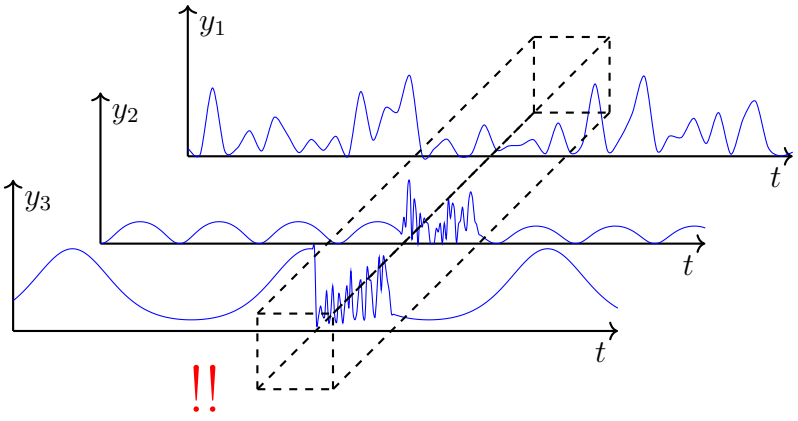
CHALMERS
UNIVERSITY OF TECHNOLOGY



Anomaly detection in communication system using machine learning

Proposed model for detecting software faults in cloud-based application

Master's thesis in Communication Engineering

ERIK EKLUND
BENGT SJÖGREN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018

Anomaly detection in communication system using machine learning

Proposed model for detecting software faults in cloud-based application

Erik Eklund
Bengt Sjögren

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communication and Antenna systems
Communication Systems Group
Chalmers University of Technology
Gothenburg, Sweden 2018

Anomaly detection in communication system using machine learning
Proposed model for detecting software faults in cloud-based application
ERIK EKLUND
BENGT SJÖGREN

Supervisor: Hannes Marcks Von Würtemberg, Ericsson AB
Supervisor: Jan Skoglund, Ericsson AB
Supervisor: Anver Hisham Unnichiriyath Siddique, Department of Electrical Engineering
Examiner: Henk Wymeersch, Department of Electrical Engineering

Master's Thesis 2018
Department of Electrical Engineering
Division of Communication and Antenna systems
Communication Systems Group
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Example scenario of three time series with two having anomalous behavior clearly highlighted as an detection.

Typeset in LaTeX
Gothenburg, Sweden 2018

Anomaly detection in communication system using machine learning
Proposed model for detecting software faults in cloud-based application
ERIK EKLUND
BENGT SJÖGREN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This report presents models to solutions for discovering anomalous behaviour in communication system software running within a cloud environment, using machine learning methods. The software is modeled as a group of collaborating subapplications, called microservices. Every microservice could be running in one or more instances for scaling purposes. The proposed solutions for solving the problem of anomaly detection is to perform classification on each microservice instance using certain output performance metrics.

Two prototype systems were built to evaluate the proposed model in two slightly different variatons of the cloud environment designed for pre-deployment preliminary testing of committed updates, as well as for live operation. The results from tests performed on the prototype indicates that the proposed type of system could be viable, and that better performance could be expected by using deep learning concepts. Several problems of the approach were also identified. A difficulty of detecting the desired class of anomalies will most likely always persist. The constant change of the microservice over many updates, as well as deployment to different customers requires training regularly.

Keywords: Machine learning, anomaly detection, outlier detection, novelty detection, cloud native, micro services.

Acknowledgements

The authors of this report would like to thank Ericsson AB for allowing us to do this research and providing us with material and supplies to do so. The authors would also like to thank the provided supervisor and the examiner from Chalmers for guidance through the process of the thesis project and input to the report.

Erik Eklund and Bengt Sjögren, Gothenburg, 2018

Contents

List of Acronyms

CNCF Cloud Native Computing Foundation
DB Database
GAN Generative Adverserial Network
IDS Intrusion detection system
iForest Isolation Forest
iTree Isolation Tree
ML Machine Learning
MS Moving Statistics
OM Operations and Maintenance
OCSVM One Class Support Vector Machine
PCA Principal Component Analysis
RBF Radial Basis Function
RRCF Robust Random Cut Forest
SVM Support Vector Machine
SVR Support Vector Regression
t-SNE t-distributed Stochastic Neighbor Embedding

1
Introduction

The first chapter introduces cloud application environments and machine learning (ML), the research areas of this thesis. Included is also a short introduction to Ericsson AB and a few of their ideas within these fields. The chapter then moves on to describe the aim of the thesis and the objectives needed to be completed, in order to reach the aim. Following these sections, previous work within the fields is presented. A scope is then defined to outline the work done in this project. A disposition of the thesis report is lastly presented at the end of the chapter.

1.1 Background

This section briefly presents the relevant fields to the thesis work. A short presentation is then included to introduce Ericsson AB and their needs connected to mentioned fields.

1.1.1 Cloud native applications

A general observation is that there is an increasing demand for applications based in "The Cloud". For users this could mean benefits such as ease of access or backup of valuable data. For the developer, it could give possibilities to avoid using own servers etc, and instead base applications on cloud servers provided by some vendor.

The *Cloud Native Computing Foundation*(CNCF) defines cloud native computing as software which uses an open source software stack in order to achieve containerization, dynamic orchestration and microservices oriented. Containerized means that each part of the environment, e.g. applications or processes are packaged in its own container. Containerization can facilitate reproducibility, transparency, and resource isolation. Dynamic orchestration is the concept of scheduling and management of containers in a way such that computational resources can be used optimally. Microservices orientation is the focus on segmenting applications into different microservices for the purpose of improving agility and maintainability of the system [1].

1.1.2 Machine learning

In broad terms, machine learning is a field in computer science about using statistical methods to create functions or applications which can progressively improve at

specific tasks using data, without being explicitly programmed [2]. With the increase in available computational power, and availability of data in general, ML has gained attention and is today used across many fields. Some examples to problems where ML methods can be applied, are within Computer vision (e.g. Facial recognition or analysis within medical imaging), stock market analysis and also Anomaly/outlier detection[3].

1.1.3 Anomaly detection

The problem of anomaly detection is that of finding patterns in data that does not conform to an expected behaviour. These patterns are referred to as different things depending on application domain, some examples are anomalies, outliers and contaminations [4]. Figure 1.1 illustrates a 2D example of data, where most points conform to two regions, but some are defined as outliers, since they deviate from the the two regions.

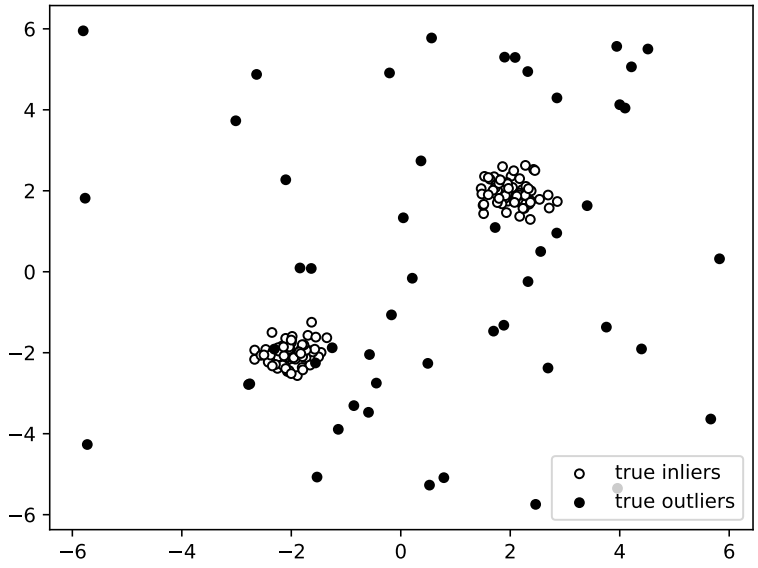


Figure 1.1: An illustration of 2D data containing anomalies

1.1.4 Implementation at Ericsson AB

Ericsson AB is a provider of information and communications technology to cellular network service providers worldwide. Ericsson AB divides their portfolio into several business areas, namely Networks, Digital Services, Managed Services and Emerging Business. The portfolio offered in these areas, being powered by 5G and Internet of Things platforms.[5]

Within the solution area, Digital Services, a system functionality that Ericsson AB incorporates into the cloud environment is their Packet Core. Packet Core has

numerous functions such as handling data, voice or message traffic, checking if a user has paid their bills or setting up their device to use the network.

Individual performance metrics of applications may be monitored manually both by viewing graphs or through alarms that has been manually set to trigger if certain metrics exceed or fall below a threshold. With more and more functionality moving into the cloud environment, the amount of metrics that needs monitoring grows larger. This makes it more difficult to monitor the data and there is also room for missed anomalies that could be very costly if not caught early enough. If the task of finding anomalies could be improved by a higher degree of automation, it could improve the overall performance level of the communication network for the subscribed users but also reduce operation cost for the operator of the network.

Anomalous behaviour in this scenario, is unexpected behaviour caused by the software, perhaps due to a bug in a recent update. If not properly detected and handled, these could cause performance degradation for customers, or even crashes.

1.2 Aim

The aim of this thesis is to present solutions for discovering anomalous behaviour in software running within a cloud environment, using machine learning methods. Furthermore, to present these solutions as autonomous systems that can operate continuously within the cloud environment. Lastly, to design the solutions such that other parts of the environment can base decisions for actions based on discoveries of the system.

1.3 Objectives

The thesis aim is categorized into four objectives, which must be addressed consecutively. The first lies in understanding the given cloud environment, and is completed by designing a descriptive model. By having a rigorously defined model of the environment, proposed solutions and how they fit into it can be defined clearly.

Secondly, it is needed to investigate what ML concepts could be applied to the problem of anomaly detection. The investigation requires a literature study into machine learning, and for anomaly detection in particular. From what is learned, proposals of how the concepts could be implemented into the defined model, can be presented as proposed additions to it. It is also a requirement, that any proposed solution is analyzed from an ethical perspective.

The third objective is to define test and evaluation methods for the proposed additions to the model. This is done using the gathered background theory in ML concepts along with an understanding of the environment through the defined model.

The final objective is to evaluate the performance of proposed solutions by creating

prototype systems and performing the defined tests and evaluation methods.

1.4 Previous work

Anomaly detection has been studied for a long time. It has been studied and used in intrusion detection systems(IDS), fraud detection, fault detection, biomedical monitoring and more[4]. There are several different techniques to anomaly detection including information theory, statistics, spectral theory and machine learning. In information theory, concepts such as *entropy*, *Kolomogorov Complexity* and *relative entropy* are used to analyze the information content in a data set. The assumption is that anomalies are sufficiently different in its information content in order to be detected. In statistical anomaly detection, the idea is to create a probabilistic model for the given data. If new data are in a low probabilistic area of the model, an anomaly could be discovered. In spectral anomaly detection, the goal is to encode the data into subspaces in order for anomalies to be more easily identified. For supervised learning, anomaly detection has been used to clean the data set before training the model to improve accuracy[6]. In this thesis, only well known anomaly detection algorithms will be used and evaluated.

1.5 Scope

As the aim and objectives state, the focus of the project will be to work with the given cloud microservices environment. Furthermore, it is to model it and to propose additions to it in the form of an ML-classification system, based on existing algorithms, rather than proposing new ways to perform outlier/anomaly detection. The thesis project will be limited to proposing solutions around the cloud environment, as it is modeled, rather than being able to propose any major changes to it. Proposed changes could simplify the implementation of a solution system, but would mean having to do more research about cloud native architecture.

The majority of the prototyping work will be done using the programming language Python [7]. Specifically, software libraries such as Scikit-Learn[8] and Keras[9] using Tensorflow[10] backend, designed for machine learning specific tasks, will be used. These libraries contain many of the relevant techniques for the thesis project, implemented into easy-to-use functionality.

Since the cloud environment consists of many different types of microservices, a limitation set for this thesis will be to implement proposed solutions on one of these types. The idea is that the same approach of designing a solution system could then apply to any type of microservice, but that there may be some slight differences for each type.

1.6 Disposition of thesis report

This report consists of six chapters. In Chapter 1, an introduction is first given to the fields in which the thesis is mainly located. The first chapter then continues on to define the aim of the thesis and the objectives which need to be completed in order to reach the aim. Some presentation of previous work and a presentation of the scope lastly gives some boundaries on where focus will be put for the report.

Chapter 2 serves to give a theoretical background to the report. Mainly, various methods used to perform machine learning are discussed, as well as some methods for preprocessing of data using statistical concepts.

In the third chapter, A model of the cloud environment along with proposed solution systems for performing ML-based anomaly detection are presented. Lastly, tests are defined to evaluate the performance of the proposed solutions. In Chapter 4, The results of the defined tests run on designed prototype systems are presented.

An analysis of the proposed model, the tests and the results of runs on the prototype are presented in Chapter 5, and finally in Chapter 6, some conclusions are drawn from the analysis.

2
Background on Machine Learning

This Chapter provides background theory to concepts of ML. The subject is often divided into two classes: Supervised and unsupervised learning. Supervised learning, where data points have known labels, is briefly introduced, and a few algorithms are presented. In the case of unsupervised learning, the focus is on outlier as well as novelty detection techniques to find data points with abnormal behaviour compared to most points.

The chapter also presents theory relating to how high dimensional data-sets can be pre-processed, as well as scaled. Lastly, background theory on how to train and validate an ML-model is presented.

2.1 Supervised learning

In supervised learning, every training point has a corresponding correct label. That way, the model learns to associate certain inputs with the the correct output. There are several algorithms for supervised learning and a subset of these, relevant to the thesis, are explained in more detail.

2.1.1 Support Vector Machine(SVM)

Given l amount of training points with corresponding labels

$$(x_1, y_1), \ldots, (x_l, y_l), \forall i = 1, \ldots, l \tag{2.1}$$

where $x_i \in \mathbb{R}^n$ and $y_i \in \{1, -1\}^l$, the SVM solves the following primal optimization problem[11]:

$$\min_{w,b,\xi} \quad \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \tag{2.2}$$

$$\text{Subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \qquad \forall i = 1, \ldots, l \tag{2.3}$$

$$\xi_i \geq 0, \qquad \forall i = 1, \ldots, l \tag{2.4}$$

and its Lagrangian dual problem is

$$\min_{\alpha} \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \tag{2.5}$$

$$\text{Subject to} \quad y^T \alpha = 0 \tag{2.6}$$

$$0 \leq \alpha_i \leq C, \qquad \forall i = 1, \ldots, l \tag{2.7}$$

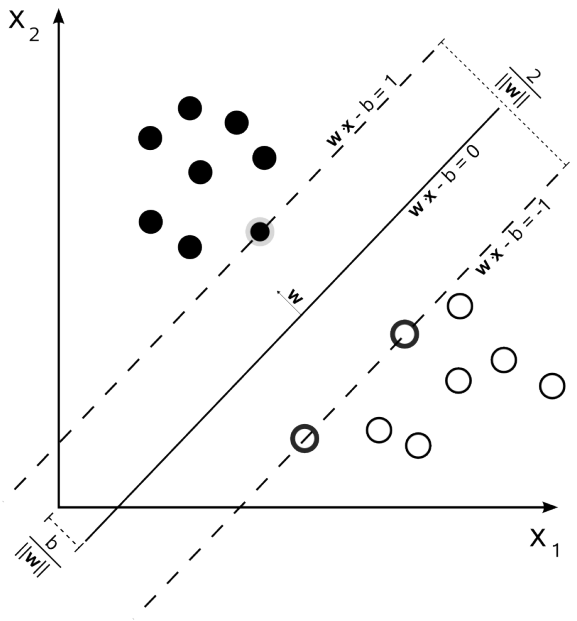


Figure 2.1: A plot of a trained decision function. By Cyc [Public domain], from Wikimedia Commons. Available: `https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png` [accessed 2018-02-15]

where $Q \in \mathbb{R}^{l \times l}$ is a positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel function. $\phi(x_i)$ is a function that maps x_i to a higher dimensional space. e is a vector with all ones and $C > 0$ is the upper bound. The prediction on a new point z is then

$$\hat{y} = \text{sign}(w^* \phi(z) + b^*) \tag{2.8}$$

$$= \text{sign}(\sum_{i=1}^{l} \alpha_i^* y_i K(x_i, z) + b^*) \tag{2.9}$$

where w^*, α^* and b^* is the solution from the optimization problems in equations 2.2-2.7[11]. Figure 2.1 illustrates an example of a trained decision function.

There are several different kernels for mapping data into different dimensional spaces. They all have varying success on different types of data and the most common ones are listed here[12]:

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$
- Radial basis function(RBF): $K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2), \gamma > 0$
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

γ, d and r are kernel parameters.

2.1.2 Support Vector Regression(SVR)

Instead of assigning a new datapoint to a class, it can be desirable to predict a value depending on the input. The prediction function is then[13]:

$$f(x) = \langle w, x \rangle + b \quad x \in \mathcal{X}, b \in \mathbb{R} \tag{2.10}$$

Where $\langle \cdot, \cdot \rangle$ is the dot product. The corresponding minimization problem is thus:

$$\min_{w,b} \quad \frac{1}{2}||w||^2 \tag{2.11}$$

$$\text{Subject to} \quad y_i - \langle w, x_i \rangle - b \leq \epsilon \tag{2.12}$$

$$\langle w, x_i \rangle + b - y_i \leq \epsilon \tag{2.13}$$

2.1.3 Artificial Neural Networks

The inspiration for Artificial Neural Networks comes from how the human brain works. The brain consists of 86 ± 8 billion[14] neurons and synapses that "fire" with different strengths in a huge network. An artificial neural network works very similar to this. It has weights that is connected to every node(neuron) in the network. These weights have been trained to produce the correct output for different inputs in the network. An example of a Neural network is seen in Figure 2.2

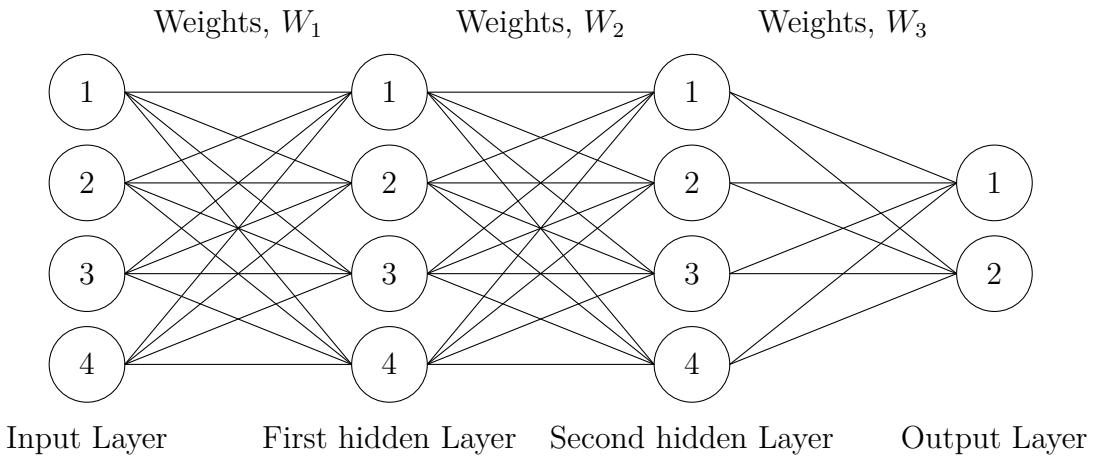


Figure 2.2: An example of how an artificial neural network with four input nodes and two output nodes. Between them is two hidden layers with four nodes each.

The hidden layers are part of the "black box" and is completely up to the learning algorithm to adjust the values of the weights in each layer.

If the loss function(often called error function) of a neural network is denoted $J(W)$, its first order derivative is then computed in terms of the neural networks weights

and then their weights are updated in the following manner[15]:

$$W^{(k+1)} = W^{(k)} - \eta \frac{\delta}{\delta W^{(k)}} J(W) \tag{2.14}$$

Where η is the learning rate.

2.1.4 Deep learning

By adding to the amount of hidden layers, the network increases in complexity and is referred to as a deep neural network. Deep learning techniques refers to different types of deep neural networks. Before deep learning, most neural network implementations would have three to five layers, whereas with deep learning, the amount of layers can be seven or more with up to millions of neurons. According to McKinsey Global Institute, deep learning techniques have been found to be of great use to increase performance where previously traditional ML methods are applied [16].

2.1.4.1 Auto-encoders

An auto-encoder is a neural network that decreases in dimensions and then increases in dimensions as it gets closer to the output nodes as seen in figure 2.3. Since the network decreases in dimensions it forces it to learn what features of the input data is needed for reconstruction in order to get the correct output. Traditionally, auto-encoders are considered to be an unsupervised algorithm but since the target output in known, it is suited to be in this section. In terms of novelty detection an auto-encoder could be trained to be able to reconstruct the nominal data correctly. If an anomaly point is entered into the trained auto-encoder, the output would be quite different from its input. If the difference is over a certain threshold, an anomaly is probably detected. The network function is trained to reproduce the input as close as possible[17], $h_{W,b}(x) \approx x$. An example of a structure for an auto-encoder is visualized in Figure 2.3

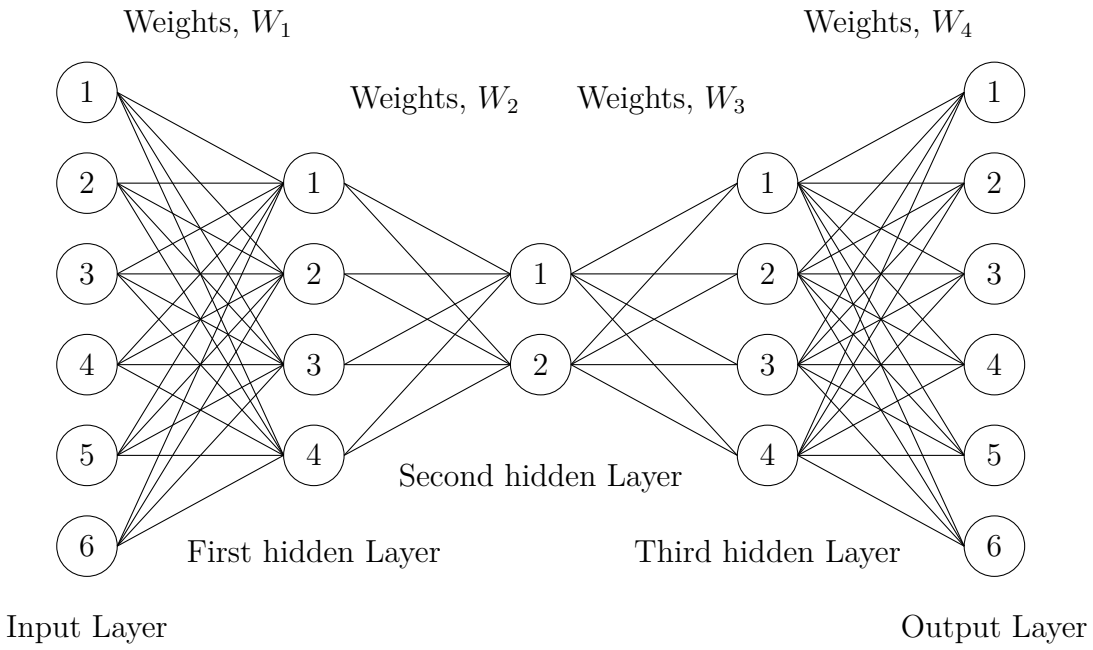


Figure 2.3: An example of the structure of an auto-encoder. This auto-encoder has six input and output nodes but the network decreases in dimension closer to the center where it only has two nodes. It then tries to decode the input from the information in the two nodes.

2.2 Unsupervised learning for outlier/novelty detection

Anomaly detection can be split up into two types, outlier and novelty detection, and their differences will be explained here. Outlier or Novelty detection are the tasks of deciding whether new observations are distributed equally to existing observations, and therefore being an *inlier*. If not, the new observation should be considered as an outlier. In novelty detection, the given training data does not contain outliers and the interest lies in finding the outliers in new observations. This method is shown in figure 2.4. In contrast for outlier detection, the training data could be polluted and it is needed to fit to the training data while ignoring deviant observations[18].

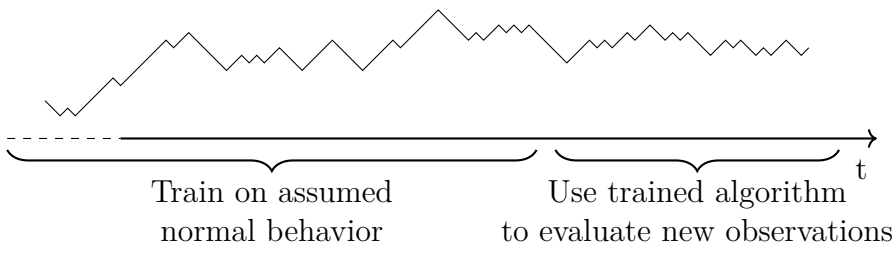


Figure 2.4: Methodology for novelty detection

2.2.1 One Class SVM(OCSVM)

The supervised learning algorithm in section 2.1.1 can be modified to be an unsupervised learning algorithm. The new algorithm is then[19]:
Solve the primal optimization problem:

$$\min_{w,\xi,\rho} \quad \frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i - \rho \tag{2.15}$$

$$\text{Subject to} \quad (w^T\phi(x_i)) \geq \rho - \xi_i \qquad \forall i = 1,\ldots,l \tag{2.16}$$

$$\xi_i \geq 0 \qquad \forall i = 1,\ldots,l \tag{2.17}$$

and its Lagrangian dual problem:

$$\min_{\alpha} \quad \sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j K(x_i,x_j) \tag{2.18}$$

$$\text{Subject to} \quad 0 \leq \alpha_i \leq C \qquad \forall i = 1,\ldots,l \tag{2.19}$$

$$\sum_{i=1}^{l}\alpha_i = 1 \qquad \forall i = 1,\ldots,l \tag{2.20}$$

And the decision function on a new point z is then:

$$f(z) = \text{sign}\left(\sum_{i=1}^{l}\alpha_i^* K(x_i,z) - \rho^*\right) \tag{2.21}$$

Where $K(x_i,z)$ is the kernel function defined in section 2.1.1. This algorithm will learn a boundary around the training points which should not contain any outliers and then be tested on unknown data points. Therefore One Class SVM falls under the novelty detection category. The boundary is not very strict and some training points might fall outside of the boundary. Either the algorithm can give a classification, $f(z) \in \{-1,1\}, -1 = \text{Anomaly}, 1 = \text{Not anomaly}$ [20] or by removing the sign function in equation 2.21 to give an anomaly score $f(z) \in \mathbb{R}^n, f(z) < 0 = \text{Anomalous}, f(z) > 0 = \text{Not anomalous}$. The greater or smaller $f(z)$ is indicates how certain it is of what class the point z belongs to.

2.2.2 Isolation Forest

By defining anomalies as observations that are few in numbers compared to the normal, as well as having different characteristics, [21] shows that anomalies are susceptible to *isolation*. Isolation in this context, is performed by using a binary tree structure (referred to as an isolation tree) where in iterations, the sample space is divided at a random value between the minimum and maximum value of a dimension chosen at random, until only a point remains within the subspace. If this process is repeated a large number of times, the average number of divisions, analogous to the depth of a binary tree, for specific observations, is shown to converge towards an average.

The method named Isolation Forest which is presented in [21] works by building

an ensemble of iTrees (called an iForest). Given a set X of data, the method generates t iTrees by subsampling ψ data points from X and creating an iTree. So ψ and t are the adjustable tuning parameters of the method, [21] proposes values of $\psi = 128$ and $t = 100$ for most applications. Performance is evaluated by computing an anomaly score s is computed using the path lengths of the trees for data points. A score of 1 will correspond to highly anomalous values, whereas values close to zero corresponds to highly regular.

2.3 Pre-processing

Several methods of pre-processing the data-set will be presented in this section. Pre-processing is useful for gaining early insights into the dataset being worked on as well as improving performance. Pre-processing can also be used to prepare raw data for input into an ML classifier. Two methods for reducing the dimensionality of data are introduced. One technique is then introduced to perform scaling on data.

2.3.1 t-distributed Stochastic Neighbor Embedding(t-SNE)

t-SNE is a relatively new algorithm for visualizing high dimensional data first introduced in 2008[22]. It maps higher dimensional data points into lower dimensional data points by minimizing the Kullback-Leibler divergance between the higher dimensional data and the lower dimensional data. $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ is the high dimensional data that will be mapped into $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$ with dimension r. The algorithm is as follows:

1. Sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I_r)$.
2. Compute p_{ij} from (2.23)
3. **for** $t = 1$ to T **do**:
   Compute q_{ij} from (2.24)
   Compute the gradient $\frac{\delta C}{\delta \mathcal{Y}}$ from (2.25)
   Compute $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$

Where η is the learning rate, T is the number of iterations and $\alpha(t)$ is the momentum.

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{2.22}$$

Where σ_i is the variance of the Gaussian that is centered on datapoint x_i.

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{2.23}$$

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{2.24}$$

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||)^{-1} \tag{2.25}$$

In this thesis, this algorithm will be implemented using `Multicore TSNE`[23] instead of `scikit-learn`[8] for improved performance.

2.3.2 Principal Component Analysis(PCA)

PCA reduces the dimensionality of a dataset $X \in \mathbb{R}^w$ to $Y \in \mathbb{R}^r$ by computing the eigenvectors of the XX^T matrix and taking the top r vectors in U[24]. The mapping to Y is then done by $Y = U_r^T X$.

2.3.3 Scaling

Scaling can improve the effectiveness of machine learning algorithms most notably in the convergence rate of gradient descent methods[25], thus decreasing the training time. There are several ways to scale a data-set[26] but only max-scaling will be used in this thesis:

$$\hat{x} = \frac{x}{\max(x)} \quad , x \geq 0 \tag{2.26}$$

This will bring $\hat{x}$ in the range $[0, 1]$.

2.4 Training and validation

For an ML-model to be implemented, data available needs to be split into three types of sets. these are sets for training, validation and lastly testing. The training set consists of sample data points used to train to fit the model. The validation set is used to get an unbiased evaluation of the model fit whilst continuously tuning hyperparameters of the ML model. Finally, the test set is used once to provide an unbiased evaluation of the model. As sets are used several time for validations, bias increases, thus risking what is refered to as model overfitting[27].

Validation and testing are large parts of the process of building an ML model, therefore tools are defined for investigating performance. An initial way to evaluate performance is by measuring the accuracy, defined as

$$\frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{2.27}$$

A confusion matrix can be used to evaluate the performance of a classifier since it will show when the algorithm predicts false positives or negatives, something that the accuracy score alone will not show. Anomalies are considered rare and therefore an algorithm would have a high accuracy score(>90%) even if it only predicted normal behavior. The layout of a confusion matrix is illustrated in Table 2.1 [28]:

Table 2.1: Confusion matrix layout

Truth\Prediction	Anomaly	Normal
Anomaly	a	b
Normal	c	d

Here, the notations a, b, c, d mean the following:

- a denotes the number of correctly classified anomalies
- b denotes the number of missed anomalies
- c denotes the number of false detections of anomalies
- d denotes the number of correctly classified normal data points

From the confusion matrix, other performance metrics for a classifier is defined. Two of these are recall and precision. Recall can be defined as

$$\text{recall} = \frac{a}{b+a} \tag{2.28}$$

and denotes the proportions of anomalies correctly identified. Precision can be defined as

$$\text{precision} = \frac{a}{c+a} \tag{2.29}$$

and denotes the proportion of anomaly cases that were correct[28].

3
System model

In this section, a model description of the proposed solutions are provided. But before this can be done, a model of the cloud application environment is given. The two solution systems which are proposed are then presented. Since the solutions share the same basic concepts, and mostly differs in where they are applied, the core concepts of both solutions are described together. Further on, the machine learning applications to the solutions are presented. Lastly, tests are defined for built prototype models for both solutions.

3.1 Application environment model

A cloud application in which it is desired to perform anomaly detection, can be described for the scope of the project as a large system made up by different types of microservices. Each microservice is designed specifically to perform different tasks within the system. A microservice can also be scaled up or down such that a variable number of instances are running simultaneously, sharing work. One microservice type will be denoted $\mathcal{A} = \{A_1, A_2, \ldots, A_{N_\mathcal{A}}\}$, where $N_\mathcal{A}$ denotes the number of instances at a given time. Instance i will be denoted A_i. The different microservices are able to output specific performance metrics, chosen by the developers which can then be fetched by querying a database server. Each A_i will have the same defined metrics. The queried metrics can be presented in graphs or other ways using analysis tools. By monitoring the metrics for anything that could indicate that something is anomalous, operators of the system could find errors. Possible actions to take in the case of anomalous behaviour, would then include triggering a downgrade to an earlier version known to be stable. The flow of data from microservice instances is illustrated in Figure 3.1.

For the model, two types of cloud environments could be identified. These look similar in that they consist of the same type of microservices, that perform the same type of tasks. The difference is their purpose and how activity is running in them. The two different types of environments are the following:

- A *pre-deployment* testing environment whose purpose is to perform scheduled defined tests using proposed updates in a simulation.
- A *live* environment which is in use by the customer, serving users continuously and receiving successful updates from the test environment.

Figure 3.2 illustrates how the environments conceptually varies in terms of when they are active or not.

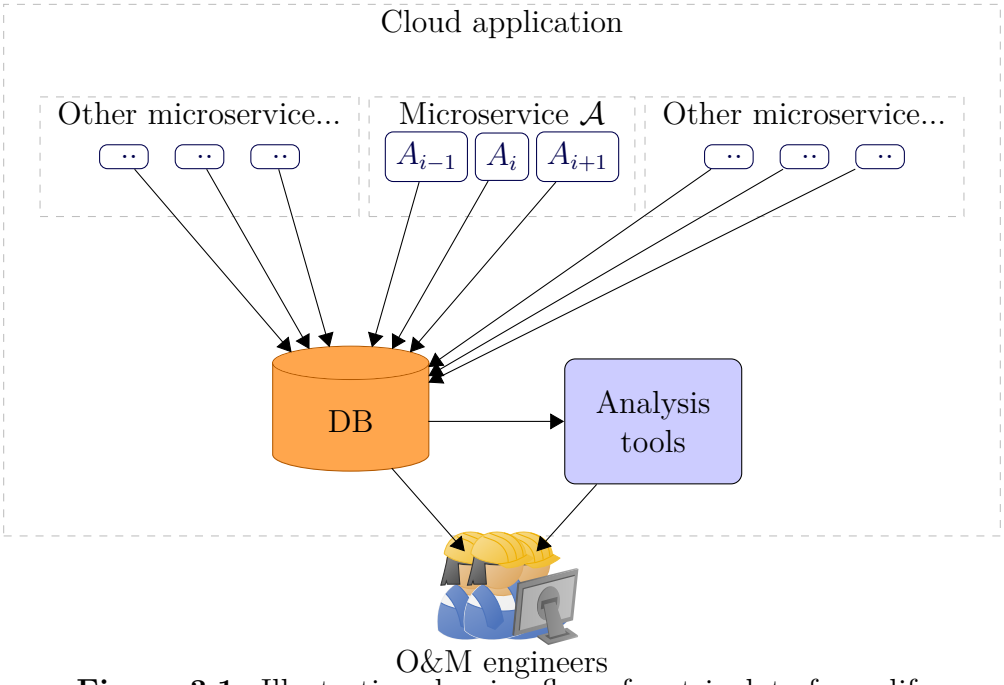


Figure 3.1: Illustration showing flow of metric data from different types of microservices to a database server, denoted DB. Analysis tools can then display graphs and gauges to help personnel in charge of operations and maintenance (O&M) to monitor the system performance. The O&M personnel will also be able to access the DB directly.

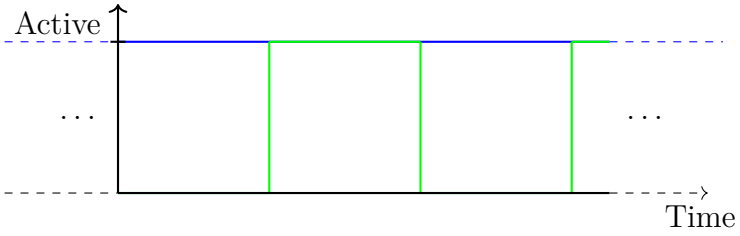


Figure 3.2: Timeline depicting how activity in the different types of environments vary over time. The blue line represents the live environment which operates continuously, and the green line represents the pre-deployment environment, which is only active during test runs.

How these two environments differ, are described in further detail in the following subsections.

3.1.1 Pre-deployment testing environment

When updates to microservices are initially committed from the development teams, they first get passed into a test environment where specific simulations are performed

using the updated microservice in a surrounding which is otherwise known not to cause anomalies. The updates will need to perform within defined constraints in order to qualify for deployment into the live environment. The constraints are defined by the developers for the specific microservice in question, with support from developers in charge of testing.

Since there are many microservices and small updates are being pushed on a regular basis, the test environment runs simulations following a schedule. This will mean that the environment is completely inactive when no tests are running, and active otherwise. After each test run, the testers and developers of the microservices have access to the test results.

3.1.2 Live environment

The live environment is the system solution which is sold to the customer, and in theory is running the latest release. The development team can run their own live environment using simulated traffic. The environment is expected to run continuously, since this is required in communications systems. Different customers will get their own respective live environments, and may not necessarily run them in the same way. Other parameters which may vary for different customers is the load put on the system as well as traffic patterns. Customers may also have their own maintenance teams.

Since the cloud application consists of various microservices, with teams working on each one, updates are committed irregularly. In a situation where an update has passed pre-deployment, and is released to $\mathcal{A}$, this may be done by running multiple instances A_i, in a way where one or more run the updated version, but most run the older version.

3.2 Anomaly detection in cloud environments

To solve the task of performing ML-based anomaly detection within a given cloud application environment, two similar solution systems are proposed. The first solution focuses on evaluating anomalous behaviour in a live environment, and the second solution will focus on evaluating anomalous behaviour during simulation runs in pre-deployment environments. The two solutions share a common flow of how to perform ML classification within a cloud environment, and this flow is described in this section.

One central design choice is that classification is performed on microservice instance level. The choice was motivated by the fact that observing on this level provides robustness to a variable number of instances over time. Each instance will be evaluated using the same classifier and comparison can then be performed between anomaly scores for instances running different versions. The flowchart in Figure 3.3 describes the overlaying structure of performing classification on a microservice instance.

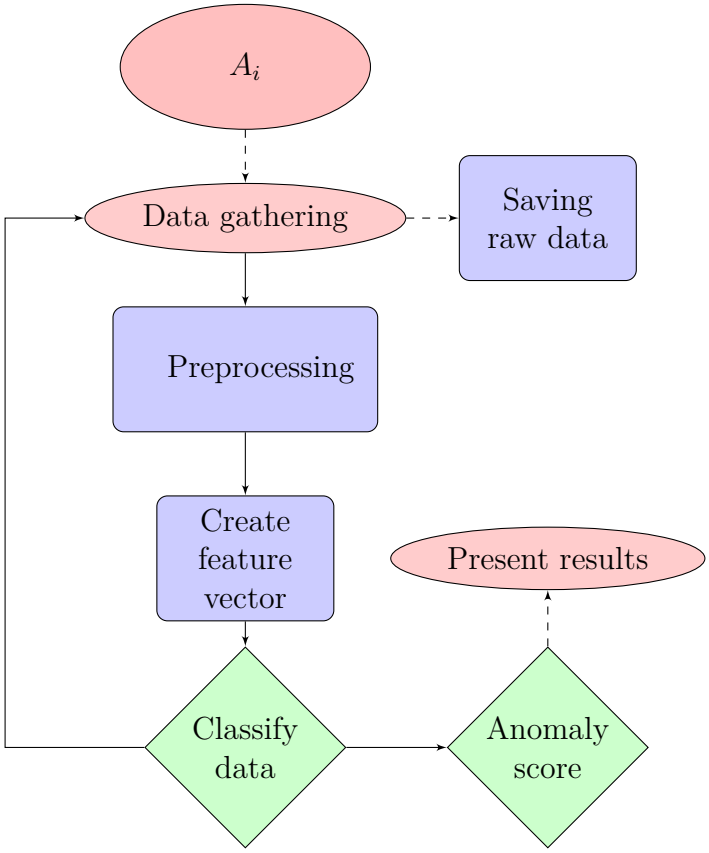


Figure 3.3: Flowchart describing how anomaly classification is done for a microservice instance. A_i is continuously queried for data, which is processed and made into feature vectors to be classified by a trained classifier. The classifier will yield an output anomaly score, which can then be presented where other performance metrics are located.

Each type of microservice has its own set of metrics which can be queried continuously. Let $q(t) = \{q_1(t), q_2(t), \ldots, q_{Nq}(t)\}$ denote the subset of metrics for a specific $\mathcal{A}$ used for a solution system. for each A_i, q is fetched and then preprocessed. The queries are also saved in order to obtain a larger data set over time. From the preprocessed queries, feature vectors are generated which can then be input into a trained classifier. The classifier will output a score of how anomalous the system is behaving given the input. These results can then be posted to be shown along with the outputs of interest for current analysis tools used by operators of the environment.

The following subsections describes the nodes of the flowchart in more detail. First, the process of how data is gathered from the DB server and preprocessed is covered. Next, feature vector design and some of the choices and trade-offs are presented. How a classifier can be trained, and then how it can operate and what the results could look like are lastly presented. In the latter subsections, some differences between the two types of environments are described.

3.2.1 Data gathering and preprocessing

For an operating instance of microservice $\mathcal{A}$, A_i, N_q queries will be sent to the DB-server to get $q(t)$. Each query corresponds to a parameter $q_i(t)$. By design, $q_i(t)$ can be queried from time t and backwards, and samples are received with a period of Δt. Let M denote how many historical samples are queried in addition to the instantaneous sample $q_i(t)$. An output matrix $Q_i(M,t)$ can be described as shown in Equation 3.1 to represent the full set of queries for time $t : t - M\Delta t$.

$$Q_i(M,t) = \begin{bmatrix} q_1(t) & q_2(t) & \dots & q_{N_q}(t) \\ q_1(t-\Delta t) & q_2(t-\Delta t) & \dots & q_{N_q}(t-\Delta t) \\ q_1(t-2\Delta t) & q_2(t-2\Delta t) & \dots & q_{N_q}(t-2\Delta t) \\ \vdots & \vdots & \ddots & \vdots \\ q_1(t-M\Delta t) & q_2(t-M\Delta t) & \dots & q_{N_q}(t-M\Delta t) \end{bmatrix} \tag{3.1}$$

After the set of data matrices has been fetched, each Q_i has to be normalized in the same way. This is done by finding a vector $z = \{z_1, z_2, \dots, z_{Nq}\}$ and calculating the normalized $Q_i(t)$ as

$$Q_{i,\text{norm}}(M,t) = Q_i(t) \bullet \frac{1}{z_i}. \tag{3.2}$$

To find z_i, all matrices $Q_i(t), \quad i = 1, 2, \dots, N_Q$ available when training, concatenated into one tall matrix Q is considered.

$$Q = \begin{bmatrix} Q_1 & Q_2 & \dots & Q_{N_Q} \end{bmatrix}^T \tag{3.3}$$

z_i will be given as the maximum of the i'th column of Q. Max scaling was used since from observing the metrics being queried it was concluded that the metrics never passed below zero and actually was zero very often. Thus, the need for min-max scaling was discarded. This scaling factor is saved and applied on new data that is acquired.

3.2.2 Feature vector design

After $Q_i(M,t)$ has been gathered and normalized, further processing is done in order to create a feature vector $x(t)$ which can be input into the classifier. Since the feature vector is varying over time, with dependent behaviour, further information could be embedded into the classifier, instead of just feeding in the instantaneous normalized version of $q(t) = \{q_1(t), q_2(t), \dots, q_{Nq}(t)\}$. For a communication system, it would, for example, be expected that more traffic happens during the day, than during the night. To embed this type of information about behaviour over time, an option could be to feed in a vectorized version of the normalized output matrix of raw data, $\text{vec}(Q_{i,\text{norm}}(M,t))$. The trade-off of this approach is that the feature vector length grows proportional to how M is chosen.

The approach chosen for the solution systems was to instead use $Q_{i,\text{norm}}(M,t)$ in

order to calculate statistical information about each metric in $q(t)$, and then to append this data in order to create $x(t)$. Another choice was to do calculate statistical information using multiple number of sample windows, yielding the possibility to gain multiple time perspectives. This is motivated by the fact that fluctuations are expected to happen over different time perspectives. For the example of user activity within a communication system, there might not only occur periodic behaviour over a full day, but also weekly, or even larger time windows.

The statistical information chosen for the solution systems were the statistical mean, and standard deviation, and the following notation is used. if one column of the matrix of raw data, when w_k historical samples have been added, is denoted by $q_{i,t-w_k\Delta t:t}$, Let

$$E[q_{i,t-w_k\Delta t:t}] = \frac{1}{w_k}\sum_{k=0}^{w_k} q_i(t - w_k\Delta t) \tag{3.4}$$

$$\sigma(q_{i,t-w_k\Delta t:t}) = \sqrt{E[(q_{i,t-w_k\Delta t:t} - E[q_{i,t-w_k\Delta t:t}])^2]} \tag{3.5}$$

denote the statistical mean, as well as the standard deviation for a metric q_i, with a sample windowsize of (w_k+1) samples. The feature vector containing the normalized original metrics, along with its moving statistics (MS) using different windowsizes, can now be illustrated as a matrix, shown in equation 3.6.

$$x_{\text{mat}}(t) = \begin{bmatrix} q_1(t) & q_2(t) & \dots & q_N(t) \\ E[q_{1,t-w_1\Delta t:t}] & E[q_{2,t-w_1\Delta t:t}] & \dots & E[q_{N,t-w_1\Delta t:t}] \\ E[q_{1,t-w_2\Delta t:t}] & \ddots & & \vdots \\ \vdots & & & \vdots \\ E[q_{1,t-w_{\max}\Delta t:t}] & & & E[q_{N,t-w_{\max}\Delta t:t}] \\ \sigma(q_{1,t-w_1\Delta t:t}) & \sigma(q_{2,t-w_1\Delta t:t}) & \dots & \sigma(q_{N,t-w_1\Delta t:t}) \\ \sigma(q_{1,t-w_2\Delta t:t}) & \ddots & & \vdots \\ \vdots & & & \vdots \\ \sigma(q_{1,t-w_{\max}\Delta t:t}) & \dots & \dots & \sigma(q_{N,t-w_{\max}\Delta t:t}) \end{bmatrix} \tag{3.6}$$

Since most ML methods implemented into software takes vectors as inputs, this is how $x(t)$ is defined. $x(t)$ is received by concatenating each column of $x_{\text{mat}}(t)$. This operation is illustrated in Equation 3.7

$$x(t) = \text{vec}(x_{\text{mat}}(t)) \tag{3.7}$$

Using two different statistical measures, and N_w number of different windows, the length of the feature vector can be formulized according to Equation 3.8.

$$l_x = N_q + 2N_qN_w \tag{3.8}$$

In Figure 3.4, a small examples using $N_q = 1$ illustrates how many elements are needed to be in the feature vector, to bring information about n number of historical samples into it.

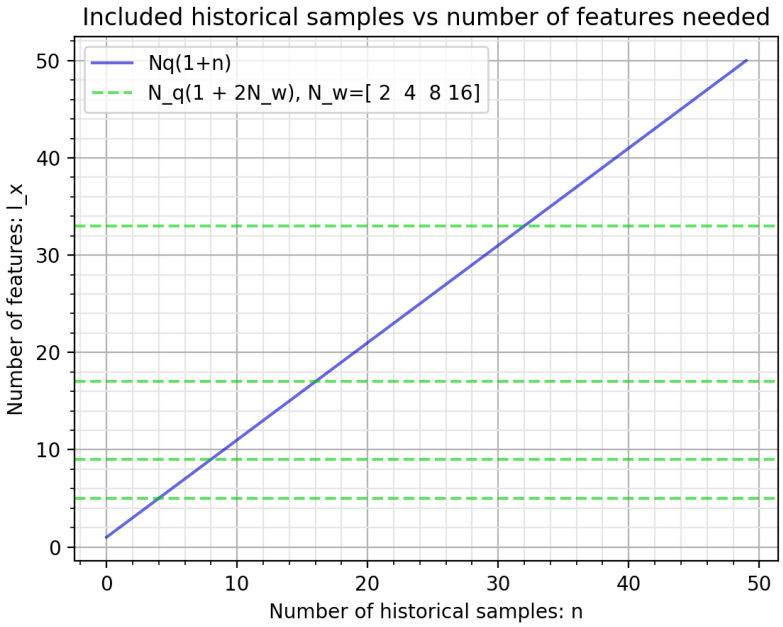


Figure 3.4: The plot illustrates the feature vector length function for the two presented approaches of looking at historical data, for the case $N_q = 1$. The solid line illustrates a linear growth as more historical samples are included into the feature vector, while the dashed line illustrates how only the number of windows, N_w affects feature length in the MS approach.

From the figure, it can be seen that using the statistical information approach presented, the amount of features will only depend on the number of windows which are desired to be included, for an arbitrary number of historical samples n.

If the age t of a subapplication instance does not fulfill the inequality

$$t \geq w_{\max} \Delta t, \tag{3.9}$$

an adaptation will have to be done to create the feature vector, since the window would exceed the amount of available historical data. The strategy that was implemented is to make the windowsize as large as possible for all w_k in the vector of windowsizes for which $w_k \Delta t \geq t$.

3.2.3 Training the classifier

Before classification can be done, a classifier will need to be trained using a saved set of samples

$$X = [x(t_1), x(t_2), \ldots, x(t_N)]^T \tag{3.10}$$

with N samples drawn, were time instances $t_1, t_2, \ldots, t_N$ can be independent. To create, and add to this set, the raw queried data is continuously saved. Because microservices can be created and terminated continuously, data has to be gathered over entire life cycles of several instances.

Depending on which ML technique is desired to create the classifier, labeled or unlabeled data is required. For a sample set X containing N samples, labels are defined as a vector

$$Y = [y_1, y_2, \ldots, y_N]^T \tag{3.11}$$

and needs to be added after recording. What the labels might look like will depend on what ML technique is used, but with the most common and realistic scenario being a binary label for anomalous or not.

If labels are not available for the dataset, this restricts the options for what types of ML techniques can be applied. If the assumption can be made that most data is not anomalous, another approach is then to redefine $x(t)$ such that one feature $x_i(t)$ is instead defined as the label $y(t)$, and removed from the feature vector. The ML classifier would then be trained to predict $y(t)$ during operation. If this prediction deviates too much from the true value, anomalous behavior is detected. This method has the advantage of being ”self labeling” and could fit in between supervised and unsupervised learning.

Once a set is generated, it could be divided into subsets for training, validation and testing. A more advanced approach to gain more from the data is to apply cross validation techniques.

3.2.4 Classification and presentation

Once the classifier is trained, it is used to evaluate all instances of the focused microservices individually, as new data is available to query. Any solution system will need to keep track of new instances initiated, so as to always check the whole microservice for anomalous behaviour.

An important difference between the two defined types of environments is that in the pre-deployment type of environment, classification will only have to be active when tests are running. Additional functionality to keep track of the schedule is therefore needed for the corresponding solution system.

The output of the classifier will differ depending on what ML technique is implemented. For the techniques used which outputs an anomaly score directly, the outputs will be the soft anomaly score $\in [0, 1]$, as well as a hard decision output $\in \{0, 1\}$. For the techniques used which performs prediction of one variable, the output will be the prediction, denoted $\hat{y}$, the measured value for the variable y, and lastly a decision function output $\in \{0, 1\}$. A detected anomaly is described by defining

$$\epsilon = aE\left[|\hat{Y} - Y|\right], \tag{3.12}$$

where $a > 0$ is a tuning parameter. The decision is then made by the following inequality:

$$f(\hat{y}_t) = \begin{cases} 1 & |\hat{y}_t - y_t| > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

The classifier output should be seen as part of the analysis tools for an environment. Therefore the output should be presented as a function of time and made available to monitoring interfaces where visualizations of current analysis tools are.

3.3 Metric choices for queries

A part of the process of implementing ML in a given problem is that of feature engineering. By choosing input parameters to include into $q(t)$, anomaly detection could be more efficient. An initial design question is why not to include all available metrics that can be queried instead of only a subset. A potential benefit of including all available metrics is that the methods used are easier to carry over to any type of microservice, without having to spend resources on feature engineering every time. Problems that could arise using this approach are in conjunction with when updates are released. An update might include changes to how some variables work or even additions/subtractions of variables, which would cause malfunctions or errors if the classifier is not re-trained. The classifier might also end up detecting the very fact that there was an update, and interpret this as the anomalous event, rather than the types of malfunctions which are desirable to find. Since continuous updating of the microservices is a large part of the cloud native design approach, the choice of approach to feature engineering was to only select a subset of metrics and that this would look different for different types of microservices.

The choice of what subset of metrics to include, was based on what type of anomalies were desired to be detected. The classifier should find behaviour indicating malfunctions, rather than just slightly updated behaviour. A potential way to create this boundary, is to focus on the experience of the user, which should be similar over updates, or in some cases improve. Detected anomalies should then be when performance degrades, but also when it improves, both of which are viable cases to trigger a roll-back and further analysis of the outcome. The decision as to what subset of metrics are valuable for a particular microservice in general, should be up

to the developers of the microservice together with testers.

Examples of metrics which are related to user experience could be different types of latency metrics or successful/unsuccessful operations. Other important factors are also how much resources are used to run the microservice, since it will relate to costs. In general, focus was put on redesigning any relevant metric that were defined as counters, and rather use their growth/decline rates.

3.4 Ethical aspects of model

When performing data gathering it is important to consider ethical aspects, information that compromises the privacy of users are not to be used. These ethical aspects will have to be respected, even if it comes at the cost of potentially decreasing performance. Not handling sensitive information within the solution could also be seen as advantageous since it reduces exposure in case of a security breach.

3.5 Generating datasets using prototypes

Two prototype solutions were constructed to handle each of the two types of cloud environments. For testing and evaluation purposes, the prototypes were tweaked compared to the described solution model. This was done so that defined datasets could be created. Instead of performing classification in a live setting, the defined datasets are used with different classifiers in a similar fashion, but separately from the prototype. Having fixed datasets throughout testing simplifies analysis between different types of classifiers.

One dataset was created for each type of environment, and these are described further in the following two subsections.

3.5.1 Pre-deployment

The prototype system evaluating the pre-deployment environment was tweaked such that data during testruns were saved to generate a defined dataset. 50 test runs were saved and these were divided into training, evaluation and test sets. 40 runs makes up the training set, 5 makes up the evaluation set and the last 5 were defined as the test set. In total there are 7294 collected data points in this data set. The training consists of 6148 normal points, the evaluation set consists of 483 normal points and 92 anomalous points and the test set consists of 443 normal points and 128 anomalous points. The amount of queries used in this dataset was 8 different queries. With window sizes 5 and 10 for the moving statistics, the resulting feature vector is 40 elements long.

The intention was to be able to test on runs that were faulty, but this intention could not be realized. Instead, the choice was to revert to adding generated noise to the data set.

Noise was added at different places in the data set. The noise was only added to the evaluation set and test set. How noise was added is described in equation 3.14.

$$Q_{\text{noise}}(t) = (1 + Q(t)) \, |\mathcal{N}(\mu, \sigma)| \tag{3.14}$$

Here, $|\mathcal{N}(\mu, \sigma)|$ describes a folded version of the normal distribution with $\mu = 0$ and $\sigma = 1$. The folded version was used because the noise could make $Q_{\text{noise}}(t) < 0$ which would not a realistic outcome for the particular environment. The $(1 + Q(t))$ was used because when $Q(t) \approx 0$, noise should still have an effect on the polluted samples. Multiplicative noise was used because the noise should have similar scale independent of what the values of the different $q_i(t)$'s are.

3.5.2 Sensitivity analysis in pre-deployment

A group of data sets were also created along side the set described in subsection 3.5.1. The purpose of these sets were to be used in tests with varying degrees of noise pollution, more specifically: $\sigma \in [0.1, 0.3, 0.5, 0.8]$ as well as absence of noise. The "no noise case" does not use equation 3.14, but rather generates features directly from $Q(t)$.

3.5.3 Live environment

A separate live environment was provided for the purpose of testing the corresponding solution prototype. In order to simulate anomalies, intentional software bugs were inserted into the microservice software. The main objective was to affect the performance of the system and hence the metrics. Whenever the micro-service was called, one of three things could happen.

1. 5% chance to pause the microservice for $X \sim \text{Uni}[0, 10]$ milliseconds and then continue with normal functionality.
2. 20% chance to calculate π with 50 bits of precision and $X \sim \text{Uni}[0, 30]$ iterations and then continue with normal functionality.
3. 75% chance to function as designed.

These injected behaviours affect performance of the system in terms of for example CPU load and response time/latencies. The motivation was not to insert faulty code that would break the system but rather to simulate a software update that yielded worse performance but was otherwise still fully functional.

The data set collected contained in total 2199 data points, with 733 normal points in the training set, 380 normal points and 353 points being anomalous in the evaluation set and 353 normal points and 380 anomalous points in the test set. The data set is also shuffled for good practice[29]. The window sizes are 5 and 10 for the moving statistics which with 13 different queries results in a feature vector that is 65 elements long.

3.6 Descriptions for evaluation tests

This section describes test setups for the datasets gathered from the prototype systems. Various ML-techniques, described in Chapter 2, were implemented to create different classifiers for comparison. How the classifiers were compared and what type of outputs are yielded is also presented.

The first test describes the process of using decomposition techniques to visualize datasets in two dimensions. The tests that follow describes implemented ML-classifiers using supervised, as well as unsupervised learning approaches on the created datasets from both prototype systems. In all classifiers presented, an evaluation is done using a confusion matrix.

3.6.1 Visualization of data using decomposition techniques

The t-SNE and PCA methods described in 2.3.1 and 2.3.2 were used to visualize the various datasets in 2-dimensional plots. In one plot, all of the high dimensional datapoints of the dataset is represented in the decomposed dimensions with the given technique. Anomalous and normal data points are denoted differently within plots. For PCA, the original `scikit-learn` implementation was used, and for t-SNE, the modified implementation was used for faster evaluation.

3.6.2 Tests using unsupervised algorithms

Two types of unsupervised classifiers are used. Those are

- OCSVM
- Isolation Forest

The OCSVM was implemented using the default values in `scikit-learn` except for `nu`. `nu` is the upper bound on training errors and a lower bound on the amount of support vectors(i.e. how strict the boundary function has to be with the training points). The rest are set to their default values. The parameter was tuned on the evaluation set and tested on the test set. The outputs of the algorithm are both the binary predicted anomaly label(equation 2.21) as well as the function value from equation 2.21 without the sign function.

The Isolation Forest classifier was implemented using default values in `scikit-learn` except for the parameter `contamination`. The parameter was tuned using the evaluation set. The results for Isolation forest are presented very similar to how the results for One Class SVM's were presented, with binary prediction value and the continuous decision function.

3.6.3 Tests using supervised algorithms

Three types of classifiers were used. Those are

- Neural network
- Auto encoder

- SVR

The neural network consits of 10 layers, where 8 are hidden layers with 32 nodes. The input layer has dimension size equal to the number of features and the output layer has dimension 1. The model was trained with a fixed number of epochs and used the "Adam" optimizer in `Keras`[30]. The loss function was "Mean Squared Error".

The auto-encoder consists of 9 hidden layers which decreases in dimensions and then increases in the second half. More precisely, the hidden layers has dimensions $[25, 20, 15, 10, 5, 10, 15, 20, 25]$. The input and output layer is equal to the number features of the data set. The model was trained with the "Adam" optimizer in `Keras` and with a fixed number of epochs. The loss function was "Mean Squared Error".

Every parameter for `scikit-learn`'s implementation of SVR was set to default values.

The classifier outputs were connected to a threshold function(equation 3.12) and during training the threshold was tuned.

3.6.4 Sensitivity analysis in pre-deployment

One of the best performing algorithms will be used for further testing and analysis using the pre-deployment data-set as described in section 3.5.2. The parameters for the algorithm will be untouched throughout the different test cases. This is to give a fair comparison between each test case.

3.6.5 Testing with unsupervised algorithms

As mentioned in section 2.2, the model should be pre-trained on not anomalous data points before giving a prediction on new data points. Therefore in the live environment case, data has to be considered normal and used to train the model before deployment at a costumer.

Compared with novelty detection, outlier detection has to be retrained every time a new set of data points has to be evaluated. Depending on the algorithm, this could cause unwanted amounts of computational power on the platform it is deployed on.

Retraining the model on regular intervals should be done to keep the model up-to-date with current system behavior.

3.6.6 Testing with supervised algorithms

Since neural networks requires a large amount of training data, the following results will not perform optimally because the size of the available data-set is relatively small. However, results will still be presented as a proof of concept for their methodology.

3.6.7 Sensitivity analysis testing

The parameters for the algorithm will be untouched throughout the different test cases. This is to give a fair comparison between each test case. Otherwise everything will be tested exactly the same way as in section 3.6.5.

4

Results from evaluation tests

This chapter presents results of the evaluation tests defined in 3.6. The visualization tests results are presented first for both the pre-deployment dataset, as well as the live environment dataset. The two sections to follow posts results from both unsupervised and supervised tests done on the pre-deployment set. Results of the defined sensitivity analysis is then shown in the section after. in the last two sections, unsupervised as well as supervised classifiers are evaluated for the live environment dataset.

4.1 Visualization using PCA and t-SNE

4.1.1 Pre-deployment set

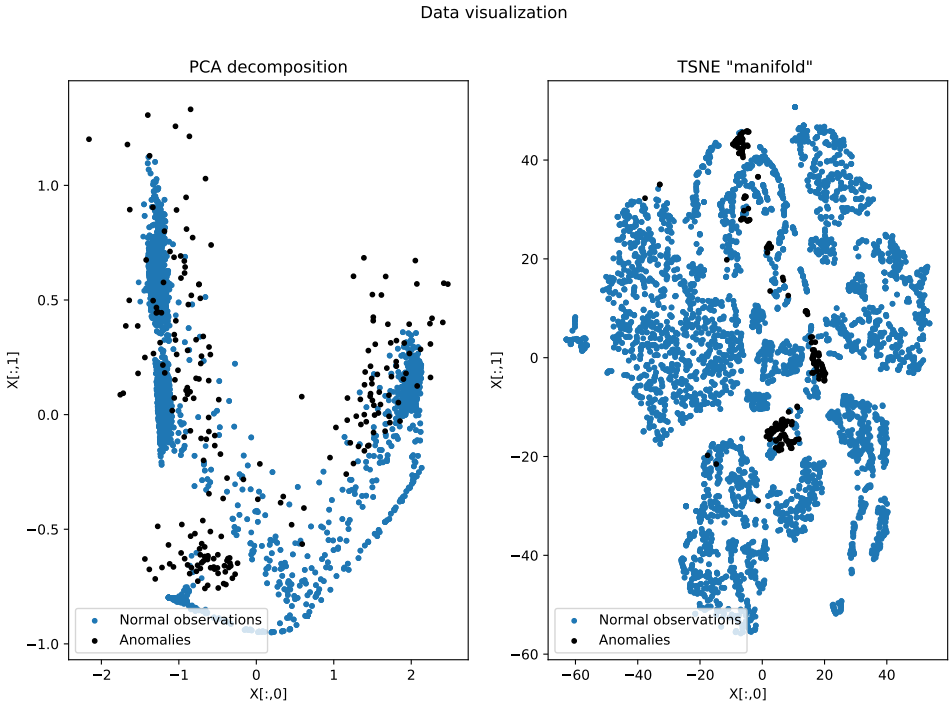


Figure 4.1: PCA and t-SNE visualizations for the pre-deployment set.

As shown in figure 4.1, the PCA decomposition didn't manage to separate the anomalous points from the normal points in a meaningful and useful way. The t-SNE manifold did however manage to cluster the anomalous points together with little overlap with the normal points.

4.1.2 Live environment set

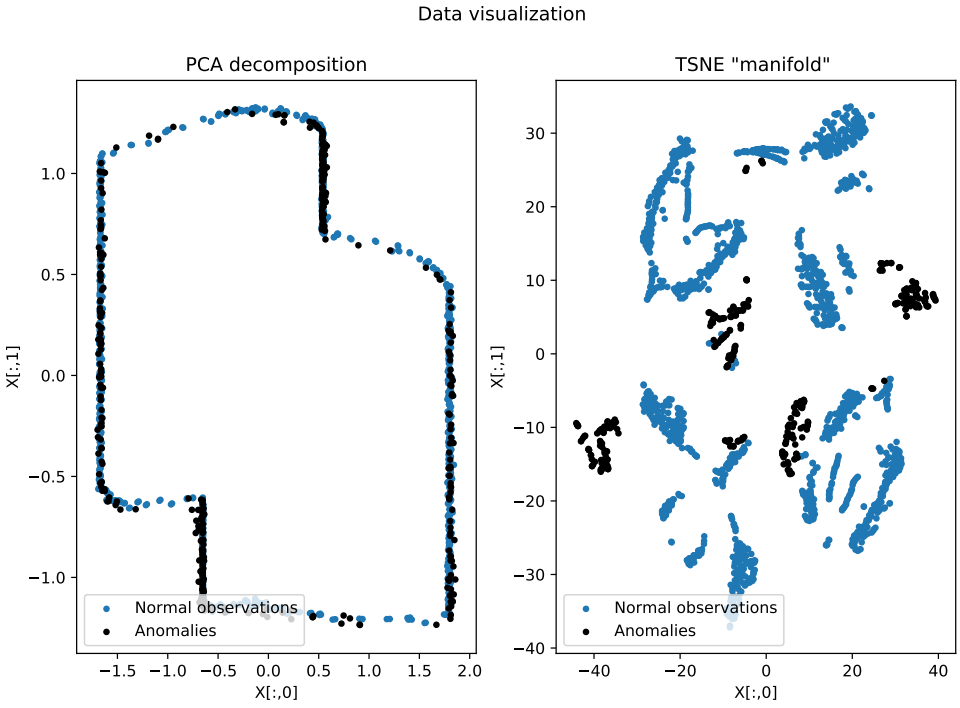


Figure 4.2: PCA and t-SNE visualizations for the live environment set.

The anomalous points in the PCA decomposition in figure 4.2 lay exactly on the normal points. The t-SNE manifold actually managed to separate the points from each other in distinct clusters.

4.2 Unsupervised learning in pre-deployment

4.2.1 One Class SVM

The tuned parameter used for `scikit-learn`'s One Class SVM is:

- `nu` = 0.08

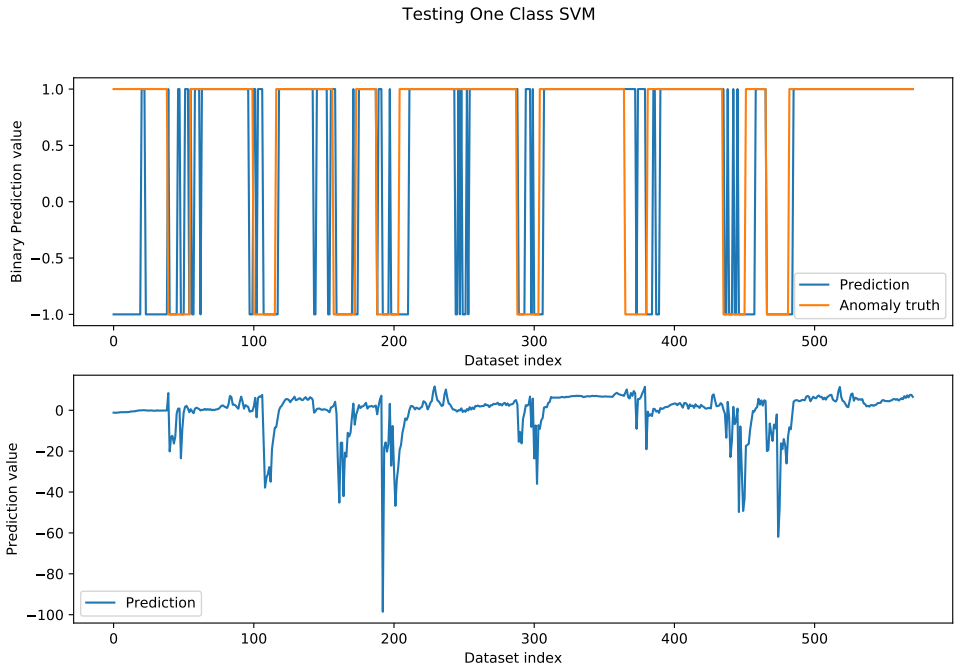


Figure 4.3: One Class SVM predictions for the pre-deployment set.

Table 4.1: Confusion matrix for One Class SVM

Truth\Prediction	Anomaly	Normal
Anomaly	85	43
Normal	84	359

Scores for One Class SVM:

- Accuracy: 77.8%
- Recall: 66.4%
- Precision: 50.3%

As shown in figure 4.3, the predicted value sometimes dips below zero and therefore causes an anomalous prediction. This can clearly be seen in the beginning of the dataset between indexes 0 and $\approx$ 40. While at other places in the dataset the prediction value is very certain of that there is an anomaly present like at indexes $\approx$ 190 to 210.

4.2.2 Isolation Forest

The parameters used for `scikit-learn`'s Isolation Forest are:

- `contamination` = 0.006

The rest are set to their default values.

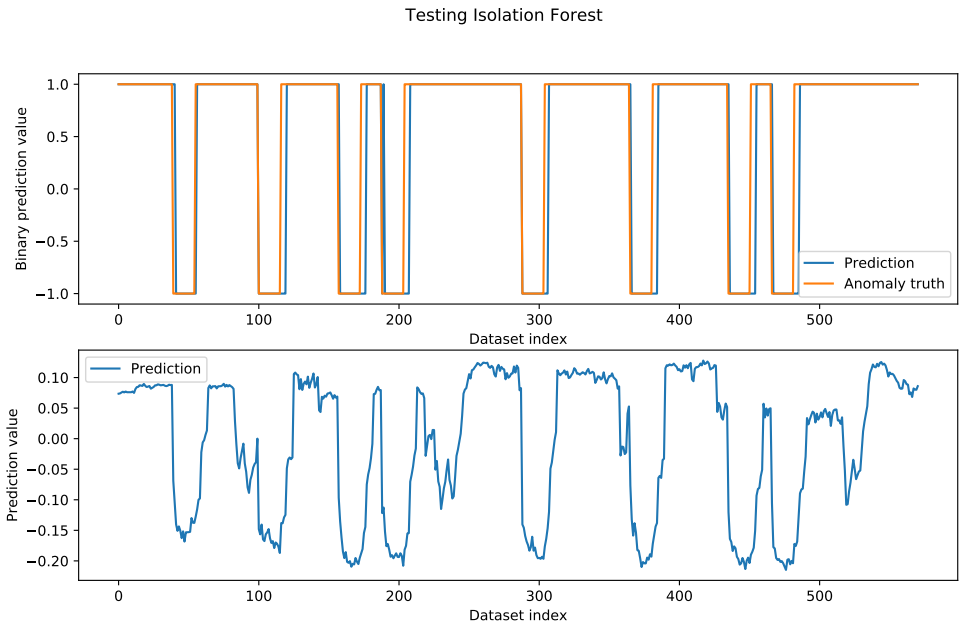


Figure 4.4: Isolation Forest predictions for the pre-deployment set.

Table 4.2: Confusion matrix for Isolation Forest

Truth\Prediction	Anomaly	Normal
Anomaly	121	7
Normal	28	415

Scores for Isolation Forest:

- Accuracy: 93.98%
- Recall: 94.5%
- Precision: 81.2%

It almost perfectly predicts where the anomalies were added in the dataset. Only a few indexes were missed both before and after the anomaly points.

4.3 Supervised learning in pre-deployment

4.3.1 Neural network

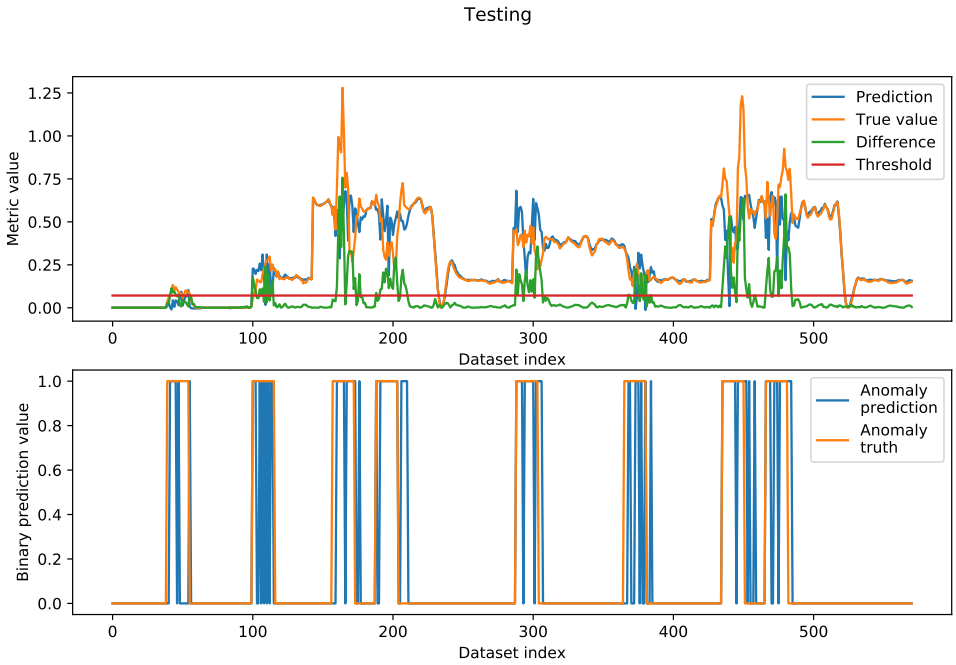


Figure 4.5: Neural network predictions for the pre-deployment set.

Table 4.3: Confusion matrix for Neural Network

Truth\Prediction	Anomaly	Normal
Anomaly	88	40
Normal	18	425

Scores for Neural Network:

- Accuracy: 89.8%
- Recall: 68.8%
- Precision: 83.0%

The prediction threshold a from equation 3.12, was set to 1.4. In figure 4.5 it can be seen that it correctly predicts where the anomalies are located. But the difference between the prediction and the true value goes up and down meaning that the binary prediction also does the same and therefore causing many spikes in the binary prediction and also affecting the accuracy score.

4.3.2 Auto-encoder

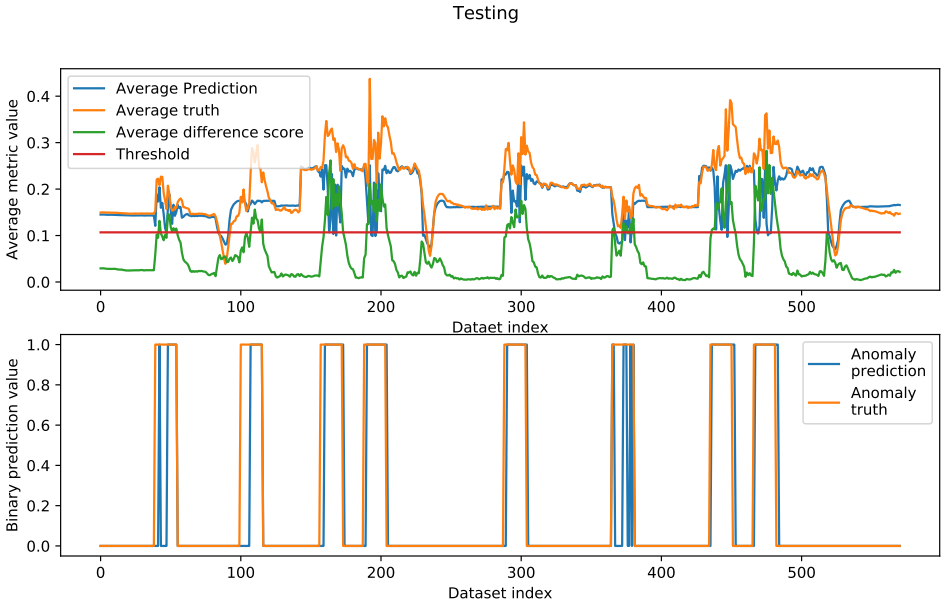


Figure 4.6: Auto-encoder predictions for the pre-deployment set.

Table 4.4: Confusion matrix for Auto-encoder

Truth\Prediction	Anomaly	Normal
Anomaly	100	28
Normal	13	430

Scores for Auto-encoder:

- Accuracy: 92.8%.
- Recall: 78.13%
- Precision: 88.5%

The size of the hidden layers of the auto-encoder is

$$[40, 25, 20, 15, 10, 5, 10, 15, 20, 25, 40] \tag{4.1}$$

The number of epochs is 1000. The threshold parameter a is 2.1. Similarly to the neural network in section 4.3.1, the auto-encoder performed very well. With the occasional missed predictions such as at around index 100 in figure 4.6.

4.3.3 SVR

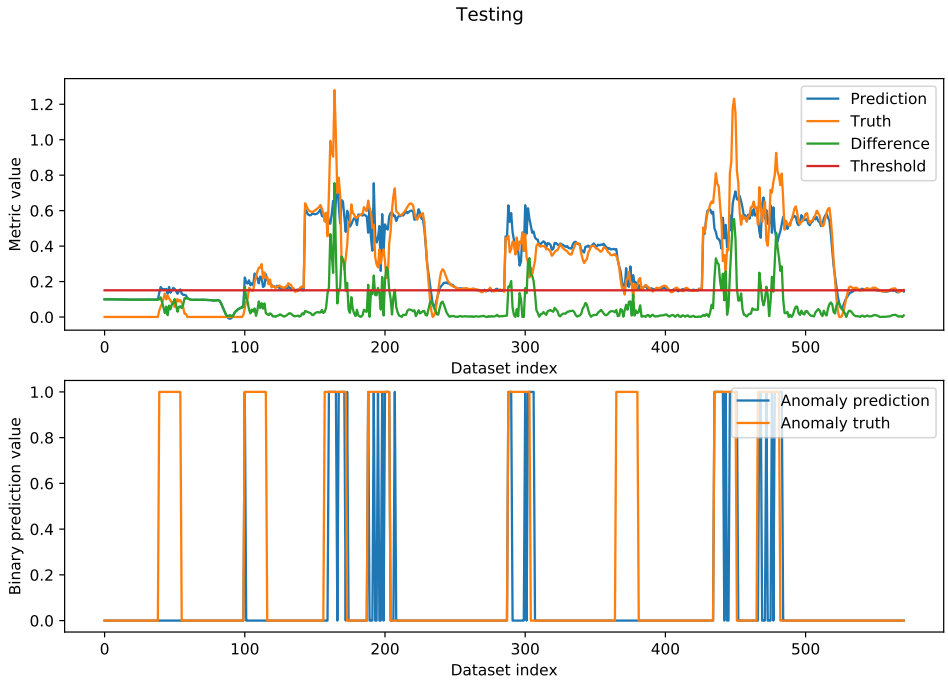


Figure 4.7: SVR predictions for the pre-deployment set.

Table 4.5: Confusion matrix for SVR in pre-deployment

Truth\Prediction	Anomaly	Normal
Anomaly	47	81
Normal	8	435

Scores for SVR:

- Accuracy: 84.4%.
- Recall: 36.7%
- Precision: 85.5%

The threshold scale parameter was set to $a = 2.5$. There are some missed predictions in the beginnings of the dataset and at index around 370 because of the threshold being a bit to large.

4.4 Sensitivity analysis

The algorithm chosen for sensitivity analysis using the pre-deployment implementation, was the neural network. The neural network had 10 hidden layers with dimension 32. The input layer had 39 layers and the output layer has 1 layer. The model was trained with 500 epochs and used the ”Adam” optimizer in `Keras`. The loss function was ”Mean Squared Error”. The prediction threshold a from equation

3.12, was set to 1.4.

When no noise was added in figure 4.8, the neural network predicted the true value very precisely. This caused the anomaly prediction threshold in equation 3.12 to be very small and predict many anomalies in the data set. The resulting accuracy score is 77.4%. Recall and precision will not say anything useful because there are no anomalies in the dataset which means they will both be 0%.

When the noise had varying degrees of standard deviation, the resulting accuracy score went from 76.7% with $\sigma = 0.1$ to 89.3% with $\sigma = 0.8$.

4.4.1 No noise

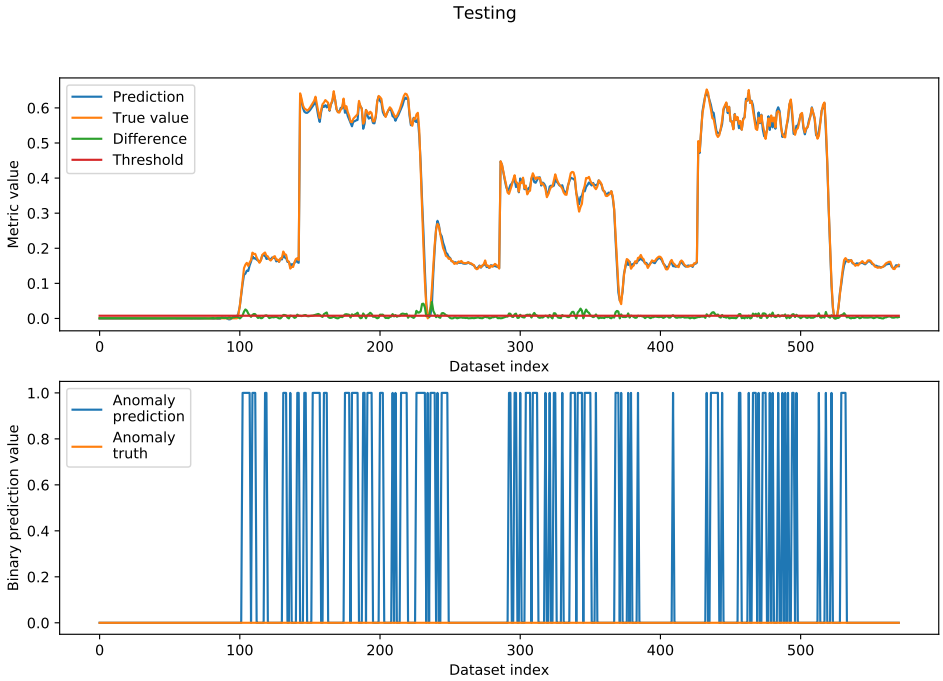


Figure 4.8: Neural network predictions on the pre-deployment set with no noise added.

Table 4.6: Confusion matrix for Neural Network with no noise added

Truth\Prediction	Anomaly	Normal
Anomaly	0	0
Normal	129	442

4.4.2 Noise with $\sigma = 0.1$

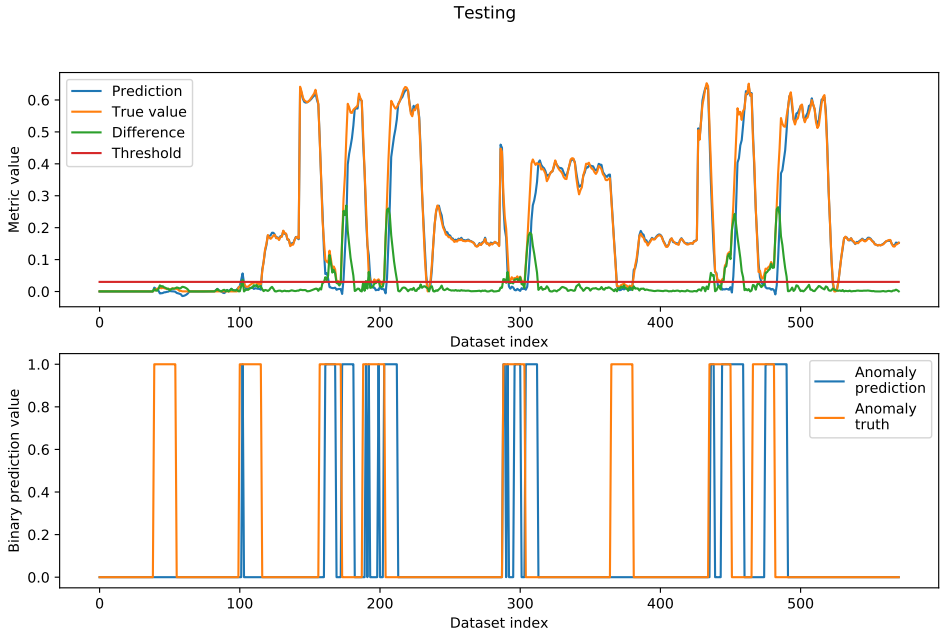


Figure 4.9: Neural network with noise with $\sigma = 0.1$ added.

Table 4.7: Confusion matrix for Neural Network with noise with $\sigma = 0.1$ added.

Truth\Prediction	Anomaly	Normal
Anomaly	40	88
Normal	45	398

Scores for neural network with $\sigma = 0.1$:

- Accuracy: 76.7%.
- Recall: 31.2%.
- Precision: 47.1%.

4.4.3 Noise with $\sigma = 0.3$

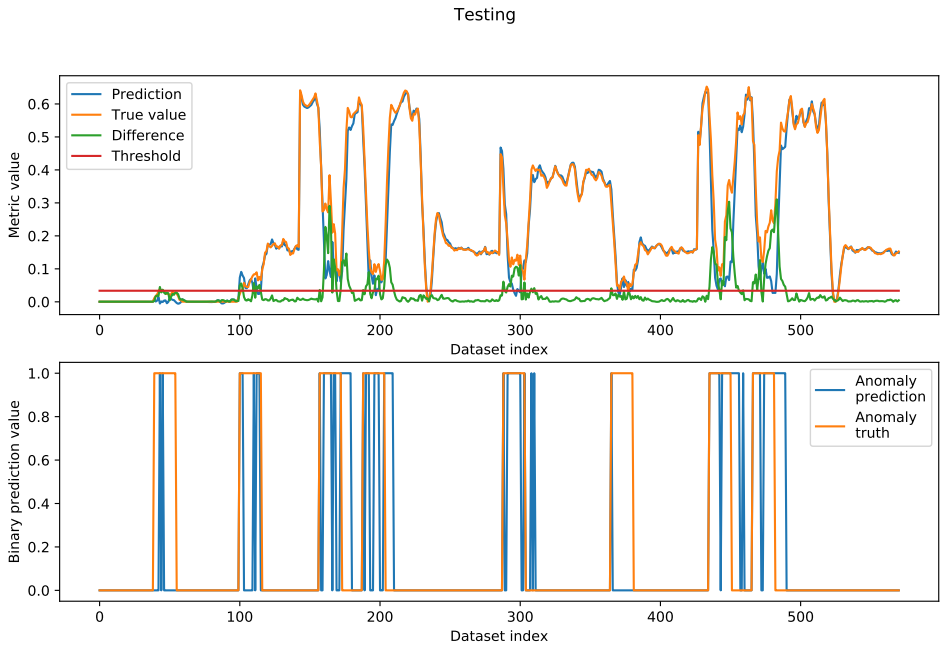


Figure 4.10: Neural network on the pre-deployment set with noise with $\sigma = 0.3$ added.

Table 4.8: Confusion matrix for Neural Network with noise with $\sigma = 0.3$ added.

Truth\Prediction	Anomaly	Normal
Anomaly	69	59
Normal	39	404

Scores for neural network with $\sigma = 0.3$:

- Accuracy: 82.8%.
- Recall: 53.9%.
- Precision: 63.9%.

4.4.4 Noise with $\sigma = 0.5$

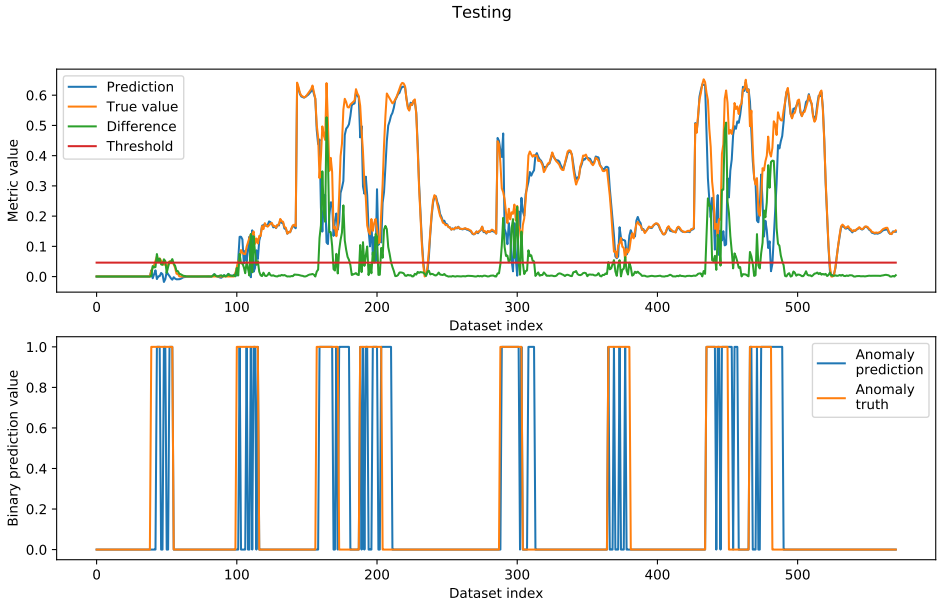


Figure 4.11: Neural network on the pre-deployment set with noise with $\sigma = 0.5$ added.

Table 4.9: Confusion matrix for Neural Network with noise with $\sigma = 0.5$ added.

Truth\Prediction	Anomaly	Normal
Anomaly	72	56
Normal	33	410

Scores for neural network with $\sigma = 0.5$:

- Accuracy: 84.4%.
- Recall: 56.3%.
- Precision: 68.6%.

4.4.5 Noise with $\sigma = 0.8$

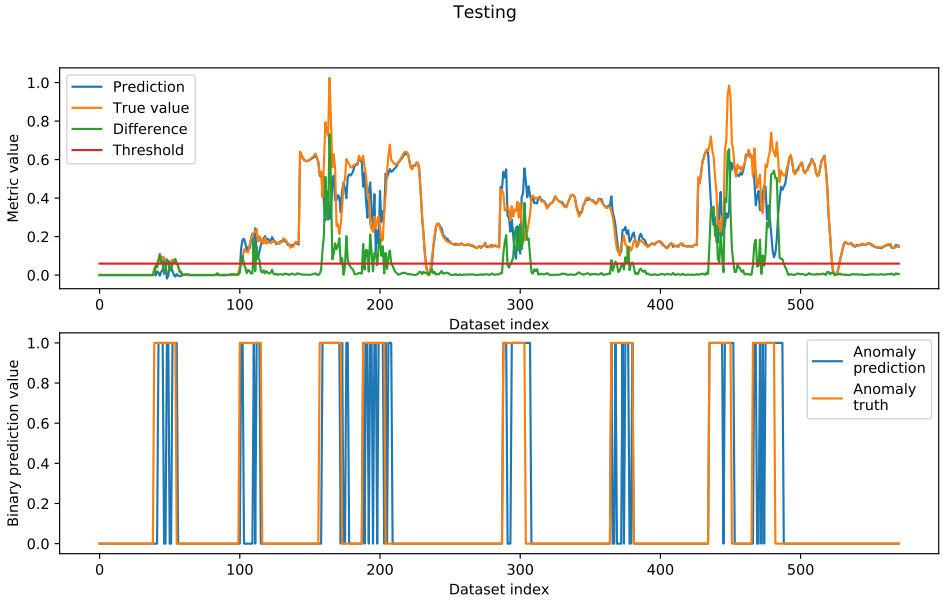


Figure 4.12: Neural network predictions on the pre-deployment set with noise with $\sigma = 0.8$ added.

Table 4.10: Confusion matrix for Neural Network with noise with $\sigma = 0.8$ added.

Truth\Prediction	Anomaly	Normal
Anomaly	87	41
Normal	20	423

Scores for neural network with $\sigma = 0.8$:

- Accuracy: 89.3%.
- Recall: 68.0%.
- Precision: 81.3%.

4.5 Unsupervised learning in live environment

4.5.1 One Class SVM

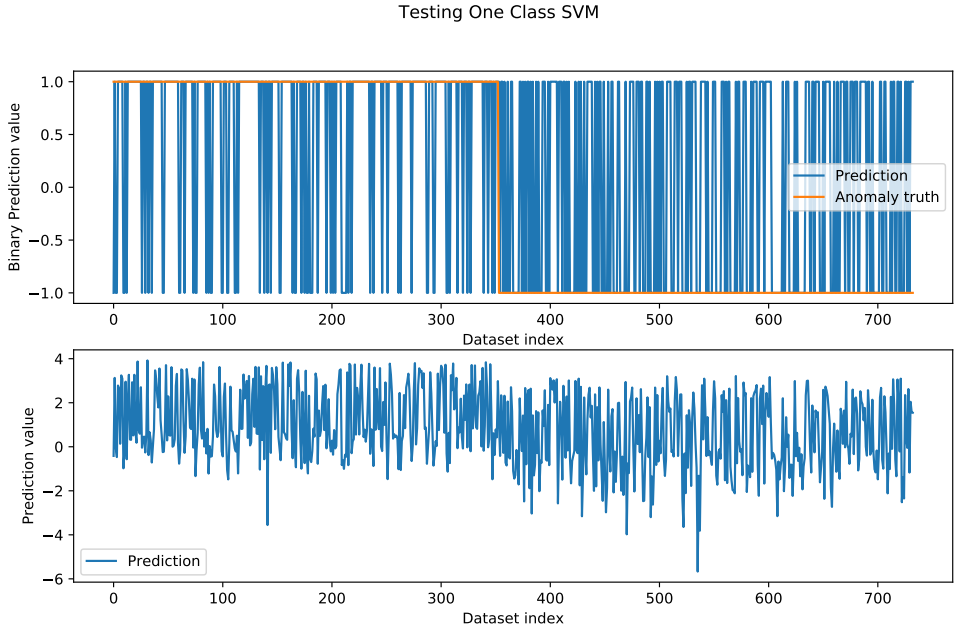


Figure 4.13: One Class SVM predicitons.

Table 4.11: Confusion matrix for OCSVM in live environment

Truth\Prediction	Anomaly	Normal
Anomaly	194	186
Normal	79	274

Scores for One Class SVM:

- Accuracy: 63.8%.
- Recall: 51.1%
- Precision: 71.1%

The tuned parameter used for `scikit-learn`'s One Class SVM is:

- `nu` = 0.18

The rest are set to default values. Half of the dataset is normal behavior and the other half in anomalous behavior. There is small decrease in the predicted value in figure 4.13 when the anomaly label changes.

4.5.2 Isolation Forest

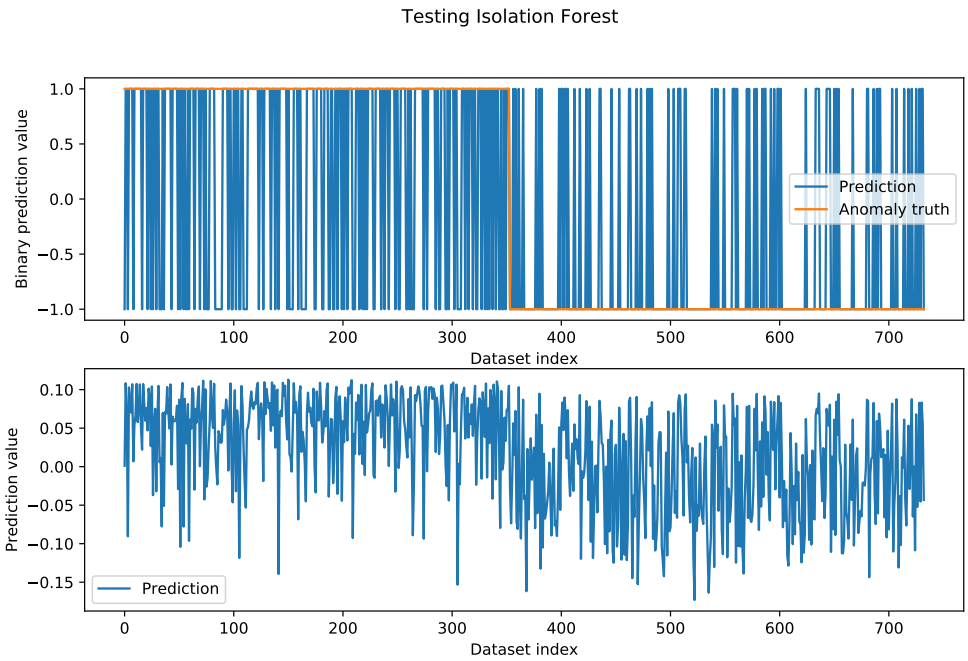


Figure 4.14: Isolation Forest predictions.

Table 4.12: Confusion matrix for Isolation Forest in live environment

Truth\Prediction	Anomaly	Normal
Anomaly	292	88
Normal	129	224

Scores for Isolation Forest:

- Accuracy: 70.4%.
- Recall: 69.4%
- Precision: 76.8%

The parameters used for `scikit-learn`'s Isolation Forest are:

- `contamination` = 0.3

The rest are set to their default values. In figure 4.14 it can be seen that similar to One Class SVM in figure 4.13, it does have a small decrease in anomaly prediction in latter half of the dataset when the true anomaly label changes.

4.6 Supervised learning in live environment

4.6.1 Neural network

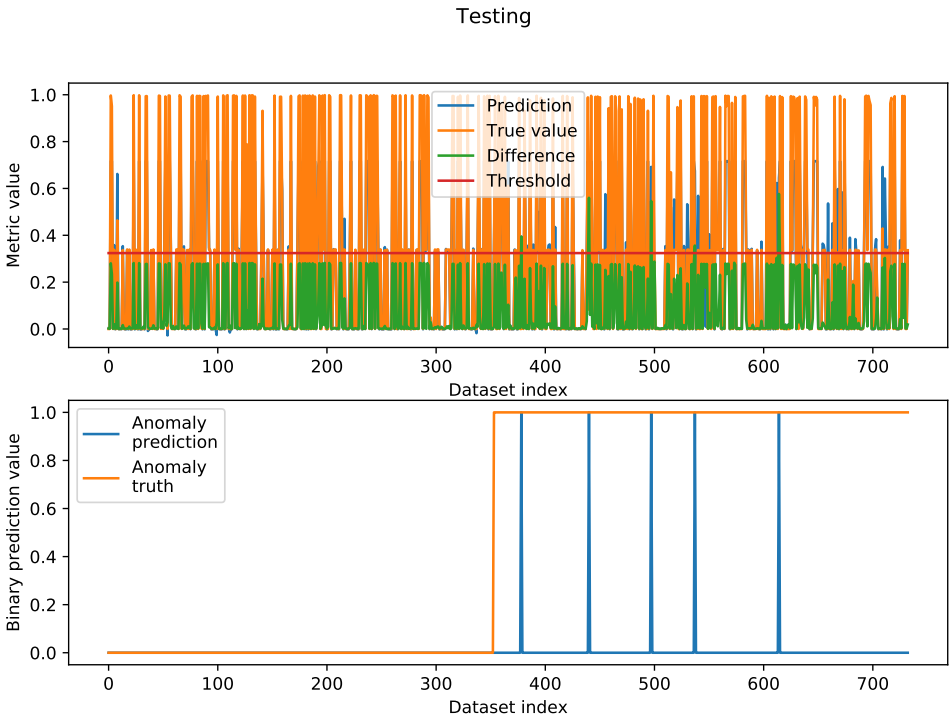


Figure 4.15: Neural network predictions in the live environment data set.

Table 4.13: Confusion matrix for Neural Network in live environment

Truth\Prediction	Anomaly	Normal
Anomaly	15	365
Normal	0	353

Scores for neural network:

- Accuracy: 50.2%.
- Recall: 3.9%
- Precision: 100%

The model was trained with 250 epochs and the prediction threshold a from equation 3.12, was set to 4. In figure 4.15, the model were able to predict 5 peaks where the data set was anomalous and none where the data set was normal.

4.6.2 Auto-encoder

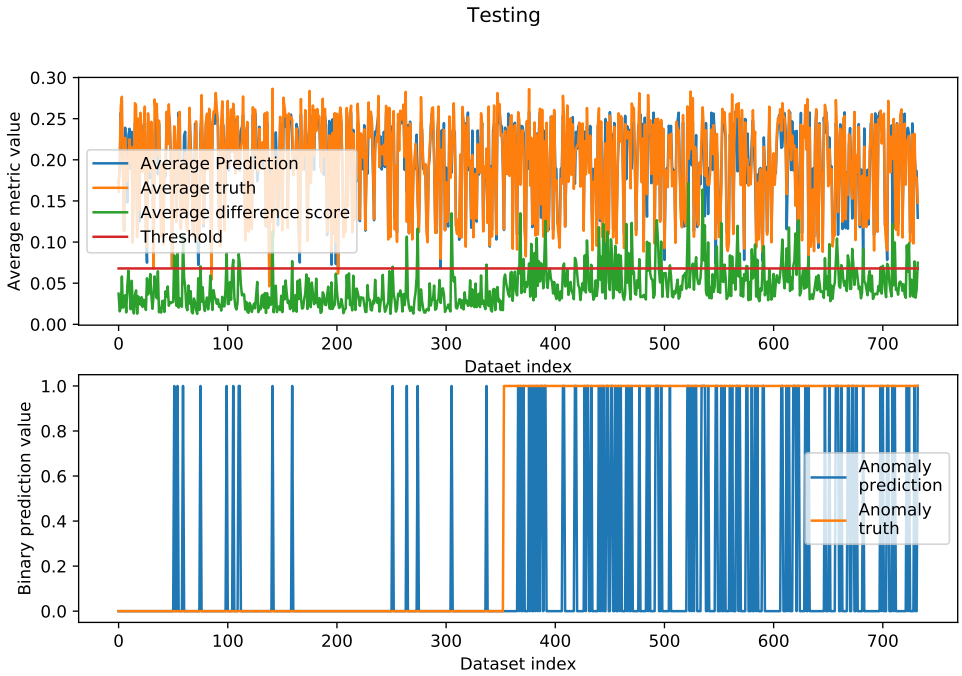


Figure 4.16: Auto-encoder predictions in live environment.

Table 4.14: Confusion matrix for Auto-encoder in live environment

Truth\Prediction	Anomaly	Normal
Anomaly	65	315
Normal	18	335

Scores for auto-encoder:

- Accuracy: 54.6%.
- Recall: 17.1%
- Precision: 78.3%

The size of the hidden layers of the auto-encoder is

$$[65, 25, 20, 15, 10, 5, 10, 15, 20, 25, 65] \tag{4.2}$$

The number of epochs is 1000. The threshold parameter a is 1.5. In figure 4.16 it can be seen that the amount of anomalous predictions in the beginning of the data set are far fewer than the anomalous predictions in the latter half of the dataset. This can also be seen in the average difference between the prediction and the true value.

4.6.3 SVR

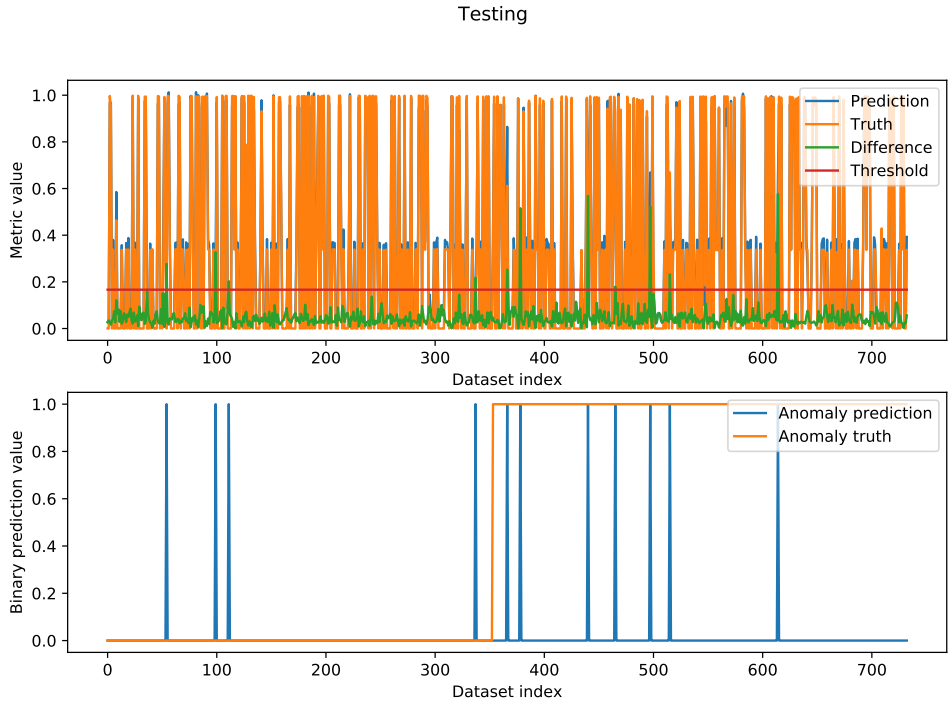


Figure 4.17: SVR predictions in the live environment.

Table 4.15: Confusion matrix for SVR in live environment

Truth\Prediction	Anomaly	Normal
Anomaly	7	373
Normal	4	349

Scores for SVR:

- Accuracy: 48%.
- Recall: 1.8%
- Precision: 63.6%

The threshold parameter a is 3.5. This algorithm predicted 7 correct anomalous points and 4 false anomalous points as shown in table 4.15 and figure 4.17.

5
Analysis

In this chapter, analysis of the previously presented results will be discussed. Each individual data set is discussed in detail with the respective methods and approaches. Some analysis is also presented about the proposed solutions as a whole, choice of metrics and issues with implementation into a real world system. Lastly, several recommended areas of future work is presented.

5.1 Design and performance of evaluation tests

5.1.1 Visualizations

On the pre-deployment set in figure 4.1, the PCA decomposition didn't manage to separate the normal observations from the anomalous observations. The anomalous observations do however spread out more than the normal observations but that is not a very robust assumption since many of them cross over the normal observations and is therefore indistinguishable from them. The t-SNE manifold were able to cluster the anomalous data points together into four distinct small clusters. But these are so close to other clusters and sometimes grouped together with other clusters that it is not good way to differentiate them.

On the live environment set in figure 4.2, the PCA decomposition was not able to distinguish the anomalous points from the normal observations. They lay both on the same path that form an odd shape in figure 4.2. The t-SNE manifold was however more successful to cluster the different data points together that should make them distinguishable. But, the clusters in figure 4.2 are very similar to other clusters of normal observations. Given the selected metrics, there is therefor no obvious way to pick out anomalous point from normal clusters if the setup is unlabeled.

5.1.2 Pre-deployment

Each algorithm tested showed promising results. They were able to predict the location of all the anomalies with confidence. The fact that they didn't have close to $\approx 100\%$ accuracy is less important when considering the setup presented in this thesis. It is still of value that the algorithm predicts anomalies at all close to the actual anomalous behaviour and that it doesn't make false anomaly predictions. In the case with One Class SVM and Isolation Forest, the lower graph in figure 4.3 and 4.4 might be more relevant for predictions. One Class SVM actually did some false

predictions in the beginning and close to the middle of the test set. But in the lower graph of figure 4.3 it can be seen that the value is very close to zero and not as small as the other anomaly predictions later in the data set. Since the decision function for One Class SVM is just the sign of the prediction value, the confidence of which the algorithm has at each data point is lost. The prediction value is therefore a more robust indicator for anomaly prediction in this case. Isolation Forest in figure 4.4 did very well and only did false prediction very close to true anomalies.

The neural network, auto-encoder and SVR also performed well. Even though they had worse accuracy score than both One Class SVM and Isolation Forest, the neural network and the auto-encoder still could predict at the correct place in the dataset where the anomalies occurred. As pointed out above, the difference between the prediction and the true value is more relevant for detecting anomalies since it shows how anomalous the datapoint is. SVR was however worse than the neural network and auto-encoder. It didn't manage to predict two places in the beginning in figure 4.7 of the test set where there were noise added and looking at the difference doesn't show any confidence that it was just barely under the threshold like it was in case of data points at index ≈ 370 to 380.

The reason these algorithms perform well given the low amount of samples in the pre-deployment dataset can be attributed to the lack of diverse training samples. The system was loaded with a very simple test case that wasn't very diverse in its operation.

5.1.3 Sensitivity analysis

In figure 4.8, the threshold is very close to zero and thus there are a lot of anomaly predictions. This is because of how the threshold is calculated in equation 3.12. The value prediction is actually very close to the true value and therefore the average difference is close to zero. This way of calculating the threshold is not ideal since it has this flaw if the is no anomaly present. Something more suitable would be to have a threshold that is the same every time(e.g $\epsilon \approx 0.1$) or a large a as the scaling factor the prediction function in equation 3.12.

When $\sigma = 0.1$ in figure 4.9, there are three missed predictions in the data set. But at other places it did predict pretty well. For $\sigma = 0.3, 0.5$ and 0.8 there is a very similar story but progressively better. These results can be explained by the way noise was added to the data set. This will be explained more in section 5.1.4.

5.1.4 Analysis of added noise in pre-deployment datasets

The way noise was added to the pre-deployment data set in equation 3.14 has some flaws. Since the noise is multiplicative, it will suppress whatever signal it is multiplied with if the folded version of normal distribution has a mean less than 1. The mean of the folded normal distribution in terms of the regular normal distribution

is

$$\mu_{FN} = \sigma_N \sqrt{\frac{2}{\pi}} e^{(-\mu_N/2\sigma_N^2)} + \mu_N \left(1 - 2\Phi\left(\frac{-\mu_N}{\sigma_N}\right)\right) \tag{5.1}$$

Where $\Phi(x)$ is cumulative distribution function for the normal distribution. Equivalently the variance for the folded normal distribution in terms of the regular normal distribution is

$$\sigma_{FN}^2 = \mu_N^2 + \sigma_N^2 - \mu_{FN}^2 \tag{5.2}$$

Since $\mu_N = 0$ for every case, a table can be constructed to show the different statistical properties of the folded normal distribution.

Table 5.1: A table showing the different folded normal properties in terms of standard deviation for the normal distribution.

σ_N	σ_{FN}	μ_{FN}
0.1	0.06	0.08
0.3	0.18	0.24
0.5	0.30	0.40
0.8	0.48	0.64
1	0.60	0.80

As can be seen in table 5.1, the noise added will on average dampen the signal it is multiplied with and with a large precision. $Z \sim \mathcal{FN}(\mu_{FN}, \sigma_{FN})$.

$$\begin{aligned} E[(1+Q(t))Z] =& E[(1+Q(t))]E[Z] && (5.3)\\ =& \mu_{FN}(1+\mu_Q) && (5.4)\\ =& \mu_{Q_{noise}} && (5.5) \end{aligned}$$

$$\begin{aligned} Var[(1+Q(t))Z] =& E\left[((1+Q(t))Z - E[(1+Q(t))Z])^2\right] && (5.6)\\ =& (1+2\mu_Q + E[Q(t)^2])(\sigma_{FN}^2 + \mu_{FN}^2) - \mu_{FN}^2(1+\mu_Q)^2 && (5.7)\\ =& \sigma_{Q_{noise}}^2 && (5.8) \end{aligned}$$

The complete derivation can be found in appendix A.1. Since both μ_{FN} and σ_{FN} are less than one, equation 5.4 and 5.7 will have an dampening effect on the noisy signal for all the values shown in table 5.1. But when $\sigma_N = 1$ the noisy signal should be unaffected by the dampening and instead make the signal noisy as intended.

5.1.5 Live environment

In this use case, it was much harder to differentiate the defined normal behavior versus the anomalous behavior. In figure 4.13, the class prediction from One Class

SVM is noisy. However, the prediction value below show a small decrease in the latter half of the dataset where the anomalous code were running. Isolation Forest in figure 4.14 follow a similar pattern where the prediction decreases slightly in the latter half of the testing set. This observation is easily missed and not at all robust for detection anomalies.

The neural network however, showed some promise in figure 4.15. Even though it predicted very few anomalous points, it didn't predict them in the wrong part of the data set. This is still not very good but at least it showed some confidence of where the anomaly was present in the data set. The auto-encoder and SVR didn't do nearly as good. The auto-encoder in figure 4.16 predicted a lot of false alarms while the SVR in figure 4.17 didn't predict much at all. If the true label on this data set was not known, the prediction in both cases wouldn't be enough to tell if there was an anomaly present or not.

The overall poor performance can be attributed to the few data points for training and not enough differentiation in the data. The injected "bugs" only affect few of the metrics used and was therefore not enough for these algorithm to make a clear differentiation between them. This could certainly be solved with more data.

5.2 Model design and choice of metrics

The test results indicate that performing anomaly detection like the model systems proposes, on individual microservice instances seems viable to solve the task of detecting anomalous behaviour in general. The tests do not directly show that anomalies related to software bugs can be found, while other classes of anomalies are ignored. The specific class of anomalies will instead become recognizable in situations where several instances of a microservice is analyzed, where a subset is running a later version. Most likely, other choices of metrics would not change these facts. Staying with the choice of a subset of key performance metrics is therefore viable to maintain a lower complexity.

A specific class of what could be classed as anomalies are performance improvements as well as large changes in general to the microservice. In the case for improvement, it is advantageous that they are detected short term and can be analyzed. In the long term, the system would need to be retrained on a regular basis as the microservice changes in how it operates over many updates. The viability of the suggested solution models should increase as a function of how patches change behaviour in terms of the selected performance metrics, over time.

5.3 Implementation in a real-world scenario

The threshold function(equation 3.12) for supervised algorithms is not ideal in an implemented system. In an implemented system, predictions should be done continuously and thus the averaging of the predicted error is not an ideal solution. A con-

stant threshold would instead be more suitable. The threshold scaling parameter(a) also has to be larger since in an implemented system the anomalies would be rarer than in the dataset tested in this thesis.

A fundamental challenge for a live solution system is that it may need to be trained locally in order to be deployed to different customers. One way to avoid the problem might be to get information about the traffic model from the customer and run training using simulations before deployment to customer. Capturing long-term cyclical behaviour and training the network for these may not be possible either cause of the regular need for retraining, and this also fortifies the need to perform simulations before customer deployment, where all types of cyclical behaviours may be simulated.

5.3.1 Ethical evaluation of proposed solutions

The solutions proposed, calls for the use of performance metrics for a microservice instance. This type of data does not connect to users, but rather how the software operates. Furthermore, if an implemented system is only available to the same personnel, which is already handling other metrics from the DB-server, then our solution is in line with what is implemented.

The suggested solutions would impact workforce as well as developers who work in areas related to investigating anomalies, if implemented in a real-world scenario. It is uncertain if it could lead to a need for less employees, but the way the solution simplifies looking for anomalies will have positive impact since less focus will have to be put on a lot of raw metrics. The workforce or developers could instead be reactive as to when the system reacts.

5.4 Future work

5.4.1 Improved system load testing

As previously mentioned in section 5.1.2 and 5.1.5, the load testing of the system was not very sophisticated. It basically called the micro-service a specified rate per second and then undid those operations a few moments later. It would be more desired to have the rate follow a traffic model that would cover more load cases and be more realistic.

5.4.2 More data

As mentioned in section 5.1.2 and 5.1.5, more training data is desired to improve the performance of the proposed model. Especially in the case of the neural network based algorithms. It also follows naturally from an improved load test from section 5.4.1, that the amount of data is required to be much larger to cover the whole distribution of loads for training.

5.4.3 Evaluate more algorithms

As mentioned in theory, there is a high chance of achieving better performance by implemented different types of Deep Learning techniques. There are numerous different algorithms for detecting anomalies that are not covered in this thesis but could be valuable for further testing and evaluation. two methods that were only briefly researched, but fell outside the scope were the following:

- **Bayesian Ridge Regression:**
  This algorithm, presented in [31] and [32], uses the posterior distribution of known data points to make new predictions. This algorithm would be applied in the same way as the supervised algorithms used in this thesis, namely to predict one known metric and if the prediction deviates to much from the truth, a probable anomaly is detected.
- **AnoGAN:**
  A version of a Generative Adversarial Network(GAN) presented in [33], where an unsupervised algorithm is trained to create a manifold of normal images of optical coherence tomography. Not only does it correctly detect anomalous images but it is also able to highlight where in the image the anomaly is. Another adaptation of GAN for anomaly detection, presented in [34], also show promise for this task. Further research has to be done in order to make this type of algorithm fit with the problem presented in this thesis.
- **Robust Random Cut Forest(RRCF):**
  An extension to Random Cut Forest algorithms such as Isolation Forest[21], presented in [35], show good results and robustness when it comes to implementing anomaly detection in an continuous training setting. The difference between RRCF and other random cut forest algorithms is how it does the dimensional cuts and how it updates the "forest" for new incoming data. RRCF is more computationally efficient than Isolation Forest and would be applied in the same way.

6
Conclusions

By identifying and modeling the cloud application infrastructure, a conclusion can be made that a system built for anomaly detection for a microservice should be performed on microservice instance level. The group of ML systems watching each microservice within the application should be designed as its own microservice. The choice of metrics should be limited to key performance indicators in order to better isolate what type of faults are desired to be detected.

The results from tests run using prototype systems with injected anomalies show that the proposed type of solution system could be viable, given that anomalies in a real-world scenario behaves 'differently enough' from normal behaviour. By further investigating modern techniques for deep learning, more advanced classifiers can be designed to outperform the tested classifiers.

In terms of problems, a few conclusions can also be drawn. One is that the problem of isolating one class of anomalies related to software updates, and not detecting other types as well, may be difficult. Another class of problems lies in retraining the classifier. The microservice will change behaviour over time as a lot of updates are done, and will regularly need retraining. The microservice may also run differently at different customers, leading to the need for customized training for each one.

Bibliography

[1] "FAQ - Cloud Native Computing Foundation," 2017. [Online]. Available: https://www.cncf.io/about/faq/

[2] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[3] Scikit-learn, "Examples - Scikit-learn documentation," 2017. [Online]. Available: http://scikit-learn.org/stable/auto_examples/index.html

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1541880.1541882

[5] Ericsson, "About us - Ericsson Corporate Information," 2018. [Online]. Available: https://www.ericsson.com/en/about-us

[6] M. R. Smith and T. Martinez, "Improving Classification Accuracy by Identifying and Removing Instances that Should Be Misclassified." [Online]. Available: http://axon.cs.byu.edu/papers/smith.ijcnn2011.pdf

[7] "Welcome to Python.org." [Online]. Available: https://www.python.org/

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[11] "1.4. Support Vector Machines — scikit-learn 0.19.1 documentation." [Online]. Available: http://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

[12] Chih-Wei Hsu, Chih-Chung Chang and C.-J. Lin, "A Practical Guide to Support Vector Classification," *BJU international*, vol. 101, no. 1, pp. 1396–400, 2008. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[13] a. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, 2004. [Online]. Available: http://www.scopus.com/scopus/inward/record.url?eid=2-s2.0-4043137356&partnerID=40&rel=R8.0.0

[14] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *The Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541, 2009. [Online]. Available: http://dx.doi.org/10.1002/cne.21974

[15] "Loss Functions and Optimization Algorithms. Demystified." [Online]. Available: https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c

[16] M. Chui, J. Manyika, M. Miremadi, N. Henke, R. Chung, P. Nel, and S. Malhotra, "Notes from the AI frontier. Insights from hundreds of use cases," Tech. Rep., 2018. [Online]. Available: https://www.mckinsey.com/mgi/

[17] A. Ng, "CS294A Lecture Notes Sparse Autoencoder," *Cs294a*, pp. 1–19, 2011. [Online]. Available: https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdfhttp://www.stanford.edu/class/cs294a/

[18] Scikit-learn, "Novely and Outlier Detection." [Online]. Available: http://scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection

[19] B. Schölkopf and R. Williamson, "Support Vector Method for Novelty Detection." *Nips*, pp. 582–588, 1999. [Online]. Available: http://www.cms.livjm.ac.uk/library/archive/GridComputing/NoveltyDetection/sch00support.pdf

[20] "2.7. Novelty and Outlier Detection — scikit-learn 0.19.1 documentation." [Online]. Available: http://scikit-learn.org/stable/modules/outlier_detection.html#outlier-detection

[21] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-Based Anomaly Detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, pp. 1–39, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2133360.2133363

[22] L. Van Der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

[23] D. Ulyanov, "Multicore-tsne," https://github.com/DmitryUlyanov/Multicore-TSNE, 2016.

[24] A. Ghodsi, "Dimensionality Reduction A Short Tutorial," 2006. [Online]. Available: https://www.math.uwaterloo.ca/~aghodsib/courses/f06stat890/readings/tutorial_stat890.pdf

[25] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015. [Online]. Available: https://arxiv.org/pdf/1502.03167.pdfhttp://arxiv.org/abs/1502.03167

[26] S. Aksoy and R. M. Haralick, "Feature Normalization and Likelihood-based Similarity Measures for Image Retrieval," no. October 2000. [Online].

Available:
http://www.cs.bilkent.edu.tr/~saksoy/papers/prletters01_likelihood.pdf
[27] J. Brownlee, "What is the Difference Between Test and Validation Datasets?" [Online]. Available:
https://machinelearningmastery.com/difference-test-validation-datasets/
[28] H. Hamilton, "Confusion Matrix - CS 831, University of Regina," 2012. [Online]. Available: http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html
[29] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700 LECTURE NO, pp. 437–478, 2012.
[30] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," pp. 1–15, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980
[31] R. Salakhutdinov, "STA 4273H: Statistical Machine Learning Lecture 1." [Online]. Available:
http://www.utstat.toronto.edu/~rsalakhu/sta4273/notes/Lecture1.pdf
[32] ——, "STA 4273H: Statistical Machine Learning Lecture 2." [Online]. Available:
http://www.utstat.utoronto.ca/~rsalakhu/sta4273/notes/Lecture2.pdf
[33] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10265 LNCS, pp. 146–147, 2017.
[34] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-Based Anomaly Detection," pp. 1–7, 2018. [Online]. Available: http://arxiv.org/abs/1802.06222
[35] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust Random Cut Forest Based Anomaly Detection On Streams," *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, 2016. [Online]. Available: http://jmlr.org/proceedings/papers/v48/guha16.pdf

A

Appendix 1

A.1 Variance of noisy data-set derivation

$$
\begin{aligned}
Var[(1+Q(t))Z] =& E\left[((1+Q(t))Z - E[(1+Q(t))Z])^2\right] && \text{(A.1)}\\
=& E\left[(1+Q(t))^2\right] E[Z^2] - && \text{(A.2)}\\
& 2E\left[\mu_{FN}(1+E[Q(t)])(1+Q(t))Z\right] + && \text{(A.3)}\\
& \left(\mu_{FN}(1+E[Q(t)])\right)^2 && \text{(A.4)}\\
=& (1+2E[Q(t)]+E[Q(t)^2])(\sigma_{FN}^2+\mu_{FN}^2) - && \text{(A.5)}\\
& 2\mu_{FN}E[Z]E[1+Q(t)](1+E[Q(t)]) + && \text{(A.6)}\\
& \left(\mu_{FN}(1+E[Q(t)])\right)^2 && \text{(A.7)}\\
=& (1+2\mu_Q+E[Q(t)^2])(\sigma_{FN}^2+\mu_{FN}^2) - 2\mu_{FN}^2(1+\mu_Q)^2 && \text{(A.8)}\\
& + \mu_{FN}^2(1+\mu_Q)^2 && \text{(A.9)}\\
=& (1+2\mu_Q+E[Q(t)^2])(\sigma_{FN}^2+\mu_{FN}^2) - \mu_{FN}^2(1+\mu_Q)^2 && \text{(A.10)}\\
=& \sigma_{Q_{noise}}^2 && \text{(A.11)}
\end{aligned}
$$

The 2nd order moment of the folded normal distribution in equation A.2 and A.5 is equal to the 2nd order moment of the regular normal distribution. If $Y \sim \mathcal{N}(\mu, \sigma)$, then

$$E[|Y|^2] = E\left[(\sqrt{Y^2})^2\right] = E[Y^2]$$