared and an anomaly score is yielded based on their difference [11]. With each input data point, the HTM continuously updates its internal state and thus it should be able to automatically deal with a concept drift. Unlike previous methods, the concept drift detection and relearning is therefore irrelevant for the HTM.

Sparse distribution representation (SDR) The crucial part of the HTM is encoding of the input data points into sparse binary input vectors called *sparse distributed representation.* The authors of HTM implementation proposed multiple encodings for various data types [41] and explored various properties of such encodings [36]. For this thesis, only encoding of real numbers is important. The idea of this encoding is explained in the Figure 2.5. Basically, the binary vector is aligned with the expected range of values and only bits that are topologically close to the encoded value are set. Such en-

[12]`https://numenta.org`

coding requires four parameters: maximum expected value, minimal expected value, length of the binary vector and number of active bits. The length of the binary vector directly determines the resolution that the real values will be treated with.

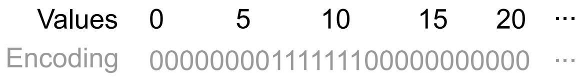


Figure 2.5: The figure illustrating the principle how the real values are encoded into the sparse binary vectors in HTM. In the figure, the value 9 is encoded.

Detecting anomalies The anomaly score for a data point $\mathbf{x}_t$ in a time point t is derived from the difference of the sparse encoding representation of the data point, $e(\mathbf{x}_t)$, and HTM's prediction for that point, $\pi(\mathbf{x}_{t-1})$. More formally, the anomaly score is inversely proportional to the number of common bits of the binary vectors $e(\mathbf{x}_t)$ and $\pi(\mathbf{x}_{t-1})$:

$$s_t = 1 - \frac{\pi(\mathbf{x}_{t-1}) \cdot e(\mathbf{x}_t)}{pop(e(\mathbf{x}_t))},$$

where $v_1 \cdot v_2$ is a dot product of some vectors v_1 and v_2 and $pop(v)$ [13] is a number of 1 bits in a vector v. This score is bounded to $\langle 0, 1\rangle$ and can be used directly for yielding anomaly scores or labels. However, authors of this method introduce the concept of *anomaly likelihood* that should be better in dealing with very noisy data. To calculate the anomaly likelihood, it is necessary to keep two windows of length W and $\tilde{W}$ with last yielded anomaly scores where $W \gg \tilde{W}$. The anomaly scores in these windows are modelled as a normal distribution with following parameters:

$$\mu_t = \frac{\sum_{i=0}^{W-1} s_{t-i}}{W}, \quad \tilde{\mu}_t = \frac{\sum_{i=0}^{\tilde{W}-1} s_{t-i}}{\tilde{W}}, \quad \sigma_t^2 = \frac{\sum_{i=0}^{W-1} (s_{t-i} - \mu_t)^2}{W-1}.$$

Then the anomaly likelihood L_t is defined as:

$$L_t = 1 - \mathcal{Q}(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}),$$

where $\mathcal{Q}$ is Q-function or tail distribution function of standard normal distribution. A data point $\mathbf{x}_t$ is labelled as anomaly if $L_t \geq 1 - \tau$ where τ is a threshold parameter.

[13] population count

Chapter 3

Experiments

The purpose of this chapter is to evaluate reviewed anomaly detection systems. In the first part, the design of experiments is presented. It includes the selection of relevant performance metrics and datasets as well as formalization of the overall experimentation process. In the second part of the chapter, selected anomaly detection systems are tested and compared according to the previously defined procedures.

3.1 Experiment design

This section starts with discussion and selection of performance metrics, continues with description of datasets used for testing and is finalized with definition of conditions under which the detection systems will be tested.

3.1.1 Performance metrics

The anomaly detection can be understood as a special case of (binary) classification problem on data with extremely unbalanced classes. This means that the well-known metrics based on the confusion matrices are usable also in the context of anomaly detection.

Confusion matrix Before discussing the specific metrics, the definition of the confusion matrix that is used in the following paragraphs of this thesis is presented. The confusion matrix is 2×2 matrix in the following form:

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix},$$

where TP is number of anomalous data points correctly labelled by the detector (*True Positives*), FP is the number of normal data points incorrectly labelled as anomalous (*False Positives*), FN is the number of anomalous data points incorrectly labelled as normal (*False Negatives*) and TN is the number of normal data points correctly labelled as normal (*True Negatives*).

Given the huge expected disproportion between classes (anomalous and normal data points), some popular classification metrics are unusable. For example, *accuracy ACC* defined as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

is completely irrelevant for anomaly detection. Even the trivial anomaly detector (null detector) that is labeling every data point as normal ($TP = 0$) would have very high accuracy because

$$ACC_{null} = \frac{TN}{TN + FN}$$

and the number of anomalous instances is assumed to be much lower than number of normal instances and therefore $TN+FN \approx TN \implies ACC_{null} \approx 1$.

On the other side, the *recall* metric (sometimes referred to as *sensitivity* or *true positive rate*) and *false positive rate* (*fall-out*, *FPR*) are very suitable metrics because they describe properties that are naturally expected from a good anomaly detection system:

$$\text{recall} = \frac{TP}{TP + FN},$$
$$\text{false positive rate} = \frac{FP}{FP + TN}.$$

The description of other metrics directly or indirectly derived from the confusion matrix follows.

F_1 **score** Comparing different anomaly detection systems using the recall and false positive rate is challenging because systems with higher recall tend to have lower FPR and the other way around. Therefore, it is suitable to combine these metrics into a single one. Traditional example of such combination of two different metrics is the F_1 score defined as harmonic mean of recall and precision:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall},$$

where precision is equal to the $\frac{TP}{TP+FP}$. In this thesis, the same idea is used for recall and false positive rate instead of recall and precision:

$$F_1 = 2 \cdot \frac{(1 - FPR) \cdot recall}{(1 - FPR) + recall}.$$

In the rest of the thesis, the F_1 score will refer to this modification.

Area Under Curve (AUC) Most anomaly detectors require definition of some threshold value that significantly influences the resulting recall and false positive rate. However, the impact on the recall and FPR is exactly the opposite. Two edge cases are recognized: threshold value that causes all data points to be labelled as anomalous ($recall = 1$, high FPR) and the opposite value that causes all data points to be labelled as normal ($recall = 0$, $FPR = 0$) [14]. Important information about the properties of an anomaly detection system can be gained by observing the changes in its behaviour (relation between $recall$ and FPR) while modifying the threshold value. This observation can be formalized by plotting the ROC[15] curve and calculating the area under this curve (AUC). The ROC can be obtained by simply plotting the 2-D points with coordinates given by $x \to FPR, y \to recall$ for various threshold settings and applying the trapezoidal rule. This definition implies that anomaly detectors with higher AUC are better (intuitively, higher recall is gained with lower increase in FPR). An example for ROC with AUC can be found in the Figure 3.1.

Usually, the points for ROC curve (performances of trained models) are generated by adjusting single threshold parameter of a learning algorithm. With increasing threshold, the recall and false positive rate should be therefore both non-decreasing. However, this assumption is not valid for the anomaly detection system with concept drift detection. Any change to the threshold parameter can significantly influence not only the anomaly detector itself but also the concept drift detector and by extension the relearning procedure (when a concept drift is detected, new model is learned on a dataset that is changing in time). Therefore, multiple different models are effectively tested together during a single evaluation pass. This can cause the recall and FPR to be changing non-monotonically with the increasing threshold.

To be able to obtain meaningful ROC and AUC, only the points that are included in the Pareto set are considered (Pareto set with preference for larger recall values and smaller FPR values). Illustration of this idea can be seen in the Figure 3.1, where the excluded points are marked with red color.

NAB Score Another interesting metric that is not widely known and is specifically designed for benchmarking streaming anomaly detection systems is the *NAB score* (Numenta Anomaly Benchmark Score) [42]. It is also a

[14]Given that the used dataset contains at least one normal and one anomalous data point

[15]Receiver Operating Characteristics

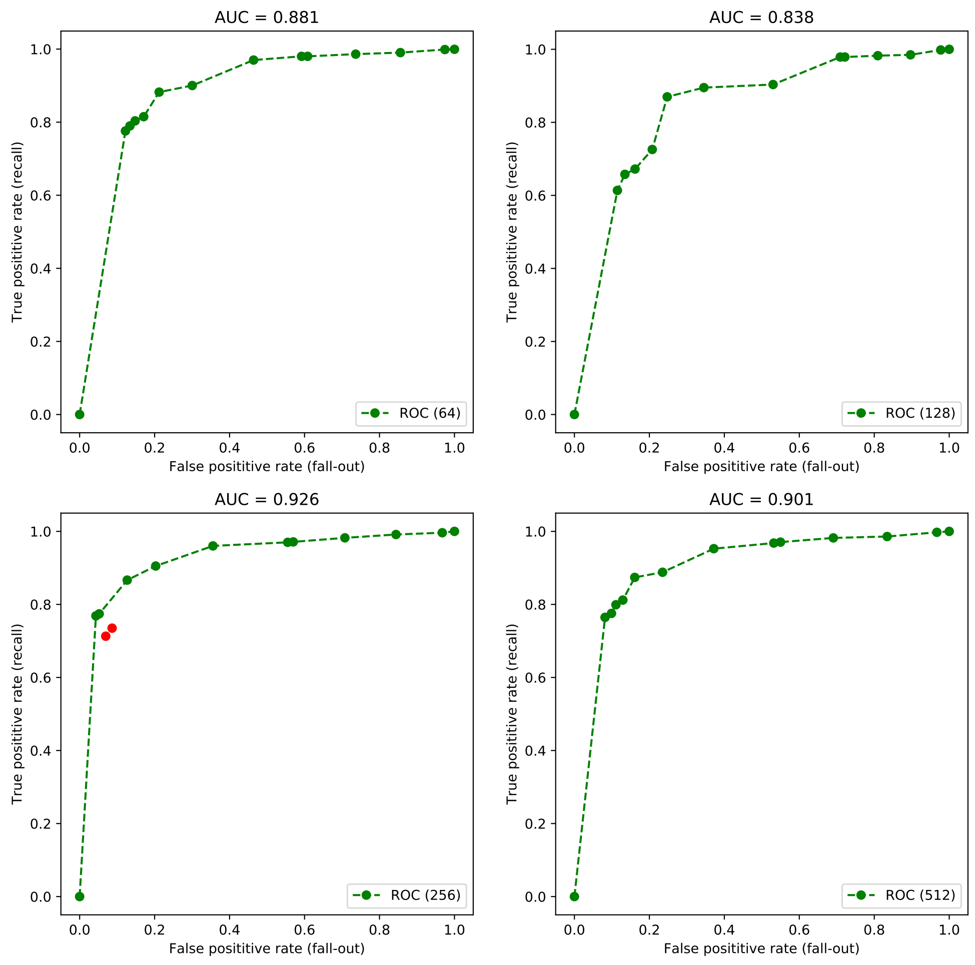


Figure 3.1: Four plots serving as an example of ROC curves with calculated AUCs. Specifically, these are ROC curves for moving average detector with window sizes $\sigma = 64, 128, 256, 512$. Excluded points (those that are not included in the Pareto set) are marked with red color.

confusion matrix-based metric that combines TP, FP, FN into an single number. Also, it can take into an account the point of a detection in time (the earlier detections are better). The formal definition of this score is out of the scope of this thesis and can be found in the original paper. Only the basic version without considering the point of a detection in time is briefly described.

The NAB score is parameterized with so called *application profiles* that enable to measure the detection performance with regard to expected application. The application profile is a vector of three positive coefficients $AP = (\alpha_{TP}, \alpha_{FN}, \alpha_{FP})$. The basic definition of a NAB score for a given detector $\mathcal{D}$, a dataset X and an application profile AP is the following:

$$nab(\mathcal{D}, X, AP) = \alpha_{TP} \cdot TP - \alpha_{FN} \cdot FN - \alpha_{FP} \cdot FP. \tag{3.1}$$

Examples of the application profiles can be $AP_1 = (2, 3, 1)$ and $AP_2 = (1, 1, 2)$. The AP_1 emphasizes that all maximum number of anomalies are catched whereas the AP_2 lowers the number false alarms.

The resulting NAB score itself is very hard to interpret (it largely depends on the underlying dataset). However, the score can be used to rank detectors on a given dataset.

The extension of this metric that includes the exact point of a detection in time requires that the testing data does not only contain the ground-truth labels for each anomaly but also a definition of anomalous windows centred around each anomaly. Then, the weighting of TP in 3.1 is extended by multiplication by a value of a sigmoidal logistic weighting function of time which is defined over each anomalous window and is decreasing. Only the first anomaly in each anomalous window is considered and given that the the sigmoidal function is decreasing, earlier detections are weighted more. This idea is illustrated in the Figure 3.2.

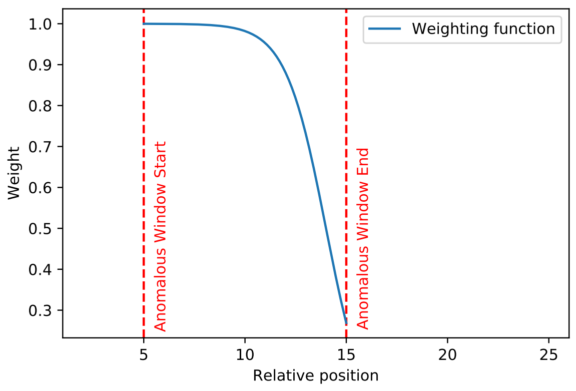


Figure 3.2: Plot illustrating the purpose of sigmoidal weighting function used in NAB scoring that assigns larger weights to earlier detection (w.r.t. to position in the anomalous window).

The F_1 score, recall, false positive rate and AUC are all well-known, interpretable metrics suitable for anomaly detection and therefore they are chosen to be used for evaluation of all performed experiments. Final ranking of the detection systems is done by the F_1 score. Regarding the NAB score, only the basic variant is calculated during the experiments as a secondary metric.

3.1.2 Datasets

All anomaly detection methods, concept drift detection methods and parameters were tested on six distinct fully labelled dataset. First two datasets are provided by Yahoo! Research [43]. Remaining four datasets comes from Numenta Anomaly Benchmark (NAB) corpus [42]. Both real and synthetic datasets are represented. Each dataset contains 1 to 100 distinct time-series.

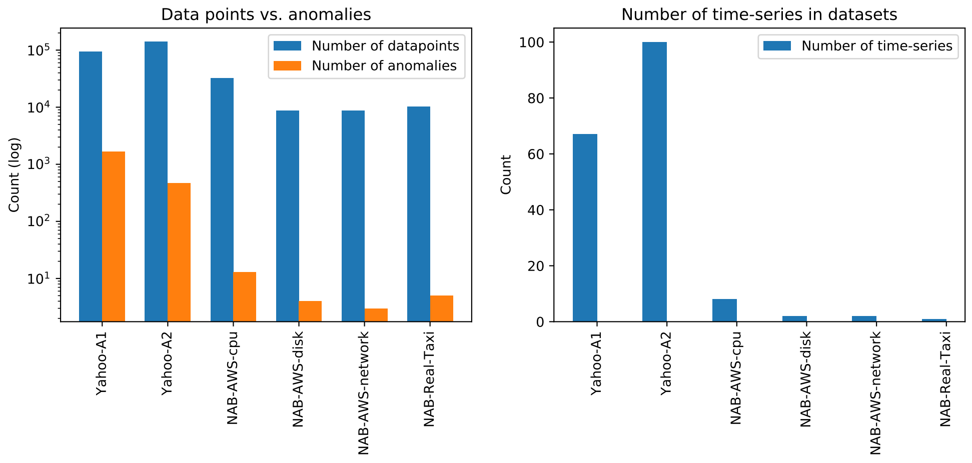


Figure 3.3: Bar charts with basic statistics for each dataset.

- *Yahoo-A1* Dataset is based on the real Yahoo! production traffic and is hand-labelled.
- *Yahoo-A2* Artificial time-series with introduced anomalies.
- *NAB-AWS-Cpu* Hand-labelled AWS (Amazon Web Services) server metrics for EC2 instances (CPU load percentage).
- *NAB-AWS-Disk* Hand-labelled AWS server metrics for EC2 instances (Disk I/O in bytes per second).
- *NAB-AWS-Network* Hand-labelled AWS server metrics for EC2 instances (Network I/O in bytes per second).
- *NAB-Real-Taxi* Number of NYC taxi passengers in the 30 minutes intervals. For each anomaly, the exact real cause is known: NYC marathon, Thanksgiving, Christmas, New Years day, and a snow storm.

All datasets (time-series) are preprocessed into the form of CSV files containing three columns: *timestamp*, *value*, *is_anomaly*. Each time-series file starts with the header (column names) and then there is one datapoint with timestamp and label per line. Number of datapoints as well as number of anomalies can be found in Table 3.1 or in Figure 3.3.

Dataset	# of TS	# of datapoints	# of anomalies	Mean TSL
Yahoo-A1	67	94866	1669	1416
Yahoo-A2	100	142100	466	1421
NAB-AWS-cpu	8	32256	13	4032
NAB-AWS-disk	2	8762	4	4381
NAB-AWS-network	2	8762	3	4381
NAB-Real-Taxi	1	10320	5	10320
	180	297066	2160	

Table 3.1: Table containing basic statistics about used datasets. TS stands for Time-Series and TSL for Time-Series Length.

3.1.3 Process

This section defines the process that is executed for each evaluated anomaly detection system. The decisions about the architectures of tested detection systems, parameter settings, and interpretation of results are discussed.

Architecture of evaluated detection systems and experiments workflow All evaluated systems fit into the definition of general online anomaly detection framework described in the beginning of the second chapter of this thesis. Therefore, there are initialized with some finite set of data points and then all the remaining data points from test dataset are fed into the detection system one-by-one. For each data point (except the initial ones), a detection label is yielded by the detection system. This label is compared with the ground-truth label and the confusion matrix is properly updated based on the result of this comparison (e.g. if the detection label is *anomaly* and the ground-truth label is *normal*, the FP (false positives) are incremented by 1).

After yielding a detection label, the procedure for concept drift detection is executed and if a drift is detected, the detection system is relearned on the last window of data points (relearning window). Four concept drift detection methods are tested:

- Ratio of anomalies in the last l data points with usage of CDF as defined in 2.1.1.1. The l chosen to be equal to the size of the relearning windows and threshold $\tau = 0.3$.
- K-S test applied to the distributions given by current and initial window (2.1.1.2). Length of compared windows is equal to the length of the relearning window. The p-value is chosen to be 0.05.
- Null concept drift detector that does detect any drifts.
- Trivial concept drift detection with relearning after each incoming data point.

The values of the concept drift detection parameters stated above were chosen to yield reasonable results during the exploratory testing of several anomaly detection methods. More rigorous choice based on some form of grid-search would be more appropriate. However, it would require infeasible amount of another calculations. The available computational resources were rather used for testing the parameters of anomaly detectors.

The concept drift detection and relearning is not incorporated in experiments with methods that are specifically designed not to require the concept drift detection.

Parameters of anomaly detectors For each anomaly detector, following parameters and their values are used:

- *Window length* $\in \{16, 32, 64, 128, 256, 512\}$. The maximum value 512 has been chosen with respect to the length of datasets.
- Expected ratio of anomalies in a dataset (*contamination*) $\in$ {0, 0.005,0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.48, 0.5, 0.64, 0.8, 0.96, 1}. The choice of this values has been made from the following reasons:
  - The contamination parameter has direct impact to sensitivity to anomalies for almost all detectors. Therefore, the contamination values sample all interval from 0 to 1 in order to properly construct the ROC curve and calculate AUC. Exact usage of the contamination parameter is explained for each method separately.
  - Real contamination in data is expected to be low, therefore the contamination interval is sampled more granularly for lower values.

The parameters that are unique for individual tested anomaly detection methods are described in relevant sections below.

Calculation of performance metrics For each dataset and anomaly detection method, a grid search is performed in the space of concept drift detection methods, window lengths, contaminations and values of special methods' parameters. For each anomaly detection method, at least 60480 experiments (passing a whole time-series through detection system with given configuration) must be made. For some methods this number is even five times higher.

$$\begin{gathered}\{\text{time-series}\} \times \{\text{window lengths}\} \times \{\text{contaminations}\} \times \\ \times \{\text{concept drift detection methods}\} \\ \rightarrow 180 \cdot 6 \cdot 14 \cdot 4 = 60480\end{gathered}$$

In context of a single dataset and single configuration, the combined confusion matrix from all time-series is obtained and then the relevant metrics are calculated (recall, false positive rate, F_1 score, AUC and NAB score).

Comparison of anomaly detection methods For each dataset and each anomaly detection method, the best configurations in three categories are selected. The categories are following:

1. Configurations with the highest F_1 score.
2. Configurations with the highest AUC.
3. Configurations with the highest NAB score with application profile ($\alpha_{TP} = 1, \alpha_{FN} = 1, \alpha_{FP} = 0.25$).

These best performing configurations in given categories are then compared between each other in context of a given dataset.

3.1.4 Configuration of evaluated detection systems

This subsection describes the configuration and parameters that are specific for given anomaly detection methods. Following anomaly detection methods are tested: Moving Average, LOF, Isolation Forest and HTM. The method of k-nearest neighbours is omitted because the LOF is a similar distance-based method that is more suitable for the task of anomaly detection.

All code used for evaluation, concept drift detection as well as anomaly detection by moving average has been written in Python using the SciPy ecosystem (NumPy, SciPy, Pandas, IPython, Jupyter) [44] libraries. For LOF and IsolationForest, the Scikit-Learn implementations were used [27]. For HTM, the Numenta implementation available from GitHub [16] has been used.

3.1.4.1 Moving Average

The moving average anomaly detector is evaluated in two variants with different scaling functions (as defined in 2.2):

- Simple moving average (constant scaling function: $\kappa(p) = 1$),
- Exponentially weighted average (exponential scaling function: $\kappa(p) = 0.8^p$).

The anomaly detection is based on the quantile function (2.3) with threshold parameter $\tau = \mathit{contamination}/2$.

[16] `https://github.com/numenta/nupic`

3.1.4.2 Local Outlier Factor

Local Outlier Factor method requires a single parameter k defining the number of considered neighbours. Following values of this parameter were tested: 1, 2, 4, 8, 16. The more neighbours are considered, the more information is available to the detector. Therefore, the expected impact of this parameter is that the performance will be increasing with the increasing number of neighbours. The parameter $\varkappa$ used for calculating the thresholding percentile is set to $\varkappa = 1 - \text{contamination}$.

3.1.4.3 Isolation Forest

For isolation forest, there is a parameter defining the number of trees in the forest (size of the ensemble). Original authors suggests to use 100 trees in the forest as a baseline that covers most cases. To verify this suggestion, the following ensemble sizes are tested in this thesis: 25, 50, 100, 125.

3.1.4.4 Hiearchical Temporal Memory

For the ongoing experiments, there is only one available implementation, provided by the Numenta company as mentioned in the previous description of the HTM. Following values of the resolution parameter are tested : 0.01, 0.001, 0.0001, 0.00001. The minimal value and maximal expected values in general have to be set according to domain knowledge about the data. In the experiments, this values are set based on some initial sample.

All experiments are performed with the raw anomaly score as well as with the anomaly likelihood.

3.2 Experiment results

This section presents results of the experiments. It is divided into two parts. At first, the best models found for each method and each dataset are presented and the influence of selected parameters that are unique to some methods is investigated. Then, the final results for all methods on all datasets are presented and compared.

The F_1 score is the primary metric. For each method and dataset, the model configuration with the best achieved F_1 score is presented and the scores are compared between all methods. As a secondary metric, the best achieved AUC for each method and dataset is stated. The optimal model configuration for the F_1 score and the AUC is not necessarily the same because both metrics describe different properties of the model. The AUC examines the overall behaviour under various values of the detection threshold (contamination) while the F_1 finds the single best model that is fully configured (including the detection threshold). In the result tables below, the stated configurations are for the F_1-optimized models.

In the ongoing paragraphs, following abbreviations are used:

- *WL* → Window Length,
- *CDD* → Concept Drift Detection,
- *ratio*, *distribution*, *eachtime*, *nulldetect* → Abbreviations of concept drift detection methods as described in the Section 3.1.3,
- *Cont* → Contamination,
- *FPR* → False Positive Rate.

3.2.1 Results of Moving Average detector

For the Moving Average detector, the two scaling functions κ were tested: constant ($\kappa(p) = 1$) and exponential ($\kappa(p) = 0.8^p$). Results for the exponential scaling were inferior to the constant scaling in almost all cases. Constant scaling function yielded higher F_1 score (1 - 10 percent) for all window lengths and all datasets. Only single exception where the exponential scaling function yielded significantly better F_1 score was the *NAB-AWS-CPU* dataset with the window length of 512, where the exponential function yielded approx. 7 percent higher score. Possible explanation for the inferior performance of the exponential scaling could be that the scaling is conflicting with the concept drift detection. The highest F_1 score for each dataset and the parameters of the related model can be found in the Table 3.2.

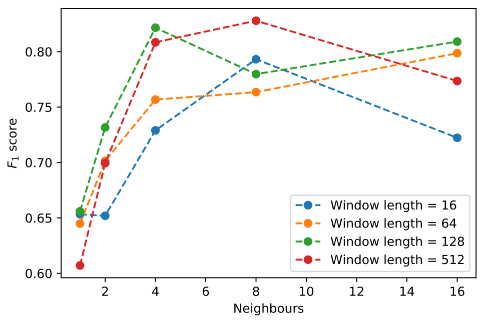
Dataset	**WL**	**CDD**	**Cont**	**SF**	**F_1**	**FPR**	**Recall**	*AUC*
Yahoo-A1	512	ratio	0.08	const	0.873	0.127	0.873	0.926
Yahoo-A2	16	eachtime	0.01	const	0.944	0.053	0.942	0.973
NAB-AWS-CPU	64	distribution	0.08	const	0.960	0.077	1.000	0.980
NAB-AWS-Disk	512	eachtime	0.8	const	0.903	0.177	1.000	0.961
NAB-AWS-Network	16	ratio	0.005	exp	0.996	0.007	1.000	0.996
NAB-Real-Taxi	32	eachtime	0.16	const	0.805	0.326	1.000	0.800

Table 3.2: Table containing optimal configurations of the moving average detector for each dataset in the terms of F_1 score. The column *SF* is *Scaling Function* and the value *const*, resp. *exp* is abbreviation for *constant scaling function*, resp. *exponential scaling function.* Last column is the best AUC achieved by the method on a given dataset (possibly with different configuration).

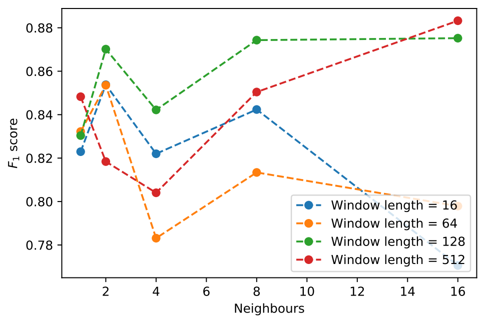
3.2.2 Results of Local Outlier Factor

As described in the previous section, the impact of the number of considered neighbours k ($k \in \{1, 2, 4, 8, 16\}$) in terms of achieved F_1 score has been explored. Models with window lengths $\in \{16, 64, 128, 512\}$ were tested. For each combination of k and window length, the model with optimal concept drift detection type and contamination value in terms of F_1 score has been picked.

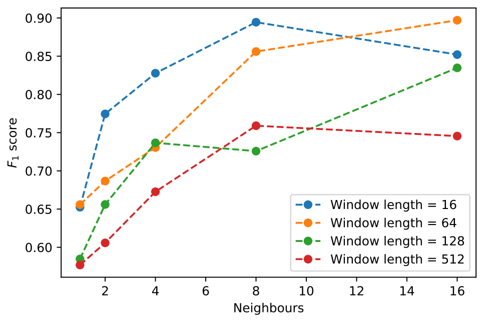
Observed results confirm the expectation that in general the larger number of neighbours increases the performance. However, in the significant number of cases, this trend is stopped or even reversed at $k = 8$. For datasets *Yahoo-A1*, *NAB-AWS-CPU* and *NAB-Real-Taxi* this trend seems to be logarithmic. The results for dataset *NAB-AWS-Disk*, resp. *NAB-AWS-Network*, are very noisy, resp. insensitive to the number of neighbours. Related plots can be found in the Figure 3.4.



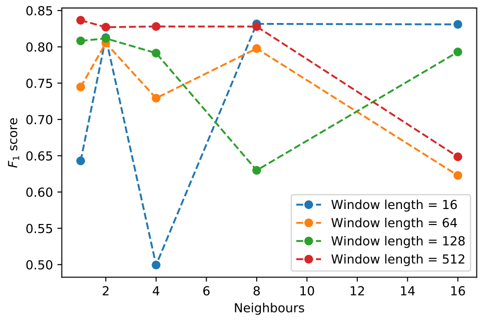
(a) Yahoo-A1



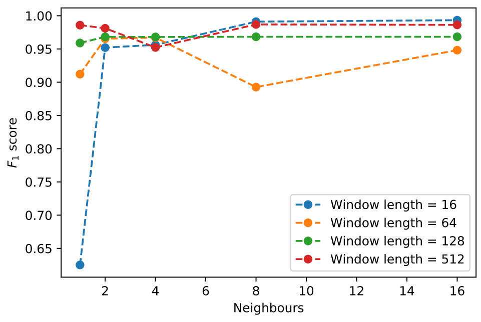
(b) Yahoo-A2



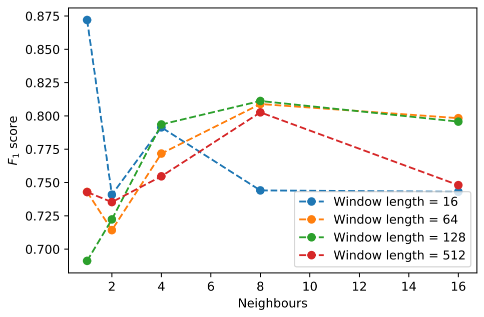
(c) NAB-AWS-CPU



(d) NAB-AWS-Disk



(e) NAB-AWS-Network



(f) NAB-Real-Taxi

Figure 3.4: Plots representing the relation between number of neighbours in LOF anomaly detector, window length and F_1 score. Each plot shows the result for a single dataset.

The best F_1 score achieved for each dataset can be found in the Table 3.3. The results for *NAB-AWS-Network* and *NAB-Real-Taxi* might seem strange at first sight (none zero recall with 0% contamination). However, this can be explained by the specific way of thresholding done by the LOF (described in 2.4). The zero contamination mean that the thresholding value is equal to the 100%-percentile and therefore no other value should be higher. However, if the anomaly values were higher than any of the values used for calculating the thresholding percentile (or the concept drift occurs), this thresholding value is exceeded.

Dataset	**WL**	**CDD**	**Cont**	**K**	**F_1**	**FPR**	**Recall**	*AUC*
Yahoo-A1	256	ratio	0.08	8	0.839	0.181	0.860	0.887
Yahoo-A2	256	eachtime	0.02	16	0.894	0.077	0.867	0.902
NAB-AWS-CPU	32	distribution	0.16	16	0.933	0.126	1.000	0.975
NAB-AWS-Disk	512	ratio	0.01	1	0.837	0.054	0.750	0.853
NAB-AWS-Network	16	ratio	0.0	16	0.993	0.013	1.000	0.993
NAB-Real-Taxi	16	nulldetect	0.0	1	0.872	0.042	0.800	0.895

Table 3.3: Table containing optimal configurations of LOF for each dataset in the terms of F_1 score. The column K is the number of neighbours. Last column is the best AUC achieved by the method on a given dataset (possibly with different configuration).

3.2.3 Results of Isolation Forest

For isolation forest, the impact of ensemble size has been tested. During the experiments included in this thesis, the suggestion of the original authors [28] (ensemble size = 100) was confirmed to be valid. Difference in F_1 score, resp. in AUC did not exceed 5%, resp. 3% for both lower and larger ensemble sizes and it cannot be said that either lower nor higher values performed consistently better. Comparison of F_1 scores and AUC for various ensemble sizes and datasets can be found in the Figure 3.5. Parameters of the model with the highest F_1 score for each dataset can be found in the Table 3.4. The suspicious result of non-zero recall with 0% contamination for *NAB-AWS-Network* can be explained in the very same way as for LOF.

Dataset	**WL**	**CDD**	**En. size**	**Cont**	**F_1**	**FPR**	**Recall**	*AUC*
Yahoo-A1	256	ratio	125	0.02	0.910	0.099	0.918	0.951
Yahoo-A2	256	eachtime	100	0.02	0.876	0.121	0.873	0.911
NAB-AWS-CPU	128	eachtime	100	0.04	0.931	0.061	0.923	0.965
NAB-AWS-Disk	256	eachtime	25	0.005	0.955	0.086	1.000	0.972
NAB-AWS-Network	512	eachtime	125	0.0	0.999	0.003	1.000	0.999
NAB-Real-Taxi	128	nulldetect	50	0.08	0.836	0.124	0.800	0.872

Table 3.4: Table containing optimal configurations of Isolation Forest for each dataset in the terms of F_1 score. Last column is the best AUC achieved by the method on a given dataset (possibly with different configuration).

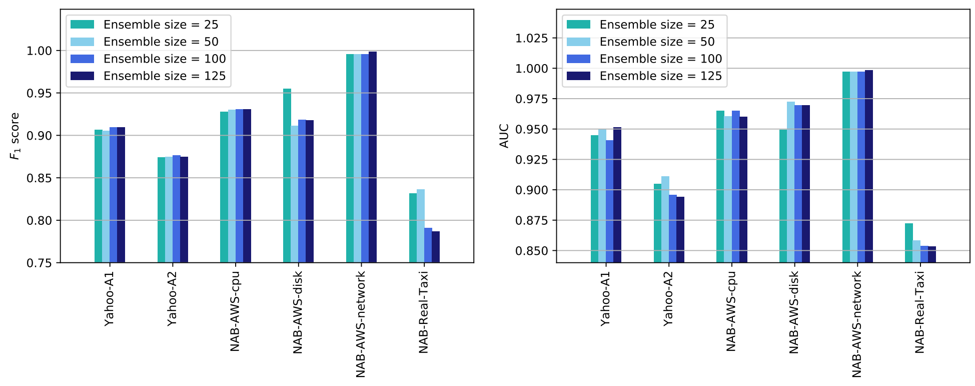


Figure 3.5: Impact of the ensemble size on the Isolation Forest performance.

3.2.4 Results of Hierarchical Temporal Memory

For all tested datasets, the usage of anomaly likelihood yielded worse F_1 scores and AUC then the usage of raw anomaly score. Therefore, all following results are for the raw anomaly score. Based on the expectation of the original authors, this conclusion can be explained by the used datasets that are not very noisy and therefore the anomaly likelihood does not have an opportunity to bring some benefit.

The experiments with various values of the resolution showed that the HTM is insensitive to this parameter on all tested datasets when the resolution is < 0.001. The differences in F_1 score were negligible ($< 0.5\%$).

Parameters of the model with the highest F_1 score for each dataset can be found in the Table 3.5.

Dataset	**Resolution**	**Cont**	$\mathbf{F_1}$	**FPR**	**Recall**	AUC
Yahoo-A1	0.01000	0.08	0.784	0.170	0.744	0.849
Yahoo-A2	0.00001	0.16	0.866	0.159	0.893	0.886
NAB-AWS-CPU	0.00001	0.08	0.912	0.099	0.923	0.931
NAB-AWS-Disk	0.00001	0.5	0.843	0.039	0.750	0.841
NAB-AWS-Network	0.00001	0.5	0.795	0.014	0.667	0.809
NAB-Real-Taxi	0.00001	0.08	0.833	0.132	0.800	0.816

Table 3.5: Table containing optimal configurations of HTM for each dataset in the terms of F_1 score. Last column is the best AUC achieved by the method on a given dataset (possibly with different configuration).

3.2.5 Results summary

The results show comparable performance of Isolation forest (best for *Yahoo-A1*, *NAB-AWS-Disk*, *NAB-AWS-Network*), LOF (best for *NAB-Real-Taxi*) and Moving Average (best for *Yahoo-A2*, *NAB-AWS-CPU*). None of the analyzed methods is significantly better than the others on any dataset (difference in F_1 and AUC between the best method and the second best method is $< 6\%$).

Surprising fact is the low performance of the HTM in comparison with other methods even if its surely the most complex one. The graphic representation of the final results can be found in the figures 3.6, 3.7, 3.8, 3.9, 3.10 and 3.11. In each of these plots, there are two sets of models. There are the best models per method in terms of F_1 score on the left side, resp. in terms of AUC on the right side. The numbers under the method names on the left side represent the NAB ranking for the models (the absolute numbers are not stated because they have no meaningful interpretation).

From the summary Table 3.6 it can be seen that for the vast majority of cases (specific method on a specific dataset), some non-trivial form of a concept drift detection yielded better results than the null concept drift detector (*nulldetect* method). Interesting is also the fact that the relearning after each new data point (*eachtime* method) is not optimal in roughly half of the cases. In these case, the relearning after each data point probably causes the underlying models to absorb some noise that is ignored by the other concept drift detection methods. An important result in the context of this thesis is that the performance of the proposed concept drift detection method based on the anomaly ratio and the complementary CDF (*ratio* method – 2.1.1.1) is very good. According to the results, this method is optimal in more cases than comparing the distribution of the data points (*distribution* method – 2.1.1.2).

Concept drift detection method	**Optimal in # cases**
nulldetect	2
eachtime	8
distribution	2
ratio	6

Table 3.6: Number of cases when the various concept drift detection methods have been found as optimal. A case is considered to be the performance (F_1 score) of a specific anomaly detection method on a specific dataset.

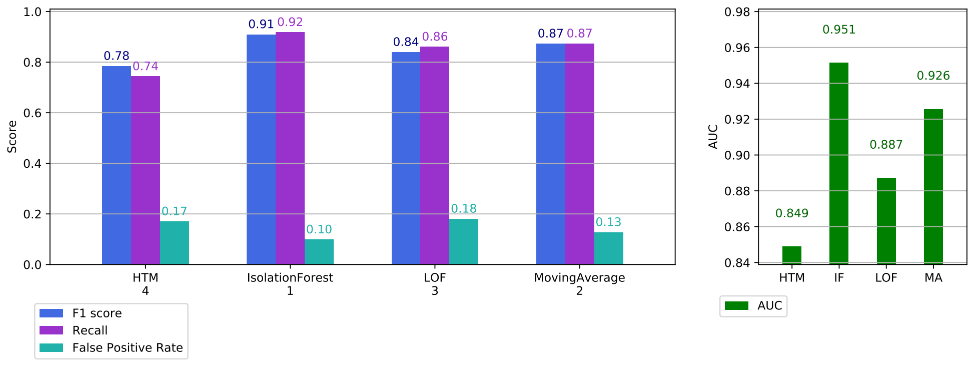


Figure 3.6: Bar chart comparing the best results of all analyzed methods on the *Yahoo-A1* dataset in terms of F_1 score, AUC and NAB ranking.

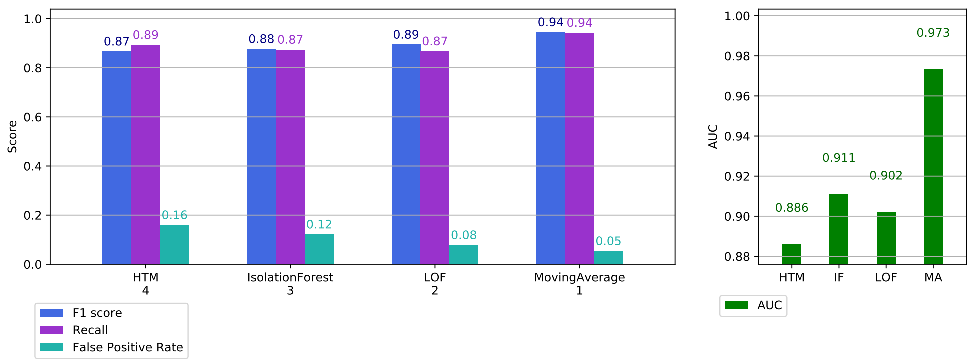


Figure 3.7: Bar chart comparing the best results of all analyzed methods on the *Yahoo-A2* dataset in terms of F_1 score, AUC and NAB ranking.

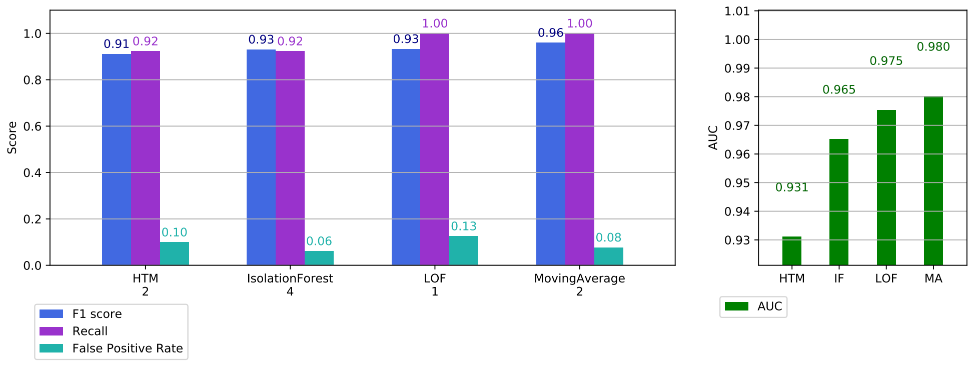


Figure 3.8: Bar chart comparing the best results of all analyzed methods on the *NAB-AWS-cpu* dataset in terms of F_1 score, AUC and NAB ranking.

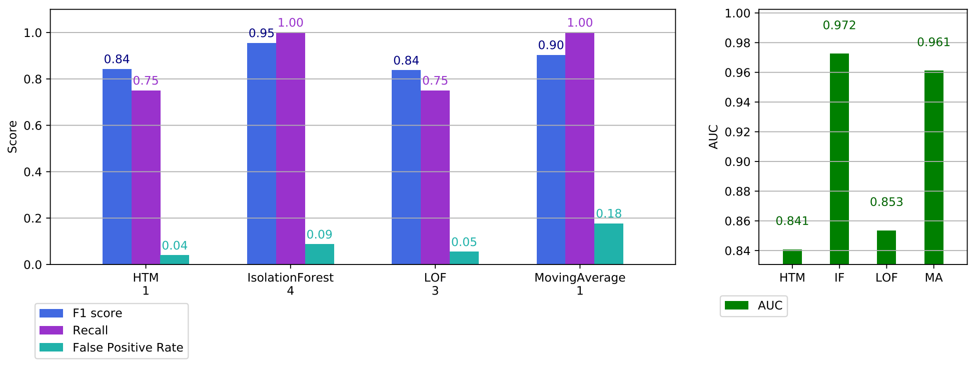


Figure 3.9: Bar chart comparing the best results of all analyzed methods on the *NAB-AWS-disk* dataset in terms of F_1 score, AUC and NAB ranking.

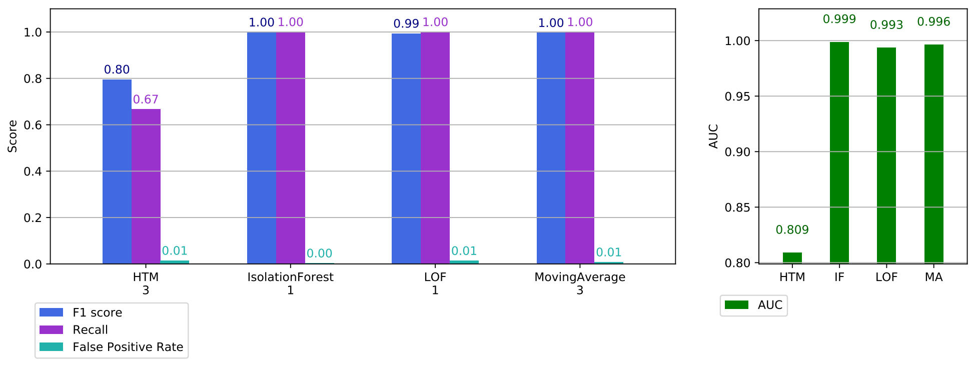


Figure 3.10: Bar chart comparing the best results of all analyzed methods on the *NAB-AWS-network* dataset in terms of F_1 score, AUC and NAB ranking.

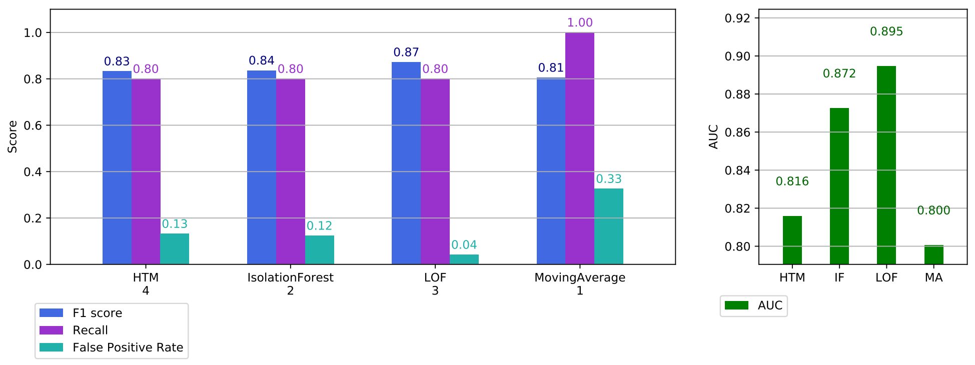


Figure 3.11: Bar chart comparing the best results of all analyzed methods on the *NAB-Real-Taxi* dataset in terms of F_1 score, AUC and NAB ranking.

The performance of the moving average is notable because it is a very simple method compared to the other ones. Especially interesting is the result on the *Yahoo-A2* and *NAB-AWS-CPU* datasets where its the best method. In the case of *Yahoo-A2* dataset, it is even the best method by the largest observable difference (5% in F_1 and 6% in AUC). This might be caused by the synthetic origin of this dataset. A representative example time-series from this dataset can be found in the Figure 3.12. It clearly shows, that the dataset consists of time-series generated by a relatively simple process with artificially inserted anomalies. This synthetic nature is apparently suitable for the moving average. The dataset *NAB-AWS-CPU* contains only point anomalies and minimal concept drift, which probably also suits to the moving average.

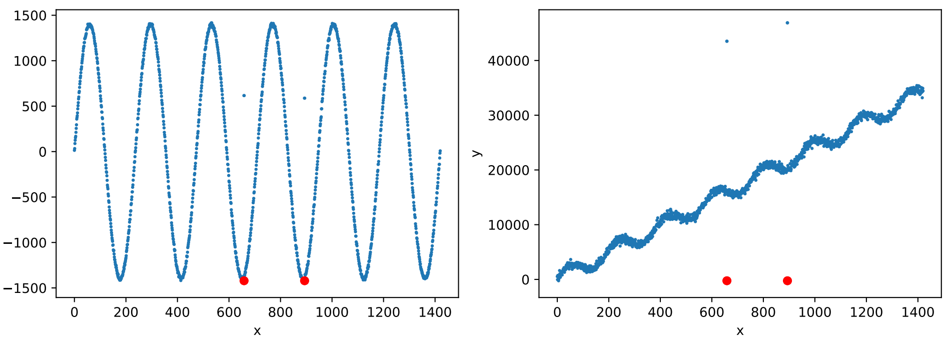


Figure 3.12: Two sample time-series from the synthetic *Yahoo-A2* dataset. The x-coordinates of the ground-truth anomalies are marked with the red points at the underside of the plots.

The very good results of the moving average suggest that the anomalies in the chosen datasets are not very hard to detect. The low relative performance of the HTM might be caused by this property of datasets. It possible that on the different datasets that contain some more complicated phenomena, the HTM could utilize its internal complexity and outperform the simpler methods.

The NAB ranking is not completely consistent with the other metrics. However, this is an expected result because the NAB score largely depends on the selected application profile and there are very small differences in F_1 score and AUC.

For other mentioned results, no possible explanation has been found. Datasets have been statistically and visually examined and no property providing such explanation emerged. Also it is not possible to say, that one method has in general e.g. higher recall or FPR.

Chapter **4**

Discussion

In the thesis, three novel approaches have been proposed:

1. concept drift detection by anomaly ratio and complementary CDF (section 2.1.1.1),
2. concept drift detection by comparing multivariate distribution of data points and yielded anomaly scores (section 2.1.1.3),
3. using other distribution than uniform one for selecting the split point of Isolation Forests (section 2.2.4.1).

The experiments confirming the feasibility of the first proposal have been performed. The future works might explore the second proposal for concept drift detecting by comparing the multivariate distributions. However, it requires the anomaly detectors to yield an anomaly score (not only labels) and therefore it is not so widely applicable. Moreover, to properly pursue this approach, a statistical test that is able to compare two multivariate distributions will have to be carefully chosen. The third proposal might be useful for the future works for which enough background domain knowledge about the data will be available.

Current experiment design could be improved by taking into an account the situations when the anomaly detectors are able to detect an anomaly but there is a delay between an anomalous data point and the moment when the detector reveals an anomaly. For example, the ground-truth anomaly label is on the data point $\mathbf{x}_t$, but the detector yields the anomaly label k data points later on $\mathbf{x}_{t+k}$. Some detectors could even make early detection (an anomaly is detected on $\mathbf{x}_{t-k}$). In current experiment design, this behaviour would result in one false negative detection and one false positive detection. This behavior could be changed by placing a window around the ground-truth label that was briefly described in the introduction to NAB scoring (3.1.1). If the detector yields several positive anomaly labels in one window, only single true positive detection is counted. No positive label yielded for the whole window would result in a single false negative detection. Next, the true detections at the beginning of the window would be scored better than at the end of the window. However, this technique brings up many questions. For example, should be the detector that is making earlier detections but with lower recall considered better than the detector with higher recall and later detections? What is the proper weighting function? How large should be the anomaly window? Should the window be centred around the anomaly or shifted backwards or forwards? Correct answers to these questions are heavily domain-dependent, however, it represents a possibly interesting topic for another work that is more focused on some specific domain.

All chosen datasets are univariate. Such data is very common in many real-world scenarios involving the online anomaly detection. However, if some future works will be dealing with multivariate data, it might by interesting to test methods like LOF or Isolation Forest that naturally supports multivariate datasets.

In the description of the Moving Average, there is a note about using this method to detect anomalous windows instead of only anomalous data points. This approach was not pursued in this thesis because there are similar problems with evaluation and acquisition of suitable datasets as described in the previous paragraphs. However, for some application domains, this could be a feasible approach. Also, the methods like LOF or Isolation Forest can be adapted to this task by considering the windows to be equivalent to the multivariate data points (e.g. a window of 16 univariate data points is a single multivariate data point with 16 dimensions).

Although that the results show good overall performance, future works might try to fine-tune the parameters of the concept drift detectors (thresholds and p-values) to achieve even better results.

In the performed experiments, the length of the relearning window was chosen to be equal to the window used for concept drift detection. This is a straightforward way that made the implementation and experiments less complicated and also no reason was found to use different lengths. However, the usage of different lengths is possible.

Also, the future works might consider testing more than one NAB application profile. The profiles are significantly domain-dependent which was undesired for this thesis, however, for other works, it might be a benefit.

Conclusion

Anomaly detection is an important problem within various research and application domains. There is a lot of literature on the topic of the general anomaly detection. However, when this problem is narrowed to the online anomaly detection in time-series, the amount of available literature is significantly lower. Moreover, there is no well-established theoretical framework around the online anomaly detection in time-series. Therefore, in the beginning, this thesis provides a consolidated view and terminology on this topic that is used to propose a design of a general anomaly detection system that enables to orthogonally combine various anomaly detection methods and concept drift detection methods.

A major part of the thesis focuses on four specific anomaly detection methods (moving average, local outlier factor, isolation forest and hierarchical temporal memory). An experimental framework has been designed in order to test interesting properties of the detectors as well as to compare them on five real-world dataset and one artificial dataset. This framework incorporates classic machine-learning metrics like recall, false positive rate, F_1 and AUC as well as less widely known metric called NAB score. Relatively large number of experiments (around six hundred thousand of evaluated time-series) were executed. The results supports various hypotheses made about the methods' behaviour. For the majority of results that do not comply with the expected behavior or were otherwise notable, explanations were proposed. Main conclusion is that no method is better than the others on all tested datasets. Only the hierarchical temporal memory lags behinds the other methods by a small amount. For the other three methods, the results show F_1 score and AUC very close or above 90% for five datasets and roughly 83% on remaining one dataset.

A significant part of the thesis deals with the problem of concept drift detection. Several methods found in the literature were described and tested. Two new approaches combining the ideas from these methods were proposed. One of these approaches was tested and the experimental results confirm its feasibility.

Bibliography

[1] Edgeworth, F. Y. XLI. On discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, volume 23, no. 143, 1887: pp. 364–375.

[2] Meriam-Webster. Definition of Anomaly by Meriam-Webster Dictionary. `https://www.merriam-webster.com/dictionary/anomaly`, [Online; Accessed: 2018-02-01].

[3] Chandola, V.; Banerjee, A.; et al. Anomaly Detection for Discrete Sequences: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, volume 24, no. 5, May 2012: pp. 823–839, ISSN 1041-4347, doi: 10.1109/TKDE.2010.235.

[4] Chandola, V.; Banerjee, A.; et al. Anomaly Detection: A Survey. *ACM Comput. Surv.*, volume 41, no. 3, Jul 2009: pp. 15:1–15:58, ISSN 0360-0300, doi:10.1145/1541880.1541882.

[5] Pimentel, M. A. F.; Clifton, D. A.; et al. Review: A Review of Novelty Detection. *Signal Process.*, volume 99, Jun 2014: pp. 215–249, ISSN 0165-1684, doi:10.1016/j.sigpro.2013.12.026.

[6] Salgado, C. M.; Azevedo, C.; et al. *Noise Versus Outliers*, chapter Noise versus Outliers. Cham: Springer International Publishing, 2016, ISBN 978-3-319-43742-2, pp. 163–183, doi:10.1007/978-3-319-43742-2_14.

[7] Hastie, T.; Tibshirani, R.; et al. *The Elements of Statistical Learning.* Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001, ISBN 978-0-387-84858-7.

[8] Assent, I.; Kranen, P.; et al. AnyOut: Anytime Outlier Detection on Streaming Data. In *Database Systems for Advanced Applications*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-29038-1, pp. 228–242.

[9] Chapelle, O.; Schlkopf, B.; et al. *Semi-Supervised Learning*. The MIT Press, first edition, 2010, ISBN 0262514125, 9780262514125.

[10] Hoens, T. R.; Polikar, R.; et al. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, volume 1, no. 1, Apr 2012: pp. 89–101, ISSN 2192-6360, doi: 10.1007/s13748-011-0008-0.

[11] Ahmad, S.; Lavin, A.; et al. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, volume 262, 2017: pp. 134 – 147, ISSN 0925-2312, doi:10.1016/j.neucom.2017.04.070.

[12] Zilberstein, S. Using Anytime Algorithms in Intelligent Systems. *AI Magazine*, volume 17, no. 3, 1996: pp. 73–83, doi:10.1609/aimag.v17i3.1232.

[13] Webb, G. I.; Lee, L. K.; et al. Understanding Concept Drift. *ArXiv e-prints*, Apr 2017, `https://arxiv.org/abs/1704.00362`.

[14] Carpenter, G. A.; Grossberg, S. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, volume 37, no. 1, 1987: pp. 54 – 115, ISSN 0734-189X, doi:10.1016/S0734-189X(87)80014-2.

[15] Grossberg, S. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, volume 1, no. 1, 1988: pp. 17 – 61, doi: 10.1016/0893-6080(88)90021-4.

[16] McCloskey, M.; Cohen, N. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, volume 24, no. C, 1989: pp. 109–165, ISSN 0079-7421, doi:10.1016/S0079-7421(08)60536-8.

[17] Hoff, T. Latency is Everywhere and it Costs You Sales - How to Crush it. http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you-sales-how-to-crush-it.html, 2009, [Online; Accessed: 2018-02-05].

[18] Gilbert, S.; Lynch, N. Perspectives on the CAP Theorem. *Computer*, volume 45, no. 2, Feb 2012: pp. 30–36, ISSN 0018-9162, doi:10.1109/MC.2011.389.

[19] Ding, Z.; Fei, M. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window. *IFAC Proceedings Volumes*, volume 46, no. 20, 2013: pp. 12 – 17, ISSN 1474-6670, doi:10.3182/20130902-3-CN-3020.00044.

[20] Reis, D. M. d.; Flach, P.; et al. Fast Unsupervised Online Drift Detection Using Incremental Kolmogorov-Smirnov Test. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, New York, NY, USA: ACM, 2016, ISBN 978-1-4503-4232-2, pp. 1545–1554, doi:10.1145/2939672.2939836.

[21] Beyer, K. S.; Goldstein, J.; et al. When Is "Nearest Neighbor" Meaningful? In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, London, UK, UK: Springer-Verlag, 1999, ISBN 978-3-540-49257-3, pp. 217–235, doi:10.1007/3-540-49257-7_15.

[22] Breunig, M. M.; Kriegel, H.-P.; et al. LOF: Identifying Density-based Local Outliers. *SIGMOD Rec.*, volume 29, no. 2, May 2000: pp. 93–104, ISSN 0163-5808, doi:10.1145/335191.335388.

[23] Ramaswamy, S.; Rastogi, R.; et al. Efficient Algorithms for Mining Outliers from Large Data Sets. *SIGMOD Rec.*, volume 29, no. 2, May 2000: pp. 427–438, ISSN 0163-5808, doi:10.1145/335191.335437.

[24] Knorr, E. M.; Ng, R. T. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, ISBN 1-55860-566-5, pp. 392–403.

[25] Angiulli, F.; Pizzuti, C. Fast Outlier Detection in High Dimensional Spaces. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '02, London, UK: Springer-Verlag, 1986, ISBN 978-3-540-45681-0, pp. 15–26, doi:10.1007/3-540-45681-3_2.

[26] Bentley, J. L. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, volume 18, no. 9, Sep 1975: pp. 509–517, ISSN 0001-0782, doi:10.1145/361002.361007.

[27] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830, ISSN 1532-4435.

[28] Liu, F. T.; Ting, K. M.; et al. Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, Washington, DC, USA: IEEE Computer Society, 2008, ISSN 2374-8486, pp. 413–422, doi:10.1109/ICDM.2008.17.

[29] Domingos, P.; Hulten, G. Mining High-speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, New York, NY, USA, 2000, ISBN 1-58113-391-X, pp. 71–80, doi:10.1145/502512.502529.

[30] Preiss, B. *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. Wiley, Jan 2000, ISBN 978-0471346135.

[31] Schlimmer, J. C.; Fisher, D. H. A Case Study of Incremental Concept Induction. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, AAAI Press, 1986, pp. 496–501.

[32] Utgoff, P. E. Incremental Induction of Decision Trees. *Machine Learning*, volume 4, no. 2, Nov 1989: pp. 161–186, ISSN 1573-0565, doi:10.1023/A:1022699900025.

[33] Hulten, G.; Spencer, L.; et al. Mining Time-changing Data Streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, New York, NY, USA: ACM, 2001, pp. 97–106.

[34] Hawkins, J.; Ahmad, S.; et al. Hierarchical Temporal Memory including HTM Cortical Learning Algorithms. Technical report, Numenta, 2011. Available from: `https://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf`

[35] Hawkins, J. *On Intelligence*. Times Books, 2004, ISBN 0-8050-7456-2.

[36] Ahmad, S.; Hawkins, J. Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory. *Computing Research Repository*, 2015, `https://arxiv.org/abs/1503.07469`.

[37] Cui, Y.; Ahmad, S.; et al. Continuous Online Sequence Learning with an Unsupervised Neural Network Model. *Neural Comput.*, volume 28, no. 11, Nov 2016: pp. 2474–2504, ISSN 0899-7667, doi:10.1162/NECO_a_00893.

[38] Hawkins, J.; Ahmad, S. Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. *Frontiers in Neural Circuits*, volume 10, 2016: p. 23, ISSN 1662-5110, doi:10.3389/fncir.2016.00023.

[39] Cui, Y.; Ahmad, S.; et al. The HTM Spatial Pooler: a neocortical algorithm for online sparse distributed coding. *bioRxiv*, 2016, doi:10.1101/085035.

[40] Cui, Y.; Surpur, C.; et al. A comparative study of HTM and other neural network models for online sequence learning with streaming data. In *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*, 2016, ISSN 2161-4407, pp. 1530–1538, doi:10.1109/IJCNN.2016.7727380.

[41] Purdy, S. Encoding Data for HTM Systems. *Computing Research Repository*, volume abs/1602.05925, 2016, `https://arxiv.org/abs/1602.05925v1`.

[42] Lavin, A.; Ahmad, S. Evaluating Real-time Anomaly Detection Algorithms - the Numenta Anomaly Benchmark. *Computing Research Repository*, volume abs/1510.03336, 2015, `https://arxiv.org/abs/1510.03336`.

[43] Yahoo! Webscope dataset: A Labeled Anomaly Detection Dataset (S5), version 1.0. Available from: `http://labs.yahoo.com/Academic_Relations`

[44] Jones, E.; Oliphant, T.; et al. SciPy: Open source scientific tools for Python. 2001, [Online; accessed 2018-04-20]. Available from: `http://www.scipy.org/`

Appendix A

Mathematical symbols

$\mathcal{N}(\mu, \sigma)$	normal (Gaussian) distribution with mean μ and standard deviation σ
$\mathbb{N}$	domain of natural numbers (not containing 0)
$\mathbb{R}$	domain of real numbers
$\mathbb{R}^{0,+}$	domain of non-negative real numbers
$\mathbb{X}, \mathbb{Y}, \mathbb{C}, \mathbb{D}$	sets
$X^{a,b}$	matrix containing a rows $\times$ b columns
x, y, σ, τ	scalar values
$\mathbf{x}, \mathbf{y}, \mathbf{x}_i, \mathbf{y}_i$	column vectors (matrices $m \times 1$ where m is the number of vector components)
X^T, resp. $\mathbf{x}^T$	transposition of matrix X, resp. vector $\mathbf{x}$
$\mathbf{x}_{i,k}$	k-th component of vector $\mathbf{x}_i$
$\{1, 2, 3, \ldots, n\}$	set containing natural numbers from 1 to n
$\mathcal{A}, \mathcal{D}, \mathcal{E}, \mathcal{F}$	morphisms

Appendix B

Acronyms

ANN	Artificial Neural Network
AUC	Area Under Curve
CDD	Concept Drift Detection
IoT	Internet Of Things
LOF	Local Outlier Factor
NAB	Numenta Anomaly Benchmark
ROC	Receiver Operating Characteristics
WL	Window Length

Appendix C

Contents of enclosed CD

```
|
├── Data ............. folder containing the instruction for obtaining datasets
├── DataPreprocessing ........... Jupyter notebooks for data preprocessing
├── Models ................ Python script for training and evaluating models
├── ResultsProcessing . Jupyter notebooks for processing of acquired results
├── ResultsRaw ............................ CSV files with acquired results
└── Thesis ...................... LaTeX source codes and PDF of the thesis
```