Anomaly Detection in Microservice Infrastructures

JONATHAN OHLSSON
Anomaly Detection in Microservice Infrastructures

JONATHAN OHLSSON
Abstract

Anomaly detection in time series is a broad field with many application areas, and has been researched for many years. In recent years the need for monitoring and DevOps has increased, partly due to the increased usage of microservice infrastructures. Applying time series anomaly detection to the metrics emitted by these microservices can yield new insights into the system health and could enable detecting anomalous conditions before they are escalated into a full incident.

This thesis investigates how two proposed anomaly detectors, one based on the RPCA algorithm and the other on the HTM neural network, perform on metrics emitted by a microservice infrastructure, with the goal of enhancing the infrastructure monitoring. The detectors are evaluated against a random sample of metrics from a digital rights management company’s microservice infrastructure, as well as the open source NAB dataset.

It is illustrated that both algorithms are able to detect every known incident in the company metrics tested. Their ability to detect anomalies is shown to be dependent on the defined threshold value for what qualifies as an outlier. The RPCA Detector proved to be better at detecting anomalies on the company microservice metrics, however the HTM detector performed better on the NAB dataset. Findings also highlight the difficulty of manually annotating anomalies even with domain knowledge. An issue found to be true for both the dataset created for this project, and the NAB dataset.

The thesis concludes that the proposed detectors possess different abilities, both having their respective trade-offs. Although they are similar in detection accuracy and false positive rates, each has different inert abilities to perform tasks such as continuous monitoring or ease of deployment in an existing monitoring setup.
Sammanfattning

Anomalitetsdetektering i tidsserier är ett brett område med många användningsområden och har undersökts under många år. De senaste åren har behovet av övervakning och DevOps ökat, delvis på grund av ökad användning av microservice-infrastrukturer. Att tillämpa tidsserieanomalitetsdetektering på de mätvärden som emitteras av dessa microservices kan ge nya insikter i systemhälsan och kan möjliggöra detektering av avvikande förhållanden innan de eskaleras till en fullständig incident.


Avhandlingen slutleder att de föreslagna detektorerna har olika förmågor, vilka båda har sina respektive avvägningar. De har liknande detekteringsnoggrannhet, men har olika inerta förmågor för att utföra uppgifter som kontinuerlig övervakning, eller enkelhet att installera i en befintlig övervakningsinstallation.
# Contents

1 Introduction ........................................... 1  
   1.1 Brief background .................................. 1  
   1.2 Microservice infrastructure ....................... 2  
   1.3 Microservice monitoring .......................... 3  
   1.4 Research Question ................................ 4  
   1.5 Objectives ........................................ 4  
   1.6 Scope .............................................. 5  
   1.7 Structure ........................................... 5  

2 Background ............................................ 6  
   2.1 Anomaly Detection .................................. 6  
   2.2 Classes of anomalies ............................... 7  
   2.3 Time Series Classification ........................ 9  
   2.4 Data reduction as anomaly detection ............... 11  
      2.4.1 PCA ........................................... 11  
      2.4.2 RPCA .......................................... 12  
      2.4.3 Previous work ................................ 14  
   2.5 Time Series Prediction ............................. 14  
      2.5.1 Hierarchical temporal memory ............... 15  
      2.5.2 Detecting anomalies using time series prediction .......................... 17  
   2.6 Normalization ...................................... 19  
   2.7 Evaluation ......................................... 20  

3 Method ................................................ 22  
   3.1 Dataset ............................................. 22  
      3.1.1 Annotating the dataset ....................... 23  
   3.2 Anomaly detector benchmarking system ............ 23  
   3.3 Anomaly detector implementations ................ 24  
      3.3.1 HTM ............................................ 24
3.3.2 RPCA .................................................. 25
3.4 Evaluation .............................................. 26
   3.4.1 Judging correct detections .................. 27
4 Results .................................................. 29
   4.1 Detector time performance .................... 29
   4.2 Detector results ................................ 30
   4.3 Classification Scores ............................ 30
   4.4 Anomaly windows detected ................... 30
   4.5 Incident Detection ............................... 31
5 Discussion .............................................. 37
   5.1 Dataset .............................................. 37
      5.1.1 NAB dataset accuracy ...................... 38
   5.2 Performance ....................................... 38
   5.3 Binary classifier measures ................... 38
   5.4 Anomaly window detection .................... 40
   5.5 Found Incidents .................................. 40
   5.6 Ethical concerns ................................ 41
6 Conclusions ............................................ 42
   6.1 Reduce cognitive load ......................... 43
   6.2 Continuous monitoring ......................... 43
   6.3 Future work ...................................... 44

Bibliography ............................................. 46
Chapter 1

Introduction

This chapter gives an introduction to the purpose and objective of the thesis, and why the results are relevant for anomaly detection research and to the industry.

1.1 Brief background

Detecting anomalies in modern distributed systems has shown itself to be a non-trivial issue. Detecting outliers, or anomalies, has been studied as early as in 1887 [14], and due to the diverse nature of data to analyze for anomalies there still is no de facto method for anomaly detection. Even for anomaly detection within a specific domain such as anomaly detection in time series, there is a huge landscape of ideas and models to detect anomalies of different kinds. Furthermore, time series data have characteristics such as seasonality, trends and noise, making true anomalies hard to define and even harder to find.

In order to operate any system it is necessary to be able to detect anomalous conditions and possible root causes. This is usually referred to as monitoring, and it is accomplished by instrumenting the software and having an external system that collects and processes the signals provided by the system under operation. It is not uncommon for a system in operation to emit hundreds of different signals per instance, making the total amount of signals to be monitored a very large number, sometimes several orders of magnitude larger than the amount of signals emitted by a single system instance.

The ultimate goal of monitoring is to correctly detect erroneous conditions before there is any business impact, while providing suf-
ficient information for the problem to be remedied.

It is obviously not an easy task to detect, from these huge sets of signals, when the system is behaving abnormally. One way of doing this is to create alerts for incidents modelled according to complex rules applied to the emitted signals. This means that the monitoring system is on the lookout for known failure scenarios. Unknown failure scenarios are usually modelled after an incident and a post-mortem analysis has taken place. It is therefore a manual process after the fact. An additional approach is to visually monitor signals using graphs. However, cognitive load places a limit on the number of graphs that can be dealt with.

If these systems could be monitored without the need for manually defined anomalies and failure scenarios, it would enable the system to scale more reliably and would make the huge number of signals more manageable. Also if the dashboards could prioritize content of interest, the cognitive load would be reduced.

1.2 Microservice infrastructure

The system being investigated in this thesis is built using a microservice architecture. Using a microservice architecture provides several benefits that are of great interest when building large scale software. Some of the benefits of a microservice architecture identified by Dragoni et al. [13] are:

- **Flexibility.** The system is able to keep up with the ever-changing business environment and is able to support all modifications that are necessary for an organization to stay competitive on the market.

- **Modularity.** The system is composed of isolated components where each component contributes to the overall system behaviour rather than having a single component that offers full functionality.

- **Evolution.** The system stays maintainable while constantly evolving and adding new features.

Like Dragoni et. al outlined above, microservice architectures enable loosely coupled designs with clear ownership of the different components, while interactions between services are governed by APIs. This
way different teams can be fully responsible for different parts of the system during the whole development cycle. Large microservice architectures, like the one investigated in this thesis, are composed of thousands of distinct microservices. Each microservice has many instances deployed in order to cope with load or to reduce latency for end users due to geographical reasons.

Building a large scale microservice architecture does introduce extra complexity [13]. The microservices need to integrate with each other, and potentially hundreds of other services. There might be dependencies between services, and thus they need to be orchestrated in correct configurations. Failures in one microservice might have a domino effect, triggering failures in other parts of the system.

Microservice architectures and DevOps practices are becoming standard in the Internet-scale software industry [5, 15]. These two concepts are key in supporting fast iteration and deployment of changes in production systems that serve millions and billions of requests per second from users all around the world, and therefore need to be highly scalable and support seamless updates.

1.3 Microservice monitoring

The microservices emit periodical data that give insights into the health and performance of the services. These signals form time series being used for monitoring. The metrics might have rules applied to them, like for example threshold values to tell if the values are outside the predefined normal behaviour. They also form dashboards of graphs for visual inspection by engineers. However, analyzing time series is an active area of research and more advanced methods for anomaly detection have been proposed through the years. The demand for analyzing time series data to detect incidents have become so necessary it’s bloomed into a full industry with companies like Datadog [12], Metricly [27] and Unomaly [39] offering monitoring as a service. Meanwhile web giants like Netflix [40], LinkedIn [23] and Twitter [38] have published their anomaly detection methods. All these companies have their own models for anomaly detection, and in recent years machine learning approaches have begun to show promise in the field. One example of this is the work by the research group Numenta [30]. Numenta has developed a neural network model called HTM which
have shown promising performance in anomaly detection \[2\], outperforming other neural network architectures like the LSTM architecture. However, all publications found on the HTM have been written by Numenta. No independent trial have been found.

Also recently, the RPCA algorithm has received attention for anomaly detection. This is the algorithm underlying Netflix anomaly detection library \[28\], however Netflix is the only source found that uses RPCA to find anomalies in time series, and they give no indication of how effective it is at this task. Historically RPCA has mainly been used for image manipulation such as background and foreground extraction, and network anomaly detection.

### 1.4 Research Question

How does an anomaly detector based on the RPCA algorithm compare to an anomaly detector based on the HTM neural network, and how do these anomaly detectors perform on real world data from the company’s microservice infrastructure?

### 1.5 Objectives

The objective of this thesis is to compare how the HTM neural network compares to RPCA as an aid in monitoring of microservice infrastructure. This requires:

- Detecting anomalies
- Detecting incidents
- Avoid falsely labeling anomalies

The two use cases for which the algorithms will be evaluated are

- Highlighting anomalous behaviour in order to reduce cognitive load during manual inspection
- Continuous monitoring for automatic anomaly detection
1.6 Scope

This thesis will focus on evaluating how well the detectors can identify anomalous behaviour within a single time series. How the detectors can be used for large scale analysis will be discussed, although the implementations will not take the demands of performing scalable analysis into account. Other domains such as collective anomalies across multiple time series, and real time anomaly detection with streaming data is beyond the scope of this thesis.

1.7 Structure

This report is divided into 6 chapters. In chapter 2 anomaly detection is more formally defined. Background is provided for previous work in general anomaly detection, classes for anomaly types are defined, along with limitations of previous work. Background theory to the utilized anomaly detector methods is explained. Finally theory on the evaluation of the anomaly detector is explained. In chapter 3 the dataset creation is explained along with implementation details of the constructed anomaly detectors. In chapter 4 the anomaly detector output is shown along with the results of running the evaluation metrics outlined in the background chapter. In chapter 5 results are highlighted along with limitations of the results gathered. In chapter 6 the anomaly detectors are judged on how well they are suited for the objectives outlined above. Finally some recommendations and inspiration is provided on how to build upon the foundations laid out by this thesis.
Chapter 2

Background

This chapter will give background to anomaly detection, some terminol-
yogy and related work in time series analysis and anomaly detec-
tion. First the general topic of anomaly detection is introduced in sec-
tion 2.1. Secondly the peculiarities of time series anomaly detection is
introduced in section 2.2. Then previous work for time series analysis
using time series classification is introduced in section 2.3 followed
by background on dimensionality reduction as a means to anomaly
detection in section 2.4. This is followed by background on predictive
models as a means to detect anomalies, and how predictions can be
used to detect anomalies, in section 2.5. Lastly some data processing
is introduced, first for normalization in section 2.6 and finally for eval-
uation in section 2.7.

2.1 Anomaly Detection

Anomaly detection refers to the problem of detecting patterns in data
that do not conform to normal behaviour. These patterns are referred
to as anomalies, outliers, discordant observations, exceptions, aber-
trations, surprises, peculiarities, or contaminants depending on the do-
main. Anomaly detection has wide application areas such as intrusion
detection in cyber-security, abnormal condition in health care and fault
detection in critical systems [7].

Following the above definition of an anomaly, it might seem simple
to detect anomalies in a system. A straight forward approach would
be to define a region of data that conforms to normal behaviour, and
declare any data that does not mach this region to be anomalous. How-
ever there are many challenges.

- Defining a normal region that encompasses every possible normal behavior is difficult \[^7\].

- The availability of labeled data for training/validation of models used by some anomaly detection techniques and time series classification is usually a major issue \[^7\,\,21\].

- The notion of an anomaly is different for different domains. For example minor fluctuations in body temperature could be a serious anomaly, where the same fluctuation in another domain like the stock market would be perfectly normal \[^7\].

Thus, anomaly detection techniques from one domain do not necessarily work in other domains. In many domains normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future, for example applying anomaly detection to a large scale microservice infrastructure. The nature and behaviour of services change continuously - dependent services are added and removed, service resources are dynamically scaled up and down, and the location of the service might change. All of these factors might affect how the service functions. These are just a few of the challenges with detecting abnormal behavior.

### 2.2 Classes of anomalies

Anomalies in data can take many forms. To provide a unified notion of different kinds of anomalies the following definition of anomalies is that of Chandola et. al \[^7\]. This divides anomalies into three main classes. An anomaly detection algorithm for time series will preferably be able to detect all these classes of anomalies.

**Point anomalies**

*Point anomalies* are data points that are different from normal data points, most researched algorithms concern this type of anomaly. Consider Figure 2.1 where the normal data would be \(c_1\) and \(c_2\). Two anomalies can easily be detected visually as \(x_1\) and \(x_2\). This is data that is significantly different from all other data, and are often referred to as
global anomalies. Considering point \( x_3 \) it does not conform to the notion of a global anomaly. It seems to be similar to the cluster \( c_2 \). However looking at the local data points in \( c_2 \), ignoring the global context, we consider \( x_3 \) to be a local anomaly. The last kind of anomaly found in Figure 2.1 are the points around \( c_3 \), the so called micro cluster. These are all data points that lie outside of the classified normal data.

![Figure 2.1: A simple two-dimensional example. It illustrates global anomalies \((x_1, x_2)\), a local anomaly \(x_3\) and a micro-cluster \(c_3\).](image)

Collective anomalies

A second class of anomalies are collective anomalies. This is a class of anomalies where the individual data points are not anomalies by themselves, however their occurrence together as a collection is anomalous. This can for example be a sequence of actions in a web server:

```latex
... http-web, buffer-overflow, http-web, ftp, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, ssh, buffer-overflow, ftp, http-web, ftp, smtp-mail, http-web...
```

where each individual event in the bolded sequence \textbf{ssh, buffer-overflow, ftp} would not be anomalous, however the collection together bear the indication of an intrusion.
Contextual anomalies

Time series data is context dependent and thus have a class of anomalies called contextual anomalies or temporal anomalies [2, 7], since each value is related to other values in time. Each value has a time attribute associated with it that determines its position in the whole series. This attribute adds constraints to how anomalies can be detected. As an example, see the fictitious depiction of the outdoor temperature over a year in Figure 2.2. The temperature $t_2$ is normal during summer, however during winter it would be an anomaly. This means we can not think of this as a simple point anomaly, since the value itself is not uncommon.

![Temperature vs Time]

Figure 2.2: Points $t_1$ and $t_2$ having the same value but during different times in the natural seasonality of the time series.

2.3 Time Series Classification

Some approaches to time series anomaly detection is to apply time series classification (TSC). Classification is the task of trying to map an input space to some discrete label, or a class [3]. There are many algorithms for classification, many hundred made specifically for analyzing time series data. There are however issues with TSC. One issue
is that most classification models use a supervised learning approach [1] and thus require labeled data, another is the need to find features in the time series data [9]. There are also issues with how research has been conducted in the area since there has been no universal evaluation [3] [18], and worse - often times new published approaches have only been evaluated on synthetic data sets with the purpose of showcasing the algorithm [3]. Attempts to remedy this has been made, for example the UCR archive [8] with labeled time series data, and a dataset compiled by Goldstein and Uchida [18] available on Harvard Dataverse [17]. However all evaluations are still carried out on the same train/test split, and classification algorithms assume patterns of interest can be correctly identified, both during the training phase and during deployment [21].

Detecting an anomaly in a time series can be seen as a binary classification task. However due to its supervised nature, most TSC algorithms will be unable to perform automatic detection of anomalies in an unlabeled dataset.

Unsupervised approaches can overcome this shortcoming, since they do not rely on labeled data for classification. Nearest Neighbor approaches have been found to be successful in anomaly detection and K-NN has been found to be the most accurate method for anomaly detection by independent researchers [7] [18] using different datasets. For example using DTW, a similarity measure for time series, combined with 1-NN was found accurate by Chandola et al. [7]. Goldstein and Uchida [18] used K-NN on multivariate data and not on time series data. Their method could still be applied to time series if preprocessing was used to extract multivariate features from the series. Proposed methods for feature extraction in time series is explained in more detail later in this section. Goldstein and Uchida also evaluated other clustering methods to find outliers, they found the CBLOF algorithm [21] and the modified uCBLOF to yield good results for global outlier detection.

Finding features

Not all classification and regression algorithms can operate on the raw time series data, they need to first extract features to be effective and one of the limiting factors when predicting a time series is the absence of features [2] [35]. Finding features in a time series is not a trivial
task. Typically time series data contain noise and redundancies \cite{9}. Some features, for example median value, will be robust and will not be affected much by outliers. Other features like max-value are intrinsically fragile. The choice of the right characteristics for a time series is crucial for the task of classification. Christ et al. \cite{9} has proposed FRESH, FeatuRe Extraction based on Scalable Hypothesis tests, a feature extraction pipeline with highly scalable performance built for classification and regression. Christ et al. showed that their feature selection performed well on binary classification, and they hypothesize that their feature selection would be accurate if applied to regression and multi-class classification problems as well.

The feature selection process involved mapping each time series to the a feature space, and then using statistical models to perform hypothesis tests and only keep features that seemed to indicate relevance. This way they reduced the number of features relevant for a time series, only retaining the most relevant ones.

\section{Data reduction as anomaly detection}

The goal of data reduction is to represent it more compactly. There are different reasons for doing this, for example when the data size is smaller it is cheaper to apply computationally expensive algorithms, as well as improving the algorithms performance. An example would be feature extraction in the previous section. However in this section the focus is on data reduction as a means of detecting anomalies.

\subsection{PCA}

Spectral techniques for anomaly detection try to approximate the data by capturing the bulk of data variability by reducing the dimensionality of the data. The key assumption is that the data can be embedded in a lower dimensional space in which normal data instances and anomalous data appears vastly different \cite{7}. See Figure 2.3 for a visual example. The most common approach to this is the Principal Component Analysis (PCA) \cite{6,7}, which decomposes an initial data matrix $M$ the following way:

$$M = L + N$$
where $L$ has low-rank and $N$ is a small perturbation matrix [6]. To solve this decomposition PCA seeks the best (in an $l^2$ sense) rank-$k$ estimation of $L$ by solving

$$\text{minimize} \quad ||M - L||$$

subject to \quad rank(L) \leq k

This can be solved efficiently using methods such as singular value decomposition (SVD) [6]. However, PCA is brittle when dealing with grossly corrupted data causing the found $L$ to be arbitrarily far from the real $L$. Corrupted data can be due to sensor failures, occlusions or simply irrelevant data to the low dimension representation [6].

![Figure 2.3: Example of PCA where data is projected to a lower dimensional space [34]. The 3-dimensional points are projected onto a 2 dimensional surface.](image)

### 2.4.2 RPCA

A more robust PCA (RPCA) has received attention for anomaly detection in time series since Netflix published an article about their anomaly detection [40]. Their approach is based on RPCA from [6], using Principal Component Pursuit by Alternating Directions (PCP-AD). Similar to the PCA algorithm above, an initial matrix $M$ is decomposed into two components:

$$M = L + S$$
where $L$ is low-rank and $S$ is sparse. Effectively $S$ contains the extracted anomalies from the original data. The task is to find a low rank matrix $L$ and a sparse matrix $S$ without any prior knowledge of the true rank of $L$ or the number of entries in $S$. It can be stated as an optimization problem like the following

$$\min_{L,S} \rho(L) + \lambda ||S||_0$$

subject to  

$$|M - (L + S)| = 0$$

where $\rho(L)$ is the rank of $L$, $||S||_0$ is the number of non-zero entries in $S$ and $\lambda$ is the coupling constant which controls the trade-off between the low-rank matrix $L$ and the sparse $S$. Brute forcing $S$ and $L$ requires testing all possible combinations of anomalies in $S$ and low-rank $L$ and would be NP-hard [33]. Candes et. al. showed in Theorem 1.2 in Robust Principal Component Analysis [6] that you actually can give guaranties for finding $S$ and $L$ provided some conditions are met [33]:

- Bounded rank of $L$
- Bounded sparsity of $S$
- Require the columns of $L$ to be incoherent far from the standard basis
- Require the non-zero entries in $S$ to be uniformly distributed

Assuming the conditions are met, the problem can be restated as the convex program

$$\min_{L,S} ||L||_* + \lambda ||S||_1$$

subject to  

$$|M - (L + S)| \leq \epsilon$$

where $||L||_*$ is the sum of singular values in $L$ and $||S||_1 = \sum_{ij} |S_{ij}|$. $\lambda$ is the same as in (2.1), and $\epsilon$ is a set of point-wise error constraints used to ameliorate the noise. These kinds of convex problems can be solved efficiently [6, 33].

Note that the sought after goal in this context is not recovering $S$ and $L$, but rather to detect anomalies. The parameters $\epsilon$ and $\lambda$ used in

\[3\]i.e. the nuclear norm, or $L^1$-norm, of $L$
the algorithm are therefore not used to achieve prefect recovery but to perform the best anomaly detection.

2.4.3 Previous work

PCA and other robust PCA methods have been applied to areas such as network intrusion detection [22, 36, 37] and anomaly detection in spacecraft components [16]. Other more recent work building on the work of Candes et. al. is for example anomaly detection in cyber networks [33]. Other work as previously motioned, is video foreground and background separation, and facial shadow removal [6].

2.5 Time Series Prediction

Another approach to detecting anomalies in time series is to predict future values. By creating a model capable of predicting the time series accurately, anomalies can be detected by looking at the difference between the prediction and the actual value. Using predictive models as anomaly detection does however require the model to be accurate, since otherwise the false positives can outweigh the benefits of the detector.

Recent approaches for time series predictions is to use artificial neural networks. Artificial neural networks is a family of algorithms responsible for major achievements in data science fields such as computer vision, speech recognition, natural language processing, and audio recognition. Google’s voice transcription service is an example, using a LSTM [4].

For time series prediction the recurrent neural networks (RNN) have shown great promise [7, 11, 26, 29, 35]. Especially promising results have been found using the long short-term memory (LSTM) model. LSTM is a neural network model that has been found to yield good prediction accuracy for time series data [11, 26, 35], outperforming other types of recurrent neural networks due to their ability to maintain long term memory of previously seen sequences.

Another neural network model that has shown great results is the HTM model [2, 10, 11, 20], which have been shown to outperform LSTM in time series prediction [11]. The HTM model is explained in more detail in section 2.5.1.
2.5.1 Hierarchical temporal memory

Hierarchical temporal memory (HTM) is a type of neural network that is vastly different from traditional neural networks. The HTM model is based on the computational principles of the neocortex [10, 11, 20]. Almost all artificial neural networks model neurons as a "point neuron", where a single weighted sum is computed of the neurons input [20]. An example can be seen in Figure 2.4A. HTM instead uses a pyramidal neuron model, which tries to simulate the way a real neuron works in the brain, with thousands of synapses, and learns by modeling the growth of new synapses. See Figure 2.4 for a depiction of how the HTM (C) compares to the biological neuron (B).

![Figure 2.4: Comparing the artificial neural network (A), the biological neuron (B), and the HTM neuron (C). Image source [20].](image)

What makes HTMs unique compared to other artificial neural networks is their ability to learn, infer and recall complex higher order patterns in a continuous fashion. Most other neural networks are trained in batch, where as HTM networks can learn sequences continuously. HTM is also robust to noise and capable of high input capacity.

In order to be able to emulate thousands of synapses HTM uses a special value encoding called a Sparse Distributed Representation (SDR). SDRs used in HTMs are binary matrices of data where each bit in the SDR represent a neuron. Just like in the brain the majority
of the neurons are inactive. In a typical representation the data might have 2048 columns with 32 neurons per column, a total of 64K neurons. These SDRs are used both internally in the HTM and to encode the network input. However, to predict similar sequences of data, the internal SDR differ from the external one in order to capture the same data in different sequences. Figure 2.5 shows a simplified example of the SDR encoding, with 21 columns with 6 cells per column. These panels represent a slice through a single cellular layer in the neocortex (Figure 2.5A). In Figure 2.5B you can see two sequences, ABCD and XBCY, being encoded into their respective SDR. Internally these sequences are translated to different SDRs, since the sub-sequence BC is contained in both sequences there is a need for different internal representations. More information about how the sequence memory works with thousands of synapses can be found in [20].

Figure 2.5: The SDR before and after learning a sequence. Note how the characters A and B have the same encoding before learning, but different after learning. Source [20].

The HTM network’s ability to handle temporal data, and its ability to continuously adapt to new data makes it well suited for time series anomaly detection where the time series may change over time. Previous research has shown that HTM based anomaly detection for time series outperform other unsupervised online algorithms, like Twitter’s
Seasonal Hybrid ESD based model and Skyline (an statistical based open source anomaly detector) [2].

In a comparative study of HTM and other neural network models for online sequence learning with streaming data, the HTM was shown to outperform the other neural networks including the LSTM [11].

HTM is the only identified neural network capable of doing true online learning. LSTMs have shown promise in time series prediction but need to be trained in batches, and has to be retrained if the data characteristics change in the time series [11]. Their predictive performance was found to be similar to each other if an appropriate re-training window was used, however the creators of the HTM highlight several benefits of their architecture like the ability to make simultaneous predictions, no need to retrain the model to adapt to new data, robustness to noise and no need for hyperparameter tuning [11].

2.5.2 Detecting anomalies using time series prediction

Like mentioned in the introduction to time series prediction, predictive models detect anomalies by using the difference between the prediction and actual value. How this difference is utilized depends on the prediction model. A simple way would be to use the residual of the prediction and the real value. Another more sophisticated approach to capture more advanced anomalies, such as the collective anomaly class, could be to consider the number of successive anomalies [29].

The HTM based anomaly detector described by Ahmad et. al. [2] used the following formula.

\[ S_t = \frac{\pi(x_{t-1}) \cdot a(x_t)}{|a(x_t)|} \]  

(2.3)

where \( S_t \) is the prediction score, \( x_t \) is the current input, \( a(x_t) \) is the SDR encoding of the input as outlined in section 2.5.1 and \( \pi(x_{t-1}) \) is the SDR representing the HTM network’s prediction of \( a(x_t) \). \(|a(x_t)|\) is the scalar norm of \( a(x_t) \), i.e. the total number of 1’s in \( a(x_t) \). Equation (2.3) will be a value on the interval \([0, 1]\) depending on how much the prediction overlaps with the actual value. If the prediction match the value exactly \( S_t \) will be 0, if the prediction and actual value are orthogonal, i.e. no common 1’s, the value will be 1.
Ahmad et. al. also added what they call an anomaly likelihood. By assuming the distribution of prediction errors defined in (2.3) follows a normal distribution, the Gaussian tail probability function is used to decide whether or not to treat a value as an anomaly. See Figure 2.6 for the intuition of the normal distribution and the Q-function. Since the algorithm was designed to work in a streaming fashion, they compute the sample mean $\mu_t$ according to (2.4) and variance $\sigma^2_t$ according to (2.5), for the last $W$ predicted error values.

$$\mu_t = \frac{\sum_{i=0}^{W-1} S_{t-i}}{W} \tag{2.4}$$

$$\sigma^2_t = \frac{\sum_{i=0}^{W-1} (S_{t-i} - \mu_t)^2}{W - 1} \tag{2.5}$$

The anomaly likelihood is defined as the complement of the tail probability:

$$L_t = 1 - Q(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}) \tag{2.6}$$

where:

$$\tilde{\mu}_t = \frac{\sum_{i=0}^{W' - 1} S_{t-i}}{W'} \tag{2.7}$$

\(^2\text{Also called the Q-function or the Survival function}\)
$W'$ is a window for calculating a short term moving average, where $W' << W$. Anomalies are based on a user defined parameter $\epsilon$ such that:

$$\text{Anomaly detected}_t = L_t \geq 1 - \epsilon \quad (2.8)$$

## 2.6 Normalization

A common practice in data pre-processing is to normalize the data. For example if different features use different scales of reference they can not be directly compared to each other. A common way of doing this is the Standard Score, also called Standardization in (2.9).

$$z = \frac{x - \mu}{\sigma} \quad (2.9)$$

where $\mu$ is the mean value of the data, and $\sigma$ is the standard deviation. These values are calculated as (2.10) and (2.11) respectively.

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n} \quad (2.10)$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n - 1}} \quad (2.11)$$

Using standardization causes not only the values to scale, but also mean centering. Algorithms like PCA are typically applied after mean centering [1].

Another method for normalization is range-based normalization. In range-based normalization, we instead map the time series to the interval $[0, 1]$ by identifying the minimum and maximum values, denoted $\text{min}$ and $\text{max}$ respectively, and apply (2.12) to each value in the time series.

$$y_i' = \frac{y_i - \text{min}}{\text{max} - \text{min}} \quad (2.12)$$

According to Aggarwal [1], standardization is usually the preferred method of normalization.

---

3 Ahmad et. al. used as an example $W' = 10$ and $W = 8000$ [2]
2.7 Evaluation

A structured and standardized method to evaluate the anomaly detectors is needed if the detectors are to be compared. Theory of the used metrics is described in this section.

One method of evaluating an anomaly detection algorithm is to use techniques from binary classifiers. In binary classification there is a positive and a negative class for each classification. The cases that can occur can conveniently be represented by a confusion matrix [24].

<table>
<thead>
<tr>
<th>Prediction outcome</th>
<th>p′</th>
<th>n′</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>p′</td>
<td>TP</td>
<td>FN</td>
<td>P′</td>
</tr>
<tr>
<td>n′</td>
<td>FP</td>
<td>TN</td>
<td>N′</td>
</tr>
<tr>
<td>total</td>
<td>P</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: A confusion matrix of a classifier

Where

TP: The number of correct classifications of positive examples (True Positive).

FN: The number of incorrect classifications of positive examples (False Negative).

FP: The number of incorrect classifications of negative examples (False Positive).

TN: The number of correct classifications of negative examples (True Negative).

Based on the confusion matrix, a number of metrics can be calculated. Among the more common are precision (2.13) and recall (2.14), which are calculated as follows.
\[ p = \frac{TP}{TP + FP} \] (2.13)

\[ r = \frac{TP}{TP + FN} \] (2.14)

Precision shows how often a classifier is correct when it classifies a value to belong to the positive class. Recall shows how often a classifier is able to classify a positive class when the actual value belonged to the positive class. If a single measure is needed to compare and evaluate a classifier the \( F_1 \)-score, also called \( F \)-score, is often used (2.15) [1, 24]. It shows the harmonic mean of the precision and recall, meaning that for the \( F \)-score to be high, both \( p \) and \( r \) must be high.

\[ F_1 = \frac{2pr}{p + r} \] (2.15)

Other common measures [1, 24] include the true positive rate

\[ TPR = r = \frac{TP}{TP + FN} \] (2.16)

the false positive rate

\[ FPR = \frac{FP}{TN + FP} \] (2.17)

the true negative rate

\[ TNR = \frac{TN}{TN + FP} \] (2.18)

and accuracy

\[ ACC = \frac{TP + TN}{TP + TN + FP + FN} \] (2.19)
Chapter 3

Method

This chapter explains how the evaluation is carried out. First it explains how the dataset is gathered and annotated. Secondly it explains how the data is processed by the anomaly detectors and the evaluation engine. It explains how the anomaly detectors are implemented, parameters used and the data encoders. A discussion on evaluation will close this chapter.

3.1 Dataset

To evaluate the anomaly detectors two datasets are used. One dataset, referred to as the NAB dataset, is collected from NAB [31] and was built by Ahamad et. al. for their real time streaming anomaly detection benchmark [2]. This dataset contains both synthetic and real world data, along with labels for anomaly times. These labels were curated using the labeling guidelines available at [31], and have a level of high accuracy.

The second dataset, referred to as the company dataset, contains data taken from the historical metrics database at the company where this thesis is carried out. A random sample of metrics is collected from the metrics database. All metrics are known to have some historical incident associated with them to increase the possibility of anomalies being present in the data. Each collected time series is manually annotated with anomaly times - if any are found. These annotations are however not exclusive, meaning there might be anomalies in the time series that are not annotated. In order to annotate the data, a simple annotation tool is created that allows the user to use a graphical inter-
face to interact with the time series and select anomaly times.

### 3.1.1 Annotating the dataset

To label the data a graphical tool is created that allowed for manual inspection of the time series, and then marking time series with anomalies. To increase the chance the time series had anomalies, past incident information is used with the assumption that incidents would correlate with anomalies. After the anomaly timestamps are created, they are expanded into anomaly windows. The rationale behind this is that an anomaly is rarely a single point to be identified, but rather a window of anomalous behavior. This is illustrated in Figure 3.1 along with an annotation.

![Figure 3.1: An example of a time series with an anomaly annotated.](image)

Using anomaly windows instead of points also makes sure early and late anomaly detections by the algorithm are accounted for. The size of the window for an anomaly was set to being 10% of the length of the time series, based on the NAB benchmark window size [2].

### 3.2 Anomaly detector benchmarking system

A data processing pipeline is constructed in order to compare the algorithms efficiently. The pipeline has the following steps:

- Read time series data as CSV files
- Predict anomalies using detectors
• Add anomaly windows from the labeled timestamps
• Save the result

A separate scoring engine is then created that read the results and calculate the final results for each of the detectors. A system overview is depicted in Figure 3.2. It should be noted that the system does not take real time analysis into account, and the whole time series to analyze is available for the detector at the start.

3.3 Anomaly detector implementations

3.3.1 HTM

The HTM Detector, as described in section 2.5.1, is a predictive model. That is, it predicts the next value in the time series and anomalies are detected by looking at how this predicted value differs from the actual value. The neural network was constructed using the open source library nupic [32], created by Numenta [30]. The detector is based on the design by Ahmad et. al [2], and constructed using the same HTM parameters.

Input mapping

The input to the HTM is a SDR. nupic provides several built in encoders that can be used to map common values to SDR representations. Two encoders were used, one for the timestamp and one for the value. For the timestamp the DateEncoder was used, as the name implies it is a SDR encoder for date encodings. Only the time of day was encoded, using the parameters width 21 and radius 9.49. The value used the RandomDistributedScalarEncoder, an encoder built for scalar values where proximate values are to be considered unrelated. Once the input values were encoded they were fed to the HTM network.

Detecting anomalies

The model takes the encoded input and returns a result, containing the anomaly score as the proportion of overlapping bits in the predicted SDR and the received one, explained in detail in (2.3) in section 2.5.2. This raw score is then used to calculate an anomaly likelihood score.
as described in (2.6) section 2.5.2, giving a score on the interval $[0, 1]$. The result is also mapped to a logarithmic scale inspired by the HTM anomaly detector in NAB [31].

### 3.3.2 RPCA

The RPCA algorithm is not a predictive model. The algorithm takes a matrix, and computes a decomposition where one component has reduced dimensionality and one is sparse. This sparse component, $S_0$, contains data points that are considered anomalies in the underlying data. It is also a batch algorithm, meaning the data need to be windowed in order to find anomalies.

The implementation of RPCA used is a modified version of the open source Netflix Surus [28] algorithm written in Java. Originally implemented to run in a distributed Hadoop cluster, it was modified to work independently in a normal Java 8 environment.

#### Input mapping

Time series data is a vector of values and since the RPCA algorithm needs a matrix, the vector must be transformed to the appropriate format. The best way to do this is for each column to represent the seasonality of the time series, like Candes et. al. stacking video frame data by column [6]. Since the time series data might not have seasonality, and the time series all have different time scales, it becomes a problem in itself to discover seasonality. A simple approach was used where a somewhat balanced matrix of size $M \times N$ was constructed by finding all the divisors $D$ of the length of the time series $L$, and then compose a matrix such that $M$ and $N$ have the minimum difference of all the divisors. This does not work if the length of the time series is a prime number of course, however this can be overcome by reducing the size of the window and disregarding the oldest data points.

The values were also standardized according to (2.9) described in section 2.6. After the data has been standardized and transformed into a matrix, the RPCA is run on the data to extract the anomalous points.

#### Detecting anomalies

Once the RPCA has decomposed the input matrix, the sparse component is transformed back into a vector. The sparse vector has anomaly
values in the same magnitude as the original data. The values are then transformed to a likelihood of a data point being an actual anomaly. A similar idea to the one used in the HTM implementation is used. Assuming a values deviation from the mean follows a normal distribution, we can calculate the probability of a value $v$ using the cumulative distribution function (CDF).

\[
\text{probability of value} = CDF(|v|) \tag{3.1}
\]

The CDF can also be expressed as the inverse of the Q-function.

\[
Q(v) = 1 - CDF(v) \tag{3.2}
\]

The CDF is the probability that a value in a normal distribution will be less than, or equal to, $v$. Referring to Figure 2.6 in section 2.5.2, it is apparent that the more the value deviates from the standard deviation the higher the probability of an anomaly will be. However, since the result in (3.1) is in the range $[0.5, 1]$, it is also adjusted to be in the desired range $[0, 1]$.

\[
anomaly score = 2(CDF(|v|) - 0.5) \tag{3.3}
\]

The result of the above equation is then used as anomaly score for the RPCA Detector.

### 3.4 Evaluation

When evaluating the anomaly detectors there are many criteria to consider. Anomalous points for each detector is defined as the anomaly score exceeding some threshold value $\varepsilon_i$. The detectors are evaluated at multiple threshold values, since both calculate their anomaly score differently and might perform best on different thresholds. For detector $i$ the $j$th value is classified as an anomaly if:

\[
anomaly score_{i,j} > \varepsilon_i
\]

The optimal value for $\varepsilon_i$ obviously depends on the desired sensitivity of the detector. A lower value will classify more values as anomalies but risk also increasing the number of false positives. In the HTM implementation in NAB, the value of $\varepsilon_i$ was dynamically set for each input file to optimize the score, and a value of $\sim 0.5$ was found to be
optimal for most input files. For the RPCA algorithm no initial $\epsilon_i$ is known, however the three sigma rule \[19\] gives a starting point for what values could be of interest.

### 3.4.1 Judging correct detections

In order to use the binary classification measures outlined in section \[2.7\] the positive and negative classes need to be defined.

#### Data point level

Using each data point as a classified instance the measures can be calculated by assigning points within a labeled window to the positive class, and all points outside the window are assigned the negative class.

#### Window level

Another, perhaps more intuitive approach, is to instead look at whether a window was detected. The intuition being that if anomalies inside the window are detected, we have detected this annotated anomaly. Thus assigning the positive class to the anomaly windows. However this does lead to issues with the negative class. It is uncertain if anomaly points outside the window are indeed anomalies, and due to the window size difference it is unclear how to determine if an area is a false positive. Therefore the negative class is ignored.
Figure 3.2: The system overview.
Chapter 4

Results

This chapter presents the results from the detector comparisons. First it shows the total time for the detectors to process all the available data. Secondly it presents the results of the anomaly detection evaluation along with baseline anomaly detectors. Further presented is output of the method pipeline described in section 3.2. Also a comparison between the proposed detectors and the existing rule based incident detection system is presented.

4.1 Detector time performance

The detectors differ in time to perform their anomaly detections. To improve the performance of the benchmarking, each detector is run in a separate OS process in parallel using the Python multiprocessing library. The benchmark system is run on a cloud computer having 64 Intel Skylake Xeon CPU @ 2.60GHz, and 57.6 GB of memory. Time measurements are made using the UNIX \textit{time} utility. \textit{sys + user} shows the total CPU time used by the entire pipeline for both datasets, using only the specified detector.

<table>
<thead>
<tr>
<th>Detector</th>
<th>real</th>
<th>sys</th>
<th>user</th>
<th>sys + user</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTM</td>
<td>16m</td>
<td>57.700s</td>
<td>567m</td>
<td>1m 19.714s</td>
</tr>
<tr>
<td>RPCA</td>
<td>0m</td>
<td>42.537s</td>
<td>16m</td>
<td>1m 6.765s</td>
</tr>
</tbody>
</table>
4.2 Detector results

Each anomaly detector is run on a large sample of different time series. Example results showing the output from each detector on a cyclical time series can be seen in Figures 4.1 and 4.2. The plots have three sections. The values section shows the time series, the raw score section is the pure output of the anomaly detector before any anomaly score is calculated. Finally the anomaly score is shown in the bottom most section. The labeled anomaly windows are highlighted with a red background.

4.3 Classification Scores

Results from binary classification evaluation on the NAB dataset can be found in Tables 4.4 and 4.5, and for the company dataset the results can be found in tables 4.6 and 4.7. Each algorithm was evaluated at different threshold values. The $P_{ratio}$ is the ratio of points classified as anomalies and $N_{ratio}$ is the ratio of points classified as non-anomalies. For an explanation of the other values, see section 2.7. As a baseline, two detectors were constructed. The One Detector and the Null Detector, and their results can be seen in table 4.2. The One Detector classified every point as an anomaly, the Null Detector did the opposite and classified every point as a non-anomaly. These give reference values needed due to the unbalanced nature of the dataset. The values 68.27%, 95.45% and 99.73% are the three sigma values $\sigma$, $2\sigma$ and $3\sigma$.

4.4 Anomaly windows detected

The result from detecting the anomaly windows in the NAB dataset can be seen in Tables 4.8 and 4.9 and for the company dataset in 4.10 and 4.11. The columns $total\_found$ and $total\_missed$ are the number of anomaly windows that were flagged as anomalies, and the $total\_tpr$ is the ratio of found windows.
Table 4.2: Two Reference detectors, the Null Detector declares no value as an anomaly and the One Detector declares every value an anomaly.

<table>
<thead>
<tr>
<th>Detector</th>
<th>F1 precision</th>
<th>TPR</th>
<th>FPR</th>
<th>TNR</th>
<th>accuracy</th>
<th>P_ratio</th>
<th>N_ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td>0.103</td>
<td>1.000</td>
<td>0.103</td>
<td>0.0</td>
<td>1.0</td>
<td>0.808</td>
<td>0.0</td>
</tr>
<tr>
<td>One</td>
<td>0.306</td>
<td>0.192</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.192</td>
<td>1.0</td>
</tr>
</tbody>
</table>

4.5 Incident Detection

Since labels of known incidents were available for the company dataset, these were correlated with anomalies detected by the two detectors. The results are shown in Table 4.3. Because the known incident labels lack the exact start and end of an incident, the proximity of a detected anomaly and an incident is used to correlate a flagged anomaly with an incident. The incident proximity use the same window creation used with anomaly windows.

Table 4.3: Found incidents at every threshold level

<table>
<thead>
<tr>
<th>Detector</th>
<th>found</th>
<th>missed</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTM</td>
<td>101</td>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>RPCA</td>
<td>101</td>
<td>0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Figure 4.1: Example of the RPCA detector on a cyclical time series. The top section is the time series, the middle section shows the raw score before anomaly score is calculated and the bottom section shows the anomaly score. The red background shows where the labeled anomaly window is.

Figure 4.2: Example of the HTM detector on a cyclical output. The layout is the same as in Figure 4.1.
Table 4.4: Binary Classification measures on the NAB dataset using the HTM Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>ε</th>
<th>F1 precision</th>
<th>TPR</th>
<th>FPR</th>
<th>TNR</th>
<th>accuracy</th>
<th>P_ratio</th>
<th>N_ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.306</td>
<td>0.192</td>
<td>1.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.192</td>
<td>1.000</td>
</tr>
<tr>
<td>0.1</td>
<td>0.247</td>
<td>0.314</td>
<td>0.380</td>
<td>0.211</td>
<td>0.789</td>
<td>0.708</td>
<td>0.226</td>
</tr>
<tr>
<td>0.2</td>
<td>0.132</td>
<td>0.484</td>
<td>0.165</td>
<td>0.021</td>
<td>0.979</td>
<td>0.802</td>
<td>0.028</td>
</tr>
<tr>
<td>0.3</td>
<td>0.104</td>
<td>0.403</td>
<td>0.123</td>
<td>0.008</td>
<td>0.992</td>
<td>0.805</td>
<td>0.010</td>
</tr>
<tr>
<td>0.4</td>
<td>0.093</td>
<td>0.412</td>
<td>0.116</td>
<td>0.005</td>
<td>0.995</td>
<td>0.807</td>
<td>0.006</td>
</tr>
<tr>
<td>0.5</td>
<td>0.097</td>
<td>0.372</td>
<td>0.109</td>
<td>0.003</td>
<td>0.997</td>
<td>0.807</td>
<td>0.003</td>
</tr>
<tr>
<td>0.6</td>
<td>0.129</td>
<td>0.356</td>
<td>0.108</td>
<td>0.003</td>
<td>0.997</td>
<td>0.807</td>
<td>0.003</td>
</tr>
<tr>
<td>0.6827</td>
<td>0.145</td>
<td>0.337</td>
<td>0.107</td>
<td>0.002</td>
<td>0.998</td>
<td>0.807</td>
<td>0.003</td>
</tr>
<tr>
<td>0.7</td>
<td>0.162</td>
<td>0.331</td>
<td>0.107</td>
<td>0.002</td>
<td>0.998</td>
<td>0.807</td>
<td>0.003</td>
</tr>
<tr>
<td>0.8</td>
<td>0.179</td>
<td>0.317</td>
<td>0.107</td>
<td>0.002</td>
<td>0.998</td>
<td>0.807</td>
<td>0.003</td>
</tr>
<tr>
<td>0.9</td>
<td>0.196</td>
<td>0.310</td>
<td>0.106</td>
<td>0.002</td>
<td>0.998</td>
<td>0.807</td>
<td>0.002</td>
</tr>
<tr>
<td>0.9545</td>
<td>0.195</td>
<td>0.309</td>
<td>0.106</td>
<td>0.002</td>
<td>0.998</td>
<td>0.807</td>
<td>0.002</td>
</tr>
<tr>
<td>0.9973</td>
<td>0.212</td>
<td>0.301</td>
<td>0.106</td>
<td>0.002</td>
<td>0.998</td>
<td>0.807</td>
<td>0.002</td>
</tr>
</tbody>
</table>

Table 4.5: Binary Classification measures on the NAB dataset using the RPCA Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>ε</th>
<th>F1 precision</th>
<th>TPR</th>
<th>FPR</th>
<th>TNR</th>
<th>accuracy</th>
<th>P_ratio</th>
<th>N_ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.306</td>
<td>0.192</td>
<td>1.000</td>
<td>1.000</td>
<td>0.000</td>
<td>0.192</td>
<td>1.000</td>
</tr>
<tr>
<td>0.1</td>
<td>0.170</td>
<td>0.294</td>
<td>0.294</td>
<td>0.223</td>
<td>0.777</td>
<td>0.653</td>
<td>0.229</td>
</tr>
<tr>
<td>0.2</td>
<td>0.163</td>
<td>0.320</td>
<td>0.270</td>
<td>0.188</td>
<td>0.812</td>
<td>0.676</td>
<td>0.195</td>
</tr>
<tr>
<td>0.3</td>
<td>0.152</td>
<td>0.335</td>
<td>0.223</td>
<td>0.133</td>
<td>0.867</td>
<td>0.718</td>
<td>0.140</td>
</tr>
<tr>
<td>0.4</td>
<td>0.161</td>
<td>0.343</td>
<td>0.202</td>
<td>0.091</td>
<td>0.909</td>
<td>0.752</td>
<td>0.099</td>
</tr>
<tr>
<td>0.5</td>
<td>0.170</td>
<td>0.360</td>
<td>0.191</td>
<td>0.078</td>
<td>0.922</td>
<td>0.760</td>
<td>0.085</td>
</tr>
<tr>
<td>0.6</td>
<td>0.164</td>
<td>0.376</td>
<td>0.180</td>
<td>0.062</td>
<td>0.938</td>
<td>0.771</td>
<td>0.069</td>
</tr>
<tr>
<td>0.6827</td>
<td>0.159</td>
<td>0.394</td>
<td>0.171</td>
<td>0.039</td>
<td>0.961</td>
<td>0.791</td>
<td>0.046</td>
</tr>
<tr>
<td>0.7</td>
<td>0.156</td>
<td>0.395</td>
<td>0.169</td>
<td>0.036</td>
<td>0.964</td>
<td>0.792</td>
<td>0.043</td>
</tr>
<tr>
<td>0.8</td>
<td>0.145</td>
<td>0.415</td>
<td>0.155</td>
<td>0.020</td>
<td>0.980</td>
<td>0.803</td>
<td>0.026</td>
</tr>
<tr>
<td>0.9</td>
<td>0.174</td>
<td>0.527</td>
<td>0.136</td>
<td>0.013</td>
<td>0.987</td>
<td>0.805</td>
<td>0.016</td>
</tr>
<tr>
<td>0.9545</td>
<td>0.129</td>
<td>0.601</td>
<td>0.129</td>
<td>0.008</td>
<td>0.992</td>
<td>0.807</td>
<td>0.011</td>
</tr>
<tr>
<td>0.9973</td>
<td>0.147</td>
<td>0.645</td>
<td>0.119</td>
<td>0.005</td>
<td>0.995</td>
<td>0.807</td>
<td>0.007</td>
</tr>
</tbody>
</table>
Table 4.6: Binary Classification measures on the company dataset using the HTM Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$F_1$ precision</th>
<th>TPR</th>
<th>FPR</th>
<th>TNR</th>
<th>accuracy</th>
<th>$P_{\text{ratio}}$</th>
<th>$N_{\text{ratio}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.222</td>
<td>0.127</td>
<td>1.000</td>
<td>1.000</td>
<td>0.127</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.1</td>
<td>0.180</td>
<td>0.220</td>
<td>0.401</td>
<td>0.407</td>
<td>0.593</td>
<td>0.565</td>
<td>0.405</td>
</tr>
<tr>
<td>0.2</td>
<td>0.184</td>
<td>0.398</td>
<td>0.082</td>
<td>0.031</td>
<td>0.969</td>
<td>0.855</td>
<td>0.037</td>
</tr>
<tr>
<td>0.3</td>
<td>0.135</td>
<td>0.330</td>
<td>0.037</td>
<td>0.014</td>
<td>0.986</td>
<td>0.865</td>
<td>0.017</td>
</tr>
<tr>
<td>0.4</td>
<td>0.124</td>
<td>0.329</td>
<td>0.027</td>
<td>0.009</td>
<td>0.991</td>
<td>0.868</td>
<td>0.011</td>
</tr>
<tr>
<td>0.5</td>
<td>0.106</td>
<td>0.298</td>
<td>0.015</td>
<td>0.007</td>
<td>0.993</td>
<td>0.869</td>
<td>0.008</td>
</tr>
<tr>
<td>0.6</td>
<td>0.133</td>
<td>0.286</td>
<td>0.014</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
<tr>
<td>0.6827</td>
<td>0.132</td>
<td>0.281</td>
<td>0.013</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
<tr>
<td>0.7</td>
<td>0.132</td>
<td>0.281</td>
<td>0.013</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
<tr>
<td>0.8</td>
<td>0.151</td>
<td>0.276</td>
<td>0.013</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
<tr>
<td>0.9</td>
<td>0.180</td>
<td>0.271</td>
<td>0.012</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
<tr>
<td>0.9545</td>
<td>0.180</td>
<td>0.270</td>
<td>0.012</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
<tr>
<td>0.9973</td>
<td>0.179</td>
<td>0.265</td>
<td>0.012</td>
<td>0.006</td>
<td>0.994</td>
<td>0.869</td>
<td>0.007</td>
</tr>
</tbody>
</table>

Table 4.7: Binary Classification measures on the company dataset using the RPCA Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$F_1$ precision</th>
<th>TPR</th>
<th>FPR</th>
<th>TNR</th>
<th>accuracy</th>
<th>$P_{\text{ratio}}$</th>
<th>$N_{\text{ratio}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.222</td>
<td>0.127</td>
<td>1.000</td>
<td>1.000</td>
<td>0.127</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.1</td>
<td>0.127</td>
<td>0.328</td>
<td>0.090</td>
<td>0.036</td>
<td>0.964</td>
<td>0.853</td>
<td>0.042</td>
</tr>
<tr>
<td>0.2</td>
<td>0.114</td>
<td>0.357</td>
<td>0.077</td>
<td>0.028</td>
<td>0.972</td>
<td>0.859</td>
<td>0.034</td>
</tr>
<tr>
<td>0.3</td>
<td>0.103</td>
<td>0.377</td>
<td>0.068</td>
<td>0.024</td>
<td>0.976</td>
<td>0.861</td>
<td>0.029</td>
</tr>
<tr>
<td>0.4</td>
<td>0.094</td>
<td>0.390</td>
<td>0.060</td>
<td>0.021</td>
<td>0.979</td>
<td>0.863</td>
<td>0.025</td>
</tr>
<tr>
<td>0.5</td>
<td>0.085</td>
<td>0.412</td>
<td>0.047</td>
<td>0.011</td>
<td>0.989</td>
<td>0.869</td>
<td>0.015</td>
</tr>
<tr>
<td>0.6</td>
<td>0.081</td>
<td>0.440</td>
<td>0.044</td>
<td>0.009</td>
<td>0.991</td>
<td>0.870</td>
<td>0.013</td>
</tr>
<tr>
<td>0.6827</td>
<td>0.078</td>
<td>0.455</td>
<td>0.041</td>
<td>0.008</td>
<td>0.992</td>
<td>0.871</td>
<td>0.012</td>
</tr>
<tr>
<td>0.7</td>
<td>0.077</td>
<td>0.459</td>
<td>0.041</td>
<td>0.008</td>
<td>0.992</td>
<td>0.871</td>
<td>0.011</td>
</tr>
<tr>
<td>0.8</td>
<td>0.074</td>
<td>0.488</td>
<td>0.038</td>
<td>0.006</td>
<td>0.994</td>
<td>0.872</td>
<td>0.010</td>
</tr>
<tr>
<td>0.9</td>
<td>0.059</td>
<td>0.524</td>
<td>0.035</td>
<td>0.004</td>
<td>0.996</td>
<td>0.873</td>
<td>0.008</td>
</tr>
<tr>
<td>0.9545</td>
<td>0.065</td>
<td>0.555</td>
<td>0.032</td>
<td>0.003</td>
<td>0.997</td>
<td>0.874</td>
<td>0.006</td>
</tr>
<tr>
<td>0.9973</td>
<td>0.085</td>
<td>0.608</td>
<td>0.025</td>
<td>0.002</td>
<td>0.998</td>
<td>0.874</td>
<td>0.004</td>
</tr>
</tbody>
</table>
Table 4.8: Anomaly window detection result on the NAB dataset using the HTM Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>total_found</th>
<th>total_missed</th>
<th>total_tpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>99</td>
<td>0</td>
<td>1.000</td>
</tr>
<tr>
<td>0.1</td>
<td>94</td>
<td>5</td>
<td>0.949</td>
</tr>
<tr>
<td>0.2</td>
<td>89</td>
<td>10</td>
<td>0.899</td>
</tr>
<tr>
<td>0.3</td>
<td>86</td>
<td>13</td>
<td>0.869</td>
</tr>
<tr>
<td>0.4</td>
<td>84</td>
<td>15</td>
<td>0.848</td>
</tr>
<tr>
<td>0.5</td>
<td>83</td>
<td>16</td>
<td>0.838</td>
</tr>
<tr>
<td>0.6</td>
<td>77</td>
<td>22</td>
<td>0.778</td>
</tr>
<tr>
<td>0.6827</td>
<td>74</td>
<td>25</td>
<td>0.747</td>
</tr>
<tr>
<td>0.7</td>
<td>72</td>
<td>27</td>
<td>0.727</td>
</tr>
<tr>
<td>0.8</td>
<td>70</td>
<td>29</td>
<td>0.707</td>
</tr>
<tr>
<td>0.9</td>
<td>68</td>
<td>31</td>
<td>0.687</td>
</tr>
<tr>
<td>0.9545</td>
<td>67</td>
<td>32</td>
<td>0.677</td>
</tr>
<tr>
<td>0.9973</td>
<td>63</td>
<td>36</td>
<td>0.636</td>
</tr>
</tbody>
</table>

Table 4.9: Anomaly window detection result on the NAB dataset using the RPCA Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>total_found</th>
<th>total_missed</th>
<th>total_tpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>99</td>
<td>0</td>
<td>1.000</td>
</tr>
<tr>
<td>0.1</td>
<td>98</td>
<td>1</td>
<td>0.990</td>
</tr>
<tr>
<td>0.2</td>
<td>97</td>
<td>2</td>
<td>0.980</td>
</tr>
<tr>
<td>0.3</td>
<td>97</td>
<td>2</td>
<td>0.980</td>
</tr>
<tr>
<td>0.4</td>
<td>95</td>
<td>4</td>
<td>0.960</td>
</tr>
<tr>
<td>0.5</td>
<td>93</td>
<td>6</td>
<td>0.939</td>
</tr>
<tr>
<td>0.6</td>
<td>93</td>
<td>6</td>
<td>0.939</td>
</tr>
<tr>
<td>0.6827</td>
<td>93</td>
<td>6</td>
<td>0.939</td>
</tr>
<tr>
<td>0.7</td>
<td>92</td>
<td>7</td>
<td>0.929</td>
</tr>
<tr>
<td>0.8</td>
<td>92</td>
<td>7</td>
<td>0.929</td>
</tr>
<tr>
<td>0.9</td>
<td>86</td>
<td>13</td>
<td>0.869</td>
</tr>
<tr>
<td>0.9545</td>
<td>83</td>
<td>16</td>
<td>0.838</td>
</tr>
<tr>
<td>0.9973</td>
<td>75</td>
<td>24</td>
<td>0.758</td>
</tr>
</tbody>
</table>
Table 4.10: Anomaly window detection result on the company dataset using the HTM Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>total_found</th>
<th>total_missed</th>
<th>total_tpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>117</td>
<td>0</td>
<td>1.000</td>
</tr>
<tr>
<td>0.1</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.2</td>
<td>107</td>
<td>10</td>
<td>0.915</td>
</tr>
<tr>
<td>0.3</td>
<td>107</td>
<td>10</td>
<td>0.915</td>
</tr>
<tr>
<td>0.4</td>
<td>107</td>
<td>10</td>
<td>0.915</td>
</tr>
<tr>
<td>0.5</td>
<td>107</td>
<td>10</td>
<td>0.915</td>
</tr>
<tr>
<td>0.6</td>
<td>103</td>
<td>14</td>
<td>0.880</td>
</tr>
<tr>
<td>0.6827</td>
<td>103</td>
<td>14</td>
<td>0.880</td>
</tr>
<tr>
<td>0.7</td>
<td>103</td>
<td>14</td>
<td>0.880</td>
</tr>
<tr>
<td>0.8</td>
<td>101</td>
<td>16</td>
<td>0.863</td>
</tr>
<tr>
<td>0.9</td>
<td>98</td>
<td>19</td>
<td>0.838</td>
</tr>
<tr>
<td>0.9545</td>
<td>98</td>
<td>19</td>
<td>0.838</td>
</tr>
<tr>
<td>0.9973</td>
<td>97</td>
<td>20</td>
<td>0.829</td>
</tr>
</tbody>
</table>

Table 4.11: Anomaly window detection result on the company dataset using the RPCA Anomaly Detector for different threshold values

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>total_found</th>
<th>total_missed</th>
<th>total_tpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>117</td>
<td>0</td>
<td>1.000</td>
</tr>
<tr>
<td>0.1</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.2</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.3</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.4</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.5</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.6</td>
<td>116</td>
<td>1</td>
<td>0.991</td>
</tr>
<tr>
<td>0.6827</td>
<td>114</td>
<td>3</td>
<td>0.974</td>
</tr>
<tr>
<td>0.7</td>
<td>114</td>
<td>3</td>
<td>0.974</td>
</tr>
<tr>
<td>0.8</td>
<td>114</td>
<td>3</td>
<td>0.974</td>
</tr>
<tr>
<td>0.9</td>
<td>114</td>
<td>3</td>
<td>0.974</td>
</tr>
<tr>
<td>0.9545</td>
<td>113</td>
<td>4</td>
<td>0.966</td>
</tr>
<tr>
<td>0.9973</td>
<td>110</td>
<td>7</td>
<td>0.940</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion

In this chapter the results are discussed, and several points are highlighted. First the dataset creation, along with findings in the NAB labeled dataset is presented. Then the timing results and its implications are addressed. The classifier and the window detection metrics are further discussed.

5.1 Dataset

The creation of the dataset with annotated anomalies illustrated the arguments of Hu et al. [21] in Time Series Classification under More Realistic Assumptions. Real data contains noise, and to determine if a data point is part of an anomaly can be extremely difficult - even with deep domain knowledge. In some instances a data point is obviously an anomaly. Global anomalies are for example easier to spot, where the data vastly differs for all other data.

However local anomalies that might have occurred previously in the series are hard to determine if they in fact are anomalies. For example, services with systematic clean up operations such as a garbage collected process or an index rebuild, will have regular patterns visible in the gathered metrics during the clean up. Metrics containing increased activity, increased latency, drop in memory usage, etc. might have patterns that resemble anomalies however they are repeating and not a cause for concern. The behaviour is expected and not the precursor to an incident.
5.1.1 NAB dataset accuracy

Inspecting the output of the two detectors, the correctness of the NAB dataset can be put into question. Consider for example the two time series in Figure 5.1 and 5.2. Disregarding the initial anomalies at the beginning of the series, since the HTM network is untrained at this point, both algorithms still flagged the same anomaly outside the labeled anomaly window using a reasonably high threshold level. Of course this might be a coincidence, but it does put into question the accuracy of the labels from the dataset.

5.2 Performance

It is clear from the time measures in Table 4.1 that the implementation of the RPCA Detector is much faster than the HTM Detector. One parameter effecting the performance of the HTM Detector is the need to process each value in sequence and thus each value will cause an update on the underlying HTM network requiring quite heavy computation. The RPCA algorithm process the entire series in one batch, and does not rely on a model which need updating.

5.3 Binary classifier measures

The binary classifier measures should be taken with some skepticism for both the NAB dataset and the company dataset, since not every point in the anomaly window is an anomaly. As discussed above, some points outside the window might still be anomalies, even in the NAB dataset that supposedly had accurate labels. The datasets are also very imbalanced, with the negative class (non-anomalies) being much larger than the positive class (anomalies). This imbalance causes the metrics to be largely dominated by the negative classes FP and TN. Bearing these limitations in mind, the results do however provide some interesting insights.

One interesting result across both datasets is that by comparing the TPR and FPR of the detectors it is evident that the anomaly windows were more "positive dense" compared to the rest of the data for both detectors. The proportion on anomaly points were higher in the labeled windows.
Figure 5.1: RPCA detector on the same time series as Figure 4.1 with noise removed at threshold level 0.8

Figure 5.2: HTM detector on the same time series as Figure 4.2 with noise removed at threshold level 0.8
Looking at the accuracy score it is clear that the accuracy increase with higher threshold values. Due to the imbalance between the positive and negative classes, a comparison must be made to the Null Detector in Table 4.2 and the accuracy is similar. However the precision for both the detectors is very high considering the skew in the dataset. The reason the Null detector has such high precision is because the detector never provides a positive prediction, causing the prediction calculation in (2.13) to be undefined. Thus, this value can be ignored.

5.4 Anomaly window detection

It can be seen from Tables 4.4 and 4.5 that the $P_{ratio}$, ratio of points classified as anomalies, differs between the detectors. The RPCA detector seems to be more sensitive than the HTM detector with a higher ratio over every threshold value. Even with the maximum threshold value of $3\sigma$, the RPCA detector had a $P_{ratio}$ of 0.007. Similarly the HTM detector had a $P_{ratio}$ 0.003 at the lower threshold 0.6. Comparing the found anomaly windows at these threshold values in Tables 4.9 and 4.8 it is clear that the HTM found more anomaly windows with a total $tpr$ of 0.778 and the RPCA had a total $tpr$ of 0.758. This shows that the RPCA has a higher sensitivity than to the HTM, since it flagged more points as anomalies yet did not identify as many of the known labeled anomalies.

However, this is not the case based on Tables 4.6 and 4.7. The evaluation shows the opposite results using the data from the company, even at the maximum threshold of $3\sigma$. The RPCA algorithm had the highest total $tpr$ and the lowest $P_{ratio}$. According to these tables the RPCA detector found more anomaly windows despite flagging fewer points overall.

5.5 Found Incidents

Both the algorithms are able to find all known incident times as shown in Table 4.3. Because the current system for detecting incidents is modeled after a set of rules and thresholds, for an incident to be detected it must be a clear global anomaly.
5.6 Ethical concerns

There are no direct ethical concerns with this project. The data used have no sensitive user information and does not pose any threat to user privacy or the company. The methods used in this project, and especially the predictive HTM network, could of course be used on other data which in theory could be used with malicious intent, for example using the ability to predict when a system is under most strain to launch an overload attack.

Regarding the social, economical and sustainability aspects, there are no major negative implications, except for automatic monitoring requiring more computing power which in turn need more energy and hardware. Positive implications could however increase the reliability of any system, which in turn could have massive implication on critical systems in areas such as health care and finance.
Chapter 6

Conclusions

As shown in the results, both the RPCA Detector and the HTM Detector performed well for anomaly detection, and both were able to perform at least as well as the current incident detection setup for the tested time series. Both detectors were able to find every known incident in the dataset, and even at high probability thresholds both algorithms were able to find a majority of the labeled anomaly windows. The RPCA algorithm seems in general be more sensitive than the HTM Detector, however since the detection of anomalies and the sensitivity of the detectors also depend on the threshold values both algorithms can be tuned to be more sensitive if desired. The sensitivity also depends on the dataset, since the HTM Detector had more flagged values than the RPCA Detector on the company dataset while the opposite was true for the NAB dataset. For the company dataset the RPCA Detector was able to detect 94% of all anomaly windows while having a false positive rate of 0.2%. The HTM Detector never reach that level of performance for the company dataset, having a minimum false positive rate of 0.6%, and finding 88% of the anomaly windows.

The big difference in resulting metrics between the datasets has been another parameter to be highlighted here. The HTM Detector performed best on a dataset curated by the research organization that created the HTM architecture, and still holds the top score in their own anomaly detector benchmark NAB. This ties back to the issues discussed in section 2.3 where algorithms are published with results on data with the purpose to showcase the algorithm. However since the RPCA has never been compared to the HTM before, it might simply be that the designed RPCA is a high-performing detector.
6.1 Reduce cognitive load

In order to reduce the cognitive load of the engineers who rely on these metrics for diagnostics, the anomaly detections need to be accurate and reliable. The results show that for the company microservice metrics analyzed, the RPCA Detector performed the best. Since the RPCA Detector is batch-based, it can be readily implemented for the visual graph dashboards already in use by engineers, since each time series in a dashboard can be analyzed as a batch. The result of the analysis can then be used to highlight time series having anomalies in recent time. A proof of concept was constructed as a simple website that could take an existing graph used for monitoring, and applied the anomaly detector specified on the series.

![Figure 6.1: Proof of concept of applying the RPCA to an existing metric used today by engineers. The top graph is the original time series, the second section shows the anomaly score with three anomaly thresholds (red = high, orange = mid, blue = low), and the last section shows the raw score before anomaly score is calculated.](image)

6.2 Continuous monitoring

For continuous monitoring of the microservices the anomaly detectors need different setups. The HTM Detector possess many desirable abilities for the continuous monitoring use case. The HTM Detector is
designed to analyze data in a streaming fashion, and thus it can be set up to continuously monitor the metrics by piping the metrics as they arrive into the model. The detector will automatically adapt to new data, learn new patterns while at the same time retaining history of previously seen sequences. One HTM Detector needs to be deployed for each time series, and this requires a large amount of computing power considering the performance metrics in Table 4.1.

The RPCA algorithm is batched-based, and thus, can only take into consideration as much history as being given to it at the time of computation. This limitation also means that analyzing each new data point as it arrives might be out of the question, and some kind of sliding window approach might be needed. How this windowing would affect the anomaly detection is still an open question, and would require further research.

### 6.3 Future work

For future work it is recommended to attempt more evaluation of how the RPCA Detector behaves with a sliding window instead of the entire time series, and how to properly discover the periodical nature of the time series when constructing the matrix to be analyzed.

Time series analysis can be extended by elaborating the analysis by dividing the dataset into time series of different characteristics and analyze each separately. This could potentially show if there are certain types of time series where one algorithm outperforms another. Perhaps this could explain why the detector performance differed substantially across the utilized datasets.

Adding more detectors to the analysis would also be beneficial to give a broader comparison of the different models in use today for time series analysis in the industry, such as the LinkedIn detector and the Twitter Detector briefly mentioned in section 1.3.

Moreover, cross referencing different time series would also be another path to go further in deep, since the microservice infrastructure has services that are co-dependent. Investigating how correlation exist between anomalies across different time series would provide valuable insight when diagnosing a system. There might even be cases where two time series are codependent and an anomaly could only be visible by analyzing both together.
Finally, the process of manually labeling anomalies for evaluation limits the size of the dataset and prone to human errors. More data with higher quality anomaly labels would be required to perform a better analysis of the detectors performance.
Bibliography


