Anomaly Detection and Exploratory Causal Analysis for SAP HANA

Master's Thesis
by

Jianqiao Jin

Chair of Pervasive Computing Systems/TECO
Institute of Telematics
Department of Informatics

&

SAP SE
SAP DBS SST Innovation & Algorithmic Analytics

Professor:	Prof. Dr. Michael Beigl
Advisor (TECO):	Dr. Nhung Ngo
Advisor (SAP):	Michael Laux

Project Period: 01/08/2018 – 31/01/2019

Abstract

Nowadays, the good functioning of the equipment, networks and systems will be the key for the business of a company to continue operating because it is never avoidable for the companies to use information technology to support their business in the era of big data. However, the technology is never infallible, faults that give rise to sometimes critical situations may appear at any time. To detect and prevent failures, it is very essential to have a good monitoring system which is responsible for controlling the technology used by a company (hardware, networks and communications, operating systems or applications, among others) in order to analyze their operation and performance, and to detect and alert about possible errors.

The aim of this thesis is thus to further advance the field of anomaly detection and exploratory causal inference which are two major research areas in a monitoring system, to provide efficient algorithms with regards to the usability, maintainability and scalability. The analyzed results can be viewed as a starting point for the root cause analysis of the system performance issues and to avoid falls in the system or minimize the time of resolution of the issues in the future.

The algorithms were performed on the historical data of SAP HANA database at last and the results gained in this thesis indicate that the tools have succeeded in providing some useful information for diagnosing the performance issues of the system.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 31. Jan. 2019 _____

Contents

1. Introduction

The chapter gives a brief introduction about SAP, a market leader in providing ERP solutions and services and emphasizes the importance and necessity of system monitoring in business applications. Based on this, the scope and the aims of the thesis are identified.

In our everyday lives nowadays, software systems play an important role in a wide area of business and the good functioning of the systems will be the key for the business to continue operating. Thus reliability of system is one of the major concerns for software engineers. The increasing size of software systems and their inherent complexity - which is essentially related to the intricate interdependencies among many heterogeneous components - pose serious difficulties to its assessment and assurance [PRT10]. In order to guarantee high reliability, system behavior needs to be monitored via tens of millions of metrics in the essential administrative areas of a system and therefore system monitoring has gained much research attention in application domains in recent years. Broadly speaking, monitoring[1] consists of collecting, processing, aggregating, and displaying real-time quantitative data about a system.

SAP SE is one of the largest vendors of enterprise resource planning (ERP) software and related enterprise applications. The company's ERP system enables its customers to run their business processes, including accounting, sales, production, human resources and finance, in an integrated environment. The integration ensures that information flows from one SAP component to another without the need for redundant data entry and helps enforce financial, process and legal controls. To this end, SAP has developed a database system called *HANA(High performance*

[1]There is no uniformly shared vocabulary for discussing all topics related to monitoring.

Analytical Appliance) that will replace the traditional relational database systems, overcome the limiting technological and conceptual factors of current enterprise application landscapes and eliminate the performance bottlenecks of operative systems to allow running transactional systems and executing complex queries and analysis in a single real-time environment. In a word, SAP HANA is a (super-fast) database system, which application and analytical systems can access in order to read and write data (super-fast). Proactive monitoring of the SAP HANA database will help the engineers to understand issues in advance and to take corrective actions and thus will lead to lesser downtime of the database, improving profitability of the business organization.

The metrics of a system are always collected as sequences of values over time, more precisely as *time series*. Time series are finite or unbounded sequences of data points in increasing order by time. As time series can be used to represent the behavior of metrics in general, the development of methods and systems for efficient transfer, storage, and analysis of time series is a necessity to enable and strengthen the system monitoring services.

1.1 Goals

Imagine that a domain expert is monitoring CPU utilization in a data center[2]. CPU utilization is the value between 0-100, which refers to percentage of CPU usage by process. At some time points, he observes that the value is relatively high compared to the normal case and then he is afraid that soon after it may not be available for new process, resulting in that new request is placed in a queue and it produces bottleneck in the system.

In such a case, the expert thinks that CPU utilization becomes *critical* and he wants to know what has caused the growth of CPU utilization and how the growth of CPU utilization actually affects the system at that moment, which may help him to have a good overview for handling resource management in the data center. However, the amount of data resulting from the data center is huge and analyzing the metrics in the data center manually is always time consuming and exhausting. As a consequence, it would be very beneficial to have some automatic methods supporting the expert by detecting unexpected behavior of a metric and giving the expert a dependency structure of the system when some unexpected behavior have been detected. Therefore, this thesis researches ways of automatically optimizing the described process of analyzing metrics of a system, e.g. SAP HANA database with the following two aims:

- **Aim 1:** detecting unexpected behavior of a metric in the system,
- **Aim 2:** investigating the inter-dependence structure of the underlying system of various metrics when some unexpected behavior in a certain metric are detected.

On-line Unsupervised Anomaly Detection Unexpected behavior of a metric are always called anomalies. A challenge, for both machines and humans, is identi-

[2]CPU utilization is displayed by some visualization tools.

fying an anomaly because the problem is often ill-posed[3] which makes it hard to tell what an anomaly is. Fortunately, metrics of a system are collected and expressed in the form of time series and anomaly detection in time series has been recently an active research area in the fields of statistics and machine learning because it is important across many industries. In time series, the assumption of *temporal continuity* plays an important role in identifying anomalies. Temporal continuity means that the pattern in the data are not expected to change abruptly, unless there are abnormal processes at work [Agg17]. One issue concerning anomaly detection is that it is difficult, even impossible to obtain a very well-sampled abnormal data [HG09], which unfortunately makes conventional classification schemes unsuitable for solving problems. Without non-representative abnormal instances available as training data, anomaly detection is largely an *unsupervised* problem where the negative class is either not present or not properly sampled in the train set. This aspect of anomaly detection tends to make it more challenging than classification problems solved always by *supervised* learning. Additionally, metrics always behave like streaming data in real applications, resulting in that the nature of data source may change over time, e.g. software upgrades and configuration changes can occur at any time and may alter the behavior of the data. Therefore, a model trained off-line is highly likely not suitable for a continuous sequence of data occurring in real time. Under the circumstance, the first goal of this thesis is to detect anomaly in univariate time series data with an algorithm which should be constantly adapt to a new definition of "normal" in an unsupervised, automated fashion.

Exploratory Causal Analysis We always think that the various metrics of a system are not independent but affect each other over time. Therefore, when anomalies in a certain metric are detected, an engineer often needs to inspect the system closely. Naturally, there are two issues he is mostly concerned with: what has caused the anomalies and how they affect the system, in other words, he wants to investigate the inter-dependence structure of the underlying system of multiple metrics. When it comes to dependency between time series data, the Pearson correlation coefficient will be the first to jump into our minds. However, it has some limitations for solving the above mentioned two issues. First, it focuses on pairwise correlation which is not applicable to a multivariate case. Second, it can only discover linear relationship but non-linear interaction between metrics is highly possible in real system. The last, but most importantly, "correlation does not imply causation" because it is a *symmetric* measurement and cannot imply that one causes the others or one is affected by the others [A$^+$95], which needs some *directional* measurements to realize.

In such cases, time series causality tools are more suitable than the Pearson correlation coefficient. The definition of causality is always based on two major principles: (1) the cause happens prior to the effect and (2) the cause makes unique changes in the effect [Wie56]. Even though there have been extensive debates on the validity and generality of these principles, most of time series causality tools are designed by assuming their correctness, therefore, a lot of philosophers and physicists insist that such tools should not be considered casual at all but only *statistical* and *associational* tools [J.P09, aJP13, BP13]. In fact, making a general causal state-

[3] An ill-posed problem is one which doesn't meet the three Hadamard criteria for being well-posed. (see: https://en.wikipedia.org/wiki/Well-posed_problem)

ment always requires a huge amount of experiments and analysis to confirm some findings, however, time series causality tools only make analysis with the available data and find some potential causal information among data, which is more like exploratory data analysis, a concept proposed by Tukey [Tuk77]. Under such circumstance, McCracken [McC16] labels the causal inference with time series causality tools as *exploratory causal analysis*, emphasizing that it is intended to determine if and what causal structure may be present in a given set of time series data but is not intended to *confirm* such structure. Anyway, the dependency structure as the primary information can give the engineers or domain-experts a deep insight of the system when anomalies are detected, which would support them to find the root causes for anomalies or make some intervention to keep the system stable. To this end, the second goal of this thesis is to construct a temporal-causal structure of the underlying system with some scalable time series causality tools.

1.2 Thesis Contributions

This work in this thesis involves several contributions; in this section, the main contributions are presented. The thesis contribution as a whole is on the application of machine learning and statistical analysis. The first major contribution is literature review on existing anomaly detection and causal inference studies for time series data. The review illustrates the types of anomaly in univariate time series data, presents several well-known anomaly detection techniques and discusses their limitations on streaming applications. In addition, the review distinguishes the exploratory causal analysis from confirmatory causal analysis in detail and introduces some widely-used time series causal inference tools.

The second contribution is the online prototypes updating anomaly detection (OPUAD) algorithm for streaming applications. One of the main advantages of this algorithm is computationally efficient, which is essential for streaming data.

The third contribution is using several variants of Granger causality to quantify the temporal-causal effect among time series data. A synthetic linear standardized VAR system and a synthetic nonlinear Henon system are generated to test the performance of these comparative methods. The MLP Lasso-GC method, which combines the multilayer perceptron neural network and lasso penalty, cannot only learn the structure of the linear system, but also adapt to the nonlinear interactions.

The OPUAD algorithm and MLP Lasso-GC method advance the filed of anomaly detection and causal inference respectively, which are two promising research areas in time series data mining. From a business perspective, they can be integrated in a monitoring service for diagnosing the performance issues of the system and to increase productivity.

1.3 Outline

This thesis is organized in 6 chapters. Each chapter starts with a brief summary, allowing the reader to rapidly understand the chapter's content. The plan of each chapter is as follows:

- Chapter 2 presents a literature overview on time series data mining, anomaly detection techniques and causal inference tools for time series.

- Chapter 3 introduces how to detect anomalies in an on-line, automated fashion, especially for streaming data in real-time. After implementation, the algorithm is evaluated by Numenta Anomaly Benchmark (NAB) which provides a controlled and repeatable environment of tools to test and measure different anomaly detection algorithms on streaming data[4].
- Chapter 4 describes various time series causality measures with their scalability and performs simulations to show the performance of the competing methods.
- In Chapter 5, the developed algorithm of anomaly detection and designed framework of exploratory causal analysis are applied to analyze the historical data of SAP HANA database.
- Finally, Chapter 6 summarizes the thesis, discusses benefits and limitations of the proposed approaches and identifies the further research directions.

[4]NAB will be introduced in Chapter 3.

2. Background & Related Work

This chapter presents the related tasks in time series data mining at first, then it is dedicated to anomaly detection for univariate time series, categorizes anomalies and surveys corresponding techniques. After that, it turns into casual inference for time series data and introduces a new concept called exploratory causal analysis.

2.1 Time Series Data Mining

In almost every scientific field, such as economic forecasting, intrusion detecting, gene expression analysis, medical surveillance etc., measurements are always performed over time. These observations lead to a collection of organized data called *time series*.

Definition 2.1. *A time series X is a sequence of observations of data points measured over a time interval*

$$X = (x_1, x_2, ..., x_N) \quad x_t \in \mathbb{R}$$

A time series is often the result of the observation of an underlying process in the course of which values are collected from measurements made at uniformly spaced *time instants* and according to a given *sampling rate*. A time series can thus be defined as a set of contiguous time instants. The series can be *univariate* as in the above definition or *multivariate* when several series simultaneously span multiple dimensions within the same time range.

Time series can cover the full set of data provided by the observation of a process or may be of considerable length. In the case of streaming, they are semi-infinite as time instants continuously feed the series. It thus becomes interesting to consider only the *subsequences* of a series.

Definition 2.2. *Given a time series $X = \{x_t\}_{t\in\mathbb{Z}}$ of length N, a subsequence S of X is a series of length $M \leq N$ consisting of contiguous time instants from X*

$$S = (x_k, x_{k+1}, ..., x_{k+M-1})$$

with $1 \leq k \leq N - M + 1$.

Therefore, we can easily conclude that the nature of time series data encompasses: large in data size, high dimensionality and continuous update. With the enormous amount of time series data present in everyday's life, it is increasingly important to develop powerful means for time series analysis and extract interesting knowledge that could help in decision-making. The analysis process is referred to as *time series data mining*.

In the context of time series data mining, there are various kinds of time series data related tasks. Attempting to catalog them all would be burdensome, if it could be done at all, and would yield the result out-of-date because of the ever-growing range of real-life problems. Major time series related tasks, however, appear to fall into the following eight broad categories:

- **Indexing**: Given a query time series Q and a similarity measure $D(Q, X)$, find the most similar time series in database DB [CKMP02, FRM94].

- **Prediction**: Given a time series $X = (x_1, x_2, ..., x_n)$, predict the k next values $(x_{n+1}, ..., x_{n+k})$ that are most likely to occur [CW94].

- **Clustering**: Given a time series database DB and a similarity measure $D(Q, X)$, find natural groupings of time series in DB [AC01, KP98].

- **Classification**: Given an unlabeled time series X, assign it to one of two predefined classes [Geu01, KP98].

- **Segmentation**: Given a time series X containing n data points, construct a model $\bar{X}$ of reduced dimensionality $\bar{d}$ ($\bar{d} << n$) so that $\bar{X}$ closely approximates X [KP98].

- **Motif Discovery**: Given a time series X, find all subsequences that occur repeatedly in the original time series [LKL+04].

- **Anomaly Detection**: Given a time series X, find all "surprising/interesting/unexpected"occurrences, not fitting the normal pattern [Wei04].

- **Causal Inference**: Given a time series database DB and identify the temporal-causal relationship between time series [ALA07].

This work is mainly dedicated to the last two categories: anomaly detection and causal inference, which are promising research directions recently due to the importance and necessity of system monitoring in business applications.

2.2 Anomaly Detection

In time series data mining, anomaly detection refers to the problem of finding occurrences in data that deviate from normal model or expected behavior as to arouse suspicions that it was generated by a different mechanism [Haw80]. Depending on context and domain these deviations can be referred to as anomalies, outliers, novelties, aberrations, surprises, peculiarities etc. Among them, anomalies and outliers are two terms most commonly used in the context of anomaly detection; sometimes interchangeably [CBK09]. In this thesis, the term used is anomalies[1]. Anomaly detection is an actual problem in various areas, such as intrusion detection, fraud detection, financial transactions, medical and public health etc. Particularly, one of the most important use cases for anomaly detection today is the application in system monitoring services which is designed to increase uptime and reduce any downtime through quick identification of any issues the minute they arise. Additionally, in time series data mining, *vertical* analysis is more important where each individual series (or dimension) is treated as a unit [Agg17], and thus **anomaly detection in this thesis is primarily performed on univariate time series**.

2.2.1 Anomaly Types

Time series consists of a set of values typically generated by continuous measurement overtime, therefore, the data values are related to each other temporally and influenced by the adjacent values of the data points, which means, temporally contextual dependency is important. Therefore, an anomaly in a time series is always a *contextual anomaly* because the data values are never be treated as independent of one another for anomaly detection of time series data.

Much of the work on time-series anomaly detection is to determine unusual changes or very large deviations from the underlying series. There are mainly two types of deviations, the one is based on the individual deviations of the data points, the other on the shapes of specific portions of the time series with respect to the other extracted portions [Agg17]. Therefore, a contextual anomaly in time series can be classified in the following two categories:

- *Type 1: point anomaly in univariate time series:* A single point can be considered as abnormal with respect to the rest of data. Sometimes, some thresholds are set to define the border between normal and abnormal. In such a case, a point violating the valid range is also referred to as *spatial anomaly* because it can be declared abnormal without contemplating other observations [ALPA17]. A point anomaly, considered as an anomaly only in a specific temporal context, but not otherwise, is not a spatial anomaly.

[1] Solutions for outliers detection, novelties detection are often used for anomaly detection and vice versa, even though there are some subtle differences in the definition of each term.

- *Type 2: collective anomaly in univariate time series:* If a group of points is declared anomalous with respect to the entire data set, it is termed a collective anomaly. The individual data points in a collective anomaly may not be abnormal by themselves, but their occurrence together as a collection is anomalous.

Examples

To give examples for each of the two types of anomalies, a synthetic time series with 200 data points is generated as Figure 2.1 shows. The generator performs like a sine wave describing a smooth periodic oscillation and the valid range is $[-1, 1]$, however at some time stamps the pattern of the data is not expected to change abruptly.

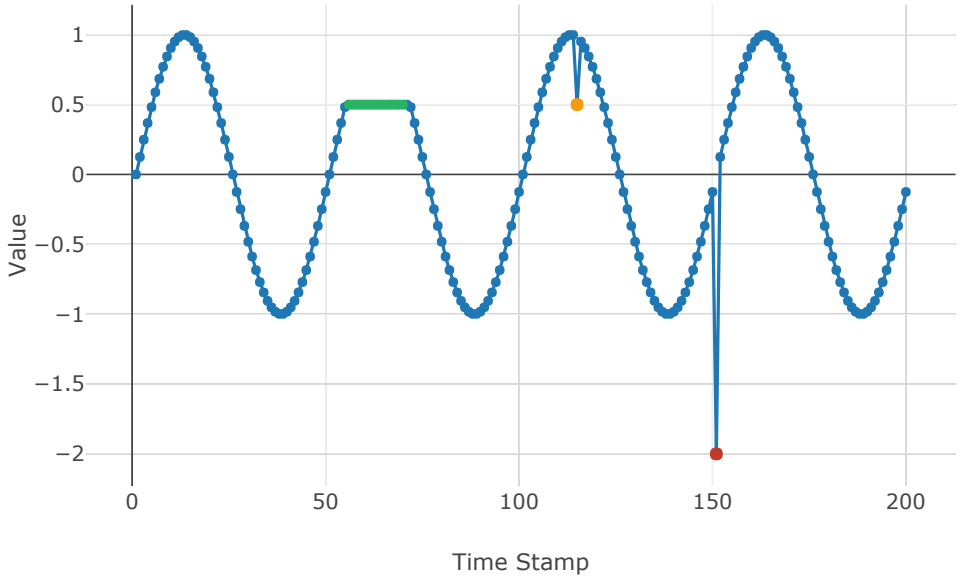


Figure 2.1: A Synthetic Time Series with Anomalies.

It is evident that there is a sudden change in the process at time stamp 115(marked in orange) and at time stamp 151(marked in red) respectively. The value of the point at time stamp 151 drops to -2 and exceeds the valid range far away, since no further observations need to be contemplated in order to make the decision, the point is easily declared a spatial anomaly, while the value of the point at time stamp 115 is abnormal considering the consecutive data values in its neighborhood, but it is valid in the range $[-1, 1]$ and it appears several times before and later, therefore the point is not a spatial anomaly, but a point anomaly. The subsequence(highlighted in green) from time stamp 56 to 72 denotes a collective anomaly because the same low value exists for an abnormally long time. It is difficult to tell that a single point in the subsequence is an anomaly because it seems normal in its neighborhood and its value is in the valid range. However, considering the whole time series, the points in this subsequence together as a group are anomalous.

To summarize, a point anomaly in time series is defined at a specific time stamp, while a collective anomaly shows very unusual behavior over multiple time stamps. Some techniques determining large deviations from previous time stamps are efficient in detecting point anomaly, but cannot differentially discover collective anomalies,

because the entire series (or subsequence) needs to be viewed from the perspective of other normal series [Agg17]. Under such circumstance, the discussion of anomaly detection techniques in the later sections is divided into two parts with respect to the different types of anomaly in time series: detection of point anomaly and detection of collective anomaly.

2.2.2 Anomaly Detection Techniques

As a human, one is capable to point out which part of given time series is abnormal, which points seem weird because a human has a neural capacity to recognize the shape of a time series. However, it is time consuming and exhausting to perform the task manually when there exist a huge amount of metrics in a system. Therefore, machine learning and data mining techniques have been proposed for solving anomaly detection of time series automatically.

Anomaly detection techniques always consist of two parts, a training phase and detection phase. During the training phase, the techniques use a set of training data to define a model which specifies what is considered normal and/or abnormal with respect to the training set. In the detection phase, new or incoming data is classified using the model from the training phase. Depending on the technique and implementation, the result is either binary or returned as a level of anomaly. A binary result implies that the tested data instance is either reported normal or abnormal, while a result as a level of anomaly is an anomaly score produced from the detection technique. Data instances with anomaly scores above some threshold level could then be classified as anomalies. Usually, there are two types of data available in the training phase, labeled or unlabeled data instances. For labeled data there are labels associated with each data instances which give information whether the instance is normal or abnormal, while for unlabeled data instances there is no such information. In the time series setting, the data instances in the training set may be a point, a subsequence or entire series, thus the labels may be associated with time-instants, with time intervals or they may be associated with the entire series [Agg17]. The training phase can be performed mainly in three ways relying on whether or not labels are available:

- *Supervised Learning:* When applying supervised learning, the data for the training phase must be labeled by some experts or algorithms before to show what is normal or not. The challenge of supervised learning is that it is usually very time consuming to label data and hard to collect all types of anomalies in a training set. Even though the challenge can be overcome, an imbalanced training set is inevitable, which means, the existence of the abnormal instances is always far less than that of the normal instances because of the fact that anomalies are rare [JAK01, JAK02, PAL04]. Learning on an imbalanced training set would produce a useless classifier.

- *Semi-supervised Learning:* Semi-supervised learning requires the training set only with normal instances. However, it is difficult to find a training set that covers all normal instances.

- *Unsupervised Learning:* Unsupervised learning does not use labeled data. Instead, this method assumes that the normal behavior is the most frequently

occurring. Normal instances are then defined as the most frequently occurring patterns, and instances deviating from these patterns are reported as anomalies.

2.2.2.1 Point Anomaly Detection

Regression models based on prediction are the most common unsupervised learning techniques of detecting anomalies at specific time stamps. The predictive model is always learned on a training set in an unsupervised manner, which uses p observations (history) to predict the $(p + 1)$th observation following them. For a test time series, the predictive model built in training phase is used to forecast the observation at each time stamp with the observations seen so far (previous p observations). The prediction error is thus selected as anomaly scores. Moving Average(MA) [Cha03], AutoRegression(AR) [FYM05, Cha03], Autoregressive Moving Average(ARMA) [Pin05] are the commonly used models.

In some cases, the time series may have some persistent trends, as a result of which it may drift away from the mean. This is referred to as the *non-stationarity* of the time series. For example, we cannot expect to predict prices today based on the prices from a hundred years back because the statistics of the price-based time series today may be very different from that a hundred years back. In such cases, the series can be de-trended by first differencing the time series before modeling. Such a modeling approach is referred to as Autoregressive Integrated Moving Average Model (ARIMA) [BGBMY01].

Probabilistic approaches have also been intensively investigated for point anomaly detection. A training time series is always assumed to be generated from some underlying probability distribution D, which represents "normality". The test data is evaluated by an estimated probability density function based on the training data. The less the probability density is, the more likely the data point is an anomaly. For the purpose of providing an estimated probability density function, Gaussian mixture models (GMMs) and kernel density estimators have proven popular. GMMs are typically classified as a parametric technique, because of the assumption that the data are generated from a weighted mixture of Gaussian distributions [CBK09, MS03]. Kernel density estimators are a non-parametric way to estimate the pdf of data as they are closely related to histogram methods [DHS00, CBK09, MS03].

2.2.2.2 Collective Anomaly Detection

Unlike the cases discussed before where anomalies are defined by a single position, a set of time stamps is always needed to be viewed collectively in order to learn whether or not they should be considered an anomaly. In other words, collective anomaly is always determined by the shape of time series. Under such circumstance, there are two possibilities described in [Agg17]:

- *Full-series anomaly:* the shape of the entire series is treated as an anomaly. It is always realized by comparing the given time series against a set of normal series. However, the noise variations within the series will mask the anomalous shape if the time series are collected over long periods of time.

- *Subsequence-based anomaly:* A single time series has typical patterns over shorter time periods as Figure 2.1 shows. Therefore, the anomalous shape is detected over small windows of the time series as deviations from these typical patterns.

In subsequence-based anomaly detection, the entire series is always divided into several subsequences and then the extracted subsequences are treated as whole series, which enables that most techniques for subsequence-based anomaly detection can always be applied to full-series anomaly detection with some subtle changes. Therefore, this section is only concerned with techniques for subsequence-based anomaly.

How to construct subsequences is important for subsequences-based anomaly detection. Mostly, the techniques divide the given time series into fixed size windows (subsequences). If the subsequences are obtained by sliding one step at a time, considering all possible overlaps, it gets computationally inefficient since the number of subsequences is nearly equal to the length of the time series. Alternatively, it can be avoided by sliding a window of fixed length of m across the time series and skipping h observations from the initial position of the current window to start the next window. However, with a large h there can be some loss of information. Thus the value of h has to be carefully chosen. Subsequently, various types of distance based, neural network based, or information theoretic methods can be applied to the extracted windows.

Distance based methods or *Nearest neighbor-based techniques* assume that normal data instances occur in dense neighborhoods, while anomalies occur far from their closest neighbors [CBK09]. The given time series are divided into subsequences in some manner at first. The anomaly score of each subsequence is calculated by the k-nearest neighbor distance of the subsequences. Distance measures could be Euclidean distance, Manhattan distance etc.

Probabilistic approaches can also be applied to detecting collective anomaly. The training time series is split into several sequences, each sequence would be viewed as a data point in a high dimension. Then, estimating the probability density function is the same to that in detecting point anomaly.

Neural network based approaches can autonomously model the underlying data, e.g. an autoencoder neural network is an unsupervised learning algorithm setting the target values to be equal to the inputs. Given a training time series and extract subsequences in some ways, an autoencoder is trained for the purpose of sequence-sequence learning, so that output is similar to input. The reconstruction error of the sequences of test data, defined to be the distance between the input and output of the learned neural network, can be related to the anomaly score. The replicator neural network (RNN) is a variant of a feed-forward deep neural network (DNN) where the input and output layers have the same size but the hidden layers have smaller sizes. Setting the target values as the input forms an auto-encoder [HHWB02]. Unlike a feed-forward DNN, a recurrent neural network (RNN[2]) contains recurrent loops where the cells' output state is fed back into the input state. Such recurrent connections give recurrent neural network the ability to have information persistence or a temporal state, therefore forming short term memory [MKB+10]. A long short-term

[2]Recurrent neural network and replicator neural network share the same abbreviation.

memory network (LSTM) is simply another form of recurrent neural network where rather than a simple recurrent loop at each recurrent cell, the LSTM introduces a more complex cell architecture for more accurately maintaining memory of important correlations [SSB14]. Malhotra et al. [MRA+16] propose a Long Short Term Memory Networks based Encoder-Decoder scheme for Anomaly Detection (EncDec-AD) that learns to reconstruct "normal" time-series behavior, and thereafter uses reconstruction error to detect anomalies.

Information theoretic methods compute the information content of a dataset using measures such as entropy, relative entropy, etc. These methods assume that anomaly significantly alters the information content of the otherwise "normal" dataset. Keogh et al. [KLR04] propose parameter-free methods based on compression theory. In such a case, the time series is divided into several subsequences using a sliding window. Each subsequence is then compared with the entire sequence using a compression-based dissimilarity method, the Kolmogorov complexity. Wang et al. [WVL+11] present a method based on the relative entropy and multinomial goodness-of-fit test. For the purpose of anomaly detection, the time series is first quantized and discretized into k values, e.g. the percentage of CPU utilization which takes value between 0 and 100 can be quantized into 10 buckets. For a subsequence, p_i denotes how possible values of this subsequence falls into ith bucket. Thus, each subsequence corresponds to a distribution $P = (p_1, ..., p_k)$. A subsequence is declared anomaly when it largely deviates from the rest of subsequences by performing the multinomial goodness-of-fit test with a threshold based on an acceptable false negative probability.

2.2.3 Streaming Applications

Most techniques discussed before separate the training phase from detection, a training time series is always viewed as a reference data to represent the normality by a learned model. Some of them e.g. distance based and information theoretic techniques are performed in an offline setting where anomalies can be detected only after seeing the entire time series. In such cases, the advantage of hindsight may be leveraged to identify abnormal time-series values or shapes [Agg17]. However, when it comes to streaming application where a continuous sequence of data occurring in real-time, the full data set is not available and a model trained offline is highly likely not suitable for detection because the data source is always not stationary. Therefore, the online setting is more realistic when dealing with the tremendous amount of data in monitoring. The detector observes each data record in sequential order as they arrive and process data and output a decision in real-time. Since data source is often non-stationary, detectors must continuously learn and adapt to the definition of an anomaly while simultaneously making decisions.

Let x_t denote the value of a real-time metric in a system at time t, e.g. CPU utilization in a data center. The detector receives a continuous stream of inputs $(..., x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}, ...)$. At each point in time t the detector needs to determine whether the behavior of the monitored metric is unusual, therefore, it is more suitable to treat the problem as *point anomaly detection*. The decision must be made in real-time, before time $t + 1$, in other words, before the next input x_{t+1} arrives, the detector has already considered the current and previous values to determine whether or not the behavior of the monitored metric is abnormal. Unlike the offline

setting, the detector can never look ahead and data is not split into train and test set.

Moreover, software upgrades and configuration changes can occur at any time and may alter the behavior of the monitored metrics, resulting in that the data source is non-stationary, in such cases, the detector must adapt to a new definition of "normal" in an unsupervised, automated fashion [ALPA17].

Early detection of anomalies is always necessary in streaming applications because sometimes an anomaly could be a precursor to failure of system. Detecting such an anomaly minutes in advance is far better than detecting it a few seconds ahead, or detecting it after the fact [ALPA17], so that it can give an alert early enough which enables the domain-experts to take some efficient actions to prevent system failure. But if a detector makes frequent inaccurate detections, the domain experts would be often asked to check the data and every time they need to inspect the data more closely, finally they will not trust the detector any more.

Take the above requirements together, we can see that anomaly detection for streaming applications is particularly challenging but very meaningful for system monitoring in business applications. Therefore, one of the goals of this thesis is to introduce an anomaly detection algorithm designed for such real-time applications in detail; see Chapter 3.

2.2.4 Anomaly Detection versus Change Detection

As mentioned in Chapter 1, the assumption of temporal continuity is critical in identifying anomalies because the values in consecutive time stamps are not expected to change very abruptly, or change in a smooth way. Therefore, the anomaly is always caused by *sudden changes* and exhibits a lack of continuity with its immediate or long-term history [Agg17]. Thus, the problem of anomaly detection in time series is highly related to the problem of change detection but they are not necessarily identical [Agg16]. The changes in a time series could happen in one of two possible ways:

- The values and trends in a time series change slowly over time, e.g. memory leak. Sometimes the phenomenon is referred to as *concept drift*[3]. Under such circumstance, the concept drift can only be detected by analysis over a long time period and it is not immediately obvious in many cases [Agg16].

- The values and trends in a time series change abruptly, so that it immediately raises doubts that something unexpected is going on and the underlying data generation mechanism has somehow changed fundamentally [Agg16].

Many scenarios of change analysis and anomaly detection in temporal data are too tightly integrated to be treated separately. Sometimes, solutions for one can be used for the other and vice versa. However, the modeling formulations of anomaly detection in temporal data are very diverse, not all of which are directly related to

[3]In predictive analytics and machine learning, the concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways.

change detection [Agg16]. Even though the detection in streaming applications is performed over time, it still makes decision just at each point in time whether the behavior of the monitored metric is abnormal or not. For change detection, it is always necessary to analyze the trend but how to determine the boundary of the trend is difficult in this case, which cannot be realized by the detection in streaming applications mentioned before. Therefore, detection in streaming application in this thesis is related to anomaly detection.

2.2.5 Conclusions

Time series data always require the analysis of each series as a unit, anomaly detection is one of the prominent research direction recently. In such a case, different types of anomalies can be defined in time-series data, depending on whether it is desirable to identify deviating points in the series (*point anomaly*), or whether it is desirable to identify unusual shape subsequences (*collective anomaly*). Since temporally contextual dependencies is important in anomaly detection with time series data, either point anomaly or collective anomaly is interpreted as *contextual anomaly*. Most existing techniques in anomaly detection are suitable in the offline setting where the entire time series are available and the training phase is always separated from detection. However, the majority of them are not applicable to streaming applications in real time. Thus, an efficient algorithm designed for real-time applications will be introduced in Chapter 3.

2.3 Exploratory Causal Analysis

When some anomalies in a certain metric are detected, it may reflect that something wrong is going on in the system, so it is necessary to learn a temporal-causal structure of the underlying system which can prevent the expert from contemplating irrelevant metrics and find some root causes for the problems, or give the expert an overall situation of the underlying system to support him to make some interventions to keep the system stable.

Let $S = \{Z_1, Z_2..., Z_D\}$ represent the D available metrics[4] in a system. Assume that at time t, an anomaly in metric Z_i is detected, i.e. $z_{i,t}$ is anomalous, which makes Z_i critical. Next the expert wants to know what has caused the anomaly and how it affects other metrics in the system. To this end, a relatively short time series is extracted from each metric in S from time stamp $t - b$ to $t + a$, in other words, each time series is a small collection of points around time stamp t because we are only concerned with what has happened around the anomaly point. The value of a and b depends on the settings of different applications. Now the problem turns out to be constructing a temporal-causal structure of S with some time series causality tools, implying the interactions among the metrics of a system from $t - b$ to $t + a$.

2.3.1 Causality Studies

The study of causality dates from Ancient Greece and has a long history in the philosophy. Since then there have been a number of philosophers and scientists continuously devoting themselves to the foundation and connotation of causality.

[4]Each metric forms a streaming data.

McCraken [McC16] gives two categories of causality: *foundational causality* and *data causality* with respect to the four primary focuses for people studying causality proposed by Holland [Hol86]: the ultimate meaningfulness of the notion of causality, the details of causal mechanisms, the causes of a given effect, and the effects of a given cause. Data causality studies are characterized by the use of experimental or observational data in discussion of causality, while foundational causality is broadly inter-disciplinary and three of the most prominent subfields are philosophical studies, natural science studies, and psychological studies. However, the two categories are not considered disjoint, e.g. a single study may include the introduction a completely new definition of "causality" and present several examples using empirical data to motivate the veracity of the definition [McC16]. Therefore, time series causality discussed in this thesis is considered a subset of data causality because there is no new definition of "causality" and causal inference is only drawn from synthetic data or available metrics of SAP HANA database.

Most modern explanations of causality may be traced back to Hume's regularities of observation, where event Y causes event X if whenever Y happens, X follows, and the relationship can be detected through observation of Y and X [KM09]. However, when it comes to inference, main concepts are rooted in probabilistic theories because few relationships are actually deterministic and causal utterances are often used in situations that are plagued with uncertainty [J.P09]. Consider a causal expression that "you fail the examination because of your laziness", it just indicates that the antecedents only tend to make the consequences more likely, but not absolutely certain. Therefore, probability theory is currently the official language of most disciplines that use causal modeling [J.P09].

2.3.2 Exploratory versus Confirmatory Causal Inference

A mathematically and statistically general definition of causality given by Wiener involves two facts [YY16]: (1) the *statistical condition*: event Y causes event X, indicating that there exist the corresponding probabilistic dependencies between Y and X, that is $P(Y_{t'}) > 0$ and $P(X_t|Y_{t'}) \neq P(X_t)$, and (2) the *temporal priority*: only the past and present may cause the future but the future cannot cause the past, which means $t' < t$. The majority of causal inference tools for time series data assume the correctness of these principles, even though there have been extensive debates on their validity and generality among philosophers and physicists over a long time [McC16]. For example, from the perspective of Pearl [J.P09], temporal precedence among variables may furnish some information about causal relationships, however, the great majority of policy-related questions *cannot* be discerned from such temporally indexed distributions, given the commitment to making no assumption regarding the presence or absence of unmeasured variables. Therefore, Pearl has distinguished *statistical parameter* from *causal parameter* as follows:

- A *statistical parameter* is any quantity that is defined in terms of a joint probability distribution of observed variables, making no assumption whatsoever regarding the existence or nonexistence of unobserved variables [J.P09].

- A *causal parameter* is any quantify that is defined in terms of a causal model and is not a statistical parameter [J.P09].

Following Pearl's definition, the time series causality tools discussed in this thesis will be classified as statistical tools rather than causal tools because we are in a situation where we have nothing but only a set of time series. In such a case, no special causal structures are already known or assumed. Moreover, the given data may not be complete or there are some outside hidden "knowledge" or "theories" that we are not very clear in the analysis phase, e.g. race or gender in social studies, therefore there is usually no effort made to determine if one time series fits any specific definition of "cause" with respect to the other time series being analyzed [McC16]. This problem is also discussed by Holland and in his opinion, such tools are associational tools, not causal tools [Hol86].

McCracke [McC16] proposes a new concept called *exploratory causal analysis*, emphasizing that the time series causality tools are only *statistical* and *associational* tools, the relationships found with such tools can only be deemed "causal" associated with the application of outside theories or assumptions, which requires an analyst to make general causal statements to be well-versed in any such theories involving the analyzed data. The term *exploratory* applied to these causal inferences is inspired by the way used by Tukey [Tuk77] to describe exploratory data analysis as McCracke said. Exploratory data analysis refers to looking at the data to see what it seems to say and drawing conclusions only from the given data, in other words, it leaves most interpretations of results to those who are experts in the subject-matter filed involved [Tuk77], which is a part of confirmatory data analysis. In the context of causal inference, confirmatory data analysis means examining the precision with which the data reflect some assumed causal structures or some general notions of causality, e.g. in physics and philosophy. Therefore, the word "potential" is very important to the exploratory causal analysis because any time series causality tool may incorrectly assign causal structure or may incorrectly not assign causal structure between time series, e.g. the temporal causal effect from time series Y to X ($Y \rightarrow X$) is obtained by a common time series causality tool, however, in fact, there is some unknown time series Z, not included in the given data, having causal impact on X and Y respectively ($Z \rightarrow X$ and $Z \rightarrow Y$) with different delays, as a consequence, the causal effect $Y \rightarrow X$ is actually spurious.

Under such circumstances, it is important to distinguish the statistical tests used in exploratory causal analysis from that in confirmatory causal analysis because some time series causality tools, such as Granger causality are concerned with developing a "testable" definition of causality, e.g. a null hypothesis H_0 is defined as "the coefficients of lagged variable of Y are jointly zero" in a VAR model[5]. Let H_0 represent that Y does not Granger cause X. If H_0 is rejected, there exists linear Granger-causality running from Y to X. But the result of the test does not mean that it *confirms* the causal structure between Y and X, i.e. it does not imply a given causal inference is confirmatory rather than exploratory. Statistical tests are also used in confirmatory causal analysis, for example, by defining a null hypothesis of some model parameter equal to a theoretically justified value. The point of such tests in confirmatory causal analysis is to *confirm* those theoretically justified models or parameter values [McC16].

[5]See Section 4.2.1.

To summarize, proof of causal relationship is often considered impossible with data alone [J.P09], so one of the main purposes of the exploratory analysis is to guide confirmatory analysis for further investigation [McC16].

2.3.3 Time Series Causal Inference Tools

Graphical models, such as Bayesian networks (BNs) first introduced in [J.P09, SGS93] model the causal structure of the system as a Graph, where variables are represented by nodes and the edges between them represent conditional dependence. The result is a directed acyclic graph (DAG) where a directed edge between two nodes means the first causes the second. Dynamic Bayesian networks (DBNs) [FMR98] extend BNs to show how the system evolves over time. For this purpose, they generally begin with a prior distribution (described by a DAG structure) as well as two more DAGs: one representing the system at time t and another at $t+1$, where these hold for any values of t. The connections between these two time stamps then describe the change over time. As before, there is usually one node per variable, with edges representing conditional independence. Note that this implies that while the system can start in any state, after that the structure and dependencies repeat themselves.

Information-theory can also be linked to Wiener's causality theorem through the development of a novel concept called *transfer entropy* (TE) [Sch00]. Intuitively, transfer entropy can be conceptualized as a model-free measure of directed information flow from one variable to another. Schreiber originally motivated transfer entropy as an alternative to lagged mutual information that takes shared information into account due to common history and input signals. Transfer entropy for the direction $Y->X$ is an information-theoretic distance measure between the transition probability that includes information from Y and the one that excludes it. Additionally, Schreiber showed that transfer entropy is able to distinguish direct from indirect causality. However, transfer entropy and some other similar approaches have mostly been applied to a bivariate setting as it is hard to estimate these measures reliably in high dimensions.

Granger causality [Gra69] applied primarily to economics, was developed by Granger (1969) to take two time series and determine whether one predicts, or causes, the other with some lag time between them. Based on this, recent work by Eichler and Didelez [ED09] focuses on time series and explicitly capturing the time elapsed between cause and effect. They define that one time series causes another if an intervention on the first alters the second at some later time. That is, there may be lags of arbitrary length between the series, and they find these lags as part of the inference process. Transfer entropy has a close connection with Granger causality, but Granger causality can be applied to measure not only linear but also non-linear temporal-causal dependencies between multivariate time series, which is the second goal of this thesis. Therefore, a number of variants which fall under the category of Granger causality are introduced in detail in Chapter 4.

2.3.4 Conclusions

Learning temporal-causal structures among multiple time series is one of the major tasks in mining time series data. In this work, it will be applied to support the engineers to understand the overall situation of the underlying system when anomalies

are detected in a certain metric. Since no special causal structures are already known or assumed and analysis is not intended to draw causal information from sources outside of the time series being analyzed, tools used for causal inference in this work are deemed as statistical and associational tools but not causal tools. Thus, causal inference is modified by a term called "exploratory causal analysis", emphasizing that there are no attempts are made to relate the interpretations of the results to more general notions of causality, e.g., in physics and philosophy. The results just imply some potential temporal-causal structure between time series. Granger causality, as a widely used concept to identify causal relationships between time series data, is chosen to solve the problem in this work; see Chapter 4.

3. Online Prototypes Updating Anomaly Detection

This chapter introduces the OPUAD (*On-line Prototypes Updating Anomaly Detection*) algorithm for streaming data based on the kernel density estimators and maximum-entropy principle. OPUAD detects anomalies in an unsupervised, automated fashion without supervision, which makes it self-adaptive and computationally efficient.

Real-time anomaly detection for streaming data is distinct from batch anomaly detection. Streaming analytics calls for models and algorithms that can learn continuously in real-time without storing the entire stream, and are fully automated and not manually supervised. Even though both supervised and unsupervised anomaly detection approaches have existed, the majority of anomaly detection methods are for batch data processing, that does not fit real-time streaming scenarios and applications.

Moreover, detecting anomalies accurately in streaming data can be difficult; the definition of an anomaly is continuously changing as systems evolve and behaviors change. For example in Figure 3.1, there is a sudden change in the data value at time stamp 9. This corresponds to an anomaly compared to the previous data points, however, subsequently, the data stabilizes at this value, and this becomes the *new normal*. Thus, the data points after time stamp 9 should not be considered as anomalies and the algorithm must adapt to the new normal automatically. From this example we can see that anomalies in time series data are always contextual anomalies because it can never be treated independent of other points and temporally contextual dependency is always important.

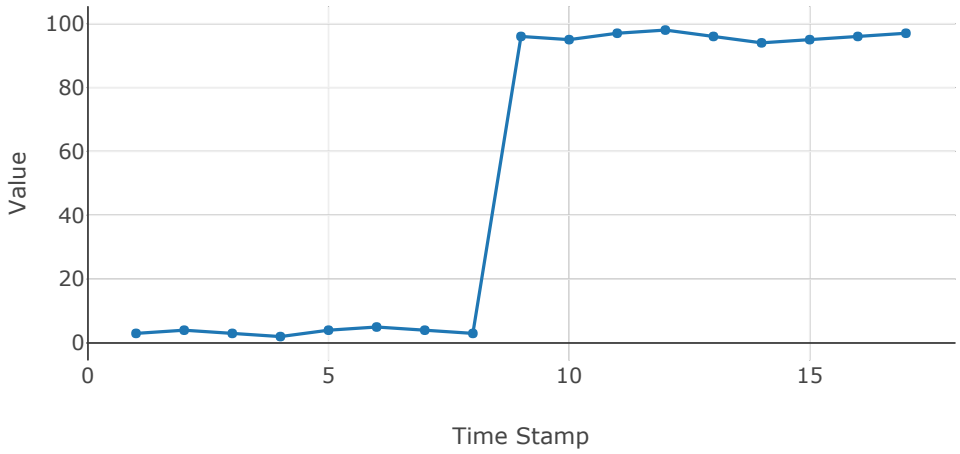


Figure 3.1: Example of System Behaviors Change.

Another critical aspect is early detection of anomalies in streaming data, as the focus lies in identifying anomalies before incurring some fatal events, e.g. the whole system has crashed and cannot work any more, it is unlike batch processing where the model is trained to look back.

Given the above requirements, Ahmad et al. [ALPA17] define the ideal characteristics of a real-world anomaly detection algorithm as follows:

- Detection must be made online; i.e., the detector receives a continuous stream of inputs $(..., x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}, ...)$, the algorithm[1] must identify state x_t as normal or anomalous before receiving the subsequent x_{t+1}.
- The algorithm must learn continuously without a requirement to store the entire stream.
- The algorithm must run in an unsupervised, automated fashion, i.e., without data labels.
- Algorithms must adapt to dynamic environments and system behaviors change automatically.
- Algorithms should make anomaly detections as early as possible.
- Algorithms should minimize false positives and false negatives.

This chapter intends to introduce the OPUAD (*On-line Prototypes Updating Anomaly Detection*) algorithm for streaming data based on the *kernel density estimators* and *maximum-entropy principle*, first proposed in [Gra90]. Kernel density estimators, as a non-parametric method have proven popular to estimate the probability density function (pdf) of a given data set, while the maximum-entropy principle is based on the premise that when estimating the probability distribution, we should select that distribution which leaves us the largest remaining uncertainty (i.e., the maximum entropy) consistent with the constraints of our prior knowledge. First, let's take a look on the kernel density estimators.

[1]The problem is treated as *point anomaly detection*; see Section 2.2.1.

3.1 Probability Density Function Estimation

Consider a data set of N data points $X = (x_1, ..., x_N)$ drawn from some unknown distribution in one dimension. The probability density function of the unknown distribution can be estimated by a Gaussian mixture model

$$f(x|\theta) = \sum_{i=1}^{M} w_i g(x|\mu_i, \sigma_i^2) \tag{3.1}$$

where w_i, $i = 1, ..., M$, are the mixture weights, the individual Gaussian components are always normalized, thus we obtain $\sum_{i=1}^{M} w_i = 1$, and $g(x|\mu_i, \sigma_i^2)$, $i = 1, ..., M$, are the component Gaussian densities

$$g(x|\mu_i, \sigma_i^2) = \frac{1}{(2\pi\sigma_i^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma_i^2}(x - \mu_i)^2\right\} \tag{3.2}$$

This method always assumes that the data points are drawn independently from the same distribution, in other words, the data points are said to be *independent and identically distributed*. However, this assumption is not applicable to the time series data because data points in a time series are always related to each other temporally and influenced by the adjacent values of the data points. Besides, some data generation mechanisms could have local nonlinearity, which will lead to non-Gaussianity of the distribution. Therefore, GMM might be a poor model of the distribution that generates the data, which can result in poor predictive performance. Under such circumstances, kernel density estimators, as *nonparametric* approaches that make few assumptions about the form of the distribution are preferable to density estimation [Bis06]. Since they are closely related to histogram methods, thus, let's first return to histogram methods.

3.1.1 Histogram Method

Given a data set $X = (x_1, ..., x_N)$ in one-dimension, a histogram method simply partitions X into different bins of equal width Δ. The set of data points falling into the i-th bin is denoted as $\mathbb{S}(i)$. Thus the probability of data points falling into i-th bins then is calculated as follows

$$p_i = \frac{|\mathbb{S}(i)|}{N}$$

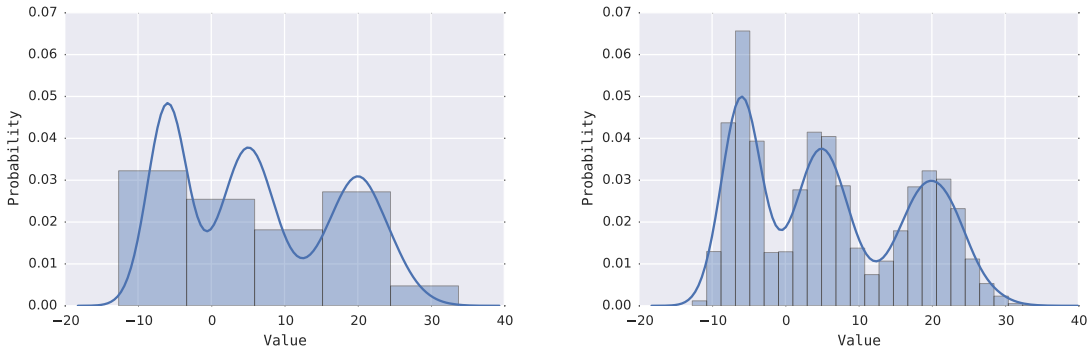
The probability density function of X is denoted as $f(x)$, and it is evident that $f(x)$ is constant over the width of each bin, so we can obtain

$$f(x) \times \Delta = \frac{n_i}{N} \quad \forall x \in \mathbb{S}(i)$$

thus

$$f(x) = \frac{n_i}{N \times \Delta} \quad \forall x \in \mathbb{S}(i) \tag{3.3}$$

for which it is easy to see that $\int f(x)\mathrm{d}x = 1$. Figure 3.2 shows a random sample actually drawn from a mixture of three Gaussians. Now we use histogram methods with different bin widths to model the distribution of the sample. From the two



(a) (b)

Figure 3.2: An illustration of the Histogram Approach to Density Estimation.

subplots we can see that the histogram methods are sensitive to the bin width, e.g. when Δ is too large (Figure 3.2a), the model is too smooth and it cannot capture the trimodal property of the dark blue curve.

Note that the histogram method has the property that, once the histogram has been computed, the data set itself can be discarded, which can be advantageous if the data set is large. Besides, it can be useful for obtaining a quick visualization. However, it is not scalable with high dimensionality. Image that now you have a D-dimensional variable, you divide each dimension into M bins, then the total number of bins will be M^D. This exponential scaling with D is an example of the curse of dimensionality. Another problem is that the estimated density has discontinuities that are due to the bin edges rather than any property of the underlying distribution that generates the data [Bis06].

From histogram methods, we have known that, to estimate the probability density at a particular location, we should consider the data points that lie within some local neighborhood of that point. With this insight, now we turn to a discussion of kernel density estimators.

3.1.2 Kernel Density Estimators

Consider a D-dimensional continuous variable $\boldsymbol{X}$ with its probability density function $f(\boldsymbol{x})$. Inspired by the histogram methods, now we consider some small region $\mathcal{R}$ and the probability mass associated with this region is given by

$$P = \int_{\mathcal{R}} f(\boldsymbol{x}) \mathrm{d}x \tag{3.4}$$

If $\mathcal{R}$ is sufficiently small such that $f(\boldsymbol{x})$ is roughly constant in $\mathcal{R}$, thus we can obtain

$$P = \int_{\mathcal{R}} f(\boldsymbol{x}) \mathrm{d}x \approx f(\boldsymbol{x}) \cdot V \tag{3.5}$$

where V denotes the volume of $\mathcal{R}$.

Now we have randomly collected a data set $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N)$ comprising N data points drawn from $f(\boldsymbol{x})$. Since each point has a probability P of falling within $\mathcal{R}$,

the probability of the total number K of points that lie inside $\mathcal{R}$ with respect to the binomial distribution[2] is given by

$$f(K, N, P) = \frac{N!}{K!(N-K)!} P^K (1-P)^K \tag{3.6}$$

Due to the properties of the binomial distribution, we can obtain

$$E(K) = NP \tag{3.7}$$
$$Var(K) = NP(1-P) \tag{3.8}$$

and for large N, we can get

$$K \approx NP \tag{3.9}$$

Combining (3.5) and (3.9), the density estimate can be formed as

$$f(\boldsymbol{x}) = \frac{K}{NV} \tag{3.10}$$

Kernel density estimators exploit the result (3.10) by fixing V and determining the number K of data points inside the small hypercube $\mathcal{R}$. Assume that $\mathcal{R}$ is centered on the point $\boldsymbol{x}$. For the purpose of counting number K of points falling within the hypercube $\mathcal{R}$, we define a variable

$$\boldsymbol{u} = \frac{\boldsymbol{x} - \boldsymbol{x}_n}{h} \tag{3.11}$$

where $\boldsymbol{u}$ is a D-dimensional vector. Let h be the length of the edge of the hypercube $\mathcal{R}$, then $V = h^D$. Since the cube is centered on $\boldsymbol{x}$, we can easily obtain that if $\boldsymbol{x}_n$ lies inside the cube, then $|u_i| \leq 1/2$, $\forall i = 1, 2, ..., D$. For convenience, we define a *kernel function*

$$k(\boldsymbol{u}) = \begin{cases} 1 & |u_i| \leq 1/2 \quad \forall i = 1, 2, ..., D \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

which means if the point $\boldsymbol{x}_n$ lies inside the cube, then $k(\boldsymbol{u})$ is one and otherwise, $k(\boldsymbol{u})$ is zero. Now it is easy to get the total number K of data points lying in this cube

$$K = \sum_{n=1}^{N} k(\frac{\boldsymbol{x} - \boldsymbol{x}_n}{h}) \tag{3.13}$$

Substituting (3.13) into (3.10), we obtain

$$f(\boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} k(\frac{\boldsymbol{x} - \boldsymbol{x}_n}{h}) \tag{3.14}$$

[2]Binomial distribution, see https://en.wikipedia.org/wiki/Binomial_distribution.

The kernel function (3.12) is also called a *Parzen window*. The estimated probability density function is sometimes called *empirical density function* because it is only estimated by the given sample data. In addition, it is not necessary to hold the assumption that data points are *independent and identically distributed*. Generally, h is considered as a smoothing parameter called the *bandwidth* and a kernel function must satisfy the following properties

- *Non-negative*: $k(\boldsymbol{u}) \geq 0$;
- *Normalization*: $\int k(\boldsymbol{u})\mathrm{d}\boldsymbol{u} = 1$;
- *Symmetry*: $k(-\boldsymbol{u}) = k(\boldsymbol{u})$.

However, the kernel function (3.12) will suffer from one of the same problems that the histogram method suffered from, namely the presence of artificial discontinuities, in this case at the boundaries of the cubes [Bis06]. In order to solve the problem of discontinuities, the Gaussian kernel is commonly used as a smoother kernel function. It is defined as follows:

$$k(\boldsymbol{u}) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{||\boldsymbol{x} - \boldsymbol{x}_n||^2}{2\sigma^2}\right\} \tag{3.15}$$

where σ determines the bandwidth of the Gaussian kernel. For a one-dimensional continuous variable X, (3.15) is easily changed to

$$k(u) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{(x - x_n)^2}{2\sigma^2}\right\} \tag{3.16}$$

As mentioned before, for a streaming data $(x_1..., x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}, ...)$, at each time stamp t, the algorithm should only consider the current and previous states to decide whether or not x_t is an anomaly, as well as perform any model updates and retraining. However, if we use a kernel density estimator to solve the problem, it does not need to update any parameters because it is a non-parametric method. When x_t is coming, it just needs to estimate the empirical density of x_t using the sequence $X^{(t)} = (x_1, ..., x_t)$ by

$$f_e^{(t)}(x) = \frac{1}{t}\sum_{\tau=1}^{t} k(x - x_\tau) \tag{3.17}$$

It is evident that when x_t is far away from the previous value, its empirical density would be very low indicating that x_t is an anomaly. However, in this case, we have to store the streaming data over time which inevitably leads to saturation of discrete memory devices finally. Besides, with larger t, the computation is always not efficient. Therefore, we want to use a few statistics to compress the information of the sequence $(x_1, ...x_t)$ and simultaneously the information loss is minimal, then the probability density could be estimated using these statistics. In this way, no matter how many data value occur in real-time, they can always be compressed in these statistics and we do not need to store the entire stream but just these statistics, and memory is no longer a problem. The compression is based on the maximum-entropy principle, which will be discussed in the next section.

3.2 Maximum Entropy Principle

Assume that in the sequence $X^{(t)} = (x_1, ...x_t)$ all the data value are different, therefore, the frequency of each value is equal, which means, $p(x_\tau) = 1/t$ and $\sum_{\tau=1}^{t} p(x_\tau) = 1$. With these frequencies, we define the *entropy* of empirical information of this sequence by

$$S_{X^{(t)}} = -\sum_{\tau=1}^{t} p(x_\tau) \log p(x_\tau) = \log t \tag{3.18}$$

So when data continuously occurs in real-time ($t \to \infty$), the entropy grows to infinity. Now we want to map the sequence $X^{(t)} = (x_1, ...x_t)$ into a few statistics $Q^{(t)} = (q_1^{(t)}, ..., q_K^{(t)})$, which will be termed *prototypes*, so that the information in the original sequence is minimally reduced. We also use the entropy to quantify the information of $Q^{(t)}$

$$S_{Q^{(t)}} = -\sum_{k=1}^{K} p(q_k^{(t)}) \log p(q_k^{(t)}) \tag{3.19}$$

Since K is fixed and much less than t ($t \to \infty$), the information entropy is generally reduced by the mapping of the sequence $X^{(t)}$ to $Q^{(t)}$. In order to maximize the entropy of $Q^{(t)}$, all the prototypes in $Q^{(t)}$ should be equally probable.

Definition 3.1. *The principle of maximum entropy dedicates that the mapping of $X^{(t)}$ into $Q^{(t)}$ minimally reduces the empirical information at time point t if a uniform probability distribution*

$$p(q_i^{(t)}) = \frac{1}{K} \qquad i = 1, ..., K$$

corresponding to the absolute maximum $S_{Q^{(t)}} = \log K$ of information entropy, is assigned to Q.

Therefore, the above definition gives an optimal solution for the mapping of $X^{(t)}$ into $Q^{(t)}$. By the prototypes $Q^{(t)}$, the probability density at time t then can be approximately estimated by

$$f_r^{(t)}(x) = \frac{1}{K} \sum_{k=1}^{K} k(x - q_k^{(t)}) \tag{3.20}$$

which can be adapted to the empirical density (3.17) when the proper prototypes are selected. In order to distinguish (3.17) and (3.20), we call (3.20) as *representative density*. Now we can define the mean square error between (3.17) and (3.20) to measure their overall discrepancy

$$\epsilon_t^2 = \int_{-\infty}^{+\infty} \left(f_r^{(t)}(x) - f_e^{(t)}(x) \right)^2 \mathrm{d}x \tag{3.21}$$

Hereby we give a definition of the optimal prototypes at time t

Definition 3.2. *The set* $(q_1^{(t)}, ..., q_K^{(t)})$ *which minimizes* ϵ_t^2 *is the optimal representation of* $X^{(t)}$ *by* $Q^{(t)}$ *at time* t *with respect to the applied kernel function.*

Now the problem is to minimize the mean square error in (3.21). For convenience, we define

$$\epsilon_t(x) = f_r^{(t)}(x) - f_e^{(t)}(x) \tag{3.22}$$

and then (3.21) can be rewritten into

$$\epsilon_t^2 = \int_{-\infty}^{+\infty} \epsilon_t(x)^2 \mathrm{d}x \tag{3.23}$$

Taking the derivative with respect to $q_l^{(t)}$ and setting this derivate to zero is straightforward,

$$\frac{\partial \epsilon_t^2}{\partial q_l^{(t)}} = 2 \int_{-\infty}^{+\infty} \epsilon_t(x) \frac{\partial \epsilon_t(x)}{\partial q_l^{(t)}} \mathrm{d}x = 0 \quad l = 1, ..., K \tag{3.24}$$

in this way, we can obtain the solution for minimizing the mean square error. For a given K, a trivial solution of (3.24) can be found by inspection of expression (3.22), for two time points:

$$\text{for} \;\; t = 1, \;\; \text{then} \quad q_1^{(t)} = q_2^{(t)} = \cdots = q_K^{(t)} = x_1 \tag{3.25}$$

$$\text{for} \;\; t = K, \;\; \text{then} \quad q_1^{(t)} = x_1, q_2^{(t)} = x_2, ..., q_K^{(t)} = x_t \tag{3.26}$$

The first case means that the first data point x_1 is coming, while the second means a certain moment when the number of data points is equal to the number of prototypes. However, when data continuously occurs in real-time, t could be much greater than K, in such a case, an explicit solution cannot be found, thus we proceed with an approximative treatment.

Assume that the solution of (3.21) is known, in other words, the optimal prototypes $Q^{(t)}$ at time t is determined and the number of data points is much greater than K, i.e. $t >> K$. Now a new data x_{t+1} is coming, then we want to map the new sequence $X^{(t+1)} = (x_1, ..., x_t, x_{t+1})$ into a set of prototypes $Q^{(t+1)}$ and find the optimal solution. At time $t+1$, the empirical density function is given by

$$\begin{aligned} f_e^{(t+1)}(x) &= \frac{1}{t+1} \sum_{\tau=1}^{t+1} k(x - x_\tau) \\ &= \frac{1}{t+1} \left(t f_e^{(t)}(x) + k(x - x_{t+1}) \right) \end{aligned} \tag{3.27}$$

So the difference between (3.27) and (3.17) is given as

$$\begin{aligned} \Delta f_e(x) &= f_e^{(t+1)}(x) - f_e^{(t)}(x) \\ &= \frac{1}{t+1} \left(k(x - x_{t+1}) - f_e^{(t)}(x) \right) \end{aligned} \tag{3.28}$$

A new data point x_{t+1} will lead to some minor changes $\Delta Q^{(t)}$ to $Q^{(t)}$

$$Q^{(t+1)} = Q^{(t)} + \Delta Q^{(t)} \tag{3.29}$$

Then, the representative density will be calculated as

$$\begin{aligned} f_r^{(t+1)}(x) &= \frac{1}{K}\sum_{k=1}^{K} k(x - q_k^{(t+1)}) \\ &= \frac{1}{K}\sum_{k=1}^{K} k(x - q_k^{(t)} - \Delta q_k^{(t)}) \end{aligned} \tag{3.30}$$

and the difference between (3.30) and (3.20) is given by

$$\begin{aligned} \Delta f_r(x) &= f_r^{(t+1)}(x) - f_r^{(t)}(x) \\ &= \frac{1}{K}\sum_{k=1}^{K} k(x - q_k^{(t)} - \Delta q_k^{(t)}) - k(x - q_k^{(t)}) \end{aligned} \tag{3.31}$$

When $\Delta q_k^{(t)} \to 0$, we can get

$$\frac{\partial}{\partial q_k^{(t)}} k(x - q_k^{(t)}) = \lim_{\Delta q_k^{(t)} \to 0} \frac{k(x - q_k^{(t)} - \Delta q_k^{(t)}) - k(x - q_k^{(t)})}{\Delta q_k^{(t)}} \tag{3.32}$$

therefore, (3.31) is rewritten as

$$\Delta f_r(x) = \lim_{\Delta q_k^{(t)} \to 0} \frac{1}{K}\sum_{k=1}^{K} \frac{\partial}{\partial q_k^{(t)}} k(x - q_k^{(t)}) \Delta q_k^{(t)} \tag{3.33}$$

The error between $f_r^{(t+1)}(x)$ and $f_e^{(t+1)}(x)$ can then be calculated as

$$\begin{aligned} \epsilon_{(t+1)}(x) &= f_r^{(t+1)}(x) - f_e^{(t+1)}(x) \\ &= f_r(x_t) + \Delta f_r(x) - (f_e(x_t) + \Delta f_e(x)) \\ &= \epsilon_t(x) + \Delta f_r(x) - \Delta f_r(x) \\ &= \epsilon_t(x) + \underbrace{\frac{1}{K}\sum_{k=1}^{K} \frac{\partial}{\partial q_k^{(t)}} k(x - q_k^{(t)}) \Delta q_k^{(t)} - \frac{1}{t+1}\left(k(x - x_{t+1}) - f_e^{(t)}(x)\right)}_{\Delta \epsilon_{t+1}(x)} \end{aligned} \tag{3.34}$$

The difference between the empirical density and representative density at time $t+1$ comprises two parts, the one is $\epsilon_t(x)$ and the other is $\Delta\epsilon_{t+1}(x)$. For the purpose of minimizing the overall discrepancy between the empirical density and representative density at time $t+1$, we just need to minimize $\Delta\epsilon_{t+1}(x)$ because we assume $\epsilon_t(x)$ is known and minimal. Therefore, we can obtain

$$\Delta\epsilon_{t+1}^2 = \int_{-\infty}^{+\infty} (\Delta\epsilon_{t+1}(x))^2 \mathrm{d}x \tag{3.35}$$

Simply, we take its derivative with respect to $\Delta q_l^{(t)}$ and setting it to zero

$$\frac{\partial \Delta\epsilon_{t+1}^2}{\partial \Delta q_l^{(t)}} = 2\int_{-\infty}^{+\infty} \Delta\epsilon_{t+1}(x)\frac{\partial \Delta\epsilon_{(t+1)}(x)}{\partial \Delta q_l^{(t)}}\mathrm{d}x = 0 \quad l = 1, ..., K \tag{3.36}$$

Now we make some derivations of (3.36) as follows:

$$\begin{aligned} 0 &= \int_{-\infty}^{+\infty} \Delta\epsilon_{t+1}(x)\left(\frac{\partial}{\partial q_l^{(t)}}k(x - q_l^{(t)})\right)\mathrm{d}x \\ &= \int_{-\infty}^{+\infty}\left(\frac{1}{K}\sum_{k=1}^{K}\frac{\partial}{\partial q_k^{(t)}}k(x - q_k^{(t)})\Delta q_k^{(t)}\right. \\ &\qquad \left. -\frac{1}{t+1}\left(k(x - x_{t+1}) - f_e^{(t)}(x)\right)\right)\left(\frac{\partial}{\partial q_l^{(t)}}k(x - q_l^{(t)})\right)\mathrm{d}x \end{aligned} \tag{3.37}$$

We write (3.37) in a compact form as solving linear equations

$$\sum_{k=1}^{K} C_{lk}^{(t)}\Delta q_k^{(t)} - B_l^{(t)} = 0; \quad l = 1, ..., K \tag{3.38}$$

with the coefficients:

$$C_{lk}^{(t)} = \int_{-\infty}^{+\infty}\frac{\partial k(x - q_k^{(t)})}{\partial q_k^{(t)}}\frac{\partial k(x - q_l^{(t)})}{\partial q_l^{(t)}}\mathrm{d}x \tag{3.39}$$

$$B_l^{(t)} = \frac{K}{t+1}\int_{-\infty}^{+\infty}\left(k(x - x_{t+1}) - f_e^{(t)}(x)\right)\frac{\partial k(x - q_l^{(t)})}{\partial q_l^{(t)}}\mathrm{d}x \tag{3.40}$$

Using this method, we do not need to store the entire stream any more because the information of the data will always be compressed in a small fixed number of prototypes with minimal information loss. When the detector receives the first data point x_1, it will initialize K prototypes $Q^{(1)} = q_1^{(1)}, ..., q_K^{(1)}$ and set $q_1^{(1)} = q_2^{(1)} = \cdots = q_K^{(1)} = x_1$. Next, every time there is a new data point x_t coming, it will update the prototypes by two steps: first it solves the following linear equations

$$\sum_{k=1}^{K} C_{lk}^{(t-1)}\Delta q_k^{(t-1)} = B_l^{(t-1)}; \quad l = 1, ..., K \tag{3.41}$$

and then estimates the new prototypes $q_k^{(t)}$

$$q_k^{(t)} = q_k^{(t-1)} + \Delta q_k^{(t-1)}; \quad k = 1, ..., K \tag{3.42}$$

The update keeps computationally efficient. In addition, the detector makes no explicit assumption of the underlying distribution of the data, which makes it have strong applicability. The only thing the detector needs to be "equipped" with is an appropriate kernel function.

Mostly, we use the Gaussian kernels (3.16) which has been introduced in the last section. With the Gaussian kernel, the coefficients $B_l^{(t-1)}$ and $C_{lk}^{(t-1)}$ are given by the following explicit expressions [Gra90]

$$C_{lk}^{(t-1)} = (1 - \frac{(q_l^{(t-1)} - q_k^{(t-1)})^2}{2\sigma^2}) \exp\left(-\frac{(q_l^{(t-1)} - q_k^{(t-1)})^2}{4\sigma^2}\right) \tag{3.43}$$

$$B_l^{(t-1)} = \frac{K}{t}\left[(x_t - q_l^{(t-1)}) \exp\left(-\frac{(x_t - q_l^{(t-1)})^2}{4\sigma^2}\right) - \frac{1}{K}\sum_{k=1}^{K}(q_k^{(t-1)} - q_l^{(t-1)}) \exp\left(-\frac{(q_k^{(t-1)} - q_l^{(t-1)})^2}{4\sigma^2}\right)\right] \tag{3.44}$$

3.3 Anomaly Likelihood

Now we consider how to determine whether or not x_t $(t > 1)$ is anomalous when it is arriving. We estimate its probability density by the prototypes $Q^{(t-1)}$ because $Q^{(t-1)}$ represents the current information of the stream without the effect of x_t:

$$f(x_t) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{(x_t - q_k^{(t-1)})^2}{2\sigma^2}\right\} \tag{3.45}$$

If x_t is an anomaly, its estimated probability density could be very small, indicating that it is less likely to be "similar" to the previous states. However, due to the fact that the estimated probability density of a "normal" data point could also be very small, it is difficult to distinguish the anomalies from "normal" data if we rely only on the estimated probability density. In such a case, we calculate the anomaly score as follows:

$$S_{\text{anomaly}}^{(t)} = -\log f(x_t) \tag{3.46}$$

Then the anomaly score is *Logarithmic loss* which is commonly used to measure the performance of a classification model where the prediction input is a probability value between 0 and 1 in the machine learning. However, for probability density, its value could be larger than 1 (see Figure 3.3). With the help of Logarithmic loss, the difference of the estimated probability density between a normal point and an anomaly point could be big enough to be distinguished.

Figure 3.4 shows an example stream and its anomaly score over time. The stream is CPU utilization of one virtual computing environment (*instance*) provided by Amazon Elastic Compute Cloud (Amazon EC2) in the Amazon Web Services (AWS) cloud. It is collected from 2014-02-23 12:00:00 to 2014-02-26 12:00:00. Note that each instance has its own types, in other words, there are various configurations of CPU, memory, storage, and networking capacity for the instances. So it is unreasonable to set a common threshold to determine the situations of all the instances. In Figure 3.4, it is obvious that CPU usage of the instance is non-stationary and there are two anomalies in the stream. Even though the anomaly score can somehow distinguish the anomaly points from normal points, it still has some limitations. First, Logarithmic loss has no boundaries so that it lacks of interpretation, which results in that the score of the second anomaly in Figure 3.4b is too high. Additionally, the anomaly score gives us an instantaneous measure of how the data points do not

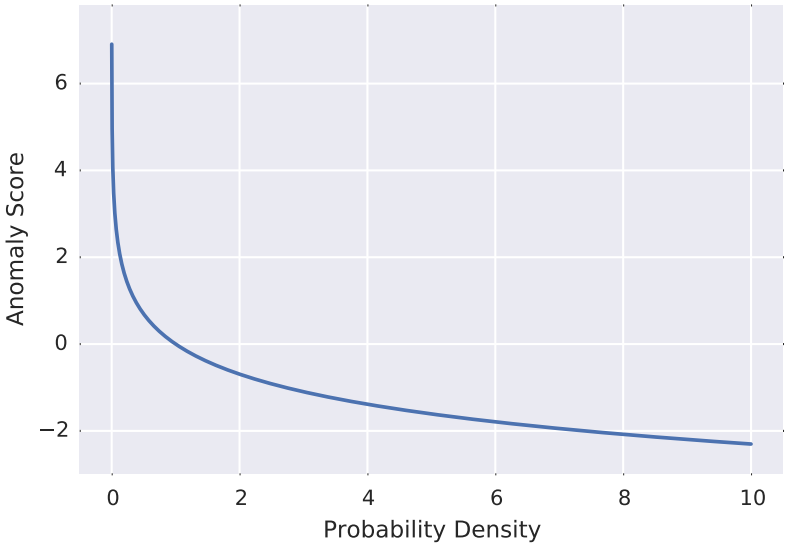


Figure 3.3: Logarithmic Loss as Anomaly Score.

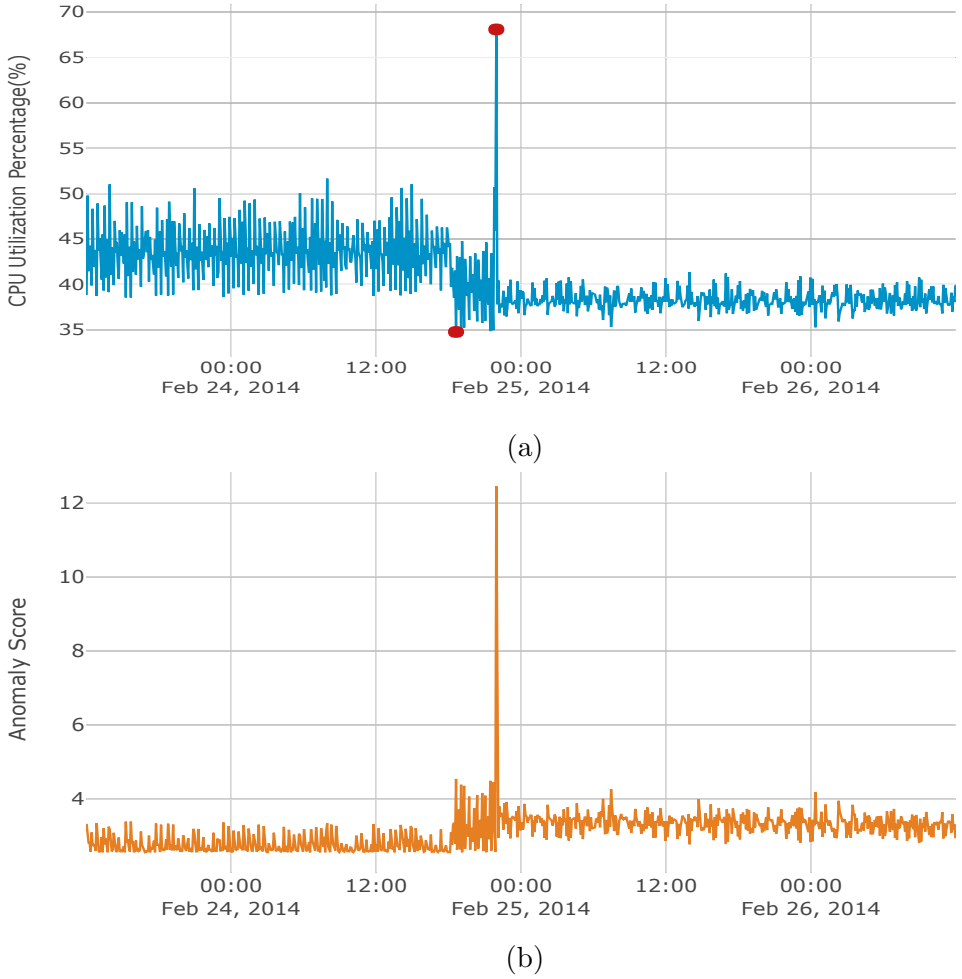


Figure 3.4: CPU Utilization of One Instance in the AWS.

match the current distribution. However, it is often incorrect when the behavior of the underlying stream is very noisy and unpredictable [ALPA17]. As an example, consider Figure 3.5. This data shows the disk write bytes of one instance in the Amazon Web Service from 2014-03-05 04:00:00 to 2014-03-15 14:00:00. The occasional jumps in this case are actually normal, however, it leads to corresponding spikes in anomaly score as shown in Figure 3.5b. From the perspective of the domain expert, the true anomaly corresponds to a sustained increase in the frequency of large disk write bytes.

In order to handle these problems, a method is proposed by Ahmad et al. [ALPA17], which takes the anomaly score as an indirect metric, to smooth it over time and normalize it into the range $[0, 1]$. Let W be a fixed size rolling window which includes the last W anomaly scores at time t. They assume that the anomaly scores in the rolling window follows a Gaussian distribution[3], where the sample mean μ and variance σ^2 can be calculated as follows:

$$\mu_t = \frac{1}{W} \sum_{i=0}^{i=W-1} S_{\text{anomaly}}^{(t-i)} \tag{3.47}$$

$$\sigma_t^2 = \frac{1}{W-1} \sum_{i=0}^{i=W-1} \left(S_{\text{anomaly}}^{(t-i)} - \mu_t \right)^2 \tag{3.48}$$

Then they use another rolling window W' which includes the last W' anomaly scores at time t and $W' << W$. By this way, W represents the distribution of the anomaly scores at time t, while W' shows the recent short term average of anomaly score near t which is defined as

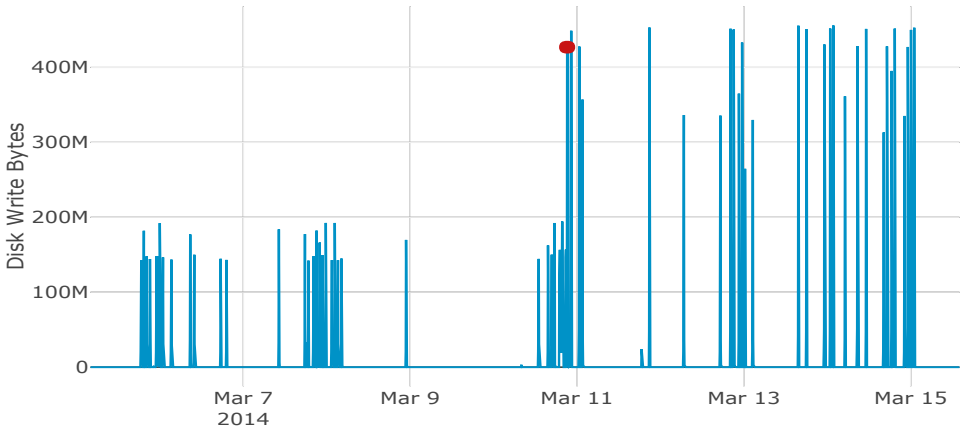
$$\tilde{\mu}_t = \frac{1}{W'} \sum_{i=0}^{i=W'-1} S_{\text{anomaly}}^{(t-i)} \tag{3.49}$$

Assume that near t the behavior of the stream is very noisy and unpredictable, thus, $\tilde{\mu}_t$ is very large. If the noisy behavior only occurs nearby the time t but the stream keeps "clean" for a long time, μ_t could be very small, thus $\tilde{\mu}_t$ is far away from μ_t and x_t is an anomaly, however, if the noisy behavior does not only occurs nearby the time t, but the stream is also inherently very noisy and unpredictable, then μ_t is similar to $\tilde{\mu}_t$, both of them are relatively large, x_t should not be declared as an anomaly. In such cases, an anomaly is not only determined by its own anomaly score, but also determined by the deviation between $\tilde{\mu}_t$ and μ_t. **If $\tilde{\mu}_t$ and the deviation between $\tilde{\mu}_t$ and μ_t are both large at the same time, then x_t is declared as an anomaly.** To this end, Ahmad et al. define a so-called *anomaly likelihood* to measure the deviation between $\tilde{\mu}_t$ and μ_t based on the *Q function*.

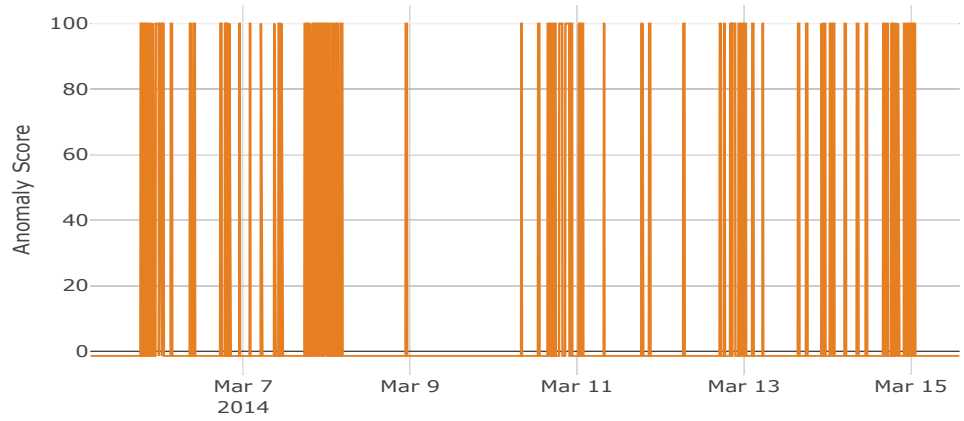
In statistics, the Q-function is the tail distribution function of the standard normal distribution (see Figure 3.6a). In other words, $Q(x)$ is the probability that a standard normal random variable takes a value larger than x. Obviously, larger x has smaller Q-value. Therefore, $\tilde{\mu}_t$ is transformed by *z-score* as follows:

$$z_t = \frac{\tilde{\mu}_t - \mu_t}{\sigma} \tag{3.50}$$

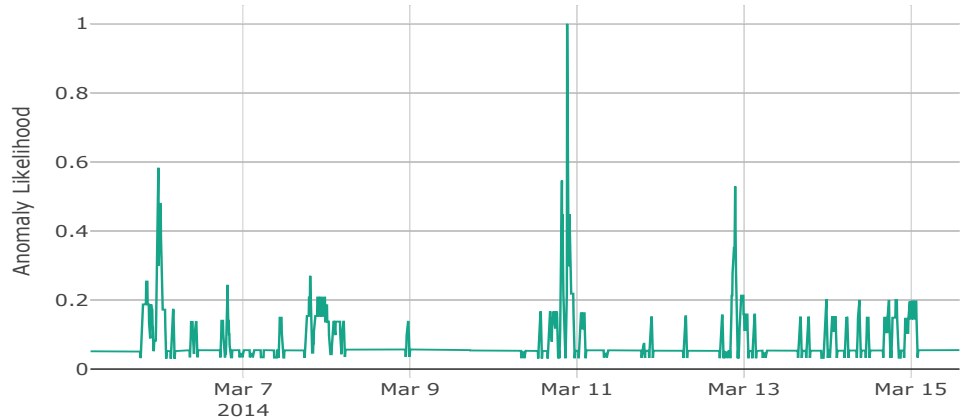
[3]As Ahmad et al. said, they have attempted to model the anomaly scores using a number of distributions and find that modeling anomaly scores as a simple normal distribution worked significantly better than others. This work just takes advantage of their ideas and does not make any other attempts.



(a)

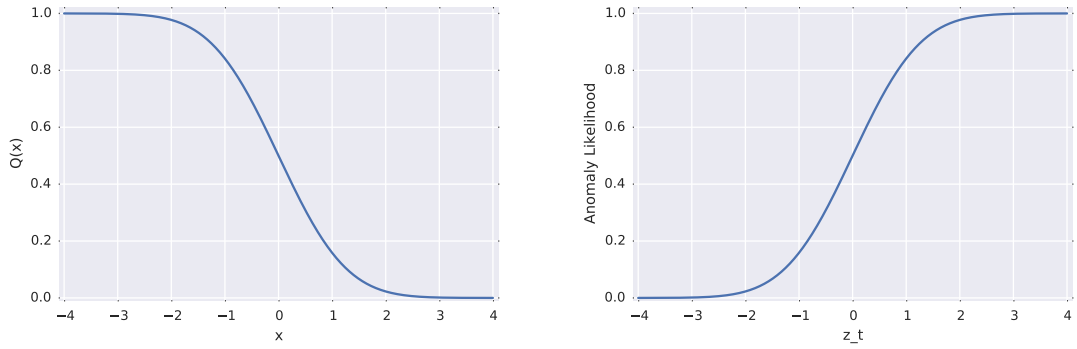


(b)



(c)

Figure 3.5: Disk Write Bytes of One Instance in the AWS.



(a) Q Function (b) Anomaly Likelihood

Figure 3.6: Q Function and Anomaly Likelihood.

Then the anomaly likelihood is calculated as

$$L^{(t)}_{\text{anomaly}} = 1 - Q(z_t) = 1 - Q(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}) \tag{3.51}$$

However, the anomaly likelihood can not help to determine a spatial anomaly[4] if the behavior of the underlying stream is very noisy. Imagine that in W' there are a lot of spikes but x_t is a spatial anomaly. In this case, despite the fact that $S^{(t)}_{\text{anomaly}}$ is very large, the small deviation between $\tilde{\mu}_t$ and μ_t could make the detector ignore the bad effect of x_t and make an incorrect decision. Since a spatial anomaly can be declared abnormal without contemplating other data points and just based on the invalid range, we use a "naïve " method to deal with it. At each time t, we calculate the valid range with respect to the maximum and minimum value of the stream so far, if x_t falls out of the valid range, its anomaly likelihood $L^{(t)}_{\text{anomaly}}$ will be directly set to 1 regardless of any other calculations.

Figure 3.5c shows an example of the anomaly likelihood on noisy disk write bytes data. The figure demonstrates that the anomaly likelihood provides clearer peaks around a noisy scenario after the disk write bytes continue to be very low for a period of time. Besides, there is a spike whose value has reached 1 indicating a spatial anomaly.

Finally, a user-defined threshold $T \in [0, 1]$ can be set to report an anomaly:

$$\text{An anomaly detected} \qquad \text{if } L^{(t)}_{\text{anomaly}} \geq T \tag{3.52}$$

3.4 Implementation

The abstract design of OPUAD (*On-line Prototypes Updating Anomaly Detection*) is shown in Figure 3.7.

For the purpose of implementation, R and Python are often good choices because both of them are open-source programming languages with a large community. R has been used primarily in academics and research, while Python provides a more

[4]Spatial anomaly has been introduced in Section 2.2.1.

general approach to data science. This work intends to use *object oriented programming techniques* which encapsulate the OPUAD algorithm as an object to bundle its parameters and methods within one unit. To this end, Python is chosen for implementation because it emphasizes productivity and code readability. Besides, coding and debugging is easier to do in Python with the "nice" syntax.

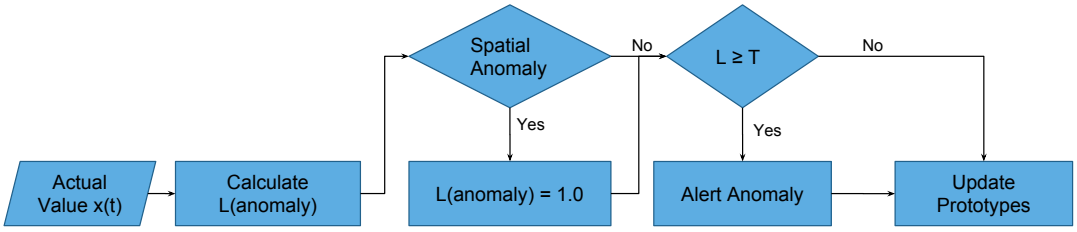


Figure 3.7: OPUAD Algorithm Abstract Design.

Thus the following dependencies are required to implement the OPUAD algorithm:

Program Language	**Library**
Python 2.7	numpy scikit-learn pandas nupic

The implementation code is shown in Appendix A.

All computations were carried out on a Dual-Core Intel Core i5 Processor running at 2.7 GHz with 3MB shared L3 cache under OS X 10.14.

3.5 Evaluation

Ahmad et al. [ALPA17] provide a benchmark called NAB[5] (*Numenta Anomaly Benchmark*) to test and measure different anomaly detection algorithms on streaming data. They are working in a company called Numenta[6] and tackling one of the most important scientific challenges of all time: reverse engineering the neocortex. Based on the principles of the cortex, they have designed a neural network called Hierarchical Temporal Memory (HTM) for sequence learning which is implemented in a machine learning platform called NUPIC[7] (*Numenta Platform for Intelligent Computing*). According to them, HTM is commonly used for prediction tasks and has been shown work well. Therefore, they have applied it to anomaly detection for streaming data. In order to test their approach, they have created NAB and provide it as a controlled open repository for researchers to evaluate and compare anomaly detection algorithms for streaming applications. So first, NAB will be briefly introduced in this section and the performance of the OPUAD algorithm will be shown later.

[5]NAB: https://github.com/numenta/NAB.
[6]Numenta: https://numenta.com.
[7]NUPIC: https://github.com/numenta/nupic.

3.5.1 Numenta Anomaly Benchmark

3.5.1.1 Bechmark Dataset

The anomaly detection algorithms for streaming applications must be operated in an unsupervised fashion, however, evaluation still needs the ground truth labels. Imagine that when a detector reports an anomaly and triggers an alert, an engineer or a domain-expert always requires to inspect the data and determine whether the reported anomaly is a true anomaly. Only when the system has totally crashed, the further inspection is no longer necessary. Therefore, ground truth labels here refer to the true anomalies, either labeled by domain-experts or labeled based on the fact. NAB saves our time on finding an appropriate data set labeled with true anomalies because it contains 58 data files[8] with ground truth labels, each with 1000–22,000 records extracted from different streams, for a total of 365,551 data points. For example, CPU utilization in Figure 3.4a and disk write bytes in Figure 3.5a are two of the 58 files, meanwhile, the red points refer to the ground truth anomaly.

3.5.1.2 NAB Scoring

According to Ahmad et al., the best anomaly detection algorithm for streaming applications must satisfy the following requirements:

- detects all anomalies present in the streaming data;
- detects anomalies as soon as possible, ideally before the anomaly becomes visible to a human;
- triggers no false alarms (no false positives).

In such a case, the traditional scoring methods, such as precision and recall which are based on the standard classification metrics: true positive (TP), false positive (FP), true negative (TN), and false negative (FN) may not be very realistic for real-time use, because they can only determine whether the algorithm behaves correct or not at an exact time point t. Imagine that there is a ground truth anomaly at time point t, but the detector reports that an anomaly occurs at time point $t+1$. If we evaluate the algorithm as usual, we will think that the algorithm gives us a false negative (FN) at time point t and a false positive (FP) at time point $t+1$. Anyway, we think that the algorithm gives us incorrect answers. However, in real applications, it is not reasonable to think that a failure occurs just at a certain exact time point. A ground truth label at time t actually means something strange is going on around t. Under such circumstance, Ahmad et al. define *anomaly windows*[9] to test algorithms [LA15]. Each window represents a range of data points that are centered around a ground truth anomaly label, see Figure 3.8. The temperature data in Figure 3.8 comes from an internal component of a large, industrial machine. There are four ground truth anomaly labels, each of them has an anomaly window highlighted by a yellow region.

[8]Some of the data files are artificially generated but most of them are real-word data from different real-applications.

[9]The window size is pre-defined by Ahmad et al. with respect to their techniques [LA15], in this work, it will not be introduced in detail.

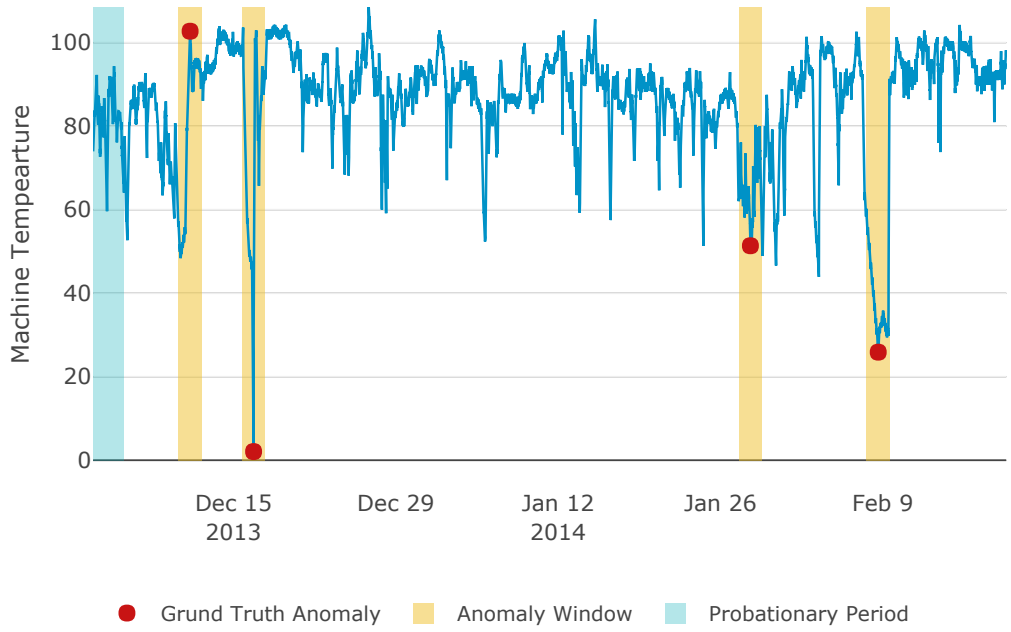


Figure 3.8: Machine Temperature.

A *scoring function* uses these windows to identify and weight true positives, false positives, and false negatives[10]. In such a case, true positive no longer means detecting anomalies correctly at an exact time point but in an anomaly window. If there are multiple detections within a window, only the earliest detection is given credit, additional positive detections within the window are ignored. The more earlier the detection in the window, the higher positive score it obtains. To this end, Ahmad et al. define a scaled sigmoidal scoring function as follows:

$$\text{Sig}(x) = 2 * (\frac{1}{1 + e^{5x}}) - 1 \tag{3.53}$$

where x is the relative position of the earliest detection within the anomaly window, the plot of this function is shown in Figure 3.9.

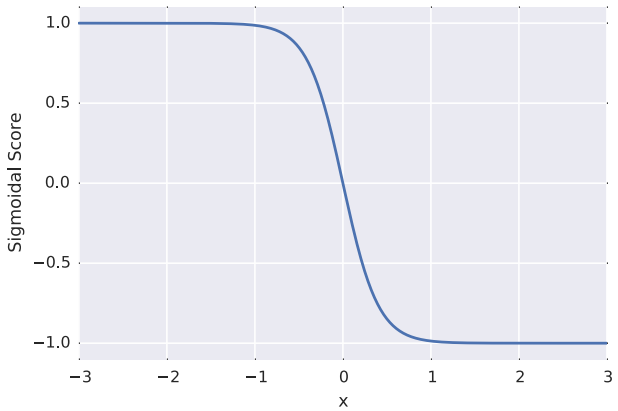


Figure 3.9: Sigmoidal Scoring Function.

[10]Since true anomalies are rare, it makes no sense to consider true negatives.

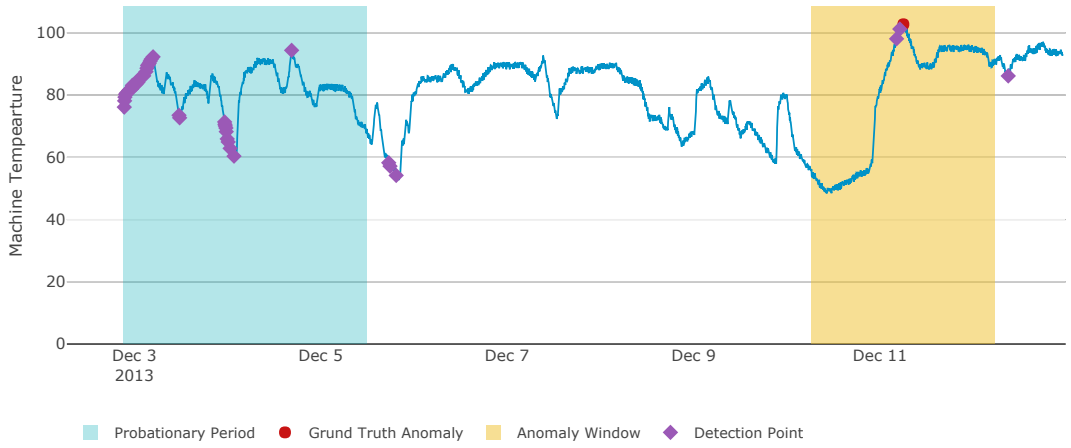


Figure 3.10: Detection Results nearby the First Anomaly Window in Figure 3.8.

As an example, we zoom in the first anomaly window in Figure 3.8 and show the detection results nearby this window, see Figure 3.10. There are two true positives within the window but we only take the first into account. The relative position of the right end of the window is set to 0 such that $\mathrm{Sig}(x = 0) = 0$. The farther the true positive (in the window) is from the right end, the higher score it is given, which enables to reward the earlier detection. The sum score of the true positives in a file is denoted as Score(TP). If there is no any detection within the window, it is a false negative and assigned a score -1, the score of all the false negatives is denoted as Score(FN). Every detection outside the window is counted as a false positive and given a scaled negative score relative to its preceding window. The sigmoidal scoring function is designed so that detections slightly after the window contribute less negative scores than detections well after the window. The shaded green region is the first 15% of the data file, representing the *probationary period*. Ahmad et al. think that an online algorithm always behaves unstable at the very beginning, therefore, during this period the detector is allowed to learn the data patterns without being scored. Since a false positive after the probationary period has no preceding anomaly window, it will be easily assigned a score -1. The sum score of the false positives is denoted as Score(FP). Therefore, the score of an online algorithm after it has been operated on a data file d is given by

$$\mathrm{Score}(d) = \mathrm{Score}(\mathrm{TP}) + \mathrm{Score}(\mathrm{FN}) + \mathrm{Score}(\mathrm{FP}) \tag{3.54}$$

Table 3.1: Weights for Measuring Anomaly Detection Performance

profile	A_{FP}	A_{FN}
Standard	0.11	1.0
Reward low FP rate	0.22	1.0
Reward low FN rate	0.11	2.0

In addition, Ahmad et al. put forward that different applications may place different emphases as to the relative importance of true positives vs. false negatives and false positives [LA15]. For example, Figure 3.8 represents an expensive industrial machine that one may find in a manufacturing plant. Even though the false positives might

always require a technician to inspect the data more closely, the false negatives will lead to machine failure in a factory, resulting in production outages and expensive costs at last. Therefore, in such a case, the cost of a false negative is far higher than the cost of a false positive. However, in some other cases, the monitored applications might be fine with the occasional missed anomaly and relatively fault tolerant. As a consequence, Ahmad et al. introduce the notion of *application profiles*. There are three different profiles in the NAB: standard profiles, reward_low_FP_rate profiles and reward_low_FN_rate profiles. Each profile A assigns different weights for false negatives A_{FN} and false positives A_{FP} (see Table 3.1) and the score for a file d with respect to a profile A is

$$\text{Score}^A(d) = \text{Score}(\text{TP}) + A_{\text{FN}}\text{Score}(\text{FN}) + A_{\text{FP}}\text{Score}(\text{FP}) \tag{3.55}$$

Then the benchmark score for a given algorithm is simply the sum of the scores over all the data files d in the corpus D:

$$\text{Score}^A(Algo) = \sum_{d \in D} \text{Score}^A(d) \tag{3.56}$$

As the score is somehow dependent on the threshold T which is set to report an anomaly, Ahmad et al. use a hill climbing algorithm[11] to find the optimal threshold $T^A_{optimal}(Algo)$ which can maximize $\text{Score}^A(Algo)$:

$$T^A_{optimal}(Algo) = \arg\max_T \text{Score}^A(Algo) \tag{3.57}$$

Then $\text{Score}^A(Algo)$ is calculated with respect to $T^A_{optimal}(Algo)$. The final reported score is a normalized NAB score computed as follows:

$$\text{Score}^A_{NAB}(Algo) = 100 \cdot \frac{\text{Score}^A(Algo) - \text{Score}^A(null)}{\text{Score}^A(perfect) - \text{Score}^A(null)} \tag{3.58}$$

where "perfect" detector means outputting all true positives, no false positives and false negatives, while "null" detector records a constant value 0.5 as anomaly likelihood for all the data points, which will have the worst performance. It follows from (3.58) that the maximum (normalized) score a detector can achieve on NAB is 100, and an algorithm with the worst performance will score 0.

The test process of an algorithm by NAB is summarized in Figure 3.11. With the intent of fostering innovation in the field of anomaly detection, NAB is designed to be an accessible and reliable framework for all to use. The NAB repository now contains source code for commonly used algorithms for online anomaly detection, as well as some algorithms submitted by the community. The algorithms evaluated now include HTM [ALPA17], Contextual Anomaly Detector (CAD OSE), Conformalized Distance-based Anomaly Detection (KNNCAD) [BI16], Multinomial Relative Entropy [WVC+11], Random Cut Forest, Twitter's Anomaly Detection, Etsy's Skyline, Bayesian Online Changepoint detection [Ada06] and EXPoSE [SER16].

[11] Hill climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making an incremental change to the solution, see https://en.wikipedia.org/wiki/Hill_climbing.

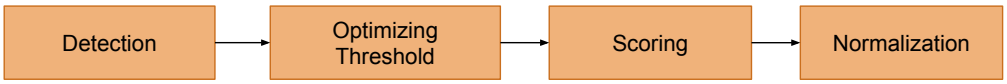


Figure 3.11: Test Process of an Algorithm by NAB.

3.5.2 Comparison of results

This work tests the OPUAD algorithm on NAB corpus using Numenta Anomaly Benchmark. Table 3.2 summarizes the NAB scores for each algorithm across all application profiles[12]. Overall we see that the HTM algorithm gets the best score

Table 3.2: NAB Scoreboard showing Results of Each Algorithm on NAB.

Detector	Standard Profile	Reward Low FP	Reward Low FN
Perfect	100	100	100
HTM	70.1	63.1	74.3
CAD OSE	69.9	67.0	73.2
OPUAD	**64.5**	**59.5**	**68.0**
KNN-CAD	58.0	43.4	64.8
Relative Entropy	54.6	47.6	58.8
Twitter ADVec	47.1	33.6	53.5
Etsy Skyline	35.7	27.1	44.5
Bayes Changepoint	17.7	3.2	32.2
EXPoSE	16.4	3.2	26.9
Null	0	0	0

and CAD OSE is the winner in the competition which was held in collaboration with IEEE WCCI[13] during the summer of 2016 by Numenta. The OPUAD algorithm described in this chapter just follows these two algorithms and takes the third place.

3.6 Conclusions

For a company which applies SAP systems or services e.g. SAP HANA database, it is essential to have a stable and healthy SAP landscape, because the company can only be as healthy as its SAP landscape. If its SAP landscape degrades, so does its productivity. For example, if the production environment goes down for an hour, it may cost the company tens of thousands. Therefore, anomaly detection is one of the most significant functions in the monitoring service, which can help to reduce the downtime of the system. To this end, this chapter proposes the OPUAD algorithm for streaming data based on the kernel density estimator and maximum entropy principle. Unlike batch processing, OPUAD detects anomalies

[12]The Scoreboard without the OPUAD algorithm is shown in https://github.com/numenta/NAB.

[13]http://www.wcci2016.org.

in an unsupervised, automated fashion without supervision, which makes it self-adaptive and computationally efficient. Its performance has been tested by Numenta Anomaly Benchmark (NAB) and the comparative result shows that OPUAD is better than the most of other algorithms included in NAB.

4. Granger Causality Analysis for Time Series Data

Temporal-casual effect between time series data is commonly quantified by Granger causality. However, its basic form is not applicable to large scale data with non-linear interactions. Therefore, this chapter introduces several variants of the Granger causality and performs simulations to discuss their scalability. Since Granger causality is a statistic tool to measure the association between time series data, the analysis in this chapter belongs to exploratory causal analysis.

4.1 Definition of Granger Causality

Granger causality, a statistical method developed by Granger(1969), is one of the causality theories widely used in recent decades to quantify the temporal-causal effect among time series due to its simplicity, robustness and extendability [BDL$^+$04, Pan04]. It is based on the common conception that the cause usually appears prior to its effect. Formally, a variable Y (the driving variable) **Granger causes** X (the response variable) if its past value can help to predict the future value of X beyond what could have been done with the past value of X only [ALA07]. Thus Granger causality uses the prediction to infer the causality, the prediction accuracy of a joint model that includes "cause" and "effect" is better than a model of "effect" alone. Generally, the definition of Granger causality is rooted in the conditional distribution [BL13] as follows:

Definition 4.1. *Given two stationary time series $X = \{x_t\}_{t\in\mathbb{Z}}$ and $Y = \{y_t\}_{t\in\mathbb{Z}}$ and considering two information sets: (i) $\mathcal{I}^*(t)$, the set of all information in the universe up to time t, and (ii) $\mathcal{I}^*_{-Y}(t)$, the set of all information in the universe excluding Y up to time t. On the basis of Granger causality, the conditional distribution of future values of X given $\mathcal{I}^*(t)$ and $\mathcal{I}^*_{-Y}(t)$ should be different. Therefore Y is defined to* ***Granger cause*** *X if*

$$P[x_{t+1} \in A|\mathcal{I}^*(t)] \neq P[x_{t+1} \in A|\mathcal{I}^*_{-Y}(t)] \tag{4.1}$$

for some measurable set $A \subseteq \mathbb{R}$ and all $t \in \mathbb{Z}$.

4.2 Linear Granger Causality

Due to the difficult modeling of the conditional distribution, linear regressions and significance tests are often employed to discover the Granger-causal relation between time series because of its simplicity, robustness with strong empirical performance in practical applications. Consequently, vector autoregression (VAR) models have evolved to be one of the dominant approaches of Granger causality [BL13].

4.2.1 Bivariate-GC Test

For simplicity of exploration, we consider Granger causality in a bivariate system at first where $\mathcal{I}^*(t)$ only consists of the information of X and Y. Let X_t and Y_t be the observations of X and Y at time t, the bivariate VAR process for the two stationary time series X and Y of length N can be described by the following equation:

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \sum_{i=1}^{L} \begin{bmatrix} \alpha_i & \beta_i \\ \gamma_i & \delta_i \end{bmatrix} \begin{bmatrix} x_{t-i} \\ y_{t-i} \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \end{bmatrix}, \tag{4.2}$$

where L is the maximal time lag, ϵ_1 and ϵ_2 refer to two white noise error terms. In that sense, time series Y is said not to **Granger causes** X if $\beta_i = 0$ for any $i = 1, ..., L$. In other words, the past values of Y do not provide any additional information on the performance of X. Similarly, X dose not **Granger causes** Y if $\gamma_i = 0$ for any $i = 1, ..., L$.

The Granger-causal relation between X and Y can be determined by statistical significant tests with the following null hypotheses: (1) $H_0^1 : \beta_1 = \cdots = \beta_L = 0$, and (2) $H_0^2 : \gamma_1 = \cdots = \gamma_L = 0$. After testing, we have four possible testing results: (1) if none of H_0^1 and H_0^2 is rejected, there is no linear Granger-causal relationship between X and Y; (2) if only H_0^1 is rejected, there exists linear Granger-causality running unidirectionally from Y to X; (3) if only H_0^2 is rejected, there exists linear Granger-causality running unidirectionally from X to Y; (4) if H_0^1 and H_0^2 are both rejected, there exists reciprocally Granger-causal relationship between X and Y.

It is evident that Granger causality is a *directional* measurement, while the Pearson correlation coefficient, the most popular tool to measure the relation between two variables, is a *symmetrical* measurement. Therefore, it is more suitable to use Granger-causal relation to support the further root cause analysis for anomalies in a critical metric and know some effects of the anomalies. Since Granger causality is a

directional measurement, how to measure Granger-causal relation will be illustrated based on one direction. In such a case, the following two models are compared to test if Y **Granger causes** X:

$$x_t = c_1 + \sum_{i=1}^{L} \alpha_i x_{t-i} + \sum_{i=1}^{L} \beta_i y_{t-i} + \epsilon_1 \tag{4.3}$$

$$x_t = c_3 + \sum_{i=1}^{L} \eta_i x_{t-i} + \epsilon_3 \tag{4.4}$$

The Eq. (4.3) is referred to as the unrestricted model (U-model) and the Eq. (4.4) as the restricted model (R-model) [BL13], because the former comprises information of X and Y but the latter only information of X. After fitting the U-model and R-model with ordinary least squares (OLS) regression, we could compare the predictive performance of two models by F-test in terms of the residual sum of squares RSS which is given by

$$RSS = \sum_i (y_i - \hat{y}_i)^2 \tag{4.5}$$

where y_i is the i-th observed value and $\hat{y}_i$ is the i-th predicted value.

The statistical significance of the difference between RSS_R and RSS_U is commonly assessed by the Fisher statistic:

$$F = \frac{(RSS_R - RSS_U)/L}{RSS_U^2/(N - 3L - 1)} \tag{4.6}$$

where RSS_R and RSS_U are the residual sum of squares from R-model and U-model respectively, N is the length of given time series data, L is the number of time lags which are included in the estimation.

The F statistic approximately follows an F distribution with degrees of freedom L and $(N - 3L - 1)$. If the F statistic is significant(i.e., greater than the critical value at $p < 0.05$ in the standard $F_{L,N-3L-1}$ distribution), then the U-model yields a better explanation of X than does the R-model, and Y is said to **Granger cause** X.

Causal Influence

Once we have concluded that a "causal relation" in the sense of Granger is present, it is usually required to assess the strength of this relationship. In [GSK+08], the causal influence is quantified with the help of the variance of the prediction errors σ^2 because if the variance of prediction error for the R-model is reduced by the inclusion of past histories of the driving variables in the U-model, then a causal relation exists. Hence, the causal influence from Y to X is quantified by

$$GC_{Y->X} = \ln \frac{\sigma^2(\epsilon_3)}{\sigma^2(\epsilon_1)} \tag{4.7}$$

Since σ^2 is typically unknown, it can be estimated by

$$\hat{\sigma}^2 = \frac{\sum_i (y_i - \hat{y}_i)^2}{n - k - 1} \tag{4.8}$$

where n is the number of observations and k is the number of the independent variables in the regression model.

Stationarity

It is important to note that Granger causality approach assumes week stationarity of underlying data.

Definition 4.2. *(Stationarity or week stationarity) The time series $X = \{x_t\}_{t\in\mathbb{Z}}$ is said to be stationary if*

i) $Var(x_t) < \infty$ *for all* $t \in \mathbb{Z}$,

ii) $E(x_t) = \mu$ *for all* $t \in \mathbb{Z}$,

iii) $\gamma_X(s,t) = \gamma_X(s+h, t+h)$ *for all* $s, h, t \in \mathbb{Z}$.

Loosely speaking, a stochastic process is stationary if its statistical properties, i.e. means and covariances, are constant over time. In other words, a stationary time series $\{x_t\}_{t\in\mathbb{Z}}$ must have three features: finite variation, constant first moment, and the second moment $\gamma_X(s,t)$ depending only on $(t-s)$ but not on s or t.

On the basis of the last point in the definition, the auto-covariance function(ACVF) of a stationary process can be rewritten as

$$\gamma_X(h) = Cov(x_t, x_{t+h}) \text{ for } t, h \in \mathbb{Z} \tag{4.9}$$

Also, when $\{x_t\}$ is stationary, we must have

$$\gamma_X(h) = \gamma_X(-h) \tag{4.10}$$

When $h = 0$, $\gamma_X(0) = Cov(x_t, x_t)$ is the variance of $\{x_t\}_{t\in\mathbb{Z}}$, so the autocorrelation(ACF) function of $\{x_t\}_{t\in\mathbb{Z}}$ is defined to be

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} \tag{4.11}$$

The **Augmented Dickey-Fuller(ADF) test** is used to test stationarity with the null hypothesis H_0 : the time series is non-stationary and the alternative hypothesis H_1 : the series is stationary. To implement the test, the following model is considered

$$y_t = c + \delta t + \phi y_{t-1} + \sum_{i=1}^{L} \beta_i \Delta y_{t-i} + \epsilon_t \tag{4.12}$$

where Δ is the differencing operator, c is the constant, δ is the deterministic trend coefficient, L is the number of lagged difference terms used in the model and ϵ_t is a white noise error. The number of lags can be determined using the Schwarz Bayesian information criterion [Sch78] or the Akaike information criterion [Aka74].

To infer about H_0, the t-statistic is used on the ϕ coefficient. The test statistic is given by the expression

$$t_{DF} = \frac{\hat{\phi} - 1}{SE(\hat{\phi})} \tag{4.13}$$

where $\hat{\phi}$ is the estimated ϕ from the fitting model and SE is the standard error. The H_0 is rejected if the test statistic is smaller than the corresponding critical value. If H_0 is not rejected, the series needs transformation to achieve stationarity. Differencing can help stabilize the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality [AA13].

However, in this work, the stationarity will always hold in the analysis because causal inference is only made with relatively short time series of each available metric in a system around anomalies; see Section 2.3. A short time series is often considered stationary. How to make causal inference with non-stationary time series is beyond the scope of this work.

Time Lag Selection

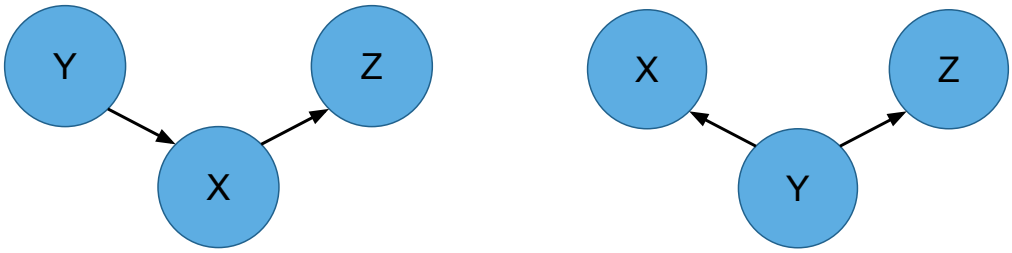
Selection of an appropriate lag is critical to inference in VAR. Lütkepohl [Lü05] indicated that using too few lags can result in autocorrelated errors while using too many lags leads to over-fitting, causing an increase in mean-square-forecast errors of the VAR model.

The most common approach to determine the lag length is to fit VAR models with $L = 0, 1, ..., L_{max}$ and choose the value of L which minimizes some model selection criteria. The three most commonly used criteria are the Akaike Information Criterion(AIC), Schwarz Bayesian Information Criterion(BIC) and the Hannan-Quinn criterion(HQ) [Lü05, Vri12]. The lag associated with the minimum value of a criterion is selected [Lü05].

4.2.2 Conditional-GC Analysis

As in many applications, such as climate science, social media, IT system, ecommerce, medical, energy etc, the data available for analysis often involve a collection of multiple univariate time series referred to as multivariate time series. Identifying temporal dependencies between multivariate time series for better understanding of causal structure among the relevant variables is thus a topic of significant interest [ALA07, LALR09].

A nature way of applying the notion of Granger causality with multivariate time series is to simply apply the bivariate-GC test to each pair of time series sequentially to determine the presence of causal relation between them and the corresponding orientation [ALA07]. However, the bivariate-GC test has some inherent limitations and might lead to incorrect conclusions, as broadly discussed in [DCB06]. For example, let us assume that there are three time series X, Y and Z. The problem arises, for example, if Y 'drives' X, while X 'drives' Z without there being a 'drive' from Y to Z(see Figure 4.1a). In this case, the Granger causality from Y to Z maybe spuriously identified as being significant. As another example, the problem also occurs if Y 'drives' both X and Z with different delays, without there being a 'drive' from X to Z(see Figure 4.1b). In this case, the Granger causality from X to Z may be spuriously as being significant.



(a) Sequential driving pattern. (b) Differentially delayed driving.

Figure 4.1: Spurious identification of significant GC by bivariate-GC test.

Conditional Granger Causality analysis [Gew84, CBD06, SE07] can mitigate the problem of spurious Granger Causality significance. This method is capable to figure out whether the interaction between two time series is direct or is mediated by another time series and whether the causal influence is simply due to different time delays in their respective driving input. The analysis now requires the estimation of multivariate vector autoregressions(MVAR), for example, if we consider Y and X as the "cause" and "effect" variable, respectively, and conditioning on variable Z, the Eq. (4.3) and Eq. (4.4) are extended by the following:

$$x_t = c_5 + \sum_{i=1}^{L} \alpha_i x_{t-i} + \sum_{i=1}^{L} \beta_i y_{t-i} + \sum_{i=1}^{L} \zeta_i z_{t-i} + \epsilon_5 \tag{4.14}$$

$$x_t = c_6 + \sum_{i=1}^{L} \eta_i x_{t-i} + \sum_{i=1}^{L} \rho_i z_{t-i} + \epsilon_6 \tag{4.15}$$

The causal influence from Y to X conditioning on Z is now defined as

$$GC_{Y->X|Z} = \ln \frac{\sigma^2(\epsilon_6)}{\sigma^2(\epsilon_5)}. \tag{4.16}$$

When the causal influence from Y to X is entirely mediated by Z, the coefficient β_i in Eq. (4.14) is uniformly zero and $\sigma^2(\epsilon_6) = \sigma^2(\epsilon_5)$. Thus we have $GC_{Y->X|Z} = 0$, meaning that no further improvement in the prediction of X can be expected by including past measurements of Y conditioning on Z. In contrary, when there is still a direct influence from Y to X, the inclusion of past measurements of Y in addition to that of X and Z leads to better prediction of X, resulting in $\sigma^2(\epsilon_5) < \sigma^2(\epsilon_6)$, and $GC_{Y->X|Z} > 0$. The statistical significance of $GC_{Y->X|Z}$ can also be assessed by the F-test. An extension of this definition for more than three time series is straightforward.

Now let $S = \{Z_1, Z_2..., Z_D\}$ be a D-dimensional stationary time series of length N and Z_i is considered as the target ("effect"). Whether or not another time series Z_j **Granger causes** Z_i is tested using the following two models:

$$z_{i,t} = a_0 + \sum_{d=1}^{D}\sum_{l=1}^{L} a_{d,l} z_{d,t-l} + e_{i,t} \quad \text{(U-model)} \tag{4.17}$$

$$z_{i,t} = b_0 + \sum_{d=1,d\neq j}^{D}\sum_{l=1}^{L} b_{d,l} z_{d,t-l} + u_{i,t} \quad \text{(R-model)} \tag{4.18}$$

The Eq. (4.18) excludes the historical information of Z_j and the null hypothesis H_0 that "Z_j does not **Granger causes** for Z_i" is rejected if the coefficients $a_{j,l},\ l = 1, ..., L$, are jointly different from 0. The formula of the F-test now has a bit change compared with (4.6):

$$F = \frac{(RSS_R - RSS_U)/L}{RSS_U/(N - L - KL - 1)} \tag{4.19}$$

where RSS_U and RSS_R are the residual sum of squares from U-model (4.17) and R-model (4.18) respectively, N is the length of the given time series data, L is the model order, KL means the number of explanatory variables in the U-model (4.17).

The causal influence in this case is also called conditional Granger causality index (CGCI) [SK16]. Then CGCI from Z_j to Z_i is defined as

$$CGCI_{Z_j->Z_i} = \ln \frac{\sigma^2(u_{i,t})}{\sigma^2(e_{i,t})} \tag{4.20}$$

4.2.3 Lasso-GC Analysis

It is well known that over-fitting will occur more prominently when the number of considered time series increases. Therefore, variable selection to reduce the number of explanatory variables in the regression model is necessary. The Lasso(Least Absolute Shrinkage and Selection Operator) algorithm [Tib96], first formulated by Robert Tibshirani in 1996, has proven popular in variable selection. It is an incremental algorithm that includes the L1-regularization term in the fitting process of over-parameterized linear models to avoid over-fitting. In linear regression, the Lasso generally minimize the sum of squared errors with a upper bound on the sum of the absolute values of the model coefficients, namely

$$\min\left(||\boldsymbol{y} - \boldsymbol{X}\beta||_2^2\right) \ \text{ subject to } \ ||\beta||_1 \leq t \tag{4.21}$$

where $\boldsymbol{y}$ is the dependent variable, $\boldsymbol{X}$ is the independent variables and t is the upper bound for the sum of the coefficients. This optimization problem is equivalent to the parameter estimation that follows

$$\hat{\beta}(\lambda) = \arg\min_{\beta}\Big(\underbrace{||\boldsymbol{y} - \boldsymbol{X}\beta||_2^2}_{\text{Loss}} + \lambda \underbrace{||\beta||_1}_{\text{Penalty}} \Big) \tag{4.22}$$

where $\lambda \geq 0$ is the parameter that controls the strength of the penalty, the larger the value of λ, the greater the amount of shrinkage.

The relation between λ and the upper bound t is a reverse relationship. As t becomes infinity, the problem turns into an ordinary least squares and λ becomes 0. Contrarily, when t becomes 0, all coefficients shrink to 0 and λ goes to infinity. Hence, choosing a good value of the tuning parameter λ is crucial. Because each tuning parameter value corresponds to a fitted model, we also refer to this task as model selection and use the generalized cross validation score to achieve the variable selection [ALA07].

If we minimize the optimization problem, some coefficients are shrunk to zero, i.e. $\hat{\beta}_j(\lambda) = 0$ for some features j depending on the parameter λ. As a result, the features with coefficient equal to zero are excluded from the model which makes Lasso a powerful method for feature selection. So unlike the conditional GC-analysis, if we apply the Lasso GC analysis, there is only one model to be considered:

$$z_{i,t} = a_0 + \sum_{d=1}^{D}\sum_{l=1}^{L} a_{d,l} z_{d,t-l} + e_{i,t} \tag{4.23}$$

and the regression task could be thus achieved by solving the following optimization problem:

$$\min_{\{a_{d,l}\}} \sum_{t=L+1}^{N} \left(z_{i,t} - \sum_{d=1}^{D}\sum_{l=1}^{L} a_{d,l} z_{d,t-l} \right)^2 + \lambda \|a_{d,l}\|_1 \tag{4.24}$$

Since only one model is considered in this case, F-test is not applicable to test the significance of Granger causality any more. But it will not hold back discovering temporal-causal relationship. Some analysts [LTTT14] propose the so-called *covariance test statistic* to test the significance of the predictor variable that enters the current lasso model, however, as mentioned in Section 2.3.2, even though statistical tests are useful for causal inference, the results of such tests should not be interpreted beyond *exploratory* causal analysis, which means, the use of a hypothesis test does not imply a given causal inference is confirmatory rather than exploratory [McC16]. Note that the objective of causal inference in this work is not to confirm some existing causal theory or structure, it just explores potential causal relationships between time series data, any finding still requires further analysis.

So under such circumstance, it makes sense to say that

$$Z_j \textbf{ Granger causes } Z_i, \quad if\ \exists\ l = 1, ..., L, \quad \hat{a}_{j,l} \neq 0\ ,$$

which means, for a given lag L, at least one individual lagged variable of Z_j is included in the fitted model and provides additional information for predicting Z_i.

Sometimes we can fix a threshold parameter $T > 0$ and say that

$$Z_j \textbf{ Granger causes } Z_i, \quad if\ \sum_{l=1}^{L} |\hat{a}_{j,l}| > T$$

and $\sum_{l=1}^{L} |\hat{a}_{j,l}|$ can also be used to quantify the casual influence from Z_j to Z_i.

4.2.3.1 Adaptive Lasso-GC Analysis

Some studies state that there exist certain scenarios where the Lasso is inconsistent for variable selection, e.g. includes noise variables and shows biased estimates for large coefficients, leading to suboptimal prediction rates. To address this issue, Zou(2006) [Zou06] proposed the Adaptive Lasso, as a regularization method to avoid overfitting penalizing large coefficients. In linear regression, the adaptive Lasso thus seeks to minimize:

$$\hat{\beta}_{adapt}(\lambda) = \arg\min_{\beta}\Big(||\boldsymbol{y} - \boldsymbol{X}\beta||_2^2 + \lambda \sum_j \hat{w}_j|\beta_j|\Big) \tag{4.25}$$

where λ is the tuning parameter, β_j are the estimated coefficients and $\hat{w}_j$ are the adaptive weights. With $\hat{w}_j$ we could perform a different regularization for each coefficient, i.e., assigning the penalty differently for each coefficient. The adaptive weights are defined as

$$\hat{w}_j = \frac{1}{|\hat{\beta}_j^{init}|} \tag{4.26}$$

where $\hat{\beta}_j^{init}$ is an initial estimate of the coefficients, usually obtained through OLS and Ridge Regression.

Notice that if $\hat{\beta}_j^{init} = 0$ then $\forall\lambda > 0$, $\hat{\beta}_j(\lambda) = 0$. In addition, those coefficients with lower initial estimates are given a relative greater penalty in Adaptive Lasso. Furthermore, if the penalization parameter λ is chosen appropriately, Adaptive Lasso is consistent for variable selection and enjoys the so-called "Oracle Property" proposed by Fan and Li(2001) [FL01], which means broadly that the procedure performs as well as if the true subset of relevant variables were known.

The optimization problem (4.24) in the Adaptive Lasso turns into

$$\min_{\{a_{d,l}\}} \sum_{t=L+1}^{N} \left(z_{i,t} - \sum_{d=1}^{D}\sum_{l=1}^{L} a_{d,l} z_{d,t-l} \right)^2 + \lambda \sum_{d=1}^{D}\sum_{l=1}^{L} \frac{|a_{d,l}|}{|\hat{a}_{d,l}^{init}|} \tag{4.27}$$

4.2.3.2 Group Lasso-GC Analysis

Either the general Lasso-GC analysis or the adaptive Lasso-GC analysis only takes the effect of the individual lagged variable into account. But when modeling the temporal-causal dependencies into a graph we do not consider whether an individual lagged variable is to be included in regression, but whether the lagged variables for a given time series as a **group** are to be included. This is the motivation for us to turn to the group Lasso, which performs variable selection with respect to model fitting criteria that penalize intra- and inter-group variable inclusion differently. In this case, variables belonging to the same group should be either selected or eliminated as a whole [LALR09]. The group Lasso proposed by Yuan and Lin [YL06] overcomes these problems by introducing a suitable extension of the lasso penalty. The estimator is defined as

$$\hat{\beta}_{group}(\lambda) = \arg\min_{\beta}\Big(||\boldsymbol{y} - \boldsymbol{X}\beta||_2^2 + \lambda \sum_{g=1}^{G} ||\beta_{\mathcal{I}_g}||_2\Big) \tag{4.28}$$

where $\mathcal{I}_g$ is the index set belonging to the gth group of variables, $g = 1, ..., G$ and $\beta_{\mathcal{I}_g} = \{\beta_j; j \in \mathcal{I}_g\}$. This penalty can be viewed as an intermediate between the l_1- and l_2-type penalty. It has the attractive property that it does variable selection at the group level and is invariant under (group-wise) orthogonal transformations like ridge regression [YL06].

If we apply group Lasso to the granger causality analysis, the optimization problem (4.24) can be rewritten as

$$\min_{\{a_{d,l}\}} \sum_{t=L+1}^{N} \left(z_{i,t} - \sum_{d=1}^{D} \sum_{l=1}^{L} a_{d,l} z_{d,t-l} \right)^2 + \lambda \sum_{d=1}^{D} ||\{a_{d,l}\}||_2 \tag{4.29}$$

where $\{a_{d,l}\} = (a_{d,1}, ..., a_{d,L})$.

In this case, we can see that each time series is viewed as a separate group and in every group there are L individuals (the lagged variables). So the D groups are of equal size. The L individual lagged variables of a time series as a whole are included or excluded in the regression. Therefore, it makes sense to say that

$$Z_j \textbf{ Granger causes } Z_i, \quad if \ \forall \ l = 1, ..., L, \quad \hat{a}_{j,l} \neq 0 \ ,$$

which means, all of the individual lagged variables of Z_j are included in the fitted model, or we can also fix a threshold parameter $T > 0$ and say that

$$Z_j \textbf{ Granger causes } Z_i, \quad if \ \sum_{l=1}^{L} |\hat{a}_{j,l}| > T.$$

4.3 Non-linear Granger Causality

It is common that many real world systems exhibit nonlinear dependence between time series so that using linear models may lead to inconsistent estimation of Granger causal interactions. Therefore, using tools to detect nonlinear Granger causality is always a challenge. Several approaches have been proposed for identification of Granger Causality in non-linear systems; among the notable ones, kernelized regression [MPS08], non-parametric techniques such as [HJ94, Pan04], non-Gaussian structural VAR [HSKP08] and generalized linear autoregressive models [YHPW09]. However these methods either perform poorly in high dimensions or do not scale to large datasets [BL13]. To tackle these challenges, an interpretable and sparse neural network model for nonlinear Granger Causality is proposed [TCF+18].

4.3.1 Neural Lasso-GC Analysis

A *nonlinear* autoregressive model allows Z_i to evolve according to more general nonlinear dynamics [TCF+18]

$$z_{i,t} = g_i(z_1^{(t-L):(t-1)}, ..., z_D^{(t-L):(t-1)}) + e_{i,t} \tag{4.30}$$

where $z_d^{(t-L):(t-1)} = (z_{d,t-L}, ..., z_{d,t-1}), \quad d = 1, ..., D$ denotes the past values of time series Z_d and g_i is a function that specifies how the past L lags influence time series

Z_i. In this context, time series Z_j does not Granger causes Z_i means that the function g_i does not depend on $z_j^{(t-L):(t-1)}$, the past lags of series Z_j.

The nonlinear dynamics can be modeled with a multilayer perceptron (MLP) neural network. Assume that for each time series Z_i, g_i takes the form of an MLP with M hidden layers and let the vector H^m denotes the values of the m-th hidden layer. The input layer has $D * L$ units and the first hidden layer has Q units. The value of the q-th unit in the first hidden layer can be written as

$$h^{1,q} = \sigma\left(\sum_{d=1}^{D}\sum_{l=1}^{L} w_{d,l}^{1,q} z_{d,t-l} + b^{1,q}\right) \tag{4.31}$$

where σ is an activation function and $b^{1,q}$ is the bias.

Therefore, the first layer weights W^1 can be written as follows

$$W^1 = \begin{bmatrix} w_{1,1}^{1,1} & \dots & w_{1,L}^{1,1} & \dots & w_{D,1}^{1,1} & \dots & w_{D,L}^{1,1} \\ w_{1,1}^{1,2} & \dots & w_{1,L}^{1,2} & \dots & w_{D,1}^{1,2} & \dots & w_{D,L}^{1,2} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,1}^{1,Q} & \dots & w_{1,L}^{1,Q} & \dots & w_{D,1}^{1,Q} & \dots & w_{D,L}^{1,Q} \end{bmatrix}$$

The outgoing weights shown in blue in Figure 4.2a represent one row of W^1.

Reform W^1 as each column d includes the weights for all the lag individuals of time series Z_d:

$$W^1_{\text{reform}} = \begin{bmatrix} w_{1,1}^{1,1} & w_{2,1}^{1,1} & \dots & w_{D,1}^{1,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,1}^{1,Q} & w_{2,1}^{1,Q} & \dots & w_{D,1}^{1,Q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,L}^{1,1} & w_{2,L}^{1,1} & \dots & w_{D,L}^{1,1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,L}^{1,Q} & w_{2,L}^{1,Q} & \dots & w_{D,L}^{1,Q} \end{bmatrix}$$

The outgoing weights shown in blue in Figure 4.2b represent one column of W^1_{reform}.

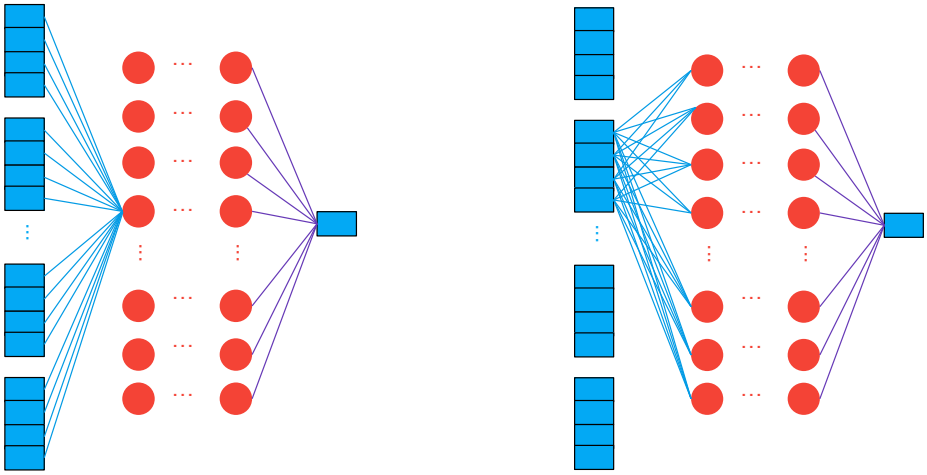
Inspired by the group Lasso-GC methods, we can apply a group lasso penalty to the columns of the matrix W^1_{reform} for g_i as in Eq. (4.30),

$$\min_{w_{d,l}^{1,q}} \sum_{t=L+1}^{N} \left(z_{i,t} - g_i(z_1^{(t-L):(t-1)}, ..., z_D^{(t-L):(t-1)})\right)^2 + \lambda \sum_{d=1}^{D} ||\{w_{d,l}^{1,q}\}||_2 \tag{4.32}$$

where $\{w_{d,l}^{1,q}\} = (w_{d,1}^{1,1}, ..., w_{d,1}^{1,Q}, ..., w_{d,L}^{1,1}, ..., w_{d,L}^{1,Q})$.

In this case, we fix a threshold parameter $T > 0$ to determine if time series Z_j Granger causes Z_i,

$$Z_j \textbf{ Granger causes } Z_i, \quad if \sum_{l=1}^{L} \sum_{q=1}^{Q} ||w_{j,l}^{1,q}||_2 > T.$$



(a) One row of W^1. (b) One column of W^1_{reform}.

Figure 4.2: Schematic for modeling Granger causality using MLP.

As discussed in Lasso algorithm, for large enough λ, the solutions to the objective function (4.32) will lead to many zero columns in W^1_{reform}. To this end, we apply *proximal gradient descent with line search* to update $w_{d,l}^{1,q}$ [1].

4.3.1.1 Gradient Descent

First, let consider unconstrained, smooth convex optimization problems in the form of

$$\min f(\boldsymbol{x}) \tag{4.33}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex, twice differentiable, with $\text{dom}(f) = \mathbb{R}$(no constraints). Assume that the problem is solvable, we denote the optimal value, $f^* = \min_{\boldsymbol{x}} f(\boldsymbol{x})$ and optimal solution as $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$.

Due to the fact that it is always not easy to solve a system of equation $\nabla f(\boldsymbol{x}^*) = 0$, *Gradient Descent* is more preferred to compute a minimizing sequence of point $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ...$ such that $f(\boldsymbol{x}^{(k)}) \to f(\boldsymbol{x}^*)$ as $k \to \infty$. At each iteration k, we want to move from our current point $\boldsymbol{x}^{(k-1)}$ to point $\boldsymbol{x}^{(k)}$ such that $f(\boldsymbol{x}^{(k)})$ is optimal. Since f is twice differentiable, we apply the quadratic approximation on $f(\boldsymbol{x}^{(k)})$ to have

$$\begin{aligned} f(\boldsymbol{x}^{(k)}) \approx & f(\boldsymbol{x}^{(k-1)}) + \nabla f(\boldsymbol{x}^{(k-1)})^T(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}) \\ & + \frac{1}{2}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)})^T \nabla^2 f(\theta(\boldsymbol{x}^{(k-1)} - \boldsymbol{x}^{(k)}) + \boldsymbol{x}^{(k)})(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}) \end{aligned} \tag{4.34}$$

[1] Note that only the first layer weights will be updated when we train the neural network in this case.

where $\theta \in [0, 1]$. By replacing $\nabla^2 f(\theta(\boldsymbol{x}^{(k-1)} - \boldsymbol{x}^{(k)}) + \boldsymbol{x}^{(k)})$ by $\frac{1}{t_k} I$, we can represent $f(\boldsymbol{x}^{(k)})$ as follows:

$$f(\boldsymbol{x}^{(k)}) \approx \underbrace{f(\boldsymbol{x}^{(k-1)}) + \nabla f(\boldsymbol{x}^{(k-1)})^T(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)})}_{\text{Linear Approximation}} + \underbrace{\frac{1}{2t_k}||\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}||_2^2}_{\text{Proximal Term}} \tag{4.35}$$

the first additive term is called linear approximation, and the second one is proximity term. Basically, the proximity term tell us that we should not go to far from $\boldsymbol{x}^{(k-1)}$, otherwise results in large $f(\boldsymbol{x}^{(k)})$. To find optimal value of $\boldsymbol{x}^{(k)}$, we take the derivative of (4.35) with respect to $\boldsymbol{x}^{(k)}$ and set to zero

$$\nabla f(\boldsymbol{x}^{(k-1)}) + \frac{1}{t_k}(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}) = 0$$

thus, we can get gradient update

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} - t_k \nabla f(\boldsymbol{x}^{(k-1)}) \tag{4.36}$$

where $k = 1, 2, ...$ is iteration number, t_k is step size at iteration k, initial $\boldsymbol{x}_0 \in \mathbb{R}^n$ is usually given. The Eq. (4.36) can also be rewritten as

$$\boldsymbol{x}^{(k)} = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}^{(k-1)}) + \nabla f(\boldsymbol{x}^{(k-1)})^T(\boldsymbol{x} - \boldsymbol{x}^{(k-1)}) + \frac{1}{2t_k}||\boldsymbol{x} - \boldsymbol{x}^{(k-1)}||_2^2 \tag{4.37}$$

After some simple algebraic manipulation and cancellation of constant terms, the Eq. (4.37) can be rewritten as

$$\boldsymbol{x}^{(k)} = \arg\min_{\boldsymbol{x}} \frac{1}{2t_k}||\boldsymbol{x} - (\boldsymbol{x}^{(k-1)} - t_k \nabla f(\boldsymbol{x}^{(k-1)}))||_2^2 \tag{4.38}$$

4.3.1.2 Proximal Gradient Descent

Now consider a function f that can be decomposed into two functions as follows:

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x}) \tag{4.39}$$

where, g is convex, differentiable and $\operatorname{dom} g = \mathbb{R}^n$, while h is convex and not necessarily differentiable, but is simple. Let us use (4.38) for the differentiable function g. Then we can obtain

$$\boldsymbol{x}^{(k)} = \arg\min_{\boldsymbol{x}} \frac{1}{2t_k}||\boldsymbol{x} - (\boldsymbol{x}^{(k-1)} - t_k \nabla g(\boldsymbol{x}^{(k-1)}))||_2^2 + h(\boldsymbol{x}) \tag{4.40}$$

where the first term signifies staying close to the gradient update for g while at the same time, making the value of h small using the second term.

Next we define *proximal operator* as a function of h and t_k as follows:

$$\operatorname{prox}_{h,t_k}(\boldsymbol{z}) = \arg\min_{\boldsymbol{x}} \frac{1}{2t_k}||\boldsymbol{x} - \boldsymbol{z}||_2^2 + h(\boldsymbol{x}) \tag{4.41}$$

Thus, Eq. (4.40) can be written as

$$\boldsymbol{x}^{(k)} = \operatorname{prox}_{h,t_k}(\boldsymbol{x}^{(k-1)} - t_k \nabla g(\boldsymbol{x}^{(k-1)})) \tag{4.42}$$

From Eq. (4.41) we can see that the proximal operator does not rely on g at all, only on h. It can be computed analytically for a lot of important h functions, such as the regularization penalty in the objective function (4.32), g can be a complicated function, all we need to do is to compute its gradient.

Now return back to the objective function (4.32), for a better understanding, it can be rewritten in a compacted form:

$$\min_{w_{d,l}^{1,q}} f(W^1) = F(W^1) + R(W^1) \tag{4.43}$$

The proximal operator for $R(W^1)$ is computed as follows

$$\text{prox}_{\lambda,t_k}(w_{d,l}^{1,q}) = \begin{cases} w_{d,l}^{1,q} - \lambda t_k \dfrac{w_{d,l}^{1,q}}{||\{w_{d,l}^{1,q}\}||_2} & ||\{w_{d,l}^{1,q}\}||_2 > \lambda t_k \\ 0 & ||\{w_{d,l}^{1,q}\}||_2 \leq \lambda t_k \end{cases} \quad \text{for} \;\; d = 1, ..., D \tag{4.44}$$

The above proximal operator is also known as the *soft thresholding operator* $\tilde{S}_{\lambda,t_k}(w_{d,l}^{1,q})$.

Therefore, using Eq. (4.42), the update of $w_{d,l}^{1,q}$ can be set as

$$w_{d,l}^{1,q(k)} = \text{prox}_{\lambda,t_k}(w_{d,l}^{1,q(k-1)} - t_k \nabla F(w_{d,l}^{1,q(k-1)})) \tag{4.45}$$

4.3.1.3 Backtracking Line Search

From (4.44) we can see that the update of $w_{d,l}^{1,q}$ depends on the step size t_k. A naive strategy is to set a constant $t_k = t$ for all iterations, however, this strategy poses two problems. A too big t can lead to divergence, meaning the learning function oscillates away from the optimal point. A too small t takes longer time for the function to converge. A good selection of the step size can make the algorithm faster to converge. In this thesis, the *backtracking line search* is considered, which is described as follows.

- First fix two parameters $0 < \beta < 1$ and $0 < \alpha \leq 0.5$.
- At each iteration k, start with $t_k = 1$, and while

$$f(W^{1(k)}) > f(W^{1(k-1)}) - \alpha t_k ||W^{1(k-1)} - W^{1(k)}||_2^2 \tag{4.46}$$

  shrink $t_k = \beta t_k$. Else perform proximal gradient descent update, see (4.45).

4.4 Causal Graph Modeling

In order to represent the dependence structure of multivariate time series, a graph $G = (V, E)$ is used in which the vertex set V stores the time series and E is a set of directed edges between vertices. A directed edge from node v_j to v_i is equivalent to "Z_j **Granger causes** Z_i conditioned on other time series". When there exists reciprocally Granger-causal relationship between Z_j and Z_i, a bi-directed edge lies between v_j and v_i.

Let A be an $K \times K$ adjacency matrix associated to G. For any pair of time series, v_i and v_j, there are two entries in A, $A[i, j]$ and $A[j, i]$, each representing an edge going in one direction. For a bi-directed edge, both entries take value 1, while for a directed edge, only one entry would take 1 and the other entry would take 0.

Now we could evaluate the performance of different approaches for Granger causality analysis by quantifying the similarity between the output causal graph G' with A' and the target causal graph $G^\star$ with $A^\star$. The accuracy of the approach is calculated based on a widely used metric in the classification problem, *AUC (Area Under the ROC Curve)*.

4.5 Simulations

A series of simulation experiments have been conducted to compare bivariate-GC analysis, conditional-GC analysis, group Lasso-GC analysis and MLP Lasso-GC analysis. Since the first three analysis methods rely on statistical models, the rich variety of library makes R the first choice for performing them, while the MLP Lasso-GC analysis is realized by python with the help of Pytorch library.

Thus the following dependencies are required to perform the simulations:

Program Language	**Library**
R 3.3	zoo vars glasso
Python 2.7	numpy scikit-learn pandas pytorch

The implementation code is shown in Appendix B.

All computations were carried out on a Dual-Core Intel Core i5 Processor running at 2.7 GHz with 3MB shared L3 cache under OS X 10.14.

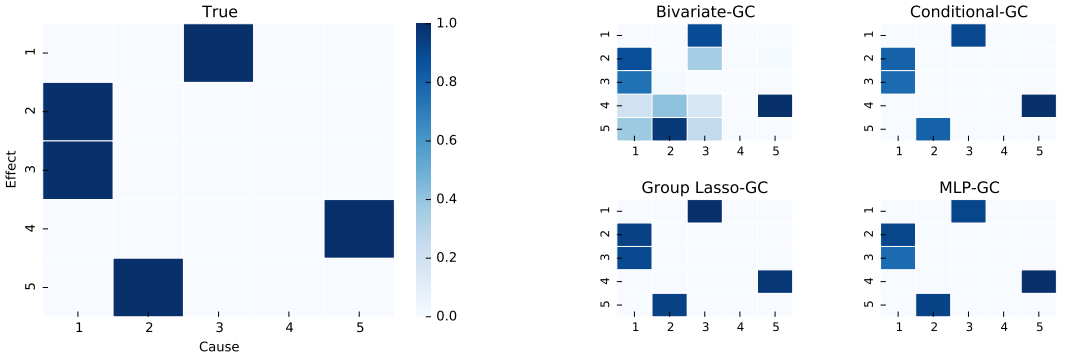
4.5.1 Linear Standardized VAR System

First, the approaches are performed on the simulated linear standardized VAR system. The data generator (written in python) is shown in Appendix B.1. The length N of time series is set to 1000 and two dimensions are considered: $D = 5$ and $D = 30$.

Figure 4.3 shows the true connectivity structure and the estimated structures of each approach[2] in a VAR system with low dimension ($D = 5$), while Figure 4.4 shows that in a VAR system with high dimension ($D = 30$). The comparison result (AUC value) is displayed in Figure 4.5.

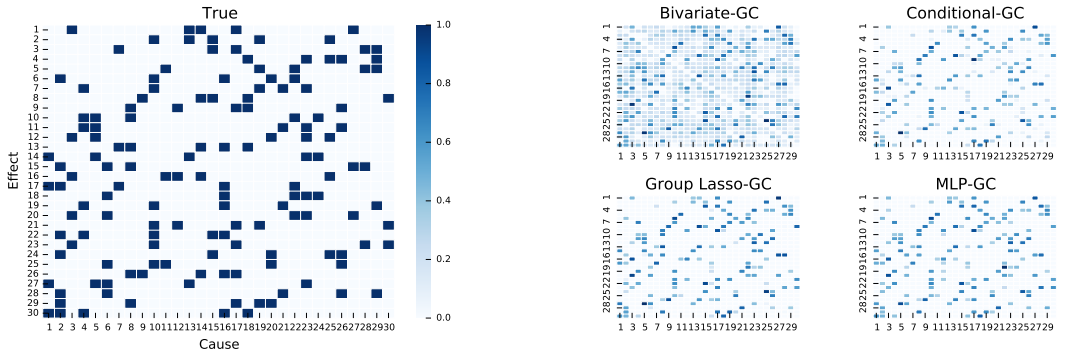
The results clearly show that conditional-GC analysis, group Lasso-GC analysis and MLP Lasso-GC analysis all perform well on the linear system, even when the number

[2]The estimated connectivity plots show not only the estimated connectivity, but also the causal influence.



(a) True Connectivity Structure. (b) Estimated Connectivity Structure.

Figure 4.3: Connectivity Structure of Standardized VAR System (D=5).



(a) True Connectivity Structure. (b) Estimated Connectivity Structure.

Figure 4.4: Connectivity Structure of Standardized VAR System (D=30).

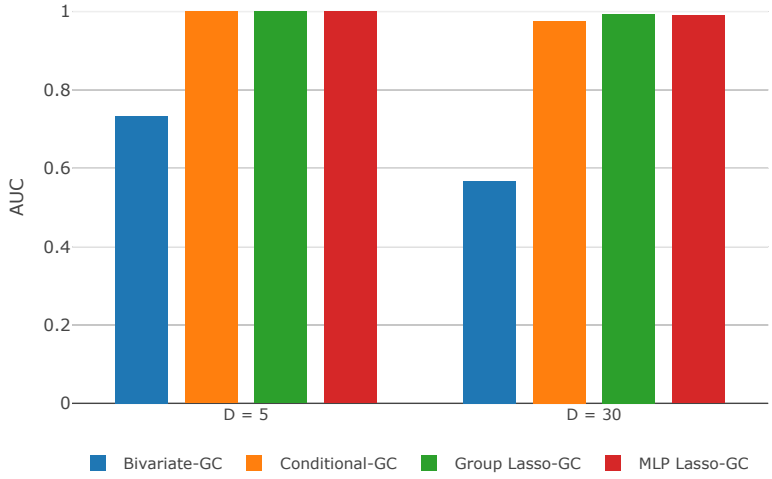


Figure 4.5: Comparison Result for Standardized VAR System.

of time series is small (D = 5), they can completely discover the true temporal-causal dependencies. For a relatively high dimension system (D = 30), group Lasso-GC analysis yields the best performance. By contrast, the bivariate-GC analysis in both case has a very poor performance. From the estimated structures, it is obvious that

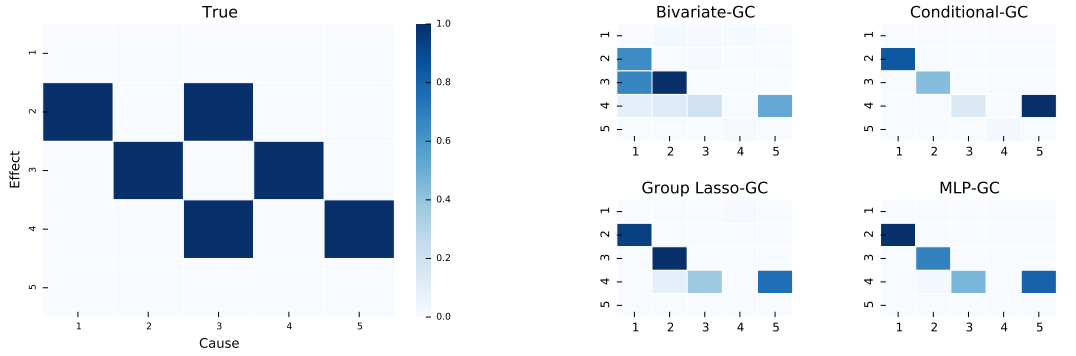
the bivariate-GC analysis discovers too many false positive dependencies, which has confirmed its limitations discussed in the previous section.

4.5.2 Nonlinear Driver-response Henon System

Next, the nonlinear Henon system [VK10] is simulated to test the performance of the GC-analysis methods. The driver-response Henon system maps of D variables, where the first and last variable in the chain of D variables drive their adjacent variable and the other variables drive the adjacent variable to their left and right,

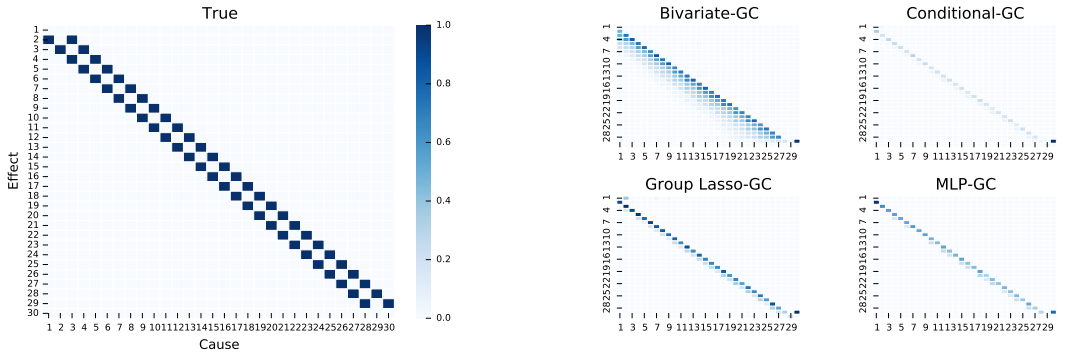
$$\begin{aligned} z_{i,t} &= 1.4 - z_{i,t-1}^2 + 0.3z_{i,t-2} \quad \text{for} \quad i = 1, D \\ z_{i,t} &= 1.4 - \left(0.5C\left(z_{i-1,t-1} + z_{i+1,t-1}\right) + (1-C)z_{i,t-1}\right)^2 + 0.3z_{i,t-2} \quad \text{for} \quad i = 2, ..., D-1 \end{aligned} \tag{4.47}$$

where C is coupling strength (this work takes $C = 1.0$). The length of time series is set to 1000 and two dimensions are considered: $D = 5$ and $D = 30$. The data generator is shown in Appendix B.2.



(a) True Connectivity Structure. (b) Estimated Connectivity Structure.

Figure 4.6: Connectivity Structure of Henon System (D=5).



(a) True Connectivity Structure. (b) Estimated Connectivity Structure.

Figure 4.7: Connectivity Structure of Henon System (D=30).

Figure 4.6 shows the true connectivity structure and the estimated structures of each approach in a Henon system with low dimension ($D = 5$), while Figure 4.7 shows that in a Henon system with high dimension ($D = 30$). The comparison result (AUC value) is displayed in Figure 4.8.

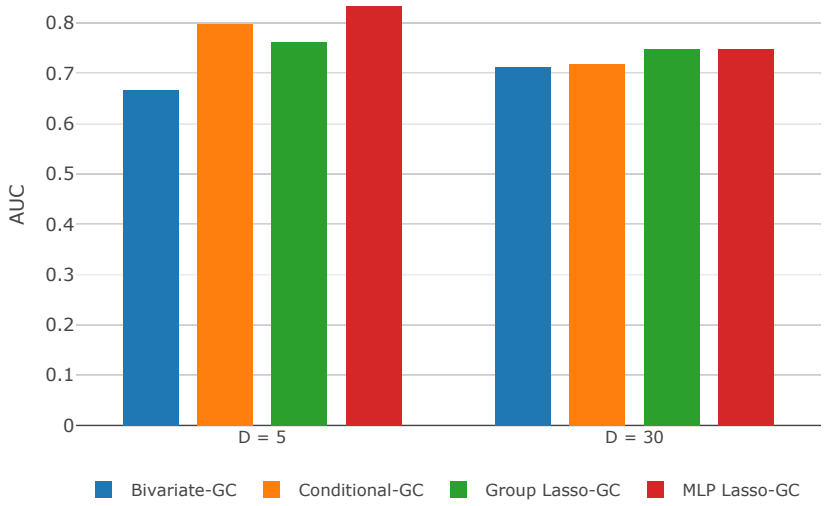


Figure 4.8: Comparison Result for Nonlinear Henon System.

In this case, all the methods have lower performance when the number of time series increases, however, MLP-GC analysis still yields the best performance compared to other methods.

4.6 Conclusions

In this chapter, the question of discovering temporal relationships between multivariate time series data has been addressed based on Granger causality analysis and graph-theoretical approaches. Four main methods are discussed: bivariate-GC analysis, conditional-GC analysis, group Lasso-GC analysis and MLP Lasso-GC analysis, among them, the first three are commonly used to deal with the linear system, while MLP Lasso-GC is applicable not only to linear time series dynamics, but also to nonlinear dependence between series.

When a system e.g. SAP HANA database is monitored, discovering temporal structures enable us to have a deep insight of the underlying system in some way. Since it is always assumed that the metrics in a system are not independent but affect each other over time, the temporal structures can help the engineer to find the most relevant metrics when anomalies are detected in a certain metric and make some interventions to keep the system stable.

5. Analysis of Historical Performance of SAP HANA

This chapter analyzes the historical performance data of the SAP HANA database across a range of key performance indicators using OPUAD algorithm and MLP Lasso-GC analysis method.

SAP HANA is an in-memory platform that combines an ACID-compliant[1] database with advanced data processing, application services, and flexible data integration services. The SAP HANA database can act as a standard SQL-based relational database. In this role, it can serve as either the data provider for classical transactional applications (OLTP) and/or as the data source for analytical requests (OLAP). Database functionality is accessed through an SQL interface.

Monitoring past and current information about the performance of the SAP HANA database is important to prevent performance issues and for root-cause analysis of problems before users are affected.

5.1 Data Collection

The dataset used in this chapter has been recorded from 01.07.2018 at 00:00 to 31.07.2018 at 23:59 of every minute in the system. The KPIs (Key Performance Indicators) to analyze historical performance data of the SAP HANA database are listed in Table 5.1.

[1] In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc.

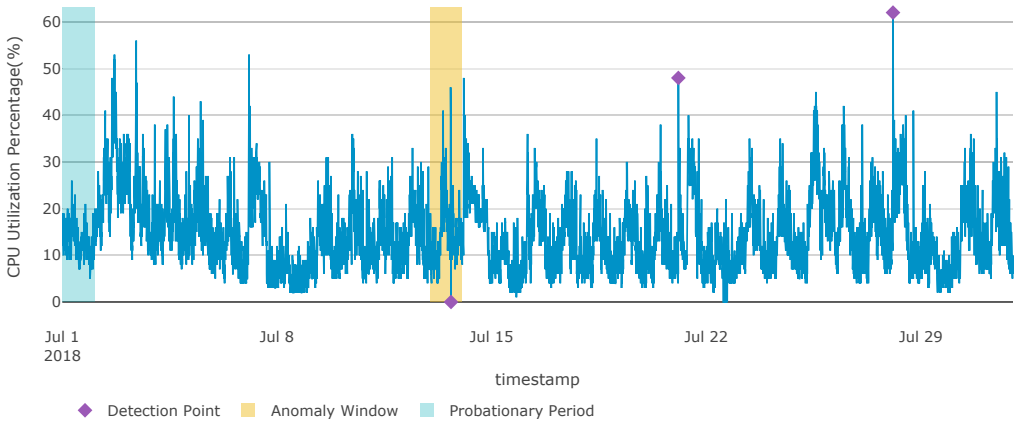
Table 5.1: KPIs for Analyzing Historical Performance of SAP HANA Database

Index	KPI	Description
1	CPU	CPU used by the database process (%)
2	MEMORY_USED	Memory used by the database process (%)
3	DISK_USED	Disk space used by data, log, and trace files belonging to the SAP HANA database (%)
4	NETWORK_IN	Bytes read from the network by all processes
5	NETWORK_OUT	Bytes written to the network by all processes
6	SWAP_IN	Bytes read from swap memory by all processes
7	SWAP_OUT	Bytes written to swap memory by all processes
8	HANDLE_COUNT	Number of open handles in the index server process
9	PING_TIME	Indexserver ping time including nsWatchdog request and collection of service-specific KPIs
10	CONNECTION_COUNT	Number of open SQL connections
11	TRANSACTION_COUNT	Number of open SQL transactions
12	BLOCKED_TRANSACTION_COUNT	Number of blocked SQL transactions
13	STATEMENT_COUNT	Number of finished SQL statements
14	PENDING_SESSION_COUNT	Number of pending requests
15	MVCC_VERSION_COUNT	Number of active MVCC versions
16	RECORD_LOCK_COUNT	Number of acquire record locks
17	CS_READ_COUNT	Number of read requests (selects)
18	CS_WRITE_COUNT	Number of write requests (insert, update, and delete)
19	CS_MERGE_COUNT	Number of merge requests
20	CS_UNLOAD_COUNT	Number of table and column unloads
21	ACTIVE_THREAD_COUNT	Number of active threads
22	WAITING_THREAD_COUNT	Number of waiting threads
23	TOTAL_THREAD_COUNT	Total number of threads
24	ACTIVE_SQL_EXECUTOR_COUNT	Total number of active SqlExecutor threads
25	WAITING_SQL_EXECUTOR_COUNT	Total number of waiting SqlExecutor threads
26	TOTAL_SQL_EXECUTOR_COUNT	Total number of SqlExecutor threads
27	THREADS_WAITING_FOR_SYSTEM_LOCK	Number of threads waiting for system lock
28	THREADS_WAITING_FOR_APPLICATION_LOCK	Number of threads waiting for application lock
29	THREAD_STATE_IS_IO_WAIT	Number of threads whose state is I/O wait
30	THREAD_STATE_IS_JOB_EXEC_WAITING	Number of threads whose state is job exec waiting
31	THREAD_STATE_IS_JOINING	Number of threads waiting for the other to end
32	THREADS_WAITING_FOR_NETWORK	Number of threads waiting for network
33	THREAD_STATE_IS_RESOURCE_LOAD_WAIT	Number of threads waiting for resource to be loaded
34	THREAD_STATE_IS_RUNNING	Number of running threads

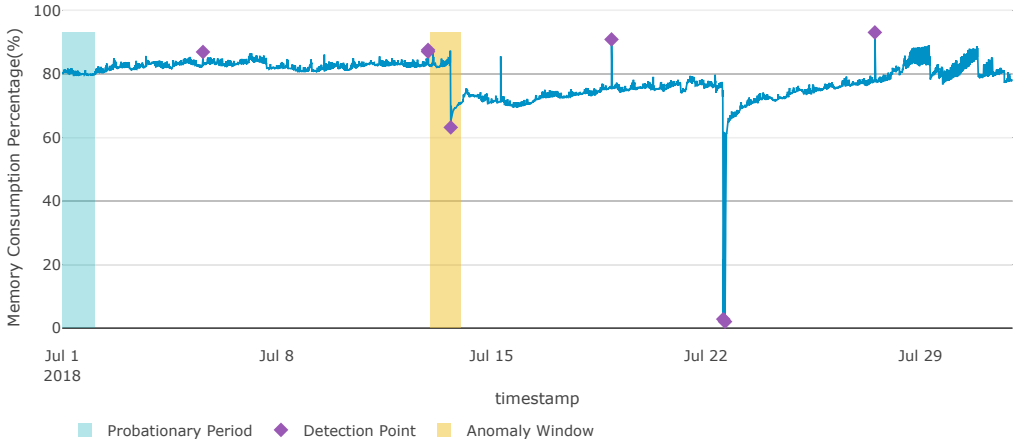
5.2 Track CPU, Memory, Disk Consumption

Issues with overall system performance can be always directly reflected in CPU usage, memory consumption and disk utilization. Therefore, monitoring these metrics can help us to observe the availability status of the SAP HANA database and detect the general symptoms shown by the system, such as poor performance, high memory usage, shortage of disk I/O etc.

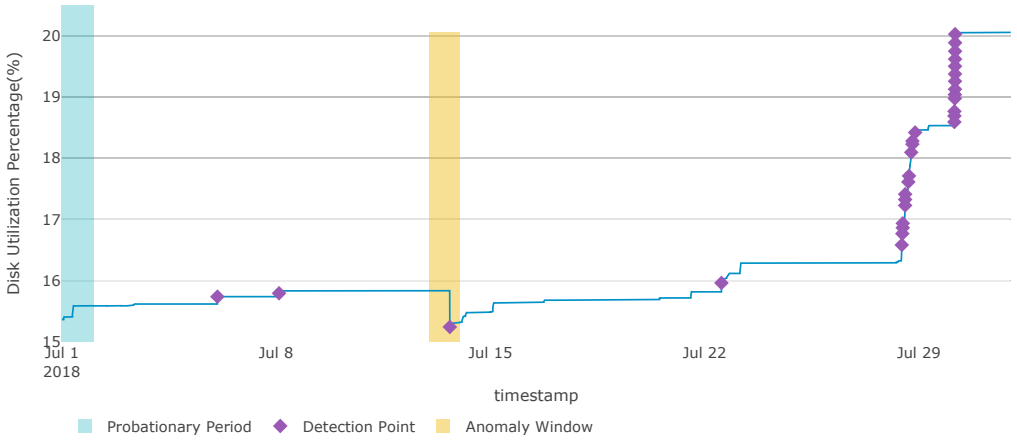
Figure 5.1 shows the anomaly detection result of these three main KPIs of SAP HANA Database by OPUAD algorithm (see Chapter 3).



(a) CPU Utilization



(b) Memory Consumption



(c) Disk Utilization

Figure 5.1: Anomaly Detection Results of Main KPIs of SAP HANA Database.

From the detection result we can see that the OPUAD algorithm has succeeded in detecting the abnormal points, such as a sudden increase or decrease, invalid values or a continuous increase. As discussed in Chapter 3, the detected anomalies act always as alerts to report that something strange is going on in the system, then a domain expert or an engineer is always required to check the corresponding alerts and determine if the detected anomalies are true positives. In other words, the monitor functions mentioned in this work can never be fully automated without humans.

5.3 Temporal Dependency Structure

According to the feedback of the SAP expert, from 01.07.2018 to 31.07.2018, the overall system has issues only in one period: from 2018-07-13 15:50:00 to 2018-07-13 16:16:00. Therefore, the data on 13.07.2018 (highlighted by a yellow region in Figure 5.1) is chosen from each KPI to analyze the overall structure of the system on that day using MLP Lasso-GC method.

However, in this period, there are three KPIs whose value keeps invariant as 0: bytes read from swap memory (Nr.6), bytes written to swap memory(Nr.7) and number of threads waiting for the other to end(Nr.31), thus they are removed from the analysis. The estimated temporal dependency structure is shown in Figure 5.2 (the KPIs are represented by their index in the Figure).

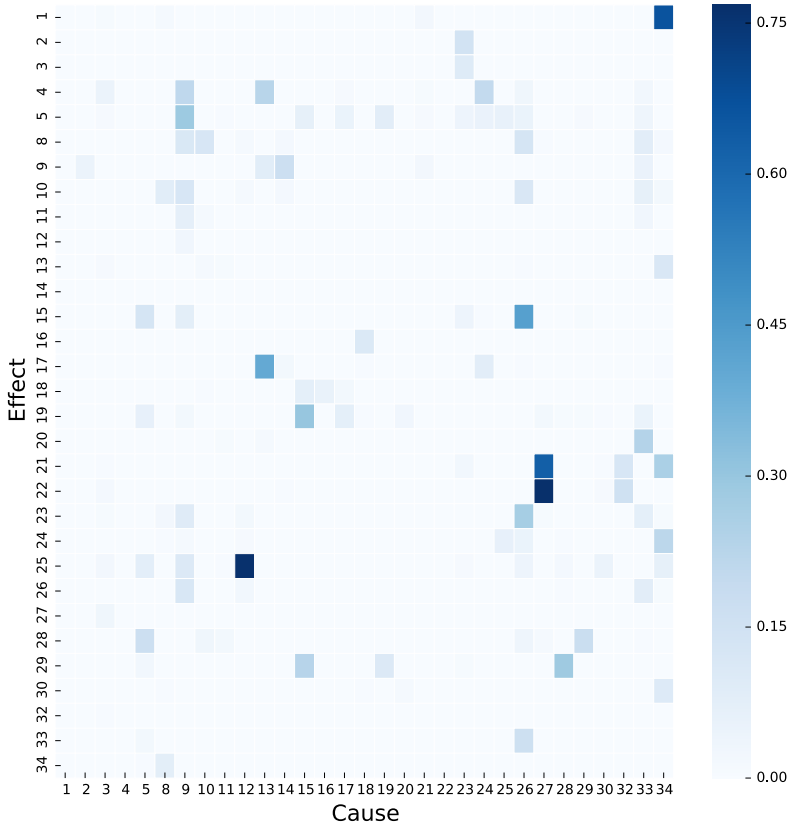


Figure 5.2: Estimated Temporal Dependency Structure of SAP HANA on 13.07.2018.

5.3.1 CPU related Cause

From Figure 5.2, we find out that the number of running threads(Nr.34) has very strong Granger causal influence on CPU utilization(Nr.1). Therefore, the interaction between them is further investigated.

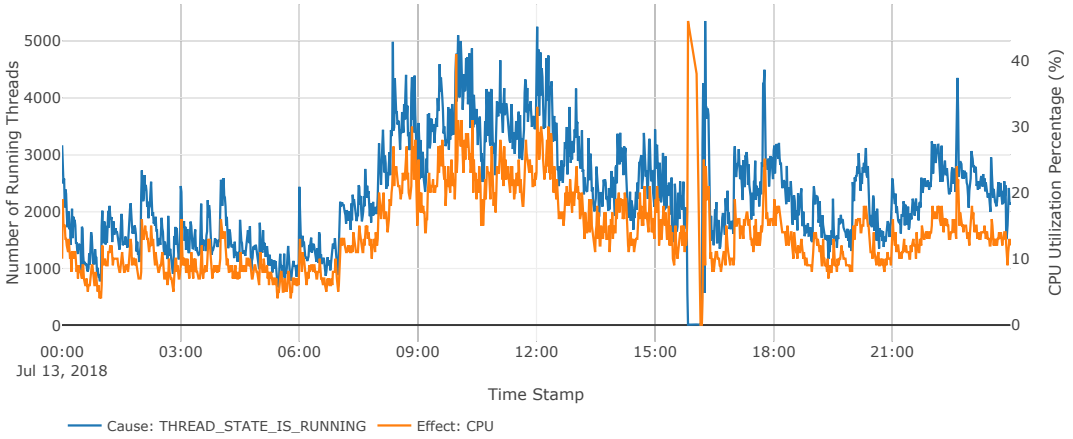


Figure 5.3: Number of Running Threads ⟶ CPU Utilization.

The interaction Figure 5.3 reflects that CPU utilization is actually highly effected by the number of running threads, and on 13.07.2018 at 15:50, the number of running threads suddenly decreased to 0 and then the state of no running threads lasted for a while, CPU utilization decreased to 0 afterwards at 16:09.

5.3.2 Memory and Disk related Cause

Total number of threads(Nr.23) Granger causes memory consumption(Nr.2) and disk utilization(Nr.3) according to Figure 5.2 and the interactions are shown in Figure 5.4 and Figure 5.5.

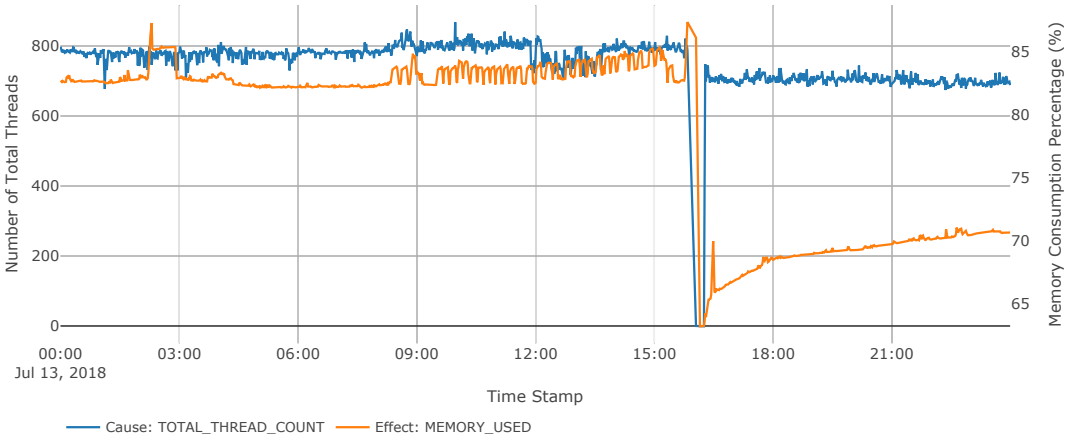


Figure 5.4: Total Number of Threads ⟶ Memory Consumption.

Even though the causal influence in both case is relatively weak, we still can find out some critical phenomenon that the memory consumption and disk usage abnormally decreased to 0 after the number of total threads has jumped to 0 at 16:03. And both

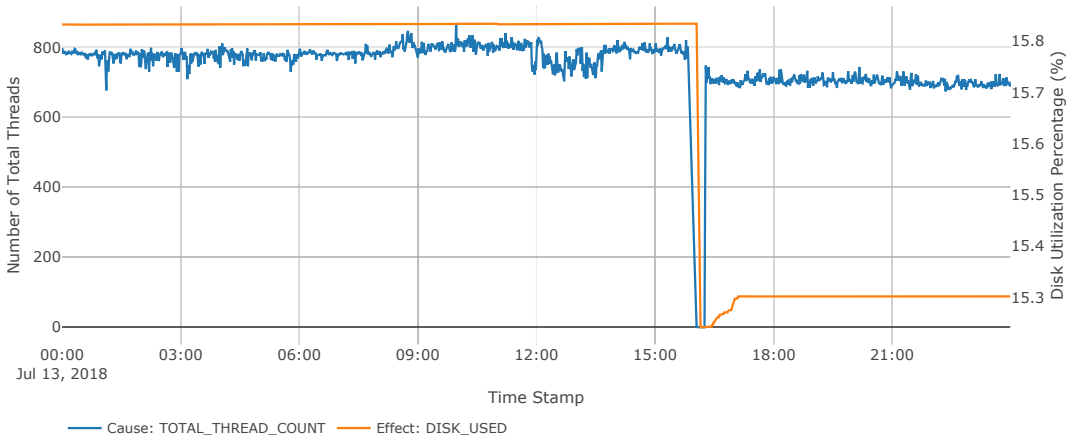


Figure 5.5: Total Number of Threads ⟶ Disk Utilization.

of them returned back to the normal state due to the fact that the number of total threads seemed to be normal again.

5.3.3 Blocked Transaction Monitoring

Blocked transactions are transactions that are unable to be processed further because they need to acquire transactional locks (record or table locks) that are currently held by another transaction. Transactions can also be blocked waiting for other resources such as network or disk.

From Figure 5.2, we see that the number of block transactions(Nr.12) strongly Granger causes the number of waiting SqlExecutor threads(Nr.25). The interaction is shown in Figure 5.6, indicating clearly that the latter depends highly on the former, more blocked transactions leads to more threads waiting to be executed.

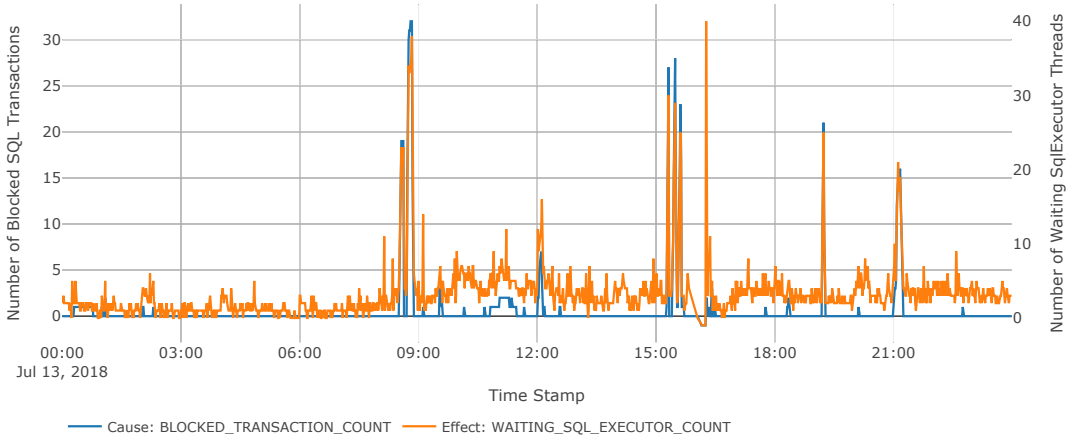


Figure 5.6: Blocked Transactions ⟶ Waiting SqlExecutor Threads.

5.3.4 Threads Monitoring

Due to the query processing mechanisms of the SAP HANA database, the threads have different state values in order to keep the consistency of the database. Figure

5.2 tells us that the number of threads waiting for system lock(Nr.27) has strong causal influence on the number of active threads(Nr.21) and waiting threads(Nr.22). System lock means that it is not possible to run any transactions by connecting to the database. To renew license keys/unlock the system, only HANA user with License Admin system privilege can connect to HANA database. When system is locked, the threads have to wait until the lock is released, resulting in that the number of waiting threads has a increase but there is a decline in the number of active threads. When the system lock is released, the threads continue to run or start to run, therefore, the number of active threads goes up. The dynamic interactions are shown in Figure 5.7 and Figure 5.8.

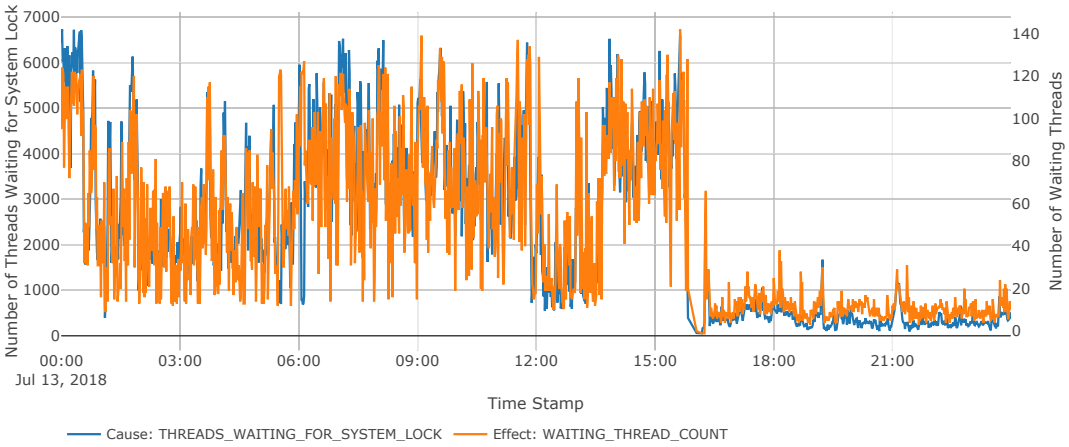


Figure 5.7: Threads Waiting for System Lock ⟶ Waiting Threads.

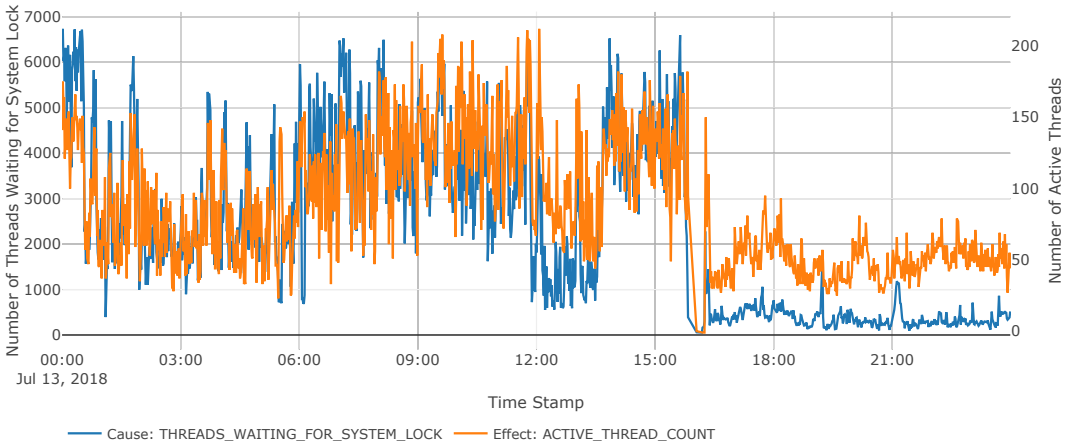


Figure 5.8: Threads Waiting for System Lock ⟶ Active Threads.

Figure 5.3 - Figure 5.8 show that the MLP Lasso-GC method is able to discover the temporal-causal dependencies of KPIs in SAP HANA database on 13.07.2018. From the estimated structure, we can really get some important information which is helpful to diagnose the performance issues.

However, as discussed in Section 2.3 of Chapter 2, analyzing the root causes of the issues still needs more data or some experiences and knowledge of the experts, e.g.

according to the analysis it is clear that the decrease of the CPU utilization was caused by the decrease of the number of the running threads, but, it is impossible to know what has happened resulting in that the number of running threads suddenly jumped to 0 only based on the available data set.

Therefore, it is more reasonable to consider anomaly detection and Granger causality analysis as a good starting point for the further root cause analysis of the system performance issues.

6. Conclusion and Future Work

This chapter summarizes the Thesis, discusses benefits and limitations of the proposed approaches and proposes some useful extensions in the future.

This thesis is mainly concentrated on two topics: anomaly detection for streaming data and Granger causality analysis among multivariate time series data, which are essential in the monitoring service[1].

For anomaly detection, an unsupervised, automated algorithm called OPUAD (Online Prototypes Updating Anomaly Detection) algorithm has been proposed based on kernel density estimators and maximum-entropy principle. It is different with the most off-line learning algorithms because it can process the input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the beginning, which satisfies the requirements of dealing with real-time streaming data. The performance of the OPUAD algorithm was evaluated by the Numetna anomaly benchmark and the result shows that OPUAD takes the third place in the scoreboard, better than the most of other anomaly detection algorithms included in NAB.

Granger causality analysis is commonly used to quantify the temporal-causal effect among time series data, however, its basic form bivariate GC test is not applicable to large scale data with non-linear interactions. Therefore, a comparison between four variants of the Granger causality: bivariate-GC analysis, conditional-GC analysis, group Lasso-GC analysis and MLP Lasso-GC analysis has been made to discuss their

[1] All code for reproducing experiments and further research may be found: https://github.com/JianqiaoJIN/master_thesis. [Jin18]

scalability. A synthetic linear standardized VAR system and a synthetic nonlinear Henon system were generated to test the performance of these four methods, which was evaluated by AUC (Area Under the ROC Curve), a widely used metric in the classification problem. The comparison results show that MLP Lasso-GC can not only perform well on the linear system, but also capture the nonlinear interactions. Even though all of the methods have a weaker performance when the number of time series increases in the nonlinear system, the result of MLP Lasso-GC is still acceptable.

The OPUAD algorithm and the MLP Lasso-GC analysis method at last were applied to the historical data of the SAP HANA database from 01.07.2018 to 31.07.2018 across a range of key performance indicators. The results showed that the two tools succeeded in providing some useful information for diagnosing performance issues of system.

6.1 Discussion

Since the OPUAD algorithm is based on kernel density estimators and maximum-entropy principle, its performance relies on the bandwidth σ and the number of prototypes. If the OPUAD algorithm is applied in a real application, these two parameters must be set appropriately to the setting of the application.

The core idea of MLP Lasso-GC analysis is using a MLP neural network for interpretation of Granger causality, therefore the structure of the network and the lag length determine the accuracy of the estimated dependency structure. In this thesis, the network only had one hidden layer with 10 units and the lag length was selected by trials. So it requires more experiments to specify the network and the lag length to pursue the high accuracy of the estimated dependency structure.

In addition, neither OPUAD nor MLP Lasso-GC can always provide 100% accuracy in all situations when monitoring the system, in other words, there is still a long way to go to develop a fully automated tool with perfect performance in the monitoring service at last. At the current time, machine learning methods just free up some work of human, but the final decisions still need to be made by the experts in the subject-matter filed involved.

6.2 Future Work

A lot of work has already been done in this thesis, but, of course, more work can always be done. Some useful extensions will be proposed here, but this list is by no means complete.

In this thesis, anomaly detection is used for univariate time series to monitor the state of each metric in a system, but in some cases, experts may just want to know the overall state of a system which requires a scalable anomaly detection algorithm for multivariate time series data. Therefore, developing a scalable algorithm for multivariate time series could also be an interesting research area.

Another obvious but useful extension to this work would be to extend the MLP Lasso-GC analysis to more algorithms with different artificial neural networks to see if there are other types of neural networks, e.g. RNN, LSTM etc. that work even

better. In addition, looking for other statistic approaches to discover the nonlinear Granger causality among large scale time series data could also be meaningful.

Furthermore, Granger causality has solved the limitations of the Pearson correlation coefficient which is not applicable to multivariate time series data. However, as the number of time series increases, it is an undeniable fact that the performance of the Granger causality would be poor. Therefore, multivariate correlation analysis could be a good choice to reduce the dimensionality of the original space and improve the quality of the results.

Finally there is also some work that can still be done from a business perspective. As discussed above there are still a few things that need to be done to get the algorithms running in a production environment. It would definitely be interesting to see how well the algorithms improve the stability of the system in a real business application.

References

[A+95] John Aldrich et al. Correlations genuine and spurious in pearson and yule. *Statistical science*, 10(4):364–376, 1995.

[AA13] Ratnadip Adhikari and RK Agrawal. An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*, 2013.

[AC01] J. Aach and G. M. Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17(6):495–508, jun 2001.

[Ada06] Ryan P. Adams. Bayesian online changepoint detection. 2006.

[Agg16] Charu C. Aggarwal. An introduction to outlier analysis. pages 1–34, dec 2016.

[Agg17] Charu C. Aggarwal. Time series and multidimensional streaming outlier detection. *Outlier Analysis*, January 2017.

[aJP13] B. Chen an J. Pearl. Regression and causation : A critical examination of six econometrics textbooks. In *Real-World Economics Review*, pages 2–20, 2013.

[Aka74] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, dec 1974.

[ALA07] Andrew Arnold, Yan Liu, and Naoki Abe. Temporal causal modeling with graphical granger methods. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*. ACM Press, 2007.

[ALPA17] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, nov 2017.

[BDL+04] A. Brovelli, M. Ding, A. Ledberg, Y. Chen, R. Nakamura, and S. L. Bressler. Beta oscillations in a large-scale sensorimotor cortical network: Directional influences revealed by granger causality. *Proceedings of the National Academy of Sciences*, 101(26):9849–9854, jun 2004.

[BGBMY01] Ana Maria Bianco, M Garcia Ben, EJ Martinez, and Victor J Yohai. Outlier detection in regression models with arima errors using robust estimates. *Journal of Forecasting*, 20(8):565–579, 2001.

[BI16] Evgeny Burnaev and Vladislav Ishimtsev. Conformalized density- and distance-based anomaly detection in time-series data. *CoRR*, abs/1608.04585, 2016.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006.

[BL13] Mohammad Taha Bahadori and Yan Liu. An examination of practical granger causality inference. pages 467–475, may 2013.

[BP13] Kenneth A. Bollen and Judea Pearl. Eight myths about causality and structural equation models. *Handbook of Causal Analysis for Social Research*, January 2013.

[CBD06] Yonghong Chen, Steven L. Bressler, and Mingzhou Ding. Frequency decomposition of conditional granger causality and application to multivariate neural field potential data. *Journal of Neuroscience Methods*, 150(2):228–237, jan 2006.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection. *ACM Computing Surveys*, 41(3):1–58, jul 2009.

[Cha03] Chris Chatfield. *The Analysis of Time Series*. Taylor & Francis Ltd., 2003.

[CKMP02] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, June 2002.

[CW94] Chris Chatfield and Andreas S. Weigend. Time series prediction: Forecasting the future and understanding the past. *International Journal of Forecasting*, 10(1):161–163, jun 1994.

[DCB06] Mingzhou Ding, Yonghong Chen, and Steven L Bressler. Granger causality: basic theory and application to neuroscience. *Handbook of time series analysis: recent theoretical developments and applications*, pages 437–460, 2006.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley John + Sons, 2000.

[ED09] Michael Eichler and Vanessa Didelez. On granger causality and the effect of interventions in time series. *Lifetime Data Analysis*, 16(1):3–32, nov 2009.

[FL01] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, 2001.

[FMR98] Nir Friedman, Kevin Murphy, and Stuart Russell. Learning the structure of dynamic probabilistic networks. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98,

pages 139–147, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[FRM94] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, May 1994.

[FYM05] Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. An anomaly detection method for spacecraft using relevance vector learning. pages 785–790, 2005.

[Geu01] Pierre Geurts. Pattern extraction for time series classification. In Luc De Raedt and Arno Siebes, editors, *Principles of Data Mining and Knowledge Discovery*, pages 115–127, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Gew84] John F. Geweke. Measures of conditional linear dependence and feedback between time series. *Journal of the American Statistical Association*, 79(388):907–915, dec 1984.

[GOV+12] Irene Gijbels, Marek Omelka, Noël Veraverbeke, et al. Multivariate and functional covariates and conditional copulas. *Electronic Journal of Statistics*, 6:1273–1306, 2012.

[Gra69] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424, aug 1969.

[Gra90] I. Grabec. Self-organization of neurons described by the maximum-entropy principle. *Biological Cybernetics*, 63(5):403–409, sep 1990.

[GSK+08] Shuixia Guo, Anil K. Seth, Keith M. Kendrick, Cong Zhou, and Jianfeng Feng. Partial granger causality—eliminating exogenous inputs and latent variables. *Journal of Neuroscience Methods*, 172(1):79–93, jul 2008.

[GVO11] Irène Gijbels, Noël Veraverbeke, and Marel Omelka. Conditional copulas, association measures and their applications. *Comput. Stat. Data Anal.*, 55(5):1919–1932, May 2011.

[GZ03] Jan G. De Gooijer and Dawit Zerom. On conditional density estimation. *Statistica Neerlandica*, 57(2):159–176, may 2003.

[Haw80] D. Hawkins. *Identification of Outliers (Monographs on Statistics and Applied Probability)*. Springer, 1980.

[HG09] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, September 2009.

[HHWB02] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. pages 170–180, 2002.

[HJ94] Craig Hiemstra and Jonathan D. Jones. Testing for linear and nonlinear granger causality in the stock price- volume relation. *The Journal of Finance*, 49(5):1639, dec 1994.

[HL14] Meng Hu and Hualou Liang. A copula approach to assessing granger causality. *NeuroImage*, 100:125–134, oct 2014.

[Hol86] Paul W. Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81(396):945–960, dec 1986.

[HSKP08] Patrik O. Hoyer, Shohei Shimizu, Antti J. Kerminen, and Markus Palviainen. Estimation of causal effects using linear non-gaussian causal models with hidden variables. *International Journal of Approximate Reasoning*, 49(2):362–378, oct 2008.

[HSP15] Kateřina Hlaváčková-Schindler and Sergiy Pereverzyev. Lasso granger causal models: Some strategies and their efficiency for gene expression regulatory networks. pages 91–117, 2015.

[JAK01] Mahesh V. Joshi, Ramesh C. Agarwal, and Vipin Kumar. Mining needle in a haystack: Classifying rare classes via two-phase rule induction. *SIGMOD Rec.*, 30(2):91–102, May 2001.

[JAK02] Mahesh V. Joshi, Ramesh C. Agarwal, and Vipin Kumar. Predicting rare classes: Can boosting make any weak learner strong? In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 297–306, New York, NY, USA, 2002. ACM.

[Jin18] Jianqiao Jin. Anomaly detection and exploratory causal analysis for sap hana. https://github.com/JianqiaoJIN/master_thesis.git, 2018.

[J.P09] J.Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Pr., 2009.

[KLR04] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 206–215, New York, NY, USA, 2004. ACM.

[KM09] Samantha Kleinberg and Bud Mishra. The temporal logic of causal structures. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 303–312, Arlington, Virginia, United States, 2009. AUAI Press.

[KP98] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, 1998.

[Lü05] Helmut Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer Berlin Heidelberg, 2005.

[LA15] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. dec 2015.

[LALR09] A. C. Lozano, N. Abe, Y. Liu, and S. Rosset. Grouped graphical granger modeling for gene expression regulatory networks discovery. *Bioinformatics*, 25(12):i110–i118, may 2009.

[LKL+04] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Jeffrey P. Lankford, and Donna M. Nystrom. Visually mining and monitoring massive time series. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04*. ACM Press, 2004.

[LLNM+09] Aurelie C. Lozano, Hongfei Li, Alexandru Niculescu-Mizil, Yan Liu, Claudia Perlich, Jonathan Hosking, and Naoki Abe. Spatial-temporal causal modeling for climate change attribution. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*. ACM Press, 2009.

[LLW09] Han Liu, John Lafferty, and Larry Wasserman. The nonparanormal: Semiparametric estimation of high dimensional undirected graphs. *J. Mach. Learn. Res.*, 10:2295–2328, December 2009.

[LTTT14] Richard Lockhart, Jonathan Taylor, Ryan J. Tibshirani, and Robert Tibshirani. A significance test for the lasso. *The Annals of Statistics*, 42(2):413–468, apr 2014.

[McC16] James M. McCracken. *Exploratory Causal Analysis with Time Series Data*. Morgan & Claypool Publishers, 2016.

[MKB+10] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. 2:1045–1048, 01 2010.

[MPS08] D. Marinazzo, M. Pellicoro, and S. Stramaglia. Kernel-granger causality and the analysis of dynamical networks. *Physical Review E*, 77(5), may 2008.

[MRA+16] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[MS03] Markos Markou and Sameer Singh. Novelty detection: a review—part 1: statistical approaches. *Signal Processing*, 83(12):2481–2497, dec 2003.

[NH98] Radford Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

[PAL04] Clifton Phua, Damminda Alahakoon, and Vincent Lee. Minority report in fraud detection. *ACM SIGKDD Explorations Newsletter*, 6(1):50, jun 2004.

[Pan04] Cees Diks & Valentyn Panchenko. Modified hiemstra-jones test for granger non-causality. Computing in Economics and Finance 2004 192, Society for Computational Economics, 2004.

[PCCT14] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, jun 2014.

[Pin05] Brandon Pincombe. Anomaly detection in time series of graphs using arma processes. *Asor Bulletin*, 24(4):2, 2005.

[PRT10] R. Pietrantuono, S. Russo, and K. S. Trivedi. Online monitoring of software system reliability. In *Proc. European Dependable Computing Conf*, pages 209–218, April 2010.

[Rö71] E. Rödel. *Fisher, R. A.: Statistical Methods for Research Workers, 14. Aufl., Oliver & Boyd, Edinburgh, London 1970. XIII, 362 S., 12 Abb., 74 Tab., 40 s*, volume 13. Wiley, 1971.

[Ran06] Jörn Rank. *Copulas: From theory to application in finance*. Risk Books, 2006.

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, mar 1978.

[Sch00] Thomas Schreiber. Measuring information transfer. *Physical Review Letters*, 85(2):461–464, jul 2000.

[SE07] Anil K. Seth and Gerald M. Edelman. Distinguishing causal interactions in neural populations. *Neural Computation*, 19(4):910–933, apr 2007.

[SER16] Markus Schneider, Wolfgang Ertel, and Fabio Ramos. Expected similarity estimation for large-scale batch and streaming anomaly detection. *Machine Learning*, 105(3):305–333, may 2016.

[SGS93] Peter Spirtes, Clark Glymour, and Richard Scheines. *Causation, Prediction, and Search*. Springer New York, 1993.

[SK16] Elsa Siggiridou and Dimitris Kugiumtzis. Granger causality in multivariate time series using a time-ordered restricted vector autoregressive model. *IEEE Transactions on Signal Processing*, 64(7):1759–1773, 2016.

[Skl96] A. Sklar. Random variables, distribution functions, and copulas: A personal look backward and forward. *Lecture Notes-Monograph Series*, 28:1–14, 1996.

[SSB14] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

[TCF+18] Alex Tank, Ian Covert, Nicholas Foti, Ali Shojaie, and Emily Fox. Neural granger causality for nonlinear time series. *arXiv preprint arXiv:1802.05842*, 2018.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[Tuk77] John W. Tukey. *Exploratory Data Analysis*. Pearson, 1977.

[TZ06] Pravin K Trivedi and David M Zimmer. Copula modeling: An introduction for practitioners. *Foundations and Trends® in Econometrics*, 1(1):1–111, 2006.

[VK10] Ioannis Vlachos and Dimitris Kugiumtzis. Nonuniform state-space reconstruction and coupling detection. *Phys. Rev. E*, 82:016207, Jul 2010.

[Vri12] Scott I. Vrieze. Model selection and psychological theory: A discussion of the differences between the akaike information criterion (AIC) and the bayesian information criterion (BIC). *Psychological Methods*, 17(2):228–243, 2012.

[Wei04] Gary M. Weiss. Mining with rarity. *ACM SIGKDD Explorations Newsletter*, 6(1):7, jun 2004.

[Wie56] N. Wiener. The theory of prediction. In *Modern mathematics for engineers, Series I*, pages 125–139. 1956.

[WVC+11] Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur, Vanish Talwar, Wade Satterfield, and Karsten Schwan. Statistical techniques for online anomaly detection in data centers. may 2011.

[WVL+11] Chengwei Wang, Krishnamurthy Viswanathan, Choudur Lakshminarayan, Vanish Talwar, Wade Satterfield, and Karsten Schwan. Statistical techniques for online anomaly detection in data centers. In *Integrated Network Management*, pages 385–392. Citeseer, 2011.

[YHPW09] Young-Ha, H.H. Permuter, and T. Weissman. Directed information, causal estimation, and communication in continuous time. jun 2009.

[YiTWM04] Kenji Yamanishi, Jun ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, may 2004.

[YL06] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, feb 2006.

[Yu12] Jie Yu. A nonlinear kernel gaussian mixture model based inferential monitoring approach for fault detection and diagnosis of chemical processes. *Chemical Engineering Science*, 68(1):506–519, jan 2012.

[YY16] Yu Yin and Dezhong Yao. Causal inference based on the analysis of events of relations for non-stationary variables. *Scientific Reports*, 6(1), jul 2016.

[Zou06] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, dec 2006.

A. OPUAD Algorithm Implementation

The following Listing shows the implementation of the OPUAD algorithm described in Chapter 3 with Python

```
import math, copy
import numpy as np
import pandas as pd
from scipy.stats import norm
from nupic.algorithms import anomaly_likelihood

class OPUAD():
    def __init__(self,T,probationaryPeriod,SPATIAL_TOLERANCE):
        # number of prototypes
        self.K = 5

        # number of data points that has been seen so far
        self.record = 0

        # threshold
        self.T = T

        # probationaryPeriod
        self.probationaryPeriod = probationaryPeriod

        # initialize the anomaly likelihood object
        numentaLearningPeriod = int(math.floor(self.
            probationaryPeriod / 2.0))
        self.anomalyLikelihood = anomaly_likelihood.
            AnomalyLikelihood(
             learningPeriod=numentaLearningPeriod,
             estimationSamples=self.probationaryPeriod-
                 numentaLearningPeriod,
             reestimationPeriod=100
        )

        # keep track of valid range for spatial anomaly detection
        self.SPATIAL_TOLERANCE = SPATIAL_TOLERANCE
```

```
        self.minVal = None
        self.maxVal = None

    def handleRecord(self, inputData):

        self.record += 1
        value = inputData['value']
        timestamp = inputData['timestamp']

        if (self.record == 1):
            """set prototypes"""
            self.q = np.repeat(value, self.K)
            if (value == 0):
                self.sigma = 1
            else:
                self.sigma = abs(value * 0.1)

        """compute anomaly score"""
        rawScore = self.calculateAnomalyScore(value)

        """compute anomaly likelihood"""
        anomalyScore = self.anomalyLikelihood.anomalyProbability(
                value, rawScore, timestamp)
        logScore = self.anomalyLikelihood.computeLogLikelihood(
           anomalyScore)
        finalScore = logScore

        """check spatial anomaly for univariate time series"""
        # check if there is a spatial anomaly
        # update max and min
        spatialAnomaly = False
        if self.minVal != self.maxVal:
            tolerance = (self.maxVal - self.minVal) * self.
               SPATIAL_TOLERANCE
            maxExpected = self.maxVal + tolerance
            minExpected = self.minVal - tolerance
            if value > maxExpected or value < minExpected:
                spatialAnomaly = True
        if self.maxVal is None or value > self.maxVal:
            self.maxVal = value
        if self.minVal is None or value < self.minVal:
            self.minVal = value

        if spatialAnomaly:
            finalScore = 1.0

        """ report anomaly """
        alertAnomaly = False
        if self.record > probationaryPeriod and finalScore >= T:
            alertAnomaly = True

        """ update prototypes """
        if (self.record > 1):
            self.updateParameter(value)

        return alertAnomaly

    def calculateAnomalyScore(self, value):
```

```
        density_sum = 0

        for i in range(self.K):
            density_sum = density_sum + norm.pdf(value, self.q[i],
                self.sigma) / self.K

        if density_sum == 0:
            anomalyScore = 100
        else:
            anomalyScore = - math.log(density_sum)

        return anomalyScore

    def updateParameter(self, value):
        C = []
        B = []

        # l -> l-th prototype

        for l in range(self.K):
            tmp_C = []
            tmp_B = 0

            # k -> k-th prototype

            for k in range(self.K):
                tmp = (self.q[k] - self.q[l])*(self.q[k]-self.q[l])
                C_lk = (1-tmp/(2*self.sigma*self.sigma))*np.exp(-
                    tmp/(4*self.sigma*self.sigma))
                tmp_C.append(C_lk)
                tmp_B = tmp_B + tmp*np.exp(-tmp/(4*self.sigma*self.
                    sigma))

            C.append(tmp_C)
            B_l = (value-self.q[l])*np.exp(-(value-self.q[l])*(
                value-self.q[l])/(4*self.sigma*self.sigma))-tmp_B/
                self.K
            B_l = B_l * self.K / (self.record)
            B.append(B_l)

        # solve the linear equation
        try:
            delta_q = np.linalg.lstsq(C, B, rcond=None)[0]
        except np.linalg.linalg.LinAlgError:
            print "Singular␣Matrix"
        self.q = self.q + delta_q
```

Listing A.1: OPUAD Algorithm for Streaming Data

B. Granger Causality Analysis Implementation

B.1 Standardized VAR System Generator

The following Listing shows the standardized VAR system generator in Chapter 4 with python

```
"""
    Generate a linear standardized VAR system
    input:
        N: time series length
        D: dimension of system
        lag: lag for VAR system
    output:
        S: simulated VAR system (N * D)
        A: adjacent matrix indicating the true connection
           column: candidate (j)
           row: target (i)
           A[i,j] = 1: j Granger causes i
           A[i,j] = 0: j does not Granger causes i
"""

def standardized_var_system(N, D, lag=1):

    def stationary_var(beta, D, lag, radius):
        bottom = np.hstack((np.eye(D * (lag-1)), np.zeros((D * (lag
            - 1), D))))
        beta_tilde = np.vstack((beta,bottom))
        eig = np.linalg.eigvals(beta_tilde)
        maxeig = max(np.absolute(eig))
        not_stationary = maxeig >= radius

        return beta * 0.95, not_stationary

    if D  == 5:
        sparsity = 0.5
```

```
    if D == 30:
        sparsity = 0.2

    beta_value = 5
    sd_e = 2.0
    radius = 0.97
    beta = np.eye(D) * beta_value
    A = np.zeros((D,D))

    # Set dependencies for each component
    num_nonzero = int(D * sparsity) - 1
    for i in range(D):
        choice = np.random.choice(D - 1, size = num_nonzero,
            replace = False)
        choice[choice >= i] += 1
        beta[i, choice] = beta_value
        A[i, choice] = 1

    # Create full beta matrix
    beta_full = beta
    for i in range(1, lag):
        beta_full = np.hstack((beta_full, beta))

    not_stationary = True
    while not_stationary:
        beta_full, not_stationary = stationary_var(beta_full, D,
            lag, radius)

    # create VAR model
    errors = np.random.normal(loc = 0, scale = sd_e, size = (D, N))
    S = np.zeros((D, N))
    S[:, range(lag)] = errors[:, range(lag)]
    for i in range(lag, N):
        S[:, i] = np.dot(beta_full, S[:, range(i - lag, i)].flatten
            (order = 'F')) + errors[:, i]

    return S.T, A
```

Listing B.1: Standardized VAR System Generator

B.2 Nonlinear Henon System Generator

The following Listing shows the nonlinear Henon system generator in Chapter 4 with python

```
"""
    Generate a nonlinear driver-response Henon system
    input:
        N: time series length
        D: dimension of system

    output:
        S: simulated Henon system (N * D)
        A: adjacent matrix indicating the true connection
           column: candidate (j)
           row: target (i)
           A[i,j] = 1: j Granger causes i
```

```
            A[i,j] = 0: j does not Granger causes i
"""

def henon_system(N, D):

    sd_e = 1.0

    # create Henon syste
    S = np.random.uniform(0,1,(D,N))

    # head and end
    for d in [0,D-1]:
        for t in range(2, N):
            S[d, t] = 1.4 - np.square(S[d, t-1]) + 0.3*S[d, t-2]

    C = 1.0 # coupling strength

    for d in range(1,D-1):
        for t in range(2,N):
            S[d, t] = 1.4 - np.square(0.5*C*(S[d-1,t-1] + S[d+1, t
                -1]) + (1-C)*S[d, t-1]) + 0.3*S[d, t-2]

    A = np.zeros((D,D))
    for d in range(1, D-1):
        A[d, d-1] = 1
        A[d, d+1] = 1

    return S.T, A
```

Listing B.2: Nonlinear Henon System Generator

B.3 Bivariate GC-Analysis

The following Listing shows the implementation of bivariate-GC analysis in Chapter 4 with R

```
require("zoo")
require("vars")

# --- Biavariate Granger causality analyzer -------#
# input:
#   S: system (type: data frame)
#   d: directory name
#
# result:
#   A_: estimated adjacent matrix
#   W:  strength of causal influence

bivariateGC_analyzer <- function(S, d){
  N <- dim(S)[1]
  D <- dim(S)[2]
  A_ <- matrix(0,D,D) # estimated connection
  W <- matrix(0,D,D) #causal influence

  # normalize
  S <- as.data.frame(scale(S, center = TRUE, scale = TRUE))
```

```
  # dataframe -> matrix
  S <- do.call("merge", lapply(1:D, function(k) as.zoo(S[k])))

  for ( i in 1:D){

    target <- S[,i]

    for (j in 1:D){
      if (j == i){
        next
      }

      candidate <- S[,j]

      # optimal lag selection
      bi_system <- merge(target, candidate)
      L <- VARselect(bi_system, lag.max = 5)$selection[1] #AIC
          criteria

      # prepare train data
      candidate_X <- do.call("merge", lapply(1:L, function(k) lag(
          candidate, -k)))
      target_X <- do.call("merge", lapply(1:L, function(k) lag(
          target, -k)))

      all <- merge(target, candidate_X, target_X)
      colnames(all) <- c("target", paste("candiate", 1:L, sep = "_"
          ), paste("target", 1:L, sep = "_"))
      all <- na.omit(all)
      target_Y <- as.vector(all[,1])
      candidate_X <- as.matrix(all[,(1:L+1)])
      target_X <- as.matrix(all[,(1:L + 1 + L)])

      # train data -> OLS regression
      U_model <- lm(formula = target_Y ~ target_X + candidate_X) #
          unstricted model
      R_model <- lm(formula = target_Y ~ target_X) # restricted
          model

      # F-test
      p <- anova(R_model,U_model)$`Pr(>F)`[2] #F-test and Pr(>F)

      if(p < 0.05){
        A_[i,j] <- 1

        sigma_U <- summary(U_model)$sigma
        sigma_R <- summary(R_model)$sigma
        W[i,j] <- log((sigma_R/sigma_U)^2)
      }

    }
  }

  return (list(A_,W))
}
```

Listing B.3: Bivariate Granger Causality Analysis

B.4 Conditional GC-Analysis

The following Listing shows the implementation of conditional-GC analysis in Chapter 4 with R

```
require("zoo")
require("vars")

# --- Conditional Granger causality analyzer -------#
# input:
#   S: system (type: data frame)
#   d: directory name
#
# result:
#   A_: estimated adjacent matrix
#   W:  strength of causal influence

conditionalGC_analyzer <- function(S, d){

  N <- dim(S)[1]
  D <- dim(S)[2]
  A_ <- matrix(0,D,D) # estimated connection
  W <- matrix(0,D,D) #causal influence

  # normalize
  S <- as.data.frame(scale(S, center = TRUE, scale = TRUE))

  # dataframe -> matrix
  S <- do.call("merge", lapply(1:D, function(k) as.zoo(S[k])))

  # optimal lag selection
  L <- VARselect(S,lag.max = 5)$selection[1] #AIC criteria

  # prepare X_train data
  name_ <- paste(names(S)[1],1:L,sep='_')
  X_train <- do.call("merge", lapply(1:L, function(k) lag(S[,1], -k
     )))
  for (i in 2:D){
    name_ <- c(name_, paste(names(S)[i],1:L,sep='_'))
    Z_i_lag <- do.call("merge", lapply(1:L, function(k) lag(S[,i],
       -k)))
    X_train <- merge(X_train, Z_i_lag)
  }

  X_train <- na.omit(X_train)
  colnames(X_train) <- name_


  for ( i in 1:D){

    target_Y <- S[(L+1):N, i]

    for (j in 1:D){
      if (j == i){
        next
      }

      candidate_X <- X_train[,((j-1)*L+1):(j*L)]
```

```
      condition_X <- X_train[,-(((j-1)*L+1):(j*L))]

      # OLS regression
      U_model <- lm(formula = target_Y ~ condition_X + candidate_X)
          # unstricted model
      R_model <- lm(formula = target_Y ~ condition_X) # restricted
         model

      # F-test
      p <- anova(R_model,U_model)$`Pr(>F)`[2] #F-test and Pr(>F)

      if(p < 0.05){
        A_[i,j] <- 1

        sigma_U <- summary(U_model)$sigma
        sigma_R <- summary(R_model)$sigma
        W[i,j] <- log((sigma_R/sigma_U)^2)
      }

    }
  }

  return (list(A_,W))
}
```

Listing B.4: conditional Granger Causality Analysis

B.5 Group Lasso-GC Analysis

The following Listing shows the implementation of group Lasso-GC analysis in Chapter 4 with R

```
require("gglasso","zoo","vars")

# --- Group Lasso Granger causality analyzer -------#
# input:
#   S: system (type: data frame)
#   d: directory name
#
# result:
#   A_: estimated adjacent matrix
#   W:  strength of causal influence

groupLassoGC_analyzer <- function(S, d){
  N <- dim(S)[1]
  D <- dim(S)[2]
  A_ <- matrix(0,D,D) # estimated connection
  W <- matrix(0,D,D) #causal influence

  # normalize
  S <- as.data.frame(scale(S, center = TRUE, scale = TRUE))

  # dataframe -> matrix
  S <- do.call("merge", lapply(1:D, function(k) as.zoo(S[k])))

  # optimal lag selection
  L <- VARselect(S,lag.max = 5)$selection[1] #AIC criteria
```

```
  # prepare X_train data
  name_ <- paste(names(S)[1],1:L,sep='_')
  X_train <- do.call("merge", lapply(1:L, function(k) lag(S[,1], -k
     )))
  for (i in 2:D){
    name_ <- c(name_, paste(names(S)[i],1:L,sep='_'))
    Z_i_lag <- do.call("merge", lapply(1:L, function(k) lag(S[,i],
       -k)))
    X_train <- merge(X_train, Z_i_lag)
  }

  X_train <- na.omit(X_train)
  colnames(X_train) <- name_
  X_train <- as.matrix(X_train)

  for (i in 1:D){
    target_Y <- as.matrix(S[(L+1):N, i])

    # OLS regression with group Lasso penalty
    group <- rep(1:(dim(X_train)[2]/L), each = L)
    cv <- cv.gglasso(x=X_train, y=target_Y, group = group, loss = "
       ls", pred.loss = "L1",lambda.factor=0.05, nfolds=5)
    pre <- coef(cv$gglasso.fit, s = cv$lambda.1se)

    # select coefficients whose value is not equal to 0
    pre <- pre[-1] # remove the intercept
    names(pre) <- name_
    pre <- pre[pre != 0]
    if (length(pre) == 0){
      next
    }

    # get the group index
    pre_index <- do.call("c",lapply(1:length(pre), function(k)
       strsplit(names(pre)[k], split = '_')) )
    pre_index <- do.call("c", lapply(1:length(pre_index), function(
       k) as.integer(pre_index[[k]][2])))
    names(pre) <- pre_index

    causes <- pre[names(pre) != i]
    if (length(causes) == 0){
      next
    }
    causes_name <- unique(names(causes))

    # record the connectivity and causal influence
    for (j in 1:length(causes_name)){
      A_[i, as.integer(causes_name[j])] <- 1
      W[i, as.integer(causes_name[j])] <- sum(abs(causes[names(
         causes) == causes_name[j]]))
    }
  }

  return (list(A_,W))

}
```

Listing B.5: Group Lasso Granger Causality Analysis

B.6 MLP Group Lasso-GC Analysis

The following Listing shows the implementation of MLP group Lasso-GC analysis in Chapter 4 with python

```
import pandas as pd
import numpy as np
import copy
import torch
import torch.nn as nn
from sklearn import metrics
from torch.autograd import Variable

class MLPGCAnalyzer(NeuralGCAnalyzer):

    """ ---- initial MLPGCAnalyzer ---- """
    "␣#␣normalize␣time␣series␣(z-score)"
    "␣#␣prepare␣train␣data␣"
    "␣#␣set␣up␣network␣for␣each␣time␣series␣"

    def __init__(self, S, d):

        self.S = S # analyzed time series
        self.d = d # directory name

        # normalize time series
        self.S = self.normalize()

        # prepare train data
        self.N, self.D = self.S.shape # N: length, D: dimension
        self.lag = 5 # network considered lag
        X_train, Y_train = self.format_ts_data()
        self.X_var = Variable(torch.from_numpy(X_train).float())
        self.Y_var = [Variable(torch.from_numpy(Y_train[:, target
           ][:, np.newaxis]).float()) for target in range(self.D)]

        # set up network for each time series
        self.hidden_units = 10
        self.nonlinearity = 'sigmoid'
        self.sequentials = [self.setNetwork(self.hidden_units, self
           .nonlinearity) for _ in range(self.D)]
        self.sequential_copy = copy.deepcopy(self.sequentials[0])

        # initialize parameters for calculating loss function
        self.loss_fn = nn.MSELoss() # loss function
        self.weight_decay = 0.01  # weight_decay for ridge penalty
        self.lam = 0.1 # weight decay for lasso penalty


    def normalize(self):
        S_centered = self.S - np.mean(self.S, axis = 0)
        sigma = np.sqrt(np.var(self.S, axis = 0))

        return np.divide(S_centered, sigma)

    def format_ts_data(self):
        N_train = self.N - self.lag
```

```
        X_train = np.zeros((N_train, self.D * self.lag))
        Y_train = np.zeros((N_train, self.D))

        for t in range(self.lag, self.N):
            X_train[t - self.lag, :] = self.S[range(t - self.lag, t
                ), :].flatten(order = 'F')
            Y_train[t - self.lag, :] = self.S[t, :]

        return X_train, Y_train

    def analyze(self):

        verbose = True

        nepoch = 1000 # number of training epochs
        loss_check = 50 # interval for checking loss
        nchecks = max(int(nepoch / loss_check), 1)

        # Prepare for training
        train_loss = np.zeros((nchecks, self.D))
        train_objective = np.zeros((nchecks, self.D))
        counter = 0
        improvement = True
        epoch = 0

        # Begin training
        while epoch < nepoch and improvement:
            improvement = self.train()

            # Check progress
            if (epoch + 1) % loss_check == 0:
                # save results
                train_loss[counter, :] = self._loss()
                train_objective[counter,:] = self._objective()

                # Print results
                if verbose:
                    print('----------')
                    print('epoch %d' % epoch)
                    print('train loss = %e' % np.mean(
                        train_objective[counter, :]))
                    print('----------')

                counter += 1

            epoch += 1

        if verbose:
            print('Done training')

        weights = self.get_weights()
        weights_est = [np.linalg.norm(np.reshape(w, newshape = (
            self.hidden_units * self.lag, self.D), order = 'F'),
            axis = 0) for w in weights]

        return weights_est


    def train(self):
```

```
        "calculate␣loss"
        loss = self._loss()
        ridge = self._ridge()

        total_loss = sum(loss) + sum(ridge)

        "␣calculate␣lasso␣penalty␣"
        penalty = self._lasso()

        [net.zero_grad() for net in self.sequentials]
        total_loss.backward()

        "␣line␣search"
        t = 0.9
        s = 0.8
        min_lr = 1e-18

        # Return value, to indicate whether improvements have been
            made
        return_value = False

        # a new network
        new_net = self.sequential_copy
        new_net_params = list(new_net.parameters())

        # Set up initial learning rate (step size t(k)), objective
            function value to beat
        self.lr = 0.001
        for target, net in enumerate(self.sequentials):

            original_objective = loss[target] + ridge[target] +
                penalty[target]
            original_net_params = list(net.parameters())


            while self.lr > min_lr:
                # Take gradient step in new params
                for params, o_params in zip(new_net_params,
                    original_net_params):
                     params.data = o_params.data - o_params.grad.
                         data * self.lr

                # group lasso -> Apply proximal operator to new
                    params (update params)
                self.prox_operator(new_net_params[0])

                # Compute objective function using new params
                Y_pred = new_net(self.X_var)
                new_objective = self.loss_fn(Y_pred, self.Y_var[
                    target]) # lasso
                new_objective += self.weight_decay * torch.sum(
                    new_net_params[2]**2) #ridge
                new_objective += self.lam * self.apply_penalty(
                    new_net_params[0]) # lasso

                diff_squared = sum([torch.sum((o_params.data -
                    params.data)**2) for (params, o_params) in zip(
                    new_net_params, original_net_params)])
```

```
                diff_squared = diff_squared.float()

                if new_objective.data.numpy() < original_objective.
                    data.numpy() - t * self.lr * diff_squared.data.
                    numpy():
                     # Replace parameter values
                     for params, o_params in zip(new_net_params,
                         original_net_params):
                          o_params.data = params.data

                     return_value = True
                     break

                else:
                     # Try a lower learning rate
                     self.lr *= s

            # Update initial learning rate for next training
                iteration
            self.lr = np.sqrt(self.lr * self.lr)

        return return_value

    def get_weights(self, p = None):
        if p is None:
            return [list(net.parameters())[0].data.numpy().copy()
                for net in self.sequentials]
        else:
            return list(self.sequentials[p].parameters())[0].data.
                numpy().copy()

    def _loss(self):
        "␣calculate␣loss␣(MSE)"
        Y_pred = [net(self.X_var) for net in self.sequentials]
        return [self.loss_fn(Y_pred[target], self.Y_var[target])
            for target in range(self.D)]

    def _ridge(self):
        "␣calculate␣ridge␣penalty␣"
        return [self.weight_decay * torch.sum(list(net.parameters()
            )[2]**2) for net in self.sequentials]

    def _lasso(self):
        "calculate␣lasso␣penalty"
        return [self.lam * self.apply_penalty(list(net.parameters()
            )[0]) for net in self.sequentials]

    def _objective(self):
        loss = self._loss()
        ridge = self._ridge()
        penalty = self._lasso()
        return [l + p + r for (l, p, r) in zip(loss, penalty, ridge
            )]

    def setNetwork(self, hidden_units = 10, nonlinearity = 'sigmoid
        '):
        net = nn.Sequential()
```

```
        "input␣layer␣->␣hidden␣layer:␣hidden_units*[D*lag]␣+␣
            hidden_units"
        net.add_module('fc', nn.Linear(self.D*self.lag,
            hidden_units, bias = True))

        "activation␣function"
        if nonlinearity == 'relu':
            net.add_module('relu', nn.ReLU())
        elif nonlinearity == 'sigmoid':
            net.add_module('sigmoid', nn.Sigmoid())
        elif nonlinearity is not None:
            raise ValueError('nonlinearity␣must␣be␣"relu"␣or␣"
                sigmoid"')

        "hidden␣layer␣->␣output␣layer:␣1*hidden_units␣+␣1"
        net.add_module('out', nn.Linear(hidden_units, 1, bias =
            True))

        return net

    def apply_penalty(self, W):
        group_loss = [torch.norm(W[:, (i * self.lag):((i + 1) *
            self.lag)], p = 2) for i in range(self.D)]
        total = sum(group_loss)

    def prox_operator(self, W):
        '''
            Apply prox operator
        '''
        C = W.data.numpy()
        h, l = C.shape
        C = np.reshape(C, newshape = (self.lag * h, self.D), order
            = 'F')
        C = self._prox_update(C)
        C = np.reshape(C, newshape = (h, l), order = 'F')

        W.data = torch.from_numpy(C)

    def _prox_update(self, W):
        '''
            Apply prox operator to a matrix, where columns each
                have group lasso penalty
        '''
        norm_value = np.linalg.norm(W, axis = 0, ord = 2)
        norm_value_gt = norm_value >= (self.lam * self.lr)

        W[:, np.logical_not(norm_value_gt)] = 0.0
        W[:, norm_value_gt] = W[:, norm_value_gt] * (1 - np.divide(
            self.lam * self.lr, norm_value[norm_value_gt][np.newaxis
            , :]))

        return W
```

Listing B.6: MLP Group Lasso Granger Causality Analysis