Preserving Performance in Anomaly Detection Models for Real-Time Univariate Streams

Master Thesis

Natalia Karpova

TU Delft

Preserving Performance in Anomaly Detection Models for Real-Time Univariate Streams

Master Thesis

by

Natalia Karpova

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday July 4, 2022 at 11:30 AM.

Student number:	5243424	
Project duration:	September 1, 2021 – July 4, 2022	
Thesis committee:	Associate Prof. dr. ir. J. A. Pouwelse,	TU Delft, (Chair)
	Prof. Dr. J. S. Rellermeyer,	Leibniz University Hannover and TU Delft, (Supervisor)
	Assistant Prof. dr. L. M. da Cruz,	TU Delft
	Msc. L. Poenaru-Olaru,	TU Delft, (Supervisor)

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

This thesis research was conducted in collaboration with ING Netherlands and concluded my two-year experience as a master's student at TU Delft. I would like to sincerely thank my supervisors, Prof. J. S. Rellermeyer and L. Poenaru-Olaru, for providing immeasurable support and teaching me to structure my research approach during the project span. I wish to thank my ING team leads, Pinar Kahraman and Evert Van Doorn, as well as the whole AIOps I3 team for helping me throughout the year with shaping my research ideas, collecting relevant data, and for providing an amazing working environment. I would also love to thank all my friends and family for their constant moral support and encouragement.

Abstract

Anomaly detection has gathered plenty of attention in the previous years. However, there is little evidence of the fact that existing anomaly detection models could show similar performance on different streaming datasets.

Within this study, we research the applicability of existing anomaly detectors to a wide range of univariate streams. We identify main dependent factors with time series that might influence the difference in performances of popular anomaly detection models across different streams, namely, time series features, data drifts, and disorder. We explore the effects that each of the dependent factors has on the performance of selected anomaly detectors. Based on our findings, we propose an adaptive threshold technique that monitors time series disorder. This technique can be integrated into built-in threshold of various anomaly detectors. We show the usability of the proposed method in improving the performance of the models on selected datasets.

Contents

List of Figures

List of Tables

I

Preliminaries

1 Introduction

1.1. Problem Description

Anomaly detection is a process of finding outliers or abnormalities within data. Since many modern services work with big data and rely on (semi)automatic techniques for data processing, automated anomaly detection has become a hot topic in the IT community in recent years. Modern services, as banks, often work with big data streams that require real-time processing and monitoring of the performance metrics or other relevant indicators. As the consequence of the above, one of the booming branches of anomaly detection services is incident analysis in real-time data streams.

Many research efforts have been done in the direction of analyzing online data streams and real-time incident detection. LinkedIn [2], Yahoo [48], Microsoft [72], and Facebook [4] - all these industries conduct own research projects in online anomaly detection and proposed own detective models. Usually it is extremely resource-consuming to obtain anomaly labels of data streams[72], and the above mentioned frameworks prioritize the usage of so-called unsupervised methods in which machine learning algorithms do not require any ground truth labels for training. In this work we as well concentrate our attention on unsupervised methods for anomaly detection in streaming data due to its particular importance in monitoring automation.

Albeit there has been a high interest to online anomaly detection in recent years with the continuous appearing of newer incident detectors in scientific literature, studies in this area suffer from several issues, from our point of view:

First of all, newly proposed models are not always evaluated in streaming scenario (sliding window scenario) [62], [48], [80] and there exists little to no consensus in the community on the exact evaluation setup for real-time streams.

There is little agreement on what test settings should qualify for near real-time processing: some of the works develop their detectors to predict one-point-ahead anomalies [72], [97] and, therefore, can be considered as real-time detectors. Nonetheless, other works [22] detect anomalies in per-batch scenario, where the detector waits until time points fill in the batch of a certain size, and only then makes anomaly prediction on the whole batch.

Lastly, we found a lack of clarity about the *generalizability* of the proposed models. Generalizability refers to the ability of anomaly detectors to exhibit a comparable performance over different set of streams. Even though there exists only limited amount of publicly accessible data of univariate time series with anomaly labels, many research papers do not test their models on all of the dataset or they rather pick only certain time series from those datasets [39], [72], [97], [56], [62]. As the result, models proposed by different research projects can not be compared over full set of available data.

All of the above issues add obstacles to creating an organized overview of the existing methods, and their applicability across different streaming settings. Our study attempts to tackle this problem by performing a generalizabiltiy study in which we have structured and tested a wide range of unsupervised online anomaly detectors on a large set of public time series data. We researched how well different models adapt to various time series data, and what time series factors might influence generalizability of the detective models. Based on the observations collected during our generalizabiltiy study, we exploited our findings by identifying the obstacles in time series nature that prevent model generalization to different streams. We proposed the ways to tackle identified problems as well as we exploited our findings by suggesting a new adaptive threshold

technique for near real-time anomaly detection applicable to a range of models and data streams. We test our thresholding technique in combination with the best-performing anomaly detectors on three public benchmarks which include a total of 12 datasets. Based on our exploratory analysis, we select a single final online anomaly detector, and we compare the performance of our final model against state-of-the-art alternatives.

1.2. Research Questions

1. Is the anomaly detectors' performance preserved across different streaming scenarios?
2. Which time series factors influence the performance differences?
3. How can we diminish the effects of these factors?

1.3. Contributions

The contribution of this work consists of several parts:

1. This work starts by structuring conventional models for unsupervised anomaly detection in univariate streams into own groupings, and researches their applicability into various online scenarios. We compare performance of anomaly detectors over wide range of datasets in order to identify their generalizability.
2. We analyze possible influences of behaviors and factors found within time series data on the predictive performance of anomaly detectors. We structure possible series factors into 3 main categories, and we research each category separately.
3. By exploiting the results of our analysis, we propose a thresholding technique that can be combined with existing anomaly detectors. We research how the addition of such thresholding technique influence the performance of existing base detectors. We also report the results in comparison to the state-of-the-art predictive models.
4. Finally, we discuss the applicability of conventional evaluation techniques in the domain of online streaming data, and discuss the alternatives.

1.4. Objectives and Constraints

Here we discuss the concrete requirements and constraints in our work:

1. Our work looks solely into univarite type of time series with one dependent variable changing over the flow of time.
2. We work only with completely *unsupervised* anomaly detectors since this class of machine learning techniques requires minimal human intervention into training process and, therefore, more resource-optimal. This also means we do not perform any hyperparameter tuning.
3. In our experiments we test our models in online settings by emulating sliding window scenario over a stream. We test both: single point anomaly detection as well as batch (interval) anomaly detection of the incoming data.
4. Since we are working with online scenario, we strive to keep anomaly predictions as near real-time case, meaning we allow maximum delay of our models to be below 100 timestamps in future. Depending on the dataset granularity, exact window size may vary per case.
5. We are interested by not limiting our research to the use-case of machine metrics. Since anomaly labeling by experts is resource-consuming exercises, there exists only limited amount of publicly available datasets with such types of data, and the true nature of time series is not always known for such datasets. Completely limiting ourselves to only one type metrics would mean that we need to exclude large parts of available data and, therefore, weaken our analysis.

2

Background

2.1. Time series

Time series are series of data points measured over time. These points are indexed over certain periods of time, or lags. Lag is a fixed but not necessarily minimal amount of passing time between time series points.

Univariate time series are represented by a single observation or variable that changes over time, whereas multivariate time series keep track of several time-dependent variables varying over time. This study looks in the case of univarite time series only.

Mathematically, univariate time series can be represented as a collection of observations over period of time. One of the notation that is used for time series analysis is shown in equation 2.1 and represents a time series X that is indexed by natural numbers.

$$X = (X1, X2, ...) \tag{2.1}$$

Another widely used notation is shown in equation 2.2 with T represents time set.

$$Y = (Y_t : t \in T), \tag{2.2}$$

2.2. Time Series Features

Time series differ in their shapes and sizes, and such differences can be quantified via time series features. In general, time series data has core components as seasonality, trend, and cycle. By seasonality, the variation in time series behavior with a period less than a year is referred, whereas more long-lasting variations in behavior are called cycle. Trend represents a moving (increasing or decreasing) pattern in time series.

Below, we describe examples of some well-known time series features.

1. *Autocorrelation* is the similarity between observations as a function of the time lag between them. Autocorrelations shows how different lagged versions of the same time series correlate to each other.

2. *Seasonality* refers to the periodic fluctuations that fit below one year interval. For instance, time series of electricity consumption will be higher over colder months of the year showing seasonal pattern.

3. *Stationarity*, on its turn, refers to the case when characteristics of time series do not change over time but stay fixed. Stationarity is usually determined by measuring mean, variance and covariance of a time series over some period: if these values stay steady, then time series is said to be stationary.

4. *Trend* represents general incline of data over higher or lower values. Often, trend is associated with slope of data that is not disrupted by seasonal behavior.

Graphical example of time series with various properties are shown in Figure 2.3.

Furthermore, as many as thousands of different metrics can be extracted from time series data with the newer properties appearing over time [32]. Such features may include some base properties as kurtosis, or more novel metrics as, for instance, Wang periodicity measure. [92]

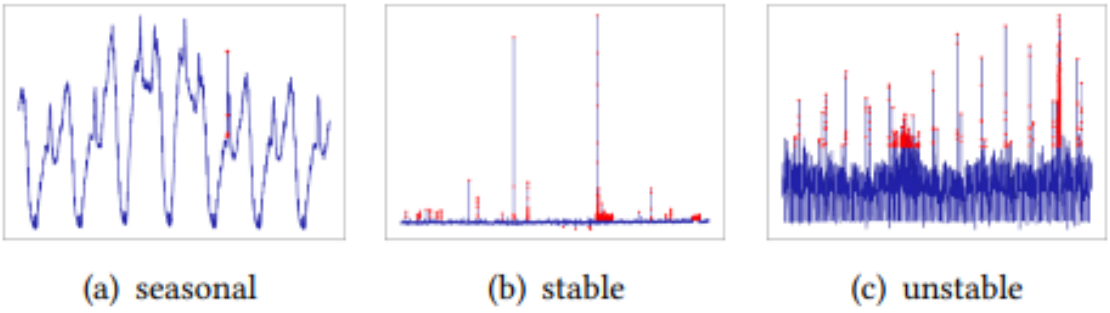


Figure 2.1: Different types of time-series. [72]

2.3. Univariate Streaming Data

Data streams are the flows of data points (single point per timestamp in case of univariate streams) with new observations appearing gradually over time usually (but not always) with fixed time intervals. Distance between adjacent time series points is called time series granularity. Time series granularity is use-case specific and can vary from milliseconds to days, month or years. For instance, when talking about data streams in medical equipment such as cardiology monitoring software [5] or machine metrics monitoring tools such as CloudWatch [1], time series granularity usually falls below 1 hour interval since it is crucial for these systems to get prompt observations and react timely. On the other hand, for tracking trends such as admissions of patients to intensive care [8] or stock market analysis on a long distance [6], granularity of days, weeks or month might be used.

The main challenges in working with such kind of data is that new observations appear gradually over time and it is impossible to have a look in future to collect missing data. In addition, since the process happen in real time, software tools must be fast enough to perform given task within time interval before getting new observation. Lastly, data might change its behaviour or its structure over time, consequently, arising the need for software tools that work with this data to be capable of adjusting themselves to the incoming flow changes in real time.

2.4. Anomalous Data

Generally, anomalies are considered to be the parts of data that do not fit into expected or accepted behavior as judged by the previously collected instances. Due to complexity of identifying what does and what does not fall under normal behavior of data, in many cases anomaly identification is subjected to human expertise.

On a higher level, it is widely accepted to classify anomalies (including time series anomalies) into three major groups [79] that are sketched in Figure 2.2 and explained below:

1. *Point or Global Anomalies*: anomaly is represented by a single point within a time interval which behaves differently in relation to the rest of time series points.

2. *Collective Anomalies*: anomaly is represented by a group of consecutive points with unexpected behavior in relation to the rest of time series.

3. *Contextual Anomalies*: anomaly can be a combination of single-point and group anomalies that have similar behavior to the rest of the time series but considered to be abnormal under specific circumstances or in a particular context.

Point or global anomalies represent the simplest type of anomalies [45] and they are researched more broadly in a scientific society. Contextual anomalies are more complex types of anomalies due to the fact that their normality depends upon underlying data knowledge or context. For example, a fluctuation of daily temperature around 30°C is not considered to be abnormal in the world unless such temperature is recorded in Canada in winter period. An example of contextual anomaly within time series is shown in Figure 2.3.

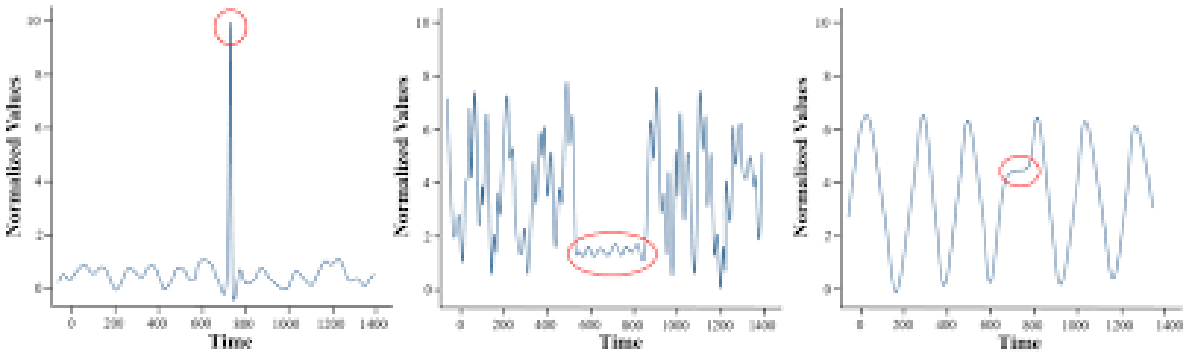


Figure 2.2: Three main types of anomalies: point (global) anomaly (left), collective anomaly (middle), contextual anomaly (right) [51]

Contextual anomaly behavior is induced by the structure of time series that has been observed before. Such structure can be define via the attributes of 2 types [27]:

Point and collective anomalies are determined using the values of behavioral attributes whereas, in order to identify contextual anomalies, contextual attributes are used. There is a rising interest to the detection of contextual anomalies, especially in the time series data [14], [56], [62].

1. *Contextual Attributes* are described as the context or neighbourhood of a time series point: it is time coordinate that determines the position of corresponding point in a time series.

2. *Behavioral Attributes* are non-contextual characteristics of a time point. For example, monthly temperature in winter in Amsterdam, Netherlands describes normal behavior of temperature data.

Anomaly classifications in time series are not limited to the three categories from [79]. More recent studies proposed to distinguish between dependent time series variable type (categorical or numerical) as well as to classify anomaly based on how its behavior corresponds to the rest of time series [31]. Nevertheless, such classification rely heavily on human expertise and, to the best of our knowledge, has little applicability into unsupervised use case.

2.5. Anomaly Detection

The process of anomaly detection strives to identify unexpected and undesired behavior in data. It can be defined as below [72]:

Given a sequence of time series real-valued points $\boldsymbol{x}=(x_1, x_2, ...)$*, the aim of time series anomaly detection is to produce the output* $\boldsymbol{y}=(y_1, y_2, ...)$ *of equal length in which* y_i *denotes the anomaly label for time series point* x_i *at time i with* $y_i \in 1, 0$*.*

Generally, automated methods for anomaly detection rely on statistical or machine learning techniques. Depending on availability of labels in data, anomaly detection methods can be classified into the following categories:

1. *Supervised Anomaly Detection* makes full use of anomaly labels in data. Supervised methods are both trained and evaluated using anomaly labels. Both classes of labeled instance, normal and anomalous, must be provided within the training data in order for supervised methods to be reliably trained. The main challenges of supervised methods are related to the fact that anomalies are far fewer compared to normal points which leads to imbalanced class distribution problems [66]. Secondly, there is a big challenge in obtaining representative labels for abnormal points [27]. Several techniques are present that tackle this problem by injecting artificial anomalies into the normal data streams, yet it is still not known how representative artificial anomalies are of a class of real-world anomalies in a particular time series.

2. *Semi-supervised Anomaly Detection.* Semi-supervised methods require only normal instances to have a label leaving anomalous points out. In addition, there exists only limited set of semi-supervised meth-

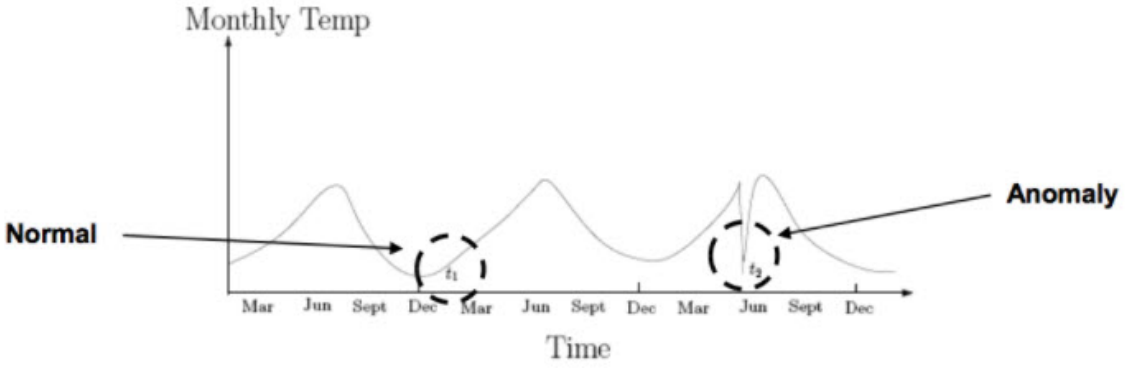


Figure 2.3: Contextual anomaly t2 in a temperature time series. Temperature at time t1 is same as that at time t2 but occurs in a different context and hence is not considered as an anomaly. [27]

ods that requires anomalous instances to be labeled for training [28]. Due to smaller amounts of labeling labour required for creating such models, they are generally more widely applicable in industry.

3. *Unsupervised Anomaly Detection.* This class of techniques do not require anomaly labels and, hence, is the most broadly applicable method out of all 3.

Depending under which category an anomaly detection method falls, different techniques for training and anomaly labeling are used. In case of supervised anomaly detection, a typical approach is to build a predictive model that oppose normal and anomalous classes represented by the labeled instances in a training data. Anomaly detector then learns the inner representation of this classes which differs depending on the exact technique that is chosen.

Semi-supervised methods, similarly to supervised methods, construct class representation but only of a single (normal) class. Then such methods are trying to find anomalies in test date basing their predictions on normal class only.

Lastly, many unsupervised methods make an implicit assumption that anomalous data is far less frequent than normal instances and try to distinguish fewer anomalous points based on the characteristics that pop out. If the implicit assumption of the quantity of anomaly points within data is wrong, unsupervised methods typically suffer from high false positive rate. [27].

It is hard to generalize the set of all existing anomaly detection techniques under limited set of categories - the above overview is a general outline of the most widely used base methods in the field. More detailed description and classification of anomaly detectors is given in Section 3.2.

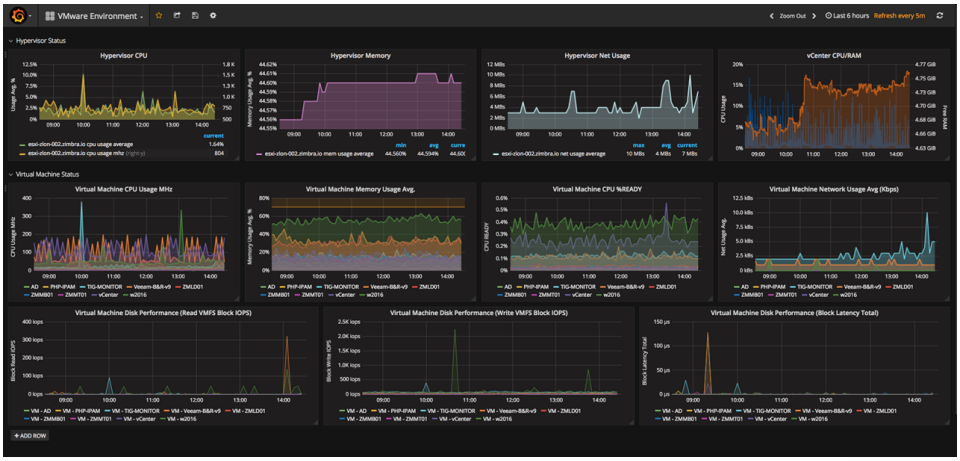


Figure 2.4: Screenshot of a layout of Grafana service.

2.6. Monitoring Services

Streaming data is used in monitoring frameworks that are programmed to track incoming streaming values and perform desired operations in real time on the input values before sending it to an end user. Many automated services require own monitoring routines in order to track performance or health indicators [72]. Monitoring services can simply visualize incoming streams to an end-user, calculate some performance metrics and aggregate values in time, or detect and alert once the behavior of the interest has been encountered. An example layout of monitoring service application is shown in Figure 2.4.

There are several challenges in designing and deploying monitoring services which are currently confronted by the research community, and the solutions are in active research.

1. *Lack of labels.* Anomaly detection requires domain expert to define the notion of anomaly as well as to label example metrics in order to obtain validation data for service creation, tuning and evaluation. Since anomalies are considered to be rare events within data, in order to get reliable evaluation of an end product, it is necessary to have many time points labeled by human experts. This puts an import constraint on monitoring service development in production scenario.

2. *Generalization.* Streaming metrics of various nature and type can be of an interest to an end-user. Building a tailor-made monitoring services per every stream is extremely inefficient and time consuming. In addition, it would require data labels to be provided by human experts per a stream of interest. Therefore, there is a huge demand for generalizable monitoring models that can be applied into various streaming scenarios with no prior optimization.

3. *Efficiency.* Monitoring system should be capable of processing of thousands of different streams in a real time. When the granularity of time series is small (in order of seconds or minutes), what is usually the case with monitoring of business or health metrics, it puts important challenges and constraints to the development of the monitoring model which should be fast enough to perform all this tasks.

Due to the *Lack of Labels* constraint, unsupervised anomaly detection methods are being studied more broadly. However, the research focus is not limited by unsupervised machine learning techniques due to the complexity of the problem. Yet, in this work we focus on unsupervised methods due to their wider applicability.

3

Related Work

In this chapter we introduce previous works and findings that are relevant to our research domain. We start by explaining modern extraction techniques for time series feature, that we later use in our generalizability analysis (Section 3.1). We then proceed by defining the main categories of machine learning models for anomaly detection: we introduce own groupings of such models based on literature survey, and we provide examples of the most popular methods in each category in Section 3.2.1. In Section 3.2.2, we explain state-of-the-art machine learning methods for anomaly detection. In the last Section 3.3, we give in-depth overview of the inner workings of our base models. We test and define our base models during the first stage of experiments described in Section 5.3.

3.1. Time Series Feature Extraction

As many as several thousands different time series features can be extracted from time series data. There exists many publicly available services and libraries for time series feature extraction and analysis, with hctsa [32] being one of the most well-known. hctsa is tailored for Matlab and provides more than 7000 properties for time series extraction. Extracting such amounts of features from lengthy time series and performing exploratory analysis upon them is resource-consuming. Therefore, attempts has been made to overcome the necessity of analyzing whole set of features by identifying the most influential and representative ones. Several studies provide reduced sets of time series features for faster and less-expensive time series analysis [48], [93], [60]. Not rarely, studies then analyse correlations between extracted time series properties and the performances of anomaly detectors on the corresponding time series. For instance, Wang et. al. made an attempt to induce rules for selecting the most optimal time series forecasting model based on time series features such as trend, seasonality, periodicity, serial correlation, skewness, kurtosis, non-linearity, self-similarity, and chaos [93]. Lately, Yahoo research group based their EGADS systems for anomaly detection within time series [48] on the features derived in Wang et. al. research.

Usually, extraction of a subset of representative features is done manually and is based on one's own expertise [61], [91]. Yet, there exist sets of prominent features extracted in alternative ways [54]. In this study, we decided to focus on feature sets proposed by 2 different extraction approaches, namely, FFORMA approach and Catch22 approach that are described in depth below.

3.1.1. FFORMA Approach

FFORMA is a complete system for anomaly detection that make use of time series features. Features are extracted as a preprocessing step and then they are exploited during prediction phase [60]. FFORMA includes ensemble of meta learners to predict anomalies and uses their weighted combination based on extracted features. We use only preprocessing part of the system to compute time series features. The total amount of FFORMA features is 42 and listed in Appendix 18. These features are subjective to manual selection and derived from the lists proposed by Hyndman et. al. in 2015 [41] and Talaga et. al. in 2018 [86].

3.1.2. Catch22 Approach

Catch22 approach starts by analyzing all dynamic time series properties present in hctsa package [54]. It then finds principal components, or the most representative groups of features, among the complete set of

hctsa features. The reduced set of time series features is extracted by analyzing performance of classification models on a large set of time series classification tasks. In such settings, the most representative features tend to exhibit strong correlation to classification performance across the given collection of time series problems. The final list of features is present in Appendix 18. Later study confirms that Catch22 extracts features with the lowest redundancy as compared to other available feature extraction packages on a different classification task [36]. The total amount of Catch22 features is 22, and they are listed Appendix 18.

3.2. Machine Learning Techniques for Anomaly Detection in Time Series

Several previous studies [87], [57], [64] have proposed classifications of machine learning models for anomaly detection into definitive groups. Usually such classifications share many categories in common, as for instance, deep learning methods or ETS methods. However, different taxonomies do not completely agree on the final set of classifications or they instantiate same machine learning methods under different categories. In addition, such classifications do not always include more recent state-of-the-art techniques. For this reason, we decided to create own grouping of machine learning methods for anomaly detection in univariate time series. Our grouping still shares many similarities with already available ones.

3.2.1. Grouping of Machine Learning Methods for Anomaly Detection

1. **Distance-based methods** - are machine learning methods that evaluate data points based on other data points that surround them. Anomaly is detected based on dissimilarity to majority of data points in the surrounding. Some well-known methods include:

   (a) *Local Outlier Factor* makes use of local density deviation (local reachability density) of a given data point with respect to its surrounding. [23] [88]

   (b) *kNN* density estimation is based on distance between a given data point and its neighbouring points. [100] [88]

2. **Clustering methods** - are methods for grouping similar data points. Clustering algorithms strives to keep points with the similar properties within the same cluster while maximizing distance between individual clusters. Data points that are left unassigned are treated as anomalies. Some well-known methods include:

   (a) *One-class SVM* is a classifier that identifies normal class of data points by finding a function that is positive for the regions with high density of normal points, and negative otherwise. Any data point falling into the 'negative' region is considered to be an anomaly. [46]

3. **ETS methods** analyze time series' error, trend, and seasonality to build statistical representation of it and detect anomalies based on own forecast of time series behavior.

   (a) *ARIMA* a model applicable for time-series analysis in order to model / predict time series points and is based on previously recorded time-series values. An example use of ARIMA for anomaly detection can be found in [59].

4. **Isolation methods** relies on the fact that anomalies are easier to separate from the rest of the data and, hence, attempts to identify and isolate outliers from real points instead of learning the behavior of the normal data. Generally, outliers are found by data partitioning: the main assumption of such methods is that outliers needs less data partitioning in order to be isolated from the normal data points. The first and the most well-known method is:

   (a) *Isolation Forest* explicitly isolates anomalies by recursively generating data partitions with binary trees and a randomly selected attribute [52] [70]

5. **Saliency methods** are based on log-amplitude spectrum of input data. Various methods include tweaking saliency methods by ,for example, taking only phase spectrum of data for efficiency reasons [38].

   (a) Spectral Residuals analyzes log-amplitude spectrum of the incoming stream. For every input data point, its spectrum is compared against average log-amplitude spectrum of the normal data. Consequently, anomalies are inferred based on high deviation from mean spectrum values. [72]

6. **Deep learning methods** include various deep neural architecture to tackle anomaly detection problems. Different models can be based, for instance, on deep feature extraction, learned signal representation, or end-to-end anomaly score learning [65].

   (a) *Autoencoder* is a neural network that reconstructs incoming signal through several hidden layers. Therefore, its output signal is a reconstruction of the most important features of the true input signal. Data points that do not fit into the Auto-Encoder output are treated as outliers [50].

   (b) *Replicator Neural Networks (RNN)* is a neural network that compresses incoming data and calculates average reconstruction error over all data features (outlying factor of the data). The outlying factor acts as a measure for detecting anomalies. [85].

7. **Combinations of methods and ensembles** - anomaly detection methods that combine several different approaches for anomaly detection in sequential or joint fashion.

   (a) *FFORMA* is an ensemble of various machine learning methods where the final label is based on the weighted labels produced by the FFORMA ensemble. Weights of the weak learners within the ensemble are assigned based on the properties of the input data and their relation to predictive models in the ensemble. [60]

3.2.2. Novel methods

In recent years there has been created numerous novel unsupervised methods for performing anomaly detection in time series, yet many of them either not directly applicable or needs modifications in the algorithm in order to be performed in online scenario. The most widely-known methods with direct integration into data flows are ETS-based methods as (S)ARIMA [99] or Exponential Smoothing [71]. More recent academia methods often make use of neural-network approaches such as Long Short-Term Memory Auto-Encoders (LSTM AE), Variational Auto-Encoders (VAE), and Convolutional Neural Networks (CNN). For example, Donut model [97] is based on VAE, and Spectrual Residuals model from Microsoft uses CNN to automatically set anomaly threshold [72]. The most recent CNN-based method for incident detection is DeepAnt [62] which uses CNN as a core of its time series predictor module. Additionally, LSTM layers can also be used as a core of DeepAnt architecture. The main drawback of using such models is related to their computational complexity and time required for retraining.

On the other side, recent anomaly detection libraries and tools proposed by the industry research often includes variety of methods for joint use as an ensemble. Not rarely, such collections include ETS-based or Neural Networks methods, nevertheless, they are not limited just by ETS and NN methods. For example, EGADS from Yahoo [48] make use of ensemble in its core. EGADS was proposed in 2015 and includes ETS weak learners (ARIMA and Exponential Smoothing) in its setup as well as other techniques as Moving Average Model.

Other recent anomaly detection models include POT/DSPOT methods that are based on Extreme Value Theory [80]. Less-known methods with direct application to data streams and a neural methods in its core include Extreme Learning Machines approaches [10]. Extreme Learning Machines may be combined with other techniques in order to improve their performance: for example, Particle Swarm Optimization is used in [63] or Grey Wolf Optimization [10] for model tuning.

Lastly, in 2021 Evolving Spiking Neural Networks were applied to the domain of anomaly detection in streaming data [56]. This methods uses time series encoding into evolving spikes representation in order to detect anomalies.

3.3. Selected Models

In this section we describe in-depth the heuristics behind the anomaly detection process of the four different anomaly detectors. These detectors were selected after initial part of experiment reported in Section 5.3 and are used in all of our tests.

3.3.1. SARIMA

Seasonal Autoregressive Integrated Moving Average, or SARIMA, is a widely used as statistical method for model forecasting. SARIMA is an ARIMA model with seasonal component. ARIMA captures different standard temporal structures in time series data. ARIMA model has three parameters, namely, p, d, q:

$$ARIMA(p,d,q) \tag{3.1}$$

p refers to autoregressive component that makes use of the dependent relationship between observation and some number of lagged observations. It is represented by lag order, or the number of lag observations included in the model.

d reflect integrated part. It reflects the number of times a raw observation is differenced. Differencing is the way of making time series stationary by subtracting an observation from a lagged observation (observation at an earlier timestamp).

q is the size of the moving average window. Moving average window moves through time series and calculates average within itself hence smoothing the data.

SARIMA, in addition to ARIMA, has seasonal factor m - the number of timestamps for a seasonal period. As well, it adds three new hyperparameters to specify autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series shown in equation 3.2.

$$SARIMA(p,d,q)(P,D,Q)_m \tag{3.2}$$

Once SARIMA model is built, it is used to forecast and consume incoming observations. Anomaly is detected based on the extent at which new real-world observation deviates from the forecasted one.

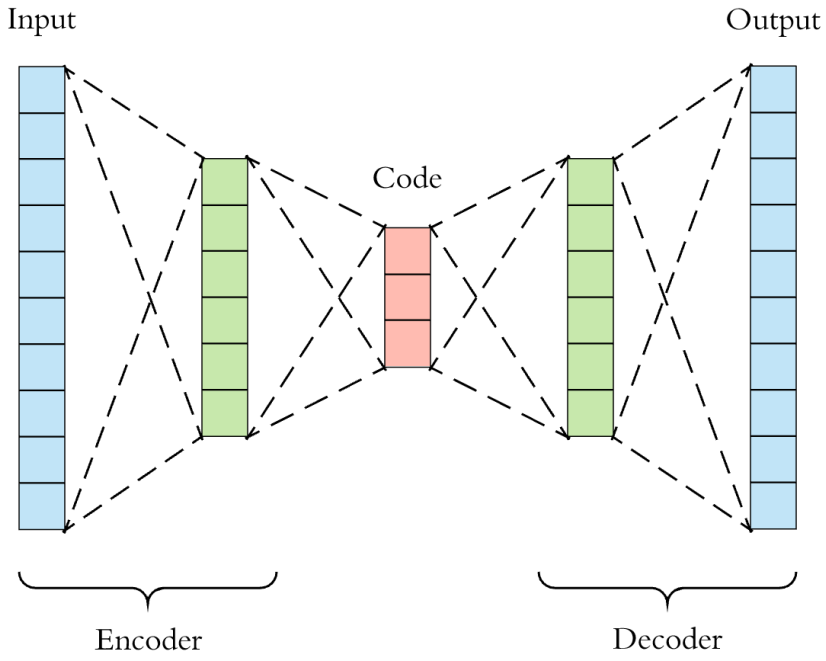


Figure 3.1: Auto-Encoder representation (from TowardsDataScience).

3.3.2. Long Short-Term Memory Auto-Encoder

Auto-Encoder-based neural networks try to learn data representations of its input by encoding them. They strive to learn optimal encoding of the data by using minimal number of parameters. Thereafter, it is possible to reconstruct data by decoding dumped values within the encoded representation. Therefore, anomaly detection technique in this model is based on the assumption that anomalies do not constitute inner or optimal description of the input data and will be lost during encode-decode process. Auto-Encoders are trained on some input signal and learn its behavior, after they are used to forecast time series value based on learned representation. If the forecast deviates significantly from actual observed value in time series, such value is considered to be anomalous since it does not correspond to the learned data encoding.

Auoencoders consist of two main parts: encoder layer that maps the input into code, and decoder that decodes it back (shown in Figure 3.1).

Long Short-Term Memory (LSTM) refers to the design of neural network structure. It is a subcategory of recurrent neural networks (RNN) class which have a feedback connection from the last network layer to the

first network layer. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate and is shown in Figure 3.2.

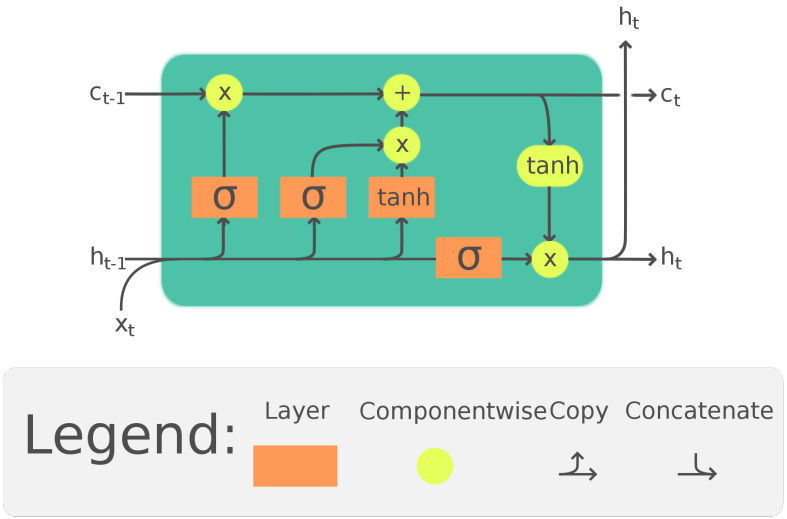


Figure 3.2: LSTM cell visualization (from [9])

RNNs are trying to keep track of arbitrary long input sequences. For this reason, they are profitable to be used within time series forecasting context since they remember previous, periodical time series structure and forecast new values based on their memory.

Input x_t from timestamp t arrives into LSTM memory cell and gets gated via input, output and forget gates with previous values. h_{t-1} represents hidden state learned by LSTM from the previous timestamp, whereas c_{t-1} defines previous cell state. Input values get concatenated with the previous hidden state in order to generate relevant values at particular time point and nullify irrelevant ones. LSTM network specifically trained to learn such hidden representations h_{t-1} to represent relevant-irrelevant values at each step.

The outputted values then are multiplied point-vise with the previous cell state. This action nullifies irrelevant components from previous state cell and the process is called forget gate.

The same outputted values from previous step ate tanh-activated to generate new memory vector that represents the extent of the updates each component of LSTM cell needs given newly incoming data. In addition, sigmoid-activated network represents input gate where it identifies which parts of new memory vector worth updating at all.

Lastly, output gate determines the new hidden state which will be fed to the next iteration by multiplying point-wise learned tanh-activated c_{t-1} state after input and forget gates with combined input and hidden values from the step.

The steps above are repeated many times and, as the very last step, linear layer is applied in order to convert hidden state from the previous step into actual forecast.

LSTM Auto-Encoders use LSTM layers to encode-decode incoming data.

3.3.3. Spectral Residuals

Spectral Residuals algorithm is a novel method that has been proposed by Microsoft in 2019 and relies on Fourier Transformation of incoming signal [72].

Fourier Transform is a mathematical transformation that decomposes function depending on space and time variables [21]. It transforms periodical signal from time domain to frequency domain and gives an insight into what frequencies occur within a signal. An example of Fourier transformation from time domain to its frequencies from [7] is shown Figure 3.3.

Mathematically, Fourier transform of a function is given in the below equation 3.3. Here, $X(\omega)$ is a function of frequency, x(t) is original input function, and ω denotes root of unity.

$$X(\omega) = \int_{-} x(t) e^{-j\omega t}\, dt \tag{3.3}$$

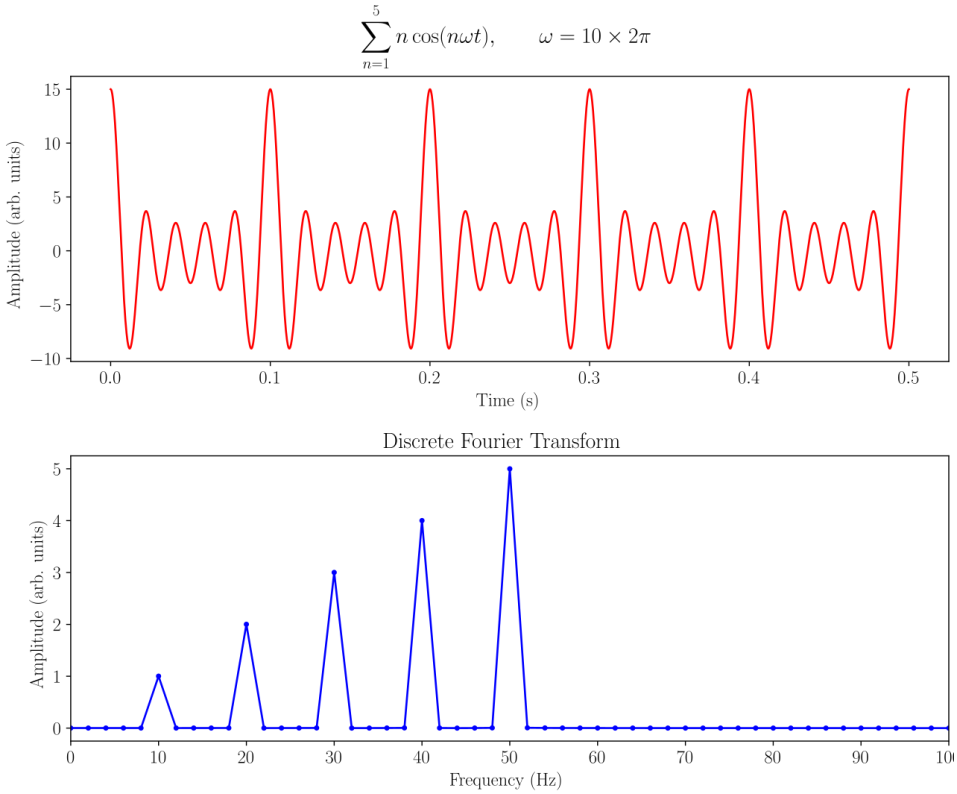


Figure 3.3: Fourier transform of cosine summation function $\sum_{i=1}^{5} ncos(nwt)$ (from Wikipedia)

The three main steps of Spectral Residuals algorithm for anomaly detection are shown below:

1. Fourier Transform to get a log amplitude spectrum.
2. Calculation of spectral residuals of the transformed signal.
3. Inverse Fourier transform to revert sequence back to spatial domain.

Given a time series sequence **x**, mathematical representation of the above algorithm is shown in equations 3.4 to 3.11, with F denoting Fourier transform and F^{-1} denoting inverse Fourier transform.
A(f) is the amplitude spectrum of the input signal **x** shown in Equation 3.4:

$$A(f) = Amplitude((F(\mathbf{x}))) \tag{3.4}$$

P(f) is the phase spectrum of the input signal **x** shown in Equation 3.5:

$$P(f) = Phase((F(\mathbf{x}))) \tag{3.5}$$

L(f) is the Log representation of the amplitude spectrum A(f) shown in Equation 3.6:

$$L(f) = log(A(f)) \tag{3.6}$$

With AL(f) being the average spectrum of L(f) calculated via convolution on input log representation L(f) and convolution matrix $h_q(f)$ shown in Equation 3.7:

$$AL(f) = h_q(f) * l(f) \tag{3.7}$$

R(f) represents the spectral residuals, or the log spectrum L(f) subtracting the averaged log spectrum AL(f) shown in Equation 3.8. It serves as a compressed representation of the initial input signal

$$R(f) = L(f) - AL(f) \tag{3.8}$$

Lastly, sequence is transformed back to spatial domain and a saliency map of the input signal is produced (example shown in Equation 3.11):

$$S(\mathbf{x}) = \left|\left|F^{-1}(exp(R(f) + iP(f)))\right|\right| \tag{3.9}$$

After generating a saliency map, the threshold on the allowed values with the map can be set in order to obtain anomaly labels: everything with a saliency value falling below this threshold is considered to be within normal deviation of a signal whereas everything exceeding this threshold is considered to be an anomaly. Graphical representation of saliency map generation and consequent anomaly detection of a time-dependent signal is shown in Figure 3.4.

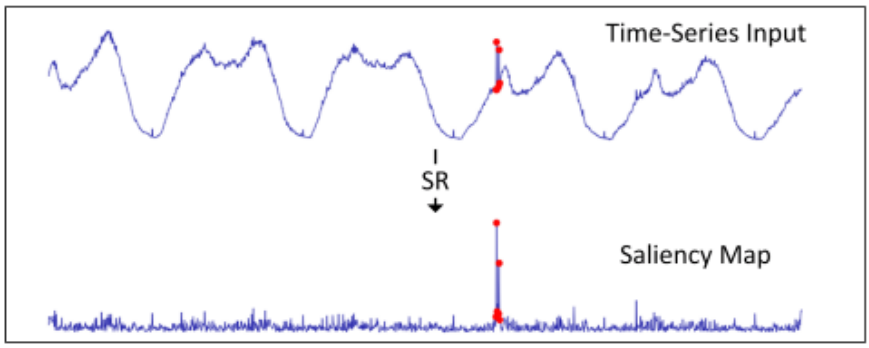


Figure 3.4: Example of SR model results (from [72])

In order to obtain real-time predictions, the above process is applied in sliding window scenario with only limited set of newly incoming time series points being labeled at particular step. According to [72], optimal window size is subjected to input data and may vary from 64 to 1440 based.

3.3.4. Particle Swarm Optimization of Extreme Learning Machines

Extreme Learning Machines
Extreme Learning Machines (ELM) are feed-forward neural networks that usually contain one hidden layer. Feed-forward neural networks are networks of consequent layers where the data gets fed into the initial layer and passes then through every next layer until it reaches very last, output layer. Parameters of hidden layers of extreme learning machines do not need to be tuned such that the hidden nodes are assigned with some values randomly in the beginning and never change these values afterwards [40]. Due to this, ELM does not work with gradient-based backpropagation to update hidden layers, but rather use Moore–Penrose inverse [77] for updating weights of the output layer. The purpose of such architecture lies in generalization and speed of the performance.

Particle Swarm Optimization
Particle Swarm Optimization (PSO) is an optimization technique that provides approximate solution for a given problem of finding global minimum/maximum [69]. It works by choosing randomly fixed set of particles on a search grid and then approximating the best solution on a grid by moving every particle within a swarm towards the most promising direction. Such direction is tailored per particle and is based on a combination of general best-seen solution so far by a swarm all together and by particular particle on its own.

PSO algorithm starts with initializing the set of particles at a certain position each with a certain velocity. Position as the velocity variable changes over time per particle. If we have N particle in our algorithm then one iteration of this algorithm will update position of some particle i via the following formula:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{3.10}$$

as well as its speed as shown below:

$$V_i(t+1) = wV_i(t) + c_1 r_1(part\_best_i - X_i(t)) + c_2 r_2(gen\_best - X_i(t)) \tag{3.11}$$

Here, $X_i(time)$ denotes the position of particle i at time $time$, and $V_i(time)$ denotes velocity of particle i at time $time$. r_1 and r_2 are two random numbers $\in$ [0.0, 1.0]; w, c_1 and c_2 are the hyperparameters of the PSO algorithm; $part\_best_i$ is the best found position by current particle i, whereas gen_best is the best found position by all particles in the swarm. Graphical representation of overall idea behind particle swarm optimization is pictured in Figure 3.5

The combined model of PSO with ELM uses swarm optimization in order to find the weights of ELM connections between the input and the first hidden layer [63] (the weights that remains fixed during ELM training routine). In addition, PSO-ELM model relies on the assumption that if particle swarm has converged on a particular solution than it has learned the concept of the problem. Therefore, when the new concept gets introduced into input data, the average error of PSO-ELM will change [33]. Thus, drift adaptation is introduced into the model by monitoring average errors of the particles and their distribution. In addition, average error values over time are smoothed via the Exponentially Weighted Moving Average (EWMA) [55]. Once particle errors deviates significantly from the distribution observed in the training part, the drift is detected and the adaptation/retraining happens.

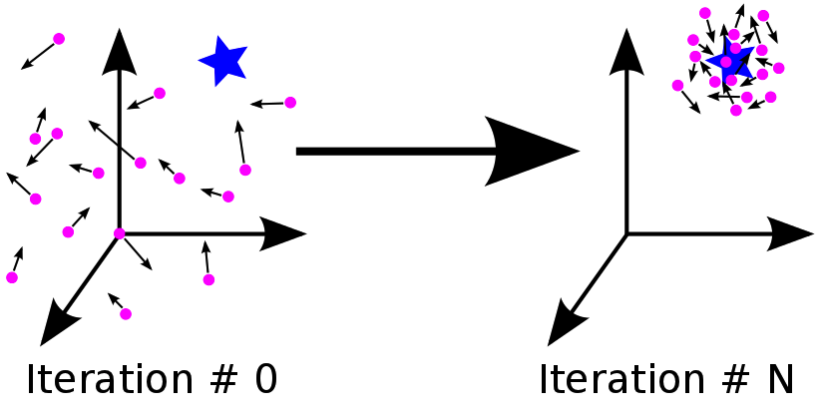


Figure 3.5: Graphical representation of particle swarm optimization algorithm (from Medium)

3.4. Drift Detection in Time Series

3.4.1. Drifted Data

Generally, data drifts are referred as changes in input data that leads to models' performance degradation. Such changes is one of the top reasons to why model accuracies decrease over time. Target concept of data streams may change making anomalous labels distinct to what have been learned by the initial anomaly detector. Changes in data concepts can emerge for many different reasons including external influence. For example, COVID'19 lockdowns over the world had influence on sale volumes of various items. These lockdowns represent external influence that provokes drifts in consumer buying behavior. Graphical representation of data drift in time series in the described scenario is given in Figure 3.6.

Drifts in data represent normal data behavior that is distinct from the previously observed one. Therefore, generally, drifts would be considered as anomalies by incident detectors which will lead to performance deterioration. Due to this, there is a rising need for model adaptation in online streaming scenarios with some studies already incorporating drift detectors into anomaly detectors [89], [43]. Unfortunately, testing such joint models is challenging since the set of benchmarks of time series with labeled anomalies and the set of benchmarks with labeled drifts are disjoint. However, previous study [95] has marked NAB dataset as having unrealistic densities of anomalies in some of their time series: in NAB some time series have many continuous regions of anomalies or anomalies located too close to each other. This might be an indication of reoccurring concepts within time series that has been labeled as group anomaly. We decided to research the applicability of drift detectors into streaming incident analysis by researching joint performance of anomaly detectors with drift detectors applied to public data.

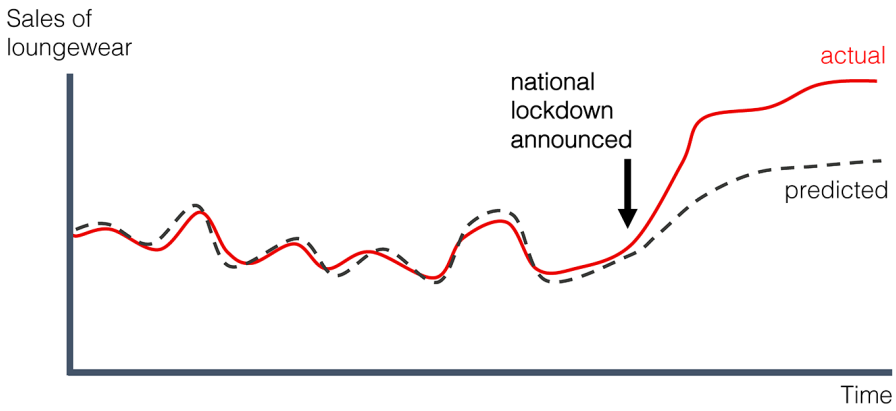


Figure 3.6: An example of concept drift in time series data caused by COVID'19. Normal old behavior is plotted by black dashed line with red line representing true drifted behavior.

3.4.2. Drift Detectors

Drift detectors are methods for identifying drifts in data. Such methods base their judgments upon statistical analysis of data over time: if distribution of some data parameters significantly changes over short period of time, this might be an indication of a drift. However, particular drift detectors base their exact technique upon various strategies. We selected and worked with the family of *error-based* drift detector due to their wide usage within scientific literature. The general technique behind error-based drift detectors is related to comparing models' classification error over various data batches. Data batches, on which model exhibit significantly higher error than what was observed in training dataset, are considered to have drifts. Below we discuss the most popular error-based drift detectors in more details:

1. **Adaptive Windowing (ADWIN)** compares distributions between 2 halves of variable-length windows of recent items by comparing their means. [19]

2. **Drift Detection Method (DDM)** tracks error rate in testing data over various batches. If such error rate hits a certain threshold, drift is detected. [33]

3. **Early Drift Detection Method (EDDM)** uses similar technique as DDM, but keeps track of the running average distance between errors on consequent test batches. [15]

4. **Exponentially Weighted Moving Average for Drift Detection (ECDD)** uses exponentially weighted moving average of model error rate. If error rate deviates significantly from the exponentially weighted error from the previous observations, drift is detected. [75]

5. **HDDA** - relies on both Hoeffding's inequality [37] and moving average to track drifts in data. [20]

6. **HDDW** - similarly relies on Hoeffding's inequality [37] but weighted moving average. [20]

In addition to common drift detectors, there exists many novel drift adaptation techniques tailored to specific use case. For example, Saurav et. al. propose Recurrent Neural Network model with integrated drift adaptation for anomaly prediction over time series [76] where RNN model is capable of training incrementally and adjusting to newly incoming observations. Xu et. al. as well integrates drift adaptation into neural-based model for anomaly detection [98] by prioritizing recent observations over older ones and increasing their corresponding weights while retraining.

There is an increasing use of drift adaptation which provoked an emerging of online libraries for drift detection as River [1] or AlibiDetect [2]. Such libraries offer a set of built-in methods for drift detection as well as stream modelling methods with integrated drift adaptation.

[1] `https://riverml.xyz/`
[2] `https://github.com/SeldonIO/alibi-detect`

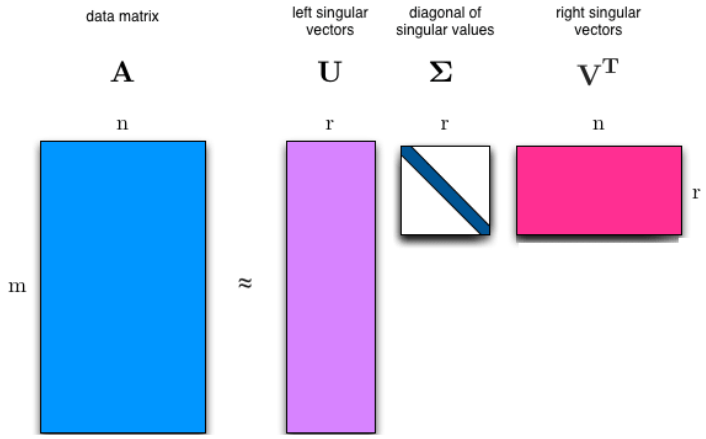


Figure 3.7: Singular Value Decomposition of matrix A (from Medium). Diagonal entries (dark blue) in Σ are singular values of A.

3.5. Time series disorder

When working with real-world data, almost always such type of data includes noise. The main question that arises is whether such real-world data indeed has any underlying structure or whether it is simply a disorder of different complexity over time. One known approach for measuring data disorder is entropy analysis. It is widely applied in EEG analysis [42], [53], stock data examination [25], [83] and other matrices [96], [13]. Generally, the term entropy comes from information theory and can be described as lack of predictability or a measure of disorder [34]. Yet there has been created various methods for entropy calculation with some of the widely-used types shown below:

1. **Permutation entropy** - measures complexity of a dynamic time series signal by capturing the order of relationships between time series singular values. Then, a probability distribution of ordinal patterns of time series is drawn from the data and corresponding signal's complexity is calculated. [16]

2. **Spectral entropy** - extracts power spectrum amplitude components of time series and calculated Shannon entropy upon it [30]

3. **Singular value decomposition (SVD) entropy** - measures the structure of correlation matrix for the component of a given signal time series. Vectors drawn from correlation matrix analysis are used to describe complexity of a signal. [25]

4. **Approximate entropy** - measure of instability of variation in the signal by finding changes in underlying per-episode behavior and by comparing these underlying sample patterns [68].

5. **Sample entropy** - is a modification of approximate entropy. It measures the regularity of a signal and is independent of the pattern length as well as it does not include self-similar patterns [73]

In vast majority of applications large periods of time series are used for entropy calculation making such experiments an offline case. However, we decided to apply entropy analysis into near real-time scenario and measure time series disorder over small intervals. Our main assumption is that entropy of anomalous regions should show different complexity to the entropy of a non-anomalous regions within the same time series. We expect that presence of an anomaly within time series batch increases or decreases disorder of this time series batch, and we expect disorder measure (entropy) to reflect this.

3.5.1. Selected Entropy: Singular Value Decomposition

Singular Value Decomposition entropy describes signal via its regularity that is determined via the number of vectors attributed to the signal [44]. Entropy value is measured using singular value matrix decomposition routine:

We can represent input signal as an array of values $[x_1, x_2, ..., x_n]$. From this array, the delay vector a_i can be constructed using delay τ and embedding dimension d_e:

$$a_i = [x_i, x_{i+\tau}, ..., x_{i+(d_e-1)\tau}] \tag{3.12}$$

Having delay vectors, the embedding space A can be constructed by using delay vectors from different signal windows:

$$A = [a(1), a(2), ..., a(N-(d_e-1)\tau)]^T \tag{3.13}$$

Then, singular matrix decomposition is performed on matrix A. Effectively, the singular value decomposition of a matrix **A** is a factorization of the form **A=UΣV*** with **U** being square unitary matrix, Σ is rectangular diagonal matrix, and **V** - square complex unitary matrix. Graphical representation of value decomposition process is shown in Figure 3.7.

The diagonal entries in Σ are known as singular values of **A**. The SVD entropy H_{SVD} is then defined in Equation 3.15 [17]

$$H_{SVD} = -\sum \bar{\sigma}_i log_2 \bar{\sigma}_i \tag{3.14}$$

where M is the amount of singular values and σ_i is normalized as below:

$$\bar{\sigma}_i = \frac{\sigma_i}{\sum_{j=1}^{M} \sigma_j} \tag{3.15}$$

Essentially, resulting singular values measure feature-richness of a given signal.

II

Experiments

4

Data

4.1. Publicly Available Data

To the best of our knowledge, there exists only 3 publicly available benchmarks with univariate time series and corresponding anomalous labels.

1. Yahoo[1] is an open-source benchmark from the datacenters of the search engine Yahoo with part of data coming from real Yahoo traffic, and part of data being synthesized. Real-time data points are labeled manually by editors, whereas labels of synthetic data are generated. The interval of time series is 1 hour. Yahoo consists of 4 subsets:

   (a) A1Benchmark - 67 time series that came from real-world observations and that were manually labeled by human experts.

   (b) A2Benchmark - 100 synthetic time series with the majority of time series having their own periodicity. Time series from this benchmark include single anomaly values.

   (c) A3Benchmark - 100 synthetic time series. Time series from this benchmark include single anomaly values, however, they are more noisy then A2 Benchmark. This benchmark also includes noise, trend, seasonality and change point data per time series.

   (d) A4Benchmark - 100 synthetic time series. The majority of the anomalies in these series are represented by a sudden transitions from an input data trend to another significantly different input data trend. Similarly to A3 Benchmark, this benchmarks includes noise, trend, seasonality and change point values.

   All time series in Yahoo dataset are rather short with the length around 2000 time points.

2. KPI[2] is a benchmark from AIOPS competition that contains multiple curves with labeled anomalies collected from different internet companies like eBay. The interval of time series ranges from 1 to 5 minutes. Time series from this dataset came from real world observations, they include numerous different types of anomalies in them, and they have varying length about 15000 to 200000 time points.

3. NAB[3] is a publicly available dataset with different data records of different nature. In total, there are 58 time series in NABdataset grouped under different categories that corresponds to time series nature: there are several artificially generated time series with and without anomalies as well as many real-world observations of EC2 metrics, taxi usage, stock tracking etc.. There is no consensus on time series granularity within NAB dataset, and it varies from minutely granularity for the artificial data to hourly granularity for click-per-cost real-world time series.

   This benchmark has its labels provided in a different form then the other datasets which make it hard to be used as a practical anomaly detection benchmark [81]. NAB gives 2 labeling techniques for labeling its data: the first labeling technique consists only of ground truth single-point anomaly labels

[1] https://yahooresearch.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly
[2] http://iops.ai/dataset_detail/?id=10
[3] https://github.com/numenta/NAB/tree/master/data

per anomaly interval, the second labeling approach places a window of the fixed size (10% of time series length) around ground truth label and identifies everything within this interval as anomaly. The second approach results in many normal time points being labeled as anomalies whereas the first approach misses large portion of anomalies points. Such labeling strategy is made to be used with NAB labeling technique introduced in [49]. It prioritizes methods that detect anomalies earlier but does not assess how good a method is in precise anomalous interval capturing. Therefore, potential users of NAB dataset are encouraged to do additional anomaly labeling of time series tailored to their use case.

Real data from NAB dataset is divided into the following categories:

(a) *realAWSCloudwatch*. Machine metrics collected by Amazon CloudWatch service as CPU utilization or disk read bytes. All time series have anomalies and their granularity is 5 minutes. All except one time series have anomalies and their granularity is 1 hour.

(b) *realAdExchange*. Web advertisement clicking rates time series. The metrics is cost-per-click (CPC time series) or cost-per-thousand-impressions (CPM time series).

(c) *realKnownCause* represents a general collection of time series with the known anomaly cause. Time series in this subset did not go through the process of manual labelling by human experts.

(d) *realTraffic*. Time traffic data from the Twin Cities Metro area in Minnesota with metrics as occupancy, speed and travel time/ The data was collected by [3]. The granularity of time series varies from 5 to 10 minutes.

(e) *realTweets*. Time series of Twitter mentions of large publicly-traded companies such as Facebook or Apple. Metrics is the number of mentions of a ticker symbol of a particular company within 5 minutes intervals.

Artificial data has the following structure:

(a) *artificialNoAnomaly*. Artificially generated data that has no anomalies. Its granularity is 5 minutes.

(b) *artificialWithAnomaly*. Artificially generated data that has anomalies. Its granularity is 5 minutes.

5

Experiments and Analysis

5.1. Model Evaluation

We split every time series in a dataset into training and testing data with the ratio 50%-50%, and we use the first part for training and the second part for testing.

We commit our evaluation in online fashion meaning that we test data in batches allowing only limited amount of the most recent timestamps from the stream to be labeled. In order to simulate streaming scenario, we use a sliding window of a fixed length in our experiments. Window of a fixed size goes through testing data part of the data, for every window we record predicted labels and store them in memory. In the end of testing data, we compare joint predictions from all of the windows against true labels on a whole test data part to obtain one overall score. Schematic representation of our testing routine is shown in Figure 5.1. We selected F1-score as a single performance measure for anomaly detectors since it is widely used across literature [72], [56]. F1-score is a single measure that calculates trade-off between precision and recall. Precision is the percentage correctly detected anomaly labels out of all hits, whereas recall is the percentage of correctly detected anomalies out of all true anomalies. The definitions of precision, recall, and F1-score are shown in (5.1), (5.2), (5.3) respectively, with TP being true positives, FP - false positives, TN - true negatives, and FN - false negatives.

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1 - score = \frac{2 * precision}{precision + recall} \tag{5.3}$$

Since we target data streams, the distribution of the upcoming data is always unknown. Therefore, hyper-parameter tuning was not performed to avoid biasing the model towards the training data.

5.2. Statistical Testing for Significance

In order to verify that the improvement shown by the top model/method is statistically significant, we use Wilcoxon Signed Rank Test [94]. The test compares the location of two sample distributions by assessing the pairwise matched samples. Wilcoxon Signed Rank Test is a non-parametric test meaning it does not assume parameterized distribution of the input data which is applicable to our experiments. We are using Wilcoxon Test in order to compare the distributions of F1-scores of anomaly detectors derived on singular time series within fixed dataset. The null-hypothesis is that the median of F1-score differences between compared models over the same time series within a dataset is zero. If Wilcoxon Signed Rank Test gives a significant p-values of less than 5% then we can reject null-hypothesis and assume that the difference is significant.

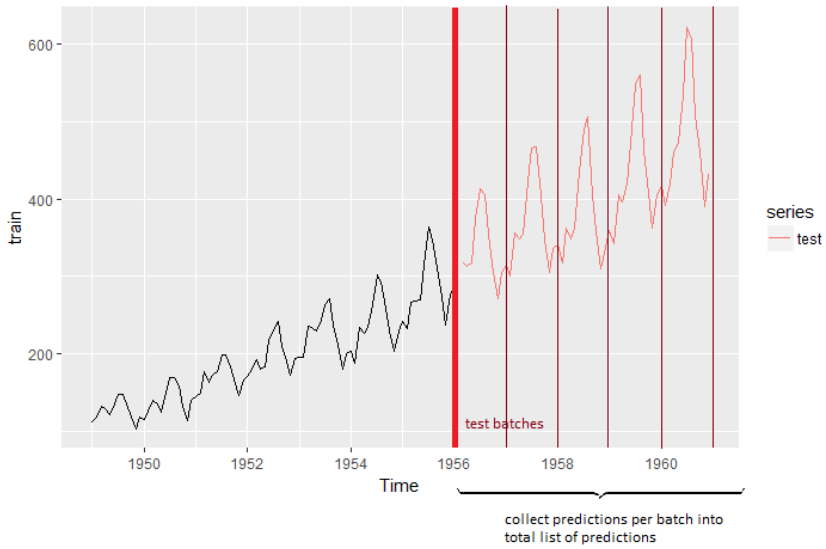


Figure 5.1: Schematic representation of training/testing procedure. All labels from test batches are collected and recorded and then used all together for the final F1-score calculation.

5.3. Base Models' Performance

We have chosen one to two machine learning algorithm from every category of our classification of machine learning methods for anomaly detection in time series (available under Section 3.2.1). We then ran the selected models on benchmark datasets *without any optimization or hyperparameter tuning since this would require anomaly labels of time series to be known.* The performance was measured in terms of F1-score, however, we also kept track of other metrics, namely, sensitivity, specificity, and the amounts of false positives and false negatives.

The decision to use several models from some categories and only a single model from other categories was based on popularity of particular methods within scientific literature. For example, to the best of our knowledge, many studies use several different ETS methods as base models or in ensemble of weak learners [48], [60], and therefore, we have tested two methods, namely Exponential Smoothing and SARIMA, from this class.

The whole set of candidate base models is listed in Table 5.1.

Selected model	Model group
LOF	Distance-based methods
DBSCAN	Distance-based methods
OC-SVM	Clustering methods
Exponential Smoothing	ETS methods
SARIMA	ETS methods
Isolation Forest	Isolation methods
LSTM Auto-Encoder	Deep learning methods
Spectral Residuals	Saliency methods
PSO-ELM	Combinations of methods and ensembles

Table 5.1: Candidate base detectors with corresponding classes from Section 3.2.1.

We first ran all selected models on Yahoo A1 dataset, which contains real data only. Such decision was made to filter out less promising models on a smaller dataset without running all detectors across all available time series. We chose Yahoo A1 due to its popularity within scientific literature as well as real nature and short length of time series that it contains. Overall F1-score on test data for each model is shown in Table 5.2.

We then selected the set of 4 best-performing models, namely, LSTM Auto-Encoder, SARIMA, Spectral Residuals, and PSO-ELM, and ran them on all public datasets (including KPI and NAB).

Our results show that when applied into sliding window scenario with the window of length 60, perfor-

Model	Total Test F1-score
LSTM AE	**0.304**
SARIMA	**0.295**
SR	**0.195**
PSO-ELM	**0.187**
Exponential Smoothing	0.169
DBSCAN	0.168
OC-SVM	0.151
Isolation Forest	0.106
LOF	0.031

Table 5.2: Base models' F1-scores on Yahoo A1 dataset.

	Total Test F1-score			
Dataset	LSTM	SARIMA	SR	PSO-ELM
Yahoo A1	**0.304**	0.295	0.195	0.187
Yahoo A2	0.008	0.298	**0.511**	0.006
Yahoo A3	0.020	0.294	**0.602**	0.055
Yahoo A4	0.0173	0.302	**0.534**	0.068
KPI	0.110	**0.160**	0.063	0.106
NAB	0.201	0.137	0.051	**0.205**

Table 5.3: Overall performance of top base models on public data.

mance of all base predictors drops below F1-score of 0.5. We show the performance of top base models on all public data in Table 5.3.

As Table 5.3 indicates, selected anomaly detectors do not adapt to different streaming scenarios, as judged by low F1-score (F1-score ≤ 0.7). We can also observe that different models exhibit their best performance on different datasets, such as Spectral Residuals model having relatively high F1-score on all 3 synthetic datasets within our benchmarks, however, failing to adapt to real data.

Finding 1: *Conventional anomaly detectors are not directly applicable to streaming scenario, and they have poor generalizability to different datasets.*

We have also performed light Bayesian Optimization (20 rounds) for all 4 base models in order to verify that our models indeed report compatible performance once tuned. We excluded Yahoo A3 and A4 dataset from this experiment due to time constraints and the fact that they both contain synthetic data whereas we are interested in real data and already have one synthetic dataset Yahoo A2. Our results with and without optimization are shown in Table 5.4.

We have observed that once being optimized for specific time series, models show around 2-fold increase in their detective performance. Nonetheless, since we are performing completely unsupervised learning, we have proceeded with SARIMA, LSTM AE, PSO-ELM, and SR models without any fine-tuning.

	Total Test F1-score							
Dataset	LSTM	LSTM-optim	SARIMA	SARIMA-optim	SR	SR-optim	PSO-ELM	PSO-ELM-optim
Yahoo A1	0.304	**0.411**	0.295	**0.422**	0.195	**0.276**	0.187	**0.323**
Yahoo A2	0.008	**0.494**	0.298	**0.427**	0.511	**0.517**	0.006	**0.507**
KPI	0.110	**0.177**	0.160	**0.173**	0.063	**0.102**	0.106	**0.316**
NAB	0.201	**0.206**	0.137	**0.139**	0.051	**0.110**	0.205	**0.211**

Table 5.4: Overall performance of top base models on public data with and without optimization. *-optim* column corresponds to the tuned model.

5.4. Data Analysis

The main purpose of performing data analysis was to find possible relations among certain time series/datasets and predictive performance of various groups of anomaly detectors instantiated via F1-scores. In case of existence of potential relation, it can be then exploited in order to either boost the performance of an existing anomaly detector or to identify time series that are more applicable for incident detection in case of fixed type of anomaly detector.

First, we have extracted certain properties (see Appendix 18) from all time series and all benchmarks. In total, 64 properties for each time series have been calculated using Catch22 and FFORMA systems described in details in Sections 3.1.2 and 3.1.1 respectively. We also used *per-time-series F1-score* for each of the 4 top-performing machine learning methods from Table 5.2 in our analysis.

The analysis consisted of several parts and had 4 main purposes:

1. Describe datasets via the distributions of time series features found within them.

2. Identify similar and more distinct datasets by calculating distances between feature distributions from different datasets.

3. Correlate the performance of base detectors on different benchmarks to the distance between these benchmarks.

4. Correlate the performance of base detectors on different time series to the features found within these time series independently of a datasets to which particular time series belongs.

For per-dataset analysis we conducted the following steps:

We have collected the properties of all time series within 3 benchmarks and looked at feature distributions between different datasets. Since our selected benchmarks contain only a small amount of time series (454 time series in total), we decided to compare feature distributions against another larger and publicly available benchmark of time series. Selected benchmark does not provide per-univariate-time-point anomaly label, however, in our analysis we are only interested in time series features and so that anomaly labels are irrelevant for the task. We have chosen UEA/UCR (University of East Anglia and University of California, Riverside) Time Series Classification Repository benchmark for this purpose [29]. Such decision was related to the fact that this repository was used in Catch22 study [54] and it contains a total of 147198 time series samples from various classes.

We examined per-feature distribution and distances among such distributions between different benchmarks including UCR benchmark. We also calculated the total distance between datasets as a total distance between all their features. There exists no single universal measure for calculating distances between two probability density functions. Due to this, we decided to use 8 different distance measures from different classes of pdfs' distance measures. These classes were introduced in [26] and depends on the base formula/technique used for calculating distance measure. We selected most common measure from every category and show them in Table 5.5.

The resulting orderings of all datasets based on distance between their pdfs are shown in Table 5.6. Since different distance measures do not have complete agreement on the pair of the most distant and the closest datasets, we calculated rank similarities among these distance orderings via each of 8 different distance measures using Kendall's rank correlation test [47]. The purpose of rank similarity analysis was in finding strongly correlated distance measures as well as finding general agreement of ranking via different distance types among each other. The results of rank correlation test are shown in Table 5.7. As Table 5.7 indicates, we have not found many strongly correlated candidates as well as there is generally only poor agreement between different distance measures on an ordering of the datasets. Notably, only rankings via Euclidian distance and squared Euclidian distances seem to have strong similarity. Such finding is explained by the similarity in distance calculation for these distance measures.

To verify that time series features distributions calculated on different benchmarks came from the same general distribution, we have performed Kolmogorov-Smirnov statistical test [18]. Our results show that there is no evidence to believe that any of features distributions from any of a benchmark came from different distribution.

After initial step of our analysis, we were not able to identify more related datasets, as judged by time series features found within them, due to the controversy of the results under different similarity measures. We then proceeded by analyzing time series features independently of the dataset they belong to.

Distance type	Selected distance	Formula
L_p Minkowsky family	Euclidian L_2	$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
L_1 family	Sorensen	$d(p,q) = \frac{\sum_{i=1}^{n}\lvert p_i - q_i\rvert}{\sum_{i=1}^{n}(p_i + q_i)}$
Intersection family	Intersection	$d(p,q) = \sum_{i=1}^{n} min(p_i, q_i)$
Inner product	Inner product	$d(p,q) = P \bullet Q$
Fidelity family	Fidelity	$d(p,q) = \sqrt{p_i q_i}$
Squared L_2 family (χ^2)	Squared Euclidian	$d(p,q) = \sum_{i=1}^{n}(p_i - q_i)^2$
Shannon's entropy family	Kullback–Leibler	$d(p,q) = \sum_{i=1}^{n} p_i ln\frac{p_i}{q_i}$

Table 5.5: Selected types of measures for distances between probability density functions from each of base classes from [26]

Dataset 1	Dataset 2	Euclidian	Sorensen	KL	Inner Product	Fidelity	Intersection	Squared Euclidian
Yahoo A1	Yahoo A2	15.898	13.788	8.225	6.524	11.202	14.740	12.049
Yahoo A1	Yahoo A3	19.859	15.339	10.824	7.873	10.367	14.740	18.175
Yahoo A1	Yahoo A4	16.683	14.342	9.076	8.501	11.597	14.740	13.053
Yahoo A1	NAB	12.312	12.978	10.426	5.822	8.790	14.740	7.810
Yahoo A1	KPI	11.345	12.412	12.459	2.478	5.432	14.740	6.568
Yahoo A2	Yahoo A3	17.731	13.227	14.641	14.481	15.823	22.000	15.233
Yahoo A2	Yahoo A4	15.935	11.848	12.982	13.038	16.993	22.000	12.220
Yahoo A2	NAB	16.314	12.580	17.707	4.525	8.857	22.000	12.758
Yahoo A2	KPI	13.787	13.838	19.048	3.358	6.467	22.000	9.356
Yahoo A3	Yahoo A4	15.165	13.906	7.544	17.332	18.621	22.000	12.017
Yahoo A3	NAB	18.498	15.413	13.847	8.906	10.412	22.000	16.017
Yahoo A3	KPI	18.471	14.188	16.961	4.677	6.868	22.000	15.844
Yahoo A4	NAB	15.866	14.697	15.459	7.769	10.253	22.000	11.939
Yahoo A4	KPI	15.514	14.624	18.645	4.233	6.746	22.000	11.589
NAB	KPI	10.119	12.059	8.631	2.980	5.709	12.540	5.562

Table 5.6: Distance measures between the datasets via 8 different distance measures between pdfs of their features.

After extracting two sets of time series features via Catch22 and FFORMA methods, we used them in per-time-series correlation analysis. We performed 2 different analysis: one for linear correlations via Pearson correlation coefficient, and another one for non-linear correlations via mutual information. The correlations we have looked into are correlations between particular time series values and corresponding F1-score of models on these time series.

The relation between two variable are considered to be strong if r-coefficient> 0.7 or r-coefficient< -0.7 [58]. As our results indicate, there are no strong linear correlations in our data. Additionally, mutual information values all fall below 0.5 indicating absence of strong non-linear correlations. However, for 90% of extracted time series features their mutual information with model scores is above 0 showing weak extent of dependability.

Since we have not encountered any clear influence of time series features on models' performance, we researched influence of other time series variable on predictive performance of the models.

Finding 2: *Neither time series features nor benchmarks these time series belong to have strong correlations to models' performances.*

	Euclidian	Sorensen	KL	Inner Product	Fidelity	Intersection	Squared Euclidian
Euclidian	1.000	0.429	0.086	0.410	0.352	0.267	0.962
Sorensen	0.429	1.000	0.086	0.257	0.124	0.216	0.390
KL	0.086	0.086	1.000	-0.276	-0.333	0.572	0.048
Inner Product	0.410	0.257	-0.276	1.000	0.867	0.267	0.448
Fidelity	0.352	0.124	-0.333	0.867	1.000	0.216	0.390
Intersection	0.267	0.216	0.572	0.267	0.216	1.000	0.267
Squared Euclidian	0.962	0.390	0.048	0.448	0.390	0.267	1.000

Table 5.7: Similarities in ranking via 8 different distance measures among Yahoo, KPI and NAB datasets.

5.5. Drift Adaptation

In this part of the analysis, we have looked into possible ways how drifts in data may influence predictive performance of base models. Unfortunately, to the best of our knowledge, there is no publicly available time series data where both anomaly labels and drifted parts are identified. Therefore, we selected a set of base drift detectors for data streams and tested their agreement in drifts they identify on our benchmarks. We selected the following drift detectors due to their popularity with the scientific literature: ADWIN [19], DDM [33], EDDM [15], ECDD [75], HDDA [37], and HDDW [37]. As our results indicate, there was no agreement found among different drift detectors on to what is considered as a drift within a stream: amount of detected drifts *per a single time series* varied from 0 (in case of DDM) to hundreds or thousands (in case of HDDA).

Since we were not able to reliably label drifts in our data, we decided to look into alternative ways of identifying and exploiting drifts in streaming data.

Our first approach relied on per-batch online performance of models by measuring whether it degrades over time. In this settings, instead of looking into one general F1-score over test data, we recorded per-batch hamming distance [1] of every anomaly window. We then calculated average hamming distance for batches before the first anomaly and after the last anomaly in testing data. We schematically represent our per-batch evaluation routine in Figure 5.2. Orange dotted line corresponds to time series batch with anomalies – in general, time series might have single or several batches with anomalies. **D1** represents average batched hamming distance before the first anomaly in time series calculated over the first 3 batches (the first 3 blue dots). **D2** represents average batched hamming distance after the last anomaly in time series calculated over the last 4 batches (the last 4 blue dots). We track D2-D1 difference per time series. In case of performance degradation, measure will *increase* due to increased percentage of mismatched labels. Average changes of hamming distance in such test settings per public dataset are shown in Table 5.8. We can see that there is no stable change in average per-batch hamming loss change over time across all anomaly detectors except for performance on Yahoo A1 dataset. However, such behavior might be explained by the fact that the vast majority of time series within Yahoo A1 have their anomalies in the very last portion of time series, not leaving representative amount of normal batches for calculating **D2** reliably.

	Increase in avg. Hamming Loss			
Dataset	LSTM	SARIMA	SR	PSO-ELM
Yahoo A1	0.141	0.125	0.126	0.133
Yahoo A2	0.337	0.0	0.004	0.0
NAB	0.090	0.083	-0.003	0.081
KPI	0.028	-0.001	0.005	-0.002

Table 5.8: Average change of per-batch Hamming loss before first and after the last anomaly encounters in time series.

The second approach relied on integrating existing drift detectors described in Section 5.2, namely ADWIN, DDM, EDDM, ECDD, HDDA, and HDDW, into our base models and comparing their performance to the same models without drift adaptation. In these experiments we again excluded Yahoo A3 and A4 dataset due to time constraints and synthetic nature of the data in these datasets. We also did not use SR model for drift adaptation experiments since the model itself does not require pre-training and, therefore, should not be updated. Table 5.9 reports performance for our base models without drift adaptation and *the highest achieved performance with one of the tested drift detectors.* Table 5.10 indicates the best performing drift detector per model and detaset.

[1] Fraction of wrong labels to all labels produced by a model.

	Total Test F1-score					
Dataset	LSTM	LSTM DA	SARIMA	SARIMA DA	PSO-ELM	PSO-ELM DA
Yahoo A1	0.304	0.300	0.295	0.295	0.187	0.187
Yahoo A2	0.008	0.011	0.298	0.355	0.006	0.006
KPI	0.110	0.094	0.160	0.242	0.106	0.107
NAB	0.201	0.187	0.137	0.142	0.205	0.207

Table 5.9: F1-score on test set for models with and without drift adaptation, where DA = drift adaptation

	Best-performing Drift Detector		
Dataset	LSTM	SARIMA	PSO-ELM
Yahoo A1	EDDM	HDDW/EDDM	EDDM
Yahoo A2	**HDDW**	**HDDW/EDDM**	HDDW/EDDM/ECDD
KPI	EDDM	**ECDD**	**HDDW**
NAB	ECDD	**HDDW**	**EDDM**

Table 5.10: Drift detectors that yield highest performance in combination with anomaly detector on selected datset. Drift detectors in **bold** improve performance in relation to anomaly detector without drift adaptation.

As the results in Table 5.9 show, we did not find any consistent improvement in performance when applying drift adaptation, yet SARIMA model seems to benefit from drift adaptation on all benchmarks. In addition, Table 5.10 indicates the most applicable set of drift detectors for anomaly detection scenario: HDDW, EDDM, and ECDD. However, due to inconsistency of the results we do not consider drift adaptation to be profitable for generalizability improvement of anomaly detectors.

Finding 3: *Drift adaptation does not always improve applicability of anomaly detectors into various streaming scenarios, yet some models, as SARIMA, benefit from it.*

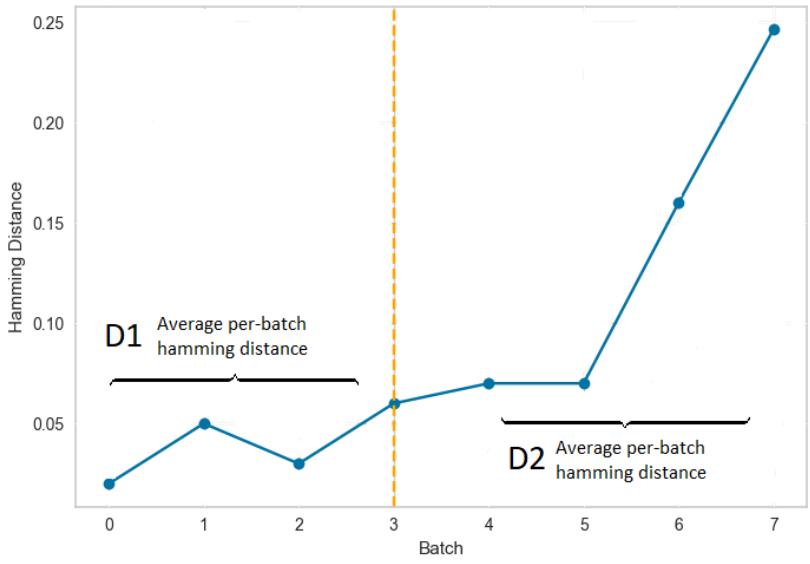


Figure 5.2: Schematic representation of evaluation of change in base model performance on time series data via per-batch hamming distance.

5.6. Entropy analysis

All the directions of our previous analysis were completely or partially based on the belief that data streams have some underlying structure that can be instantiated via features or disrupted via drifts. Our base models, as SARIMA or LSTM AE are trying to reconstruct underlying representation of a stream and detect outliers.

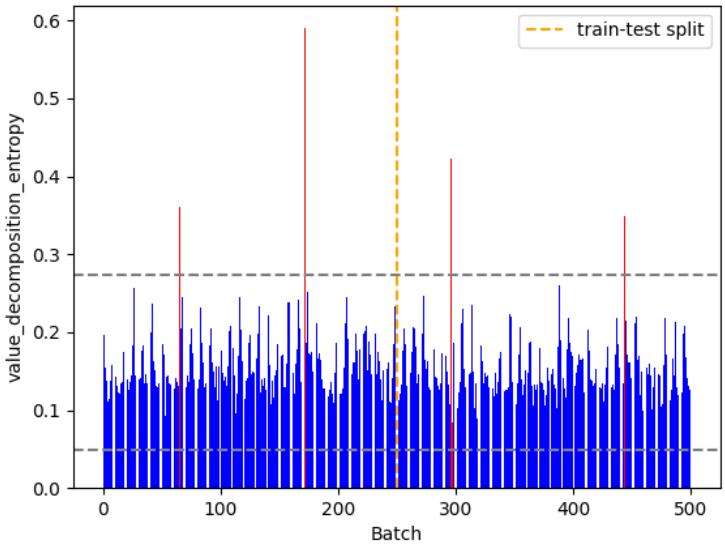


Figure 5.3: Example of how entropy model pops out anomalous batches. Accepted boundaries are shown with dashed grey lines, blue color of a batch indicates 0 anomalies whereas red color indicates presence of anomalies within a batch.

In this current direction we offer to treat streams as disorder. If our streams do not have structure and consist of disordered observations, we can detect anomalies in such settings by looking at data complexity: even though there is no underlying structure, anomalies exhibit different behavior from what was recorded before. Such difference in behavior might be identifiable via complexity comparison of consequent stream batches. For example, anomaly situation when all monitoring machines went off will have lower complexity in comparison to behavior before anomaly has happened since zero-signal does not need a lot of information to be reconstructed. On the other hand, if we see anomalous increase in CPU usage created by, for example, additional processes, the behaviour of such processes adds additional complexity to the stream disorder.

We tried to exploit this direction by identifying per-batch stream complexity via different measures of per-batch entropy.

We defined and measured entropy in batched scenario where time series are split into non-intersecting consequent batches of equal length and disorder in every batch is measured. We set a hard static threshold for upper and lower bounds of per batch entropy by calculating mean per-batch entropy value on a training data and allowing a deviation within ± standard deviation boundaries. We re-labeled training set *per-batch* where batch with at least one anomaly got label of 1 and 0 otherwise. We ran experiments on NAB and KPI datasets in order to find the most suitable entropy in a given scenario. Our results, reported in Table 5.11, show that value decomposition entropy is the most suitable measure for identifying anomalous batches. Figure 5.3 shows the capability of entropy model to highlight anomalous batches.

Thereafter, we proceeded with SVD entropy only and we created entropy model for comparison with already existing baseline models. Our entropy model works by calculating per-batch entropies from a train part of time series and then it calculates upper and lower bounds via $mean \pm factor * std$ on a range of batched entropies from train set. We again used per-batch evaluation scenario for comparing baselines with entropy model. In such settings, applying solely toy entropy model without any additional ML-based anomaly detectors performs comparably well to our base detectors within the same settings (see results in Table 5.12).

Finding 4: *SVD entropy analysis is capable of highlighting anomalous batches within the signal.*

We then ran brute-search through possible window sizes and boundary factors to find the best matching parameters for KPI dataset. Our results show that the window of 55 is the best choice (F1-score = 0.453), however, window of 35 is almost as good (F1-score = 0.451) when using factor of 2.5 (see Figure 5.5 (a)). This result means that entropy windowing can be integrated into real-time streams.

Figure 5.5 (b) shows same brute-search applied to NAB dataset. In this case, there is no gradual change of F1-score over adjacent values of windows and factors. In addition, the range of F1-scores obtained by model is lower than in case of KPI. Therefore, in case of NAB data, optimal window for entropy application might be

Entropy Measure	F1-score KPI
Value Decomposition entropy	**0.421**
Sample entropy	0.266
Approximate entropy	0.105
Spectral entropy	0.093
Permutation entropy	0.024

Table 5.11: Analysis of anomaly detection per-batch performance of various entropy measures from Section 3.5.

Model	F1-score NAB	F1-score KPI
SR	0.149	0.262
SARIMA	0.304	0.531
LSTM AE	0.205	0.263
PSO-ELM	0.312	0.266
Entropy std model	0.190	0.421

Table 5.12: Per-batch performance analysis of toy entropy model versus 3 best-performing baseline models from Section 5.3.

obtained by chance.

Figure 5.5 (c) and (d) apply same brute-force search on Yahoo A1 and A2 datasets. For A1 data, optimal window is 85 with a factor of 1.0 whereas for A2 the optimal window is 25 with a factor of 1.8.

Interestingly, we were not capable of finding optimal window size for Yahoo synthetic benchmarks A3 and A4 - our heatmaps suggest that optimal window lies in bottom-right corner of the map meaning maximally large windows with maximally small factors (near 0.0) that would label every incoming point as anomaly due to high restrictiveness. Corresponding heatmaps are shown in Figure 5.5 (e) and (f).

Finding 5: *SVD entropy can be applied into near-real time scenario with batch sizes being below 100 time points.*

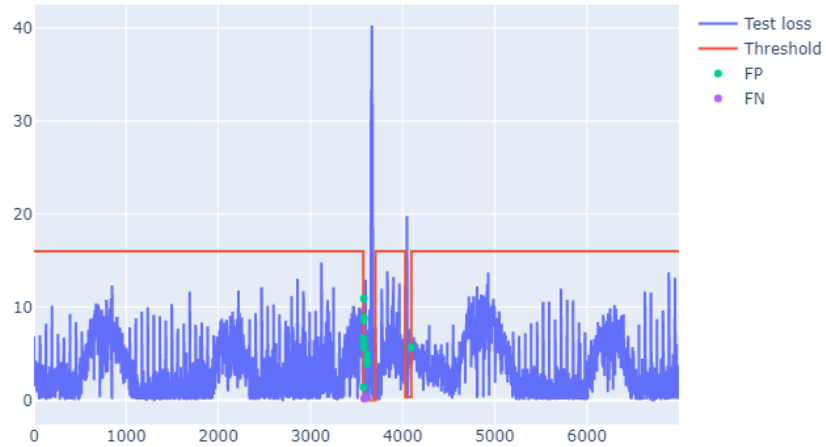


Figure 5.4: An examples of dynamic threshold adaptation for LSTM AE. Threshold is shown by the red line and is applied on LSTM AE forecasting loss.

After we have found the optimal window size that can fit into our definition of near real-time scenario for time series, we decided to integrate entropy-based threshold into existing base methods. Since each of the base methods ground their detection strategy upon scoring incoming points from a univariate stream, we can apply entropy analysis to adjust models' threshold over data stream dynamically in relation to disorder measure of a particular batch. We integrated entropy threshold into existing (rigid) threshold strategies for every model. This was done in several steps:

1. We calculate per-batch anomaly scores on a training part of time series.

2. Based on these scores, we set minimum and maximum threshold for non-anomalous batch values by calculating mean and standard deviations of per-batch entropy scores:

$$threshold\_bottom = \overline{entropy\_scores} - f * \sigma$$

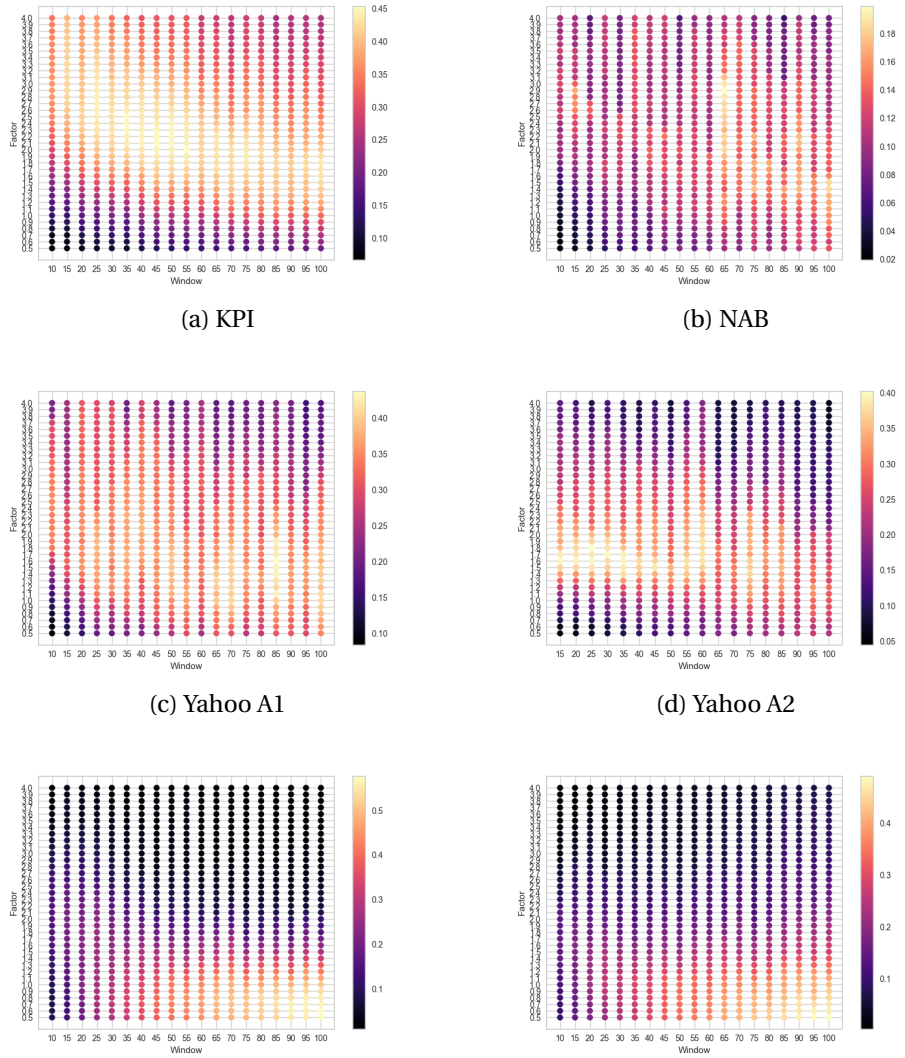
$$threshold\_top = \overline{entropy\_scores} + f * \sigma$$

Factor f is subjected to particular time series, yet our experiments show that the factors between 1.0 and 2.5 yield the best performance.

3. When making predictions in streaming settings, we first populate current anomaly batch up until it is full and then test whether its disorder measure falls within normal deviation of entropies.

   (a) **If this is the case**, then we proceed with built-in anomaly detection strategy of a model.

   (b) **Otherwise**, we shrink built-in threshold value by the same magnitude as current entropy batch deviates from mean entropy.



(a) KPI (b) NAB

(c) Yahoo A1 (d) Yahoo A2

(e) Yahoo A3 (f) Yahoo A4

Figure 5.5: Colormap of entropy model performance on public datasets (Yahoo, NAB, KPI) with varying window sizes and factor values. Colorbar indicates F1-score performed in batched fashion with the corresponding window size.

Proposed strategy aims to minimize anomaly threshold in presence of entropy abnormalities, meaning that we allow more points to be classified as anomalies when entropy of data changes. Dynamic threshold adaptation integrated within LSTM AE model is shown in Figures 5.4.

	Total F1-score									
	KPI		NAB		Yahoo A1		Yahoo A2		Murex ING	
Model	BT	ET	BT	ET	BT	ET	BT	ET	BT	ET
SARIMA	0.173	**0.180**	0.139	**0.194**	**0.422**	0.208	**0.427**	0.343	**0.175**	0.131
LSTM AE	0.177	**0.214**	0.206	**0.234**	**0.411**	0.277	**0.494**	0.233	**0.129**	0.078
SR	0.102	**0.277**	0.110	**0.171**	**0.276**	0.182	**0.512**	0.228	0.112	0.112
PSO-ELM	0.220	**0.316**	0.123	**0.211**	**0.323**	0.303	**0.507**	0.304	**0.144**	0.133

Table 5.13: Performance of **tuned** models with built-in (BT) and entropy (ET) threshold strategies on publicly available data.

	Total F1-score									
	KPI		NAB		Yahoo A1		Yahoo A2		Murex ING	
Model	BT	ET	BT	ET	BT	ET	BT	ET	BT	ET
SARIMA	0.160	0.161	0.137	**0.174**	**0.295**	0.191	**0.298**	0.105	**0.081**	0.078
LSTM AE	0.110	**0.132**	0.201	**0.219**	**0.304**	0.184	0.008	0.008	0.046	**0.059**
SR	0.063	**0.079**	0.051	**0.055**	**0.195**	0.155	**0.511**	0.302	**0.104**	0.100
PSO-ELM	0.106	0.106	0.205	0.205	**0.187**	0.183	0.006	0.006	0.061	**0.104**

Table 5.14: Performance of **raw** models with built-in (BT) and entropy (ET) threshold strategies on publicly available data.

	Total F1-score					
	LSTM		SARIMA		PSO-ELM	
Dataset	BT	ET	BT	ET	BT	ET
Yahoo A2	0.011	**0.014**	**0.355**	0.125	-	-
KPI	-	-	0.242	**0.251**	0.107	**0.131**
NAB	-	-	0.142	**0.158**	0.207	**0.220**

Table 5.15: Performance of raw models with built-in (BT) and entropy (ET) threshold and with the most applicable drift detector from Table 5.10 (if any) on publicly available data. "-" indicates that drift adaptation did not improve model performance (see Table 5.10) and, therefore, entropy threshold has not been tested in such settings.

	P-values from Wilcoxon Signed Rank Test			
	Raw models		Tuned models	
Model	KPI	NAB	KPI	NAB
SARIMA	0.551	0.055	0.014	$4.2e^{-6}$
LSTM AE	0.031	0.009	0.007	0.031
SR	0.119	0.344	0.001	$1.7e^{-5}$
PSO-ELM	0.863	0.922	0.065	$7.8e^{-5}$

Table 5.16: P-values of Wilcoxon Signed Rank Test performed on F1-score distributions (with and without entropy threshold) of raw and optimized models per dataset.

Our last step was to test joint models (baselines + entropy threshold) on publicly available data. Our results are reported in Table 5.13 for ***tuned*** models and in Table 5.14 for ***raw*** models. In addition, for the combinations of base models and datasets that benefit from drift adaptation (see Table 5.10), we ran experiments with both dynamic entropy threshold and drift detectors. We report results in Table 5.15 for raw models only. We also performed statistical test (Wilcoxon Signed Rank Test) described in Section 5.2 to test significance of

change when applying entropy-based threshold over the datasets. These results are reported in Table 5.16 for the dataset where we observed improvement, namely, NAB and KPI, and for both raw and optimized models.

Results indicate that in application to KPI and NAB public data, entropy-based threshold boost performance of models from our selection, as judged by total F1-score. Such boost in performance is significant for all optimized models except PSO-ELM on KPI data, as indicated by p-values ≤ 0.05. The average improvement is around 30% and fluctuates between 0% to 50% for different model types and different datasets. In case of Yahoo dataset, we only tested our models on benchmarks A1 and A2 since we could not find optimal window for other 2 benchmarks. In case of Yahoo, entropy-based threshold decreases models' performance. Additionally, we used internal ING Murex dataset for test, where entropy-based threshold did not improve performance either.

Finding 6: *Entropy-based threshold seems to improve models' performance on closer-to-real-case data, as in case of KPI and NAB datasets, whereas it decreases performance for datasets with larger granularity, as Yahoo or ING Murex.*

5.7. Comparison to state-of-the-art models

Based on all finding and observations from the previous stages of our research, we have decided to combine them all together into a single model and compare this model to the recent state-of-the-art anomaly detectors found in scientific literature. We selected Online Evolving Spiking Neural Networks model (OeSNN-UAD) proposed in 2021 by Piotr S. Maciąga et. al. [56] due to its recentness and large collection of anomaly detectors it was compared to. We compare our model to all base models used in [56] that include recently invented (2019) DeepAnt model [62] as well as some older anomaly detectors outlined below:

1. *Numenta* and *NumentaTM* [12] are the algorithms based on hierarchical temporal memory network (HTM) and anomaly likelihood calculation part for anomaly detection.

2. *HTM JAVA* [35] is a Java implementation of the Numenta algorithm.

3. *Skyline* [84] is an ensemble algorithm for anomaly detection. Majority-voting is used to identify anomaly label of a time point.

4. *TwitterADVec* [67] is a Seasonal Hybrid ESD (S-H-ESD) -based algorithm [74] for anomaly detection.

5. *EGADS* [48] an ensemble algorithm for anomaly detection in time series proposed by Yahoo. It uses time series learning and modelling in order to identify deviations in incoming streams from the learned model and detect anomalies based on it. It is capable of detecting three distinct anomaly types, namely: outliers, sudden change points in values and anomalous sub-sequences of time series.

6. *DeepAnT* [62] is neural methods for anomaly detection that can have in its core long short-term layers (LSTM) or convolutional (CNN) layers. It works by modelling time series and then detecting anomalies in true observed values.

7. *Bayesian Changepoint* [11] time series change point detection algorithm based on Bayesian inference and it analyses a probability distribution of the time series partitions created from earlier detected change points. For the algorithms to work reliably, it is desired to have sudden change points in stream distributions over time.

8. *EXPected Similarity Estimation (EXPoSE)* [78] distribution-based anomaly detector. It detects anomalies if current input distribution of input deviates to what has been observed before.

9. *KNN CAD* [24] makes used of nearest-neighbours classification technique for anomaly detection in univariate streams.

10. *Relative Entropy* [90] Kullback-Leibler divergence based methods for anomaly detection. It compares previously observed and current distributions of time series points and detects anomaly if divergence is large. This technique resembles our per-batch entropy calculation.

11. *ContextOSE* [82] extracts values from time series inputs and creates context properties from them. Anomalous label is given to such points which context differs significantly to baseline context.

Table 5.17: Average F-measure values obtained for Numenta Anomaly Benchmark stream data using the unsupervised anomaly detectors (marked with *) presented in [56] and using proposed PSO-ELM-ET detector. The results in bold are the best for each data files category. The results for the detectors marked with * were reported in [56].

Dataset category	*Bayesian Changepoint**	*Context OSE**	*EXPoSE**	*HTM JAVA**	*KNN CAD**	*Numenta**	*NumentaTM**	*Relative Entropy**	*Skyline**	*Twitter ADVec**	*Windowed Gaussian**	*DeepAnT**	*OeSNN-UAD**	*PSO-ELM-ET*
Artificial no Anomaly	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Artificial with Anomaly	0.009	0.004	0.004	0.017	0.003	0.012	0.017	0.021	0.043	0.017	0.013	0.156	**0.427**	0.346
Real Ad Exchange	0.018	0.022	0.005	0.034	0.024	0.040	0.035	0.024	0.005	0.018	0.026	0.132	0.234	**0.362**
Real AWS Cloud	0.006	0.007	0.015	0.018	0.006	0.017	0.018	0.018	0.053	0.013	0.06	0.146	0.369	**0.404**
Real Known Cause	0.007	0.005	0.005	0.013	0.008	0.015	0.012	0.013	0.008	0.017	0.006	0.2	0.324	**0.350**
Real Traffic	0.012	0.02	0.011	0.032	0.013	0.033	0.036	0.033	0.091	0.020	0.045	0.223	0.340	**0.341**
Real Tweets	0.003	0.003	0.003	0.010	0.004	0.009	0.010	0.006	0.035	0.018	0.026	0.075	**0.310**	0.212
Averaged	*0.008*	*0.009*	*0.006*	*0.018*	*0.008*	*0.018*	*0.018*	*0.016*	*0.034*	*0.015*	*0.025*	*0.133*	*0.286*	***0.288***

We did not run or pre-train any of the above detectors by ourselves since, in majority, they report rather poor performance on NAB benchmark [62] in comparison to other prominent models. We use earlier reported results from [56] as the point of reference.

In selecting components of our model we relied on the following conclusions from our previous observations as well as on architectural constraints of our base models:

1. We have observed that on longer time series that are closer to real data, entropy-based threshold boosts performance, therefore, we decided to use a model with entropy thresholding.

2. In order for model to be comparable to OeSNN-UAD, we need to perform point-based predictions, yielding anomaly predictions one-by-one.

3. Ideally, we would like our model to be capable to retrain in real-time as well, since it might be profitable to use it with drift adaptation for some datasets.

Based on the above observations, we decided to use PSO-ELM model with entropy-based threshold (PSO-ELM-ET). We removed normalization from the original implementation of PSO-ELM since this would require us to know and normalize point in future incoming time series points. Instead of using non-intersecting batches for entropy calculation, we used sliding batches with the stride of 1 on the test part of the time series, we calculate per-batch entropy mean and standard deviation values same way we did it before on disjoint batches. We used extensive hyperparameter tuning of our model where parameters such as entropy window size, factor f as well as some internal PSO-ELM hyperparameters like training window size have been tuned. We performed our hyperparameter optimization the same way it was done for OeSNN-UAD.

Table 5.17 shows performance of PSO-ELM-ET against previous models on all subsets of NAB dataset. As Table 5.17 indicates, our model is capable of reaching state-of-the-art overall performance as well as outperforms previous models on a number of NAB subset.

Finding 7: *Proposed PSO-ELM-ET model has comparable performance to the state-of-the-art detectors on NAB dataset. Yet, this model is not a universally best choice for other datasets.*

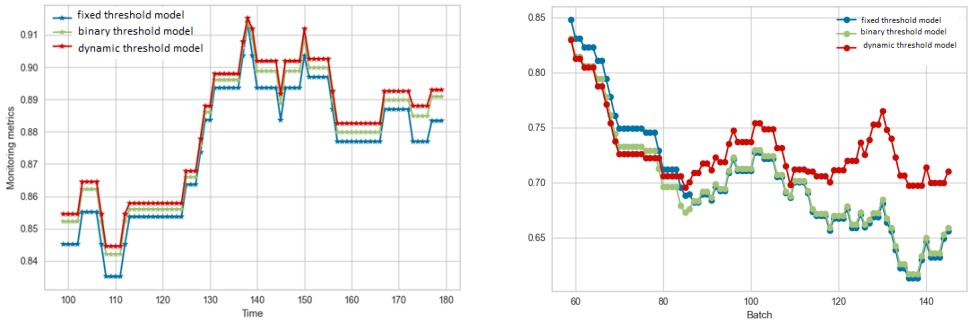
5.8. Evaluation of online models

In this section we would like to discuss evaluation routines used in online anomaly detection domain and their representativeness of actual model behavior after deployment.

The vast majority of current and previous studies on online anomaly detection uses single total F1-score as their measure of models' performance. This is exactly the reason why we used such evaluation technique in our study - to assess and score our models such that they can be compared directly with the existing detectors. In addition, some studies use single overall evaluation scores as precision, recall, balanced accuracy, and Matthews correlation coefficient [56], [39], [72]. All these scores reflect or combine proportions of correctly or wrongly classified anomalies by some detector within test part of the data.

Nevertheless, such single-value scores do not give any insight into actual models' performance during its life-cycle once deployed. For example, a model with average stable performance over its whole lifespan will yield nearly identical metrics as a model that initially had high accuracy or precision which drastically drops over time in production. Therefore, we looked into alternative ways of measuring models' performances and evaluating them over time that can give insight into how model behavior changes over time.

To the best of our knowledge, there is only very limited amount of studies that evaluates online anomaly detectors over time and use such matrices for detector comparisons. One immediate approach would be to look into per-batch predictor metricise over time, where for example F1-scores or precision are calculated on small disjoint but consequent batches of time series values. With this approach, it becomes possible to observe changes of predictive behavior of models over time and identify models with stable performance, or models which performance degrades over time. In addition, such approach still allows for identifying strictly superior models: such model will have all per-batch metrics above less-accurate models. We provide visual representations in Figure 5.6 (a) and (b).



(a) (b)

Figure 5.6: Per-batch F1-score of PSO-ELM model with various thresholding techniques on a KPI time series.

In Figure 5.6 a) model with dynamic threshold is superior to another two models on every batch, even though performance of the model is unstable over time interval. In such case, it can be concluded, that dynamic threshold model is the best choice out of 3 options for the selected time series. However, as Figure 5.6 b) shows, on a different time series there is no best-fitting model, since dynamic threshold model performs the worst in the beginning but gains advantage over time. Additionally, it can be observed that in case of time series from Figure 5.6 b), their performance starts decreasing almost immediately after the first test batch which a single F1-score over test set would not indicate.

Therefore, we think it is profitable to evaluate and track performance of online detectors via per-batch metrics over time. Then, the extent of the deviation within such metrics, overall trend, and average distance to another model metrics can be used to evaluate online model.

Finding 8: *Modern evaluation techniques for online anomaly detectors are not representative of a model lifespan when deployed. Nevertheless, when tracking performance over time, more complex analysis involving human-in-the-loop is needed to reliably assessing online performance.*

III

Discussion and Conclusions

5.9. Discussion

Within this work, we have structured existing anomaly detection techniques using one single general classification. We have selected the most popular methods out of each category and we have tested their applicability to various known public streaming benchmarks. We researched what variables that are present within the benchmark time series data, might have an influence on the predictive performance of anomaly detectors and their generalizability. We exploited our findings and proposed one final PSO-ELM-ET model: a combination of an already existing model based on particle swarm optimization of extreme learning machines and an entropy-dependent threshold technique that adjusts rigid anomaly threshold over a span of time based on batched entropies. Below, we discuss our findings in more detail and we draw corresponding conclusions.

We have found that some types of anomaly detectors do not adapt well to real-time streams: for instance, clustering, isolation, or distance-based methods did not show competitive performance in our experiments. Our best-performing online anomaly detectors include neural-based models, such as Auto-Encoders or Extreme Learning Machines, and signal-processing models such as SARIMA or Spectral Residuals. One reason for this might be that all of our selected base models make use of time dependency within the time series either via LSTM layers or by using time series characteristics as frequencies or seasonality during model creation and prediction.

Our **Finding 1** indicates that conventional anomaly detectors do not generalize well into various streaming scenarios as shown by their low performance on the joint set of anomaly detection benchmarks (F1-score ≤ 0.7). We did not find any strong patterns or accordance among models' performances: each benchmark has its own top-performing model which is not universal across other datasets.

Neither did we find any similarities or dissimilarities across publicly available benchmarks, that could be correlated to the detective performance of the models. **Finding 2** shows that we did not find any strong relation between particular time series features (independently of the dataset) and models' performances on the given data. Nevertheless, our analysis is highly limited by the number of publicly available time series and bias, if any, present within them. Our final set consists of 456 time series out of 3 public benchmarks with the vast majority of them (367) coming from Yahoo benchmark. Not only such a small amount of time series limits our correlation research on time series feature influence on models' performance, but particular dataset biases may lower result soundness. As an example, a recent study from 2021 [95] argues that modern anomaly detection time series benchmarks suffer from flaws such as unrealistic anomaly densities, mislabeled ground truth, or location of anomaly groups (that tends to be towards the very end of a time series). Such flaws also may have an influence on our results in general, and correlation analysis in particular. We reckon that one useful extension of our study might be to perform a large-scale correlation analysis with thousands of labeled time series.

Our **Finding 3** indicate that, in general, there is no consistent improvement of anomaly detectors when combined with drift detectors, however, individual anomaly detectors may exhibit a performance boost when combined with drift adaptation. We have also identified a set of the best-suitable drift detectors for particular models in application to streaming data. Our findings show that EDDM, ECDD, and HDDW adapt the best to our base anomaly detectors.

Our contribution consists of exploiting time series disorder, we have shown that it is feasible to decompose time series into per-batch entropy values over *short periods of time* with the window size suitable to our definition of near real-time anomaly detection (**Finding 4** and **Finding 5**), and that such short-period entropy indicators are capable of highlighting anomalous batches. We did not find a single overall window that yields the best performance over all benchmarks - in order to achieve the most representative entropy-decomposition values over time series batches, it is necessary to search for an optimal entropy window within such time series. We found entropy windows to be in a range of 35 to 85 time points, with some datasets completely failing to optimal window search and, therefore, being inapplicable to entropy analysis. We used such short-period entropy measures for dynamically adjusting rigid anomaly threshold measure in base models. We tested various entropy measures with singular values decomposition entropy proving to be the best choice in our case, leaving behind more popular choices for time series entropy analysis, such as permutation and spectral entropies. Such a result might be influenced by the short window size constraint: to the best of our knowledge, previous studies on entropy analysis within time series data applied it to significantly larger time periods.

Based on our complete exploratory analysis, we selected and proposed one final anomaly detection model for data streams (**Finding 6**). Our model did not show reliable accuracy on anomaly benchmarks (accuracy 0.7). Nevertheless, when tuned with optimization routines given within the literature, it is capable of showing performance comparable to state-of-the-art online anomaly detectors, as **Finding 7** indicates.

We believe that our findings indicate there is an overall little generalization of online anomaly detectors to different time series types, and that there is no one-suits-all solution. In order to achieve the highest performance, not only a detector should be tuned per time series, but also different detector types better capture anomalies in different time series. We selected PSO-ELM-ET model as the final choice due to it seemingly better applicability to lengthy and closer-to-real time series, as those in NAB or KPI detasetsas, well as due to its speed of training.

Lastly, we argue in **Finding 8** that a popular single F1-score measure over the test set is not representative of the true changing behavior of an anomaly detector in real time and, therefore, cannot be used alone to evaluate online anomaly detectors. We believe that per-batch metrics (as F1-score measure) capture changes in models' performance over their life cycle and should be used for assessing online anomaly detectors. Nonetheless, interpretation of such metrics is not always straightforward, it requires human-in-the-loop intervention for model comparison and it is highly use-case specific.

5.10. Future Work

Our analysis suffers from the limited amount of available data. Therefore, one interesting improvement of our work lies in applying our analysis to a larger set of time series. Such time series can be artificially generated or can be of real nature, yet, in our opinion, the real data type is the most interesting one. In addition, possible correlations between models' performance and types of data (real vs anomalous) can be researched as well as other data type groupings based on the granularity of time series or their nature (machine metrics, traffic metrics, etc.).

Additionally, our analysis works solely with univariate data whereas the applicability of our findings to the multivariate case might is of high interest.

Last but not the least, more in-depth research into possible evaluation metrics for online anomaly detectors will be a valuable input for the domain.

5.11. Conclusion

In our work, we researched the applicability of time series anomaly detectors to online scenarios with fully unsupervised learning. We have tested selected detectors out of several groups and we identified the best-performing models for univariate cases, namely, SARIMA, LSTM AE, PSO-ELM, and Spectral Residuals. However, our response to the first research question from Section 1.2 is we observed poor preservation of models' performance over different datasets.

To answer the second research question from Section 1.2, we analyzed various dependencies among performance scores of anomaly detectors, represented by F1-score on a full test set, and certain time series properties. We researched several categories of such properties, such as time series features, data drifts, and time series disorder, and their possible influence on models' performance. We found time series disorder, as measured via singular value decomposition entropy, to be the most promising and interesting direction for improving the performance preservation of anomaly detectors.

We proposed an adaptive threshold method to diminish time series disorder influence on the performance of anomaly detectors as the response to the last research question from Section 1.2. The method is capable of adjusting in near real-time in relation to the entropy measure of incoming time series batches, and we show its applicability to all base models on 2 out of 3 datasets. We also combined all our findings together by selecting one final model for anomaly detection in streaming data and by testing it against state-of-the-art benchmarks on NAB dataset to show the competitiveness of our method.

Lastly, we discuss possible problems and solutions in evaluating online anomaly detectors via single F1-score.

Feature	From Method	Description
DN_HistogramMode_5	Catch22	Mode of z-scored distribution (5-bin histogram)
DN_HistogramMode_10	Catch22	Mode of z-scored distribution (10-bin histogram)
CO_f1ecac	Catch22	First 1/e crossing of autocorrelation function
CO_FirstMin_ac	Catch22	First minimum of autocorrelation function
CO_HistogramAMI_even_2_5	Catch22	Automutual information, m = 2, τ = 5
CO_trev_1_num	Catch22	Time-reversibility statistic, $\langle(x_{t+1} - x_t)^3\rangle t$
MD_hrv_classic_pnn40	Catch22	Proportion of successive differences exceeding 0.04σ []
SB_BinaryStats_mean_longstretch1	Catch22	Longest period of consecutive values above the mean
SB_TransitionMatrix_3ac_sumdiagcov	Catch22	Trace of covariance of transition matrix between symbols in 3-letter alphabet
PD_PeriodicityWang_th0_01	Catch22	Periodicity measure of [92]
CO_Embed2_Dist_tau_d_expfit_meandiff	Catch22	Exponential fit to successive distances in 2-d embedding space
IN_AutoMutualInfoStats_40_gaussian_fmmi	Catch22	First minimum of the automutual information function
FC_LocalSimple_mean1_tauresrat	Catch22	Change in correlation length after iterative differencing
DN_OutlierInclude_p_001_mdrmd	Catch22	Time intervals between successive extreme events above the mean
DN_OutlierInclude_n_001_mdrmd	Catch22	Time intervals between successive extreme events below the mean
SP_Summaries_welch_rect_area_5_1	Catch22	Total power in lowest fifth of frequencies in the Fourier power spectrum
SB_BinaryStats_diff_longstretch0	Catch22	Longest period of successive incremental decreases
SB_MotifThree_quantile_hh	Catch22	Shannon entropy of two successive letters in equiprobable 3-letter symbolization
SC_FluctAnal_2_rsrangefit_50_1_logi_prop_r1	Catch22	Proportion of slower timescale fluctuations that scale with linearly rescaled range fits
SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1	Catch22	Proportion of slower timescale fluctuations that scale with DFA (50% sampling)
SP_Summaries_welch_rect_centroid	Catch22	Proportion of slower timescale fluctuations that scale with linearly rescaled range fits
FC_LocalSimple_mean3_stderr	Catch22	Mean error from a rolling 3-sample mean forecasting
hurst	FFORMA	Hurst exponent
seasonality	FFORMA	strength of seasonality
series_length	FFORMA	length of time series
unitroot_pp	FFORMA	test statistic based on Phillips-Perron test
unitroot_kpss	FFORMA	test statistic based on KPSS test
stability	FFORMA	stability
nperiods	FFORMA	number of seasonal periods in the series
seasonal_period	FFORMA	length of seasonal pertiod
trend	FFORMA	strength of trend
spike	FFORMA	spikiness
linearity	FFORMA	linearity
curvature	FFORMA	curvature
e_acf1	FFORMA	first ACF value of remainder series
e_acf10	FFORMA	sum of squares of first 10 ACF values of remainder series
x_pacf5	FFORMA	sum of squares of first 5 PACF values of original series
diff1x_pacf5	FFORMA	sum of squares of first 5 PACF values of differenced series
diff2x_pacf5	FFORMA	sum of squares of first 5 PACF values of twice-differenced series
nonlinearity	FFORMA	nonlinearity
lumpiness	FFORMA	lumpiness
alpha	FFORMA	ETS(A,A,N) $\hat{\alpha}$
beta	FFORMA	ETS(A,A,N) $\hat{\beta}$
flat_spots	FFORMA	number of flat spots, calculated by discretizing the series into 10 equal sized intervals and counting the maximum run length within any single interval
entropy	FFORMA	spectral entropy
crossing_points	FFORMA	number of times the time series crosses the median
arch_lm	FFORMA	ARCH LM statistic
x_acf1	FFORMA	first ACF value of the original series
x_acf10	FFORMA	sum of squares of first 10 ACF values of original series
diff1_acf1	FFORMA	first ACF value of the differenced series
diff1_acf10	FFORMA	sum of squares of first 10 ACF values of differenced series
diff2_acf1	FFORMA	first ACF value of the twice-differenced series
diff2_acf10	FFORMA	sum of squares of first 10 ACF values of twice-differenced series
hwalpha	FFORMA	ETS(A,A,A) $\hat{\alpha}$
hwbeta	FFORMA	ETS(A,A,A) $\hat{\beta}$
hwgamma	FFORMA	ETS(A,A,A) $\hat{\gamma}$
peak	FFORMA	strength of peak
through	FFORMA	strangth of through
arch_acf	FFORMA	sum of squares of the first 12 autocorrelations of z^2
garch_acf	FFORMA	sum of squares of the first 12 autocorrelations of r^2
arch_r2	FFORMA	R^2 value of an AR model applied to z^2
garch_r2	FFORMA	R^2 value of an AR model applied to r^2
seas_pacf	FFORMA	partial autocorrelation coefficient at first seasonal lag
sediff_acf1	FFORMA	first ACF value of seasonally differenced series

Table 18: Full set of time series features used for correlation analysis.

Bibliography

[1] Aws cloudwatch. URL https://aws.amazon.com/cloudwatch/.

[2] Luminol. URL https://github.com/linkedin/luminol.

[3] Minnesota department of transportation. URL https://www.dot.state.mn.us/.

[4] Prophet. URL https://research.facebook.com/blog/2017/02/prophet-forecasting-at-scale/.

[5] Cardiac software. URL https://www.medicalexpo.com/prod/northeast-monitoring/product-110013-732396.html.

[6] E*trade. URL https://us.etrade.com/home.

[7] Wikipedia fourier transform. URL https://en.wikipedia.org/wiki/Fast_Fourier_transform#/media/File:FFT_of_Cosine_Summation_Function.svg.

[8] Intensive care admission data from dutch government. URL https://coronadashboard.government.nl/landelijk/intensive-care-opnames.

[9] Wikipedia lstm. URL https://en.wikipedia.org/wiki/Long_short-term_memory.

[10] Sara Abdelghafar, Ashraf A. Darwish, Aboul Ella Hassanien, Mohamed Yahia, and A. A. S. Zaghrout. Anomaly detection of satellite telemetry based on optimized extreme learning machine. *Journal of Space Safety Engineering*, 6:291–298, 2019.

[11] Ryan P. Adams and David J. C. Mackay. Bayesian online changepoint detection. *arXiv: Machine Learning*, 2007.

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

[13] Mosabber Uddin Ahmed and Danilo P. Mandic. Multivariate multiscale entropy: a tool for complexity analysis of multichannel data. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 84 6 Pt 1:061918, 2011.

[14] Daniel B. Araya, Katarina Grolinger, Hany F. ElYamany, Miriam A. M. Capretz, and Girma T. Bitsuamlak. Collective contextual anomaly detection framework for smart buildings. *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 511–518, 2016.

[15] Manuel Baena-Garc, José Avila, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. Early drift detection method. 2005.

[16] Christoph Bandt and Bernd Pompe. Permutation entropy: a natural complexity measure for time series. *Physical review letters*, 88 17:174102, 2002.

[17] F. S. Bao, Xin Liu, and Christina Zhang. Pyeeg: An open source python module for eeg/meg feature extraction. *Computational Intelligence and Neuroscience*, 2011, 2011.

[18] Vance W. Berger and Yanyan Zhou. Kolmogorov–smirnov tests. 2005.

[19] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SDM*, 2007.

[20] Isvani Inocencio Frías Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales Bueno, Agustín Alejandro Ortiz Díaz, and Yailé Caballero Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27:810–823, 2015.

[21] Thomas Kelman Boehme and Ronald N. Bracewell. The fourier transform and its applications. *American Mathematical Monthly*, 73:685, 1966.

[22] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. Sand in action: Subsequence anomaly detection for streams. *Proc. VLDB Endow.*, 14:2867–2870, 2021.

[23] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. SIGMOD '00, page 93–104, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132174. doi: 10.1145/342009.335388. URL `https://doi.org/10.1145/342009.335388`.

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

[25] Petre Caraiani. The predictive power of singular value decomposition entropy for stock market dynamics. *Physica A-statistical Mechanics and Its Applications*, 393:571–578, 2014.

[26] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. 2007.

[27] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, 2009.

[28] Dipankar Dasgupta and L.F. Nino. A comparison of negative and positive selection algorithms in novel pattern detection. *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, 1:125–130 vol.1, 2000.

[29] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The ucr time series classification archive, October 2018. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`.

[30] Jürgen Fell, Joachim Röschke, Klaus Mann, and Cornelius Schäffner. Discrimination of sleep stages: a comparison between spectral and nonlinear eeg measures. *Electroencephalography and clinical neurophysiology*, 98 5:401–10, 1996.

[31] Ralph Foorthuis. On the nature and types of anomalies: a review of deviations in data. *International Journal of Data Science and Analytics*, pages 1 – 35, 2021.

[32] Ben D. Fulcher and Nick S. Jones. hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell systems*, 5 5:527–531.e3, 2017.

[33] João Gama, Pedro Medas, Gladys Castillo, and Pedro Pereira Rodrigues. Learning with drift detection. In *SBIA*, 2004.

[34] Robert M. Gray. Entropy and information theory. In *Springer New York*, 1990.

[35] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10, 2016.

[36] Trent Henderson and Ben D. Fulcher. An empirical evaluation of time-series feature sets. *ArXiv*, abs/2110.10914, 2021.

[37] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[38] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[39] Ganghui Hu, Jing Wang, Yunxiao Liu, Wang Ke, and Youfang Lin. Ccad: A collective contextual anomaly detection framework for kpi data stream. *Communications in Computer and Information Science*, 2021.

[40] Guangbin Huang, Qin-Yu Zhu, and Chee Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501, 2006.

[41] Rob J. Hyndman, Earo Wang, and Nikolay Pavlovich Laptev. Large-scale unusual time series detection. *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, pages 1616–1619, 2015.

[42] Massimiliano Ignaccolo, Miroslaw Latka, Wojciech Jernajczyk, Paolo Grigolini, and Bruce J. West. The dynamics of eeg entropy. *Journal of Biological Physics*, 36:185–196, 2010.

[43] Meenal Jain and Gagandeep Kaur. Distributed anomaly detection using concept drift detection based hybrid ensemble techniques in streamed network data. *Clust. Comput.*, 24:2099–2114, 2021.

[44] Herbert F. Jelinek, Luke A. Donnan, and Ahsan H. Khandoker. Singular value decomposition entropy as a measure of ankle dynamics efficacy in a y-balance test following supportive lower limb taping. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2439–2442, 2019.

[45] Fan Jiang, Ying Wu, and Aggelos K. Katsaggelos. Detecting contextual anomalies of crowd motion in surveillance video. *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 1117–1120, 2009.

[46] Baihong Jin, Yuxin Chen, Dan Li, Kameshwar Poolla, and Alberto Sangiovanni-Vincentelli. A one-class support vector machine calibration method for time series change point detection. In *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–5, 2019. doi: 10.1109/ICPHM.2019.8819385.

[47] Carl F. Kossack and M. G. Kendall. Rank correlation methods. *American Mathematical Monthly*, 57:425, 1950.

[48] Nikolay Pavlovich Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[49] Alexander Lavin and Subutai Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 38–44, 2015.

[50] Longyuan Li, Junchi Yan, Haiyang Wang, and Yaohui Jin. Anomaly detection of time series with smoothness-inducing sequential variational auto-encoder. *IEEE Transactions on Neural Networks and Learning Systems*, 32(3):1177–1191, 2021. doi: 10.1109/TNNLS.2020.2980749.

[51] Benjamin Lindemann, Benjamin Maschler, Nada Sahlab, and Michael Weyrich. A survey on anomaly detection for technical systems using lstm networks. *Comput. Ind.*, 131:103498, 2021.

[52] F. Liu, K. Ting, and Z. Zhou. Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[53] Xiaofeng Liu, Aimin Jiang, N. Xu, and Jianru Xue. Increment entropy as a measure of complexity for time series. *Entropy*, 18:22, 2016.

[54] Carl Henning Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, 33:1821 – 1852, 2019.

[55] Ya lun Chou. Statistical analysis, with business and economic applications. 1975.

[56] Piotr S. Maciag, Marzena Kryszkiewicz, Robert Bembenik, and Jesús López Lobo. Unsupervised anomaly detection in streamdatawith online evolving spiking neural networks. 2021.

[57] Ruchi Makani and B.V.R. Reddy. Taxonomy of machine leaning based anomaly detection and its suitability. *Procedia Computer Science*, 132:1842–1849, 2018. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.05.133. URL https://www.sciencedirect.com/science/article/pii/S1877050918308652. International Conference on Computational Intelligence and Data Science.

[58] S. David McSwain, Donna S Hamel, P. Brian Smith, Michael A. Gentile, Saumini Srinivasan, Jon N. Meliones, and Ira M Cheifetz. End-tidal and arterial carbon dioxide measurements correlate across all levels of physiologic dead space. *Respiratory care*, 55 3:288–93, 2010.

[59] H. Zare Moayedi and M. A. Masnadi-Shirazi. Arima model for network traffic prediction and anomaly detection. *2008 International Symposium on Information Technology*, 4:1–6, 2008.

[60] Pablo Montero-Manso, George Athanasopoulos, Rob J. Hyndman, and Thiyanga S. Talagala. Fforma: Feature-based forecast model averaging. *International Journal of Forecasting*, 36:86–92, 2020.

[61] Fabian Mörchen. Time series feature extraction for data mining using dwt and dft. 2003.

[62] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas R. Dengel, and Sheraz Ahmed. Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, 7:1991–2005, 2019.

[63] Gustavo H. F. M. Oliveira, Rodolfo Carneiro Cavalcante, George G. Cabral, Leandro L. Minku, and Adriano Oliveira. Time series forecasting in the presence of concept drift: A pso-based approach. *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 239–246, 2017.

[64] Salima Omar, Asri Md. Ngadi, and Hamid H. Jebur. Machine learning techniques for anomaly detection: An overview. *International Journal of Computer Applications*, 79:33–41, 2013.

[65] Guansong Pang, Chunhua Shen, L. Cao, and A. V. Hengel. Deep learning for anomaly detection. *ACM Computing Surveys (CSUR)*, 54:1 – 38, 2021.

[66] Clifton Phua, Kate Smith-Miles, Vincent Cheng-Siong Lee, and R. Gayler. Adaptive spike detection for resilient data stream mining. In *AusDM*, 2007.

[67] Phyks. Introducing practical and robust anomaly detection in a time series | twitter blogs. 2015.

[68] Steven M. Pincus. Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences of the United States of America*, 88:2297 – 2301, 1991.

[69] Riccardo Poli, James Kennedy, and Tim M. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007.

[70] Yu Qin and YuanSheng Lou. Hydrological time series anomaly pattern detection based on isolation forest. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1706–1710, 2019. doi: 10.1109/ITNEC.2019.8729405.

[71] Jamal Raiyn and Tomer Toledo. Real-time road traffic anomaly detection. *Journal of Transportation Technologies*, 04:256–266, 2014.

[72] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Q. Zhang. Time-series anomaly detection service at microsoft. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[73] Joshua Richman and J. Randall Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American journal of physiology. Heart and circulatory physiology*, 278 6: H2039–49, 2000.

[74] Bernard A Rosner. Percentage points for a generalized esd many-outlier procedure. *Technometrics*, 25: 165–172, 1983.

[75] Gordon J. Ross, Niall M. Adams, Dimitris K. Tasoulis, and David J. Hand. Exponentially weighted moving average charts for detecting concept drift. *ArXiv*, abs/1212.6018, 2012.

[76] Sakti Saurav, Pankaj Malhotra, T. R. Vishnu, Narendhar Gugulothu, Lovekesh Vig, Puneet Agarwal, and Gautam M. Shroff. Online anomaly detection with concept drift adaptation using recurrent neural networks. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, 2018.

[77] Karsten Schmidt. Computing the moore–penrose inverse of a matrix with a computer algebra system. *International Journal of Mathematical Education in Science and Technology*, 39:557 – 562, 2008.

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

[79] Kamran Shaukat, Talha Mahboob Alam, Suhuai Luo, Shakir Shabbir, Ibrahim A. Hameed, Jiaming Li, Syed Konain Abbas, and Umair Javed. A review of time-series anomaly detection techniques: A step to future perspectives. 2021.

[80] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouët. Anomaly detection in streams with extreme value theory. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[81] Nidhi Singh and Craig Olinsky. Demystifying numenta anomaly benchmark. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1570–1577, 2017.

[82] M. Smirnov. Contextual anomaly detector, 2016. URL `https://github.com/smirmik/CAD`.

[83] Vladimir N. Soloviev, Andrii O. Bielinskyi, and Viktoria Solovieva. Entropy analysis of crisis phenomena for djia index. In *ICTERI Workshops*, 2019.

[84] A. Stanway. Etsy skyline, 2015. URL `https://github.com/etsy/skyline`.

[85] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[86] Thiyanga S. Talagala, Rob J. Hyndman, and George Athanasopoulos. Meta-learning how to forecast time series. 2018.

[87] Mingyan Teng. Anomaly detection on time series. *2010 IEEE International Conference on Progress in Informatics and Computing*, 1:603–608, 2010.

[88] Mingyan Teng. Anomaly detection on time series. In *2010 IEEE International Conference on Progress in Informatics and Computing*, volume 1, pages 603–608, 2010. doi: 10.1109/PIC.2010.5687485.

[89] Maurras Ulbricht Togbe, Yousra Chabchoub, Aliou Boly, Mariam Barry, Raja Chiky, and Maroua Bahri. Anomalies detection using isolation in concept-drifting data streams. *Comput.*, 10:13, 2021.

[90] Chengwei Wang, Krishnamurthy Viswanathan, Choudur K. Lakshminarayan, Vanish Talwar, Wade Satterfield, and Karsten Schwan. Statistical techniques for online anomaly detection in data centers. *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 385–392, 2011.

[91] Xiaozhe Wang, Kate Smith-Miles, and Rob J. Hyndman. Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery*, 13:335–364, 2005.

[92] Xiaozhe Wang, Anthony Wirth, and Liang Wang. Structure-based statistical features and multivariate time series clustering. *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 351–360, 2007.

[93] Xiaozhe Wang, Kate Smith-Miles, and Rob J. Hyndman. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomputing*, 72:2581–2594, 2009.

[94] Robert F. Woolson. Wilcoxon signed-rank test. 2005.

[95] Renjie Wu and Eamonn J. Keogh. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *ArXiv*, abs/2009.13807, 2021.

[96] Ning Xinbao. Multiscale entropy analysis of complex physiologic time series. *Beijing Biomedical Engineering*, 2007.

[97] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 187–196. International World Wide Web Conferences Steering Committee, 2018.

[98] Rongbin Xu, Yongliang Cheng, Zhiqiang Liu, Ying Xie, and Yun Yang. Improved long short-term memory based anomaly detection with concept drift adaptive method for supporting iot services. *Future Generation Computer Systems*, 112:228–242, 2020. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2020.05.035. URL `https://www.sciencedirect.com/science/article/pii/S0167739X20302235`.

[99] Yin Zhang, Zihui Ge, Albert G. Greenberg, and Matthew Roughan. Network anomography. In *IMC '05*, 2005.

[100] Puning Zhao and Lifeng Lai. Analysis of knn density estimation. *ArXiv*, abs/2010.00438, 2020.