UPPSALA
UNIVERSITET

UPTEC F 19040
Examensarbete 30 hp
Juni 2019

Machine learning to detect anomalies in datacenter

Filip Lindh

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Machine learning to detect anomalies in datacenter

Filip Lindh

This thesis investigates the possibility of using anomaly detection on performance data of virtual servers in a datacenter to detect malfunctioning servers. Using anomaly detection can potentially reduce the time a server is malfunctioning, as the server can be detected and checked before the error has a significant impact. Several approaches and methods were applied and evaluated on one virtual server: the K-nearest neighbor algorithm, the support-vector machine, the K-means clustering algorithm, self-organizing maps, CPU-memory usage ratio using a Gaussian model, and time series analysis using neural network and linear regression. The evaluation and comparison of the methods were mainly based on reported errors during the time period they were tested. The better the detected anomalies matched the reported errors the higher score they received. It turned out that anomalies in performance data could be linked to real errors in the server to some extent. This enables the possibility of using anomaly detection on performance data as a way to detect malfunctioning servers. The most simple method, looking at the ratio between memory usage and CPU, was the most successful one, detecting most errors. However the anomalies were often detected just after the error had been reported. Support vector machine were more successful at detecting anomalies before they were reported. The proportion of anomalies played a big role however and K-nearest neighbor received higher score when having a higher proportion of anomalies.

Handledare: Mikael Björn
Ämnesgranskare: Koen Tiels
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F** ***

Populärvetenskaplig sammanfattning

Stor del av många företags produktion är idag digitaliserad och således är behovet av datorkraft stort. Större företag kan använda serverhallar, ofta med så kallade virtuella servrar, som ansvarar för olika uppgifter. Ibland uppstår fel hos servrarna som kan resultera i att dom inte fungerar optimalt. Dessa fel kan förbli oupptäckta under en tid. Då antalet virtuella servrar kan vara mycket högt blir mänsklig övervakning av alla dessa både tidsineffektift och dyrt. Avvikelsedetektering av performance data är ett kostandseffektivt sätt att ständigt övervaka servrarna. Avvikelsedetektering analyserar performance datan och flaggar avvikande beteende utan mänsklig närvaro. Detta kräver dock att felen på servrarna har en inverkan på performance datan som kan detekteras. Detta examensarbete har undersökt möjligheten att genom avvikelsedetektering spåra fel hos en virtuell server i en av Sandvik Coromants två datorhallar i Gimo.

Det visade sig att avvikelser i performance data kunde kopplas till fel som hade rapporterats in hos serven. Vetskapen om detta samband skulle kunna innebära framtida möjligheter för avvikelsedetektering som ett sätt att detektera servrar som inte fungerar normalt. Att dessa fel kunde detekteras kan innebära att även andra typer av fel, som inte rapporteras in i eventloggen, skulle kunna detekteras på samma sätt. Detta skulle leda till att felen kan bli åtgärdade snabbare.

Acknowledgements

I would like to express my appreciation to my supervisor, Mikael Björn at Sandvik Coromant, for his guidance and the possibility to spend one semester working at Sandvik Coromant. It has been a great pleasure! I would also like to thank my subject reviewer Koen Tiels at Uppsala University for the very passionate participation. His careful inputs and feedback have been highly valuable for me and this thesis.

Contents

1 Introduction

Anomaly detection refers to the task of tracking anomalies in data. In other words to detect abnormal observations deviating from the majority of the data. Anomaly detection is widely applied in all kinds of areas, as deviating data very often indicates something interesting, regardless of the context. For companies, it can be a way to quickly detect a potential error in a system or a machine. Especially in cases where big parts of the activity/production is automated. Anomaly detection can for example be applied on performance data from servers in big datacenters. This example is worked out in this thesis.

1.1 Background

As companies get digital the need for computational power and storage increases. Accordingly datacenters are often essential for big companies. Many datacenters use virtualization, meaning the operating system is separated from the underlying hardware, through a layer called hypervisor. In a datacenter the number of so-called virtual servers may be very high. Superintending every server at all times is time consuming and inefficient. Nevertheless it is often of the company's interest to detect a malfunctioning server quickly, to fix it and thus minimize the negative impact it might have.

Sandvik Coromant is a world-leading company in the area of tools and tooling systems for industrial metal cutting, advanced steels and alloys, and know-how for the metal cutting industry. The company has two factories placed in Gimo, producing drilling tools and inserts. Adjacent to these two factories there are two datacenters. The two datacenters are virtual with lots of virtual servers running on them. These virtual servers are managing different tasks, for example ordering systems, databases etc. Accordingly if one of these virtual servers would start to perform poorly that might have a negative impact on the task it is performing. As it is today a misbehaving server needs to be noticed by the staff before being investigated, meaning it might take some time before the malfunctioning server is noticed and fixed. One possibility for Sandvik Coromant to quickly detect malfunctioning servers is to use anomaly detection on performance data.

1.2 Scope

The primary objective of this thesis intends to investigate the possibility of using anomaly detection on performance data to detect when a virtual server is malfunctioning. More specifically, the task is to build models that analyzes performance data and flag abnormal data. The aim is to look into different methods and approaches to anomaly detection and compare these. To evaluate the capability of the methods, the results will be compared to the event log of the server. An event log displays historical events like errors along with the time they were reported.

In this thesis the main focus will lie on anomaly detection. How the datacenter work, what is affecting the performance data, and what are the underlying causes of the errors will not be addressed.

2 Background on anomaly detection

This chapter provides basic theory about anomaly detection. This includes a comprehension of general problems one needs to consider, general concepts like different types of anomalies and an overview of approaches and methods to apply anomaly detection.

2.1 Anomaly detection in general

When collecting and analyzing data some observations might stand out from the majority. These are called anomalies or outliers. Anomaly detection refers to the wide field of detecting these outliers, meaning finding observations that are inconsistent with the rest of the dataset.

Anomaly detection can be applied in a lot of different situations. It can be fraud detection for credit cards, intrusion detection, recognizing spam email, medical problems, or finding abnormal performance data in a datacenter. What makes anomaly detection so widely applicable is the fact that abnormal data often are a sign of something, regardless situation. An anomaly in an MRI picture may indicate a tumour, anomalies in data collected from a sensor may indicate a fault in the equipment or occurrence of a notable event, anomalous behaviour of a bank card may indicate a fraud etc [1].

Anomaly detection has been known and applied since the early 1900s in statistics but has grown with increased computer power and the development of machine learning. Also the demand for anomaly detection has increased as more areas get automated and digitized.

There are different approaches and methods to apply anomaly detection, depending on context and type of data. The data can be univariate or multivariate, binary, discrete or continuous, labeled or unlabeled. Is it time series analysis, one has one (or more) sequence in time, consisting of data samples with a certain frequency, where the anomaly detection intends to search for anomalies in that time series context [1]. Anomaly detection methods can also investigate how data of different parameters relate to each other. This could for example be investigating the ratio between CPU and memory usage.

When applying anomaly detection there are some general problems often having to be dealt with:

- Deciding where to draw the line between normal and abnormal data. This might not be obvious since the boundary is often blurred. Being too generous with classifying data as normal might result in a high false negative rate, meaning missing anomalies. The opposite case on the other hand will result in high false alarm rate, with normal data being classified as anomalies.
- The boundary between normal and abnormal data might change over time. Thus the boundaries at a certain time might not be optimal later on.
- It can be hard to distinguish between anomalies and noise. Data often contain noise, which may have similar characterization as anomalies but are not relevant to detect.

2.2 Types of anomalies

To get a better understanding of anomaly detection it is important to know what types of anomalies there are. Before applying anomaly detection one must know and decide what types to look for. Listed below are the three main kinds of anomalies [1].

- Point anomalies. When an individual point differs from the rest of the dataset, it is considered a point anomaly. This is the most simple type of anomaly since it does not take context into account. However it is widely used and is the type of anomaly that has been most researched [1]. An illustrative example is shown in figure 1.
- Contextual anomalies. A contextual anomaly is, like the name suggests, an anomaly with respect to the context. This means it is not an anomaly in itself (point anomaly), but with the current context taken into account it is. In time series data for example, time is the context (see figure 2). If the temperature in Sweden was to be analyzed, 25 degrees might be perfectly normal or abnormal depending on the context, meaning the time of the year.
- Collective anomalies. A collective anomaly is a type only considered as an anomaly when occurring as a group. The individual points are not anomalies but forming a group together is considered as an anomaly. This is illustrated in figure 3.

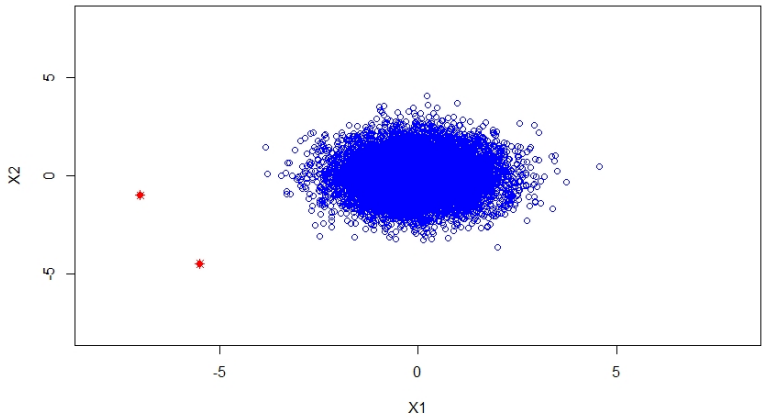

Figure 1: An example of point anomalies. Two variables, X1 and X2 plotted against each other, forming a cluster. The two red points outside the cluster are point anomalies.

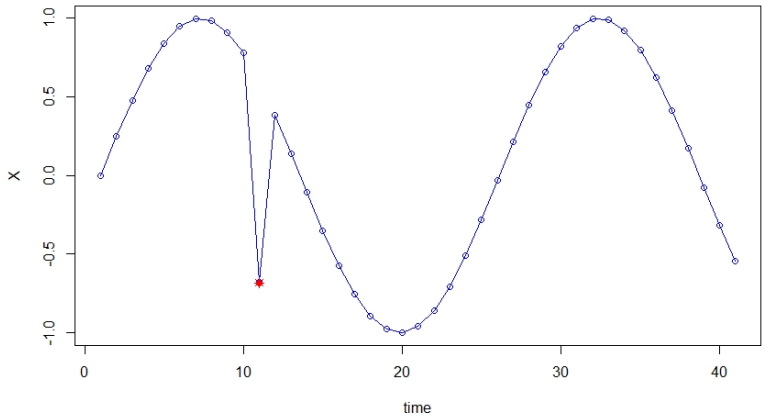

Figure 2: An example of a contextual anomaly. The red point does not have an abnormal x-value. However with the context (time) taken into account, it is considered an anomaly.

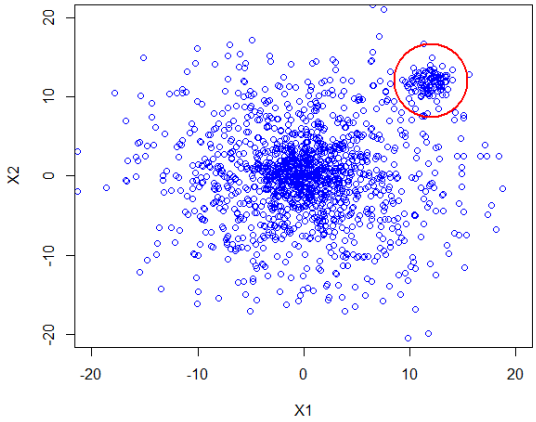

Figure 3: An example of a collective anomaly. The cluster of points marked by the red circle is a collective anomaly, as the points are dense in a low dense region.

2.3 Types of learning

Except for different types of anomalies there are also different ways in which methods learn to detect them, i.e. different kinds of learning. The two main ones are *supervised learning* and *unsupervised learning*. These are general for machine- and statistical learning and are not limited only to anomaly detection.

Before explaining these different types of learning in further detail below, a simple example will be given to get a sense of the difference. Imagine having a set of photos presenting two categories, cars and humans. If there is *prior knowledge* about the photos, meaning every picture is labeled 'car' or 'human', supervised learning can be applied. In this case a supervised algorithm uses the labeled photos to learn to distinguish humans from cars. When trained properly it can categorize new photos in the future as either 'car' or 'human'. If there is no prior knowledge available, meaning only a set of photos with no labels, unsupervised learning can be applied. In this case the algorithm will go through all pictures trying to sort them out into two categories, based on their properties.

2.3.1 Supervised learning

In supervised learning the dataset is labeled, meaning every data point is marked with the class it belongs to. Methods using supervised learning utilize the labeled training data to learn to categorize data properly. In general it can be expressed as for every element in the observation vector, x (input data) there is an associated response measurement y, revealing the class. Applying supervised learning in anomaly detection requires labeled instances from both normal and anomaly classes. When the algorithm is fully trained it can be used to categorize new data, based on their properties [2][3, section 2.1.4].

The response vector, y in the training dataset can also have a numerical value. This is called regression and means that the algorithm intends to learn to determine the numerical output value as accurate as possible. These two types of supervised learning, regression and classification, are explained further in section 2.4.

Supervised learning can be very effective when applied, but requires labeled data with a good distribution over both the normal and abnormal regions. Labeled data in anomaly detection is usually not available and labeling is often very time consuming or even impossible. Also the ratio between normal and abnormal data is often very uneven. In some cases however this can be solved by weighting the classes in the training data [2].

The term *Supervised anomaly detection* is analogous to supervised learning but is sometimes used in the anomaly detection field, to clarify that the problem consists of normal and anomaly classes [1].

2.3.2 Unsupervised learning

The opposite case to supervised learning is unsupervised learning. The vector of observations x is available but the associated response vector, y is not, thus the data is unlabeled. In anomaly detection this is analogous to *unsupervised anomaly detection* or *unsupervised clustering* and means that no prior knowledge about the data is available. This makes them widely applicable. The algorithms must learn to detect outliers without having a labeled training set available meaning the problem of defining what should be considered an anomaly is completely up to the method to define and solve. Many methods using unsupervised techniques do not need training data. Instead a complete dataset is used and analyzed at once, as in the earlier example with the photos [2][3, section 2.1.4].

Unsupervised clustering assumes that classification can be based on distance between points. Normal data points are clustered together in one or many groups, i.e. regions with high probability density. Outliers on the other hand are points far away from these regions. Unsupervised clustering assumes that anomalies are unusual. In anomaly detection unsupervised learning is the most common approach as labeled data are usually scarce or non existent [2].

Except for supervised and unsupervised anomaly detection, there is a third approach called *semisupervised anomaly detection*. It does not use labeled data and is thus similar to unsupervised anomaly detection. It assumes that all the data points in the training set are normal. Thus instead of having a training set consisting of observation points of different labels, all training data are considered normal, or at least the vast majority. New data points that deviate from the training data are flagged anomalous. In contrast to unsupervised anomaly detection, semisupervised anomaly detection requires that training and test data is split up [1]. Note that semisupervised anomaly detection is not analogous to semisupervised learning, where the training set consists of both labeled and unlabeled data.

2.4 Regression versus classification problems

In supervised learning there are two types of problems, regression and classification problems. What distinguishes them is the output variables they are trained to predict. These variables can be quantitative or qualitative. Quantitative variables have numerical values, for example age, income, salary. Qualitative variables could be gender, employed or unemployed etc. Problems having a quantitative output are called regression problems, while qualitative ones are called classification problems. Anomaly detection often falls under the classification category since the output consists of two classes, 'normal' or 'anomaly'. Classification problems like this, only having two classes are called two-class or binary problems. Many methods however can be adjusted to apply both for regression and classification problems [3, section 2.1.5].

2.5 Principal component analysis

Some problems consist of very large numbers of dimensions/variables. Principal component analysis, PCA is a dimension reduction technique. If having a multivariate dataset of variables possibly correlated to each other, execution of PCA will convert them into a set of uncorrelated variables, called principle components. The first principle component is placed along a line of highest variance in the dataset. In this way it catches as much of the variance as possible and thus as much information of the dataset as possible in one dimension. Accordingly the first principle component is the first eigenvector of the covariance matrix. The second principle component will be orthogonal to the first one, the third one orthogonal to the first two etc. PCA is used frequently in exploratory data analysis as it can reveal inner structures and variance of the data. If the problem is suitable for PCA a small number of principal components can gather information from very many dimensions. In this way a lower number of principal components can replace a higher number of original features. The big drawback with PCA however is that it is linear and the variables should be linearly correlated to each other to get good results [2][4].

2.6 Approaches to anomaly detection

Most anomaly detection methods can be categorized into statistical-based, nearest neighbor-based, clustering-based and classification-based [1]. These four categories/approaches will be addressed in this section. Each approach mentioned above is broad, including multiple different methods.

A couple of methods from every approach are briefly introduced here. Some of them were not optimal for this project which is discussed in the discussion, section 6.2. The ones used in this thesis however are introduced here but explained in further detail in the theory (section 3).

2.6.1 Statistical anomaly detection

Statistical anomaly detection is based on the assumption that normal instances lie in regions of high probability while anomalies lie outside of these, in low probability regions. The basic principle is to collect data and fit them to a model that predicts the probability of the regions. New data points can then be classified either normal or abnormal depending on the probability of their region. There are two main different ways to fit a model to data: parametric and nonparametric. Parametric methods assume a certain distribution of the data, while nonparametric methods do not [1].

Parametric methods are more accurate when the distribution of the data is known. After discovering which parametric distribution the data are sampled from, a probability density function (PDF) can be set up. Having a PDF function all that remains is to optimize the parameters describing the generation of data, to make the fit as accurate as possible. This often makes them easier to write down and faster to compute. However if the assumptions regarding the distribution and parameters are not correct, parametric methods are more likely to fail [1]. There are many types of distributions with different PDFs. One of the most common, simple and well-known is the Gaussian distribution. The PDF in one dimension is defined as:

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

Where x is the investigated variable, μ is the mean value and σ is the standard deviation. This can be generalized to many dimensions. The PDF is then given by the multivariate Gaussian distribution defined as:

$$p(\mathbf{x}) = \frac{exp(-0.5(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma} (\mathbf{x}-\boldsymbol{\mu})}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \tag{2}$$

where Σ is the covariance matrix which needs to be positive definite. Finding point anomalies using this approach is simple. A minimum probability is set, for example points more than 3σ away from the mean value, and all points falling outside of the interval are classified as anomalies. The technique is accurate when having the variables generated from a Gaussian distribution but not otherwise [1].

Nonparametric methods do not assume a certain distribution of the data as parametric methods do. Therefore they are not as accurate but they are preferable when the data distribution is unknown. The fact that they make no assumptions of the distribution makes them robust and able to handle data that might deviate from the original distribution, meaning a distribution where the parameters are not fixed over time [1][3]. The most basic non-parametric statistical techniques are histogram-based, also referred as frequency-based. In the univariate case this simply consists of two steps. At first a histogram of the training data is built. Second step is to detect anomalies based on the histogram, meaning data points falling outside of the existing bins created from the training data. The anomaly classification can also be based on the height of the bins, meaning data falling into low frequent bins are flagged as anomalies. One tricky part is to choose the right width of the bins. The narrower they are, the more points will fall outside or in low frequent bins. Too wide on the other hand might result in missing anomalies. Again this is a balance between low false alarm rate and low false negative rate. Similar approaches can be applied in two dimensions, using a 2D histogram. It can be likened to the density of the two variables [1].

2.6.2 Nearest neighbor-based anomaly detection

Nearest neighbor methods assume that normal data are gathered close together in high dense regions, while anomalies are further away. By calculating the distance from each point to its neighbors the idea is to get a measure on whether it is close or far away from the groups, i.e. the normal data. Having continuous problems, the Euclidean distance is often used. Nearest neighbor is a nonparametric approach which can be applied in mainly two ways: K nearest neighbors (KNN) which uses the distance to its K nearest neighbors, and density methods which instead use the

density. KNN is more straightforward than density methods, simply calculating the anomaly score based on the summed distances to every data point's K closest neighbors [1][2].

Density based methods estimate the density at every point's position. The density is calculated using the distance to the neighbors and is therefore similar to KNN. Density can be viewed as the inverse of the distance to the nearest neighbors. Varying densities in a dataset makes a straightforward density method perform weakly, as illustrated in figure 4. However this can be solved by calculating the relative density. In this case the density at a point is compared to the densities of its neighbors and thus local deviations can be detected. Local outlier factor, LOF is a local density method. The anomaly score is then calculated as a ratio between the density of the point and its neighbors [1].

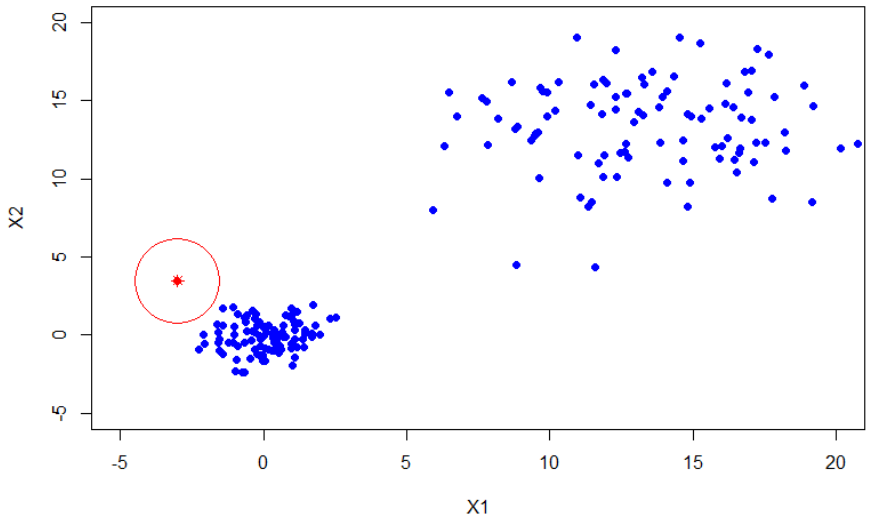

Figure 4: An example of local outlier factor (LOF). The blue points are normal data, forming two clusters. The red point marked with a circle is an anomaly according to LOF. For a global density method or KNN this red point could have been classified normal, as all points are treated equally and the variance in the right cluster is much higher.

2.6.3 Clustering-based anomaly detection

Clustering methods arrange data points into clusters according to certain courses of action, forming groups of data points with similar characteristics. There are two main approaches to apply clustering for anomaly detection, relying on two different assumptions. The first one assumes that normal data is grouped together in clusters, while anomalies fall outside of these. The second approach proceeds from the assumption that anomalous points lie far away from the center of the clusters they belong to, their so called centeroids.

Using the first assumption, the principles and definitions of forming the clusters depend on the method. Density-based spatial clustering of applications with noise (DBSCAN), is an example of such a method, forming clusters based on amount of neighbors and distances. In DBSCAN all the data points falls into four categories: core points, directly reachable points, reachable points and outliers. These are based on two variables. Epsilon, ϵ which specifies the radius of neighborhood around every point and $minPts$ which determines the minimum number of points in an area to be counted as a cluster [5]. The clustering principle of DBSCAN is shown and described in figure 5.

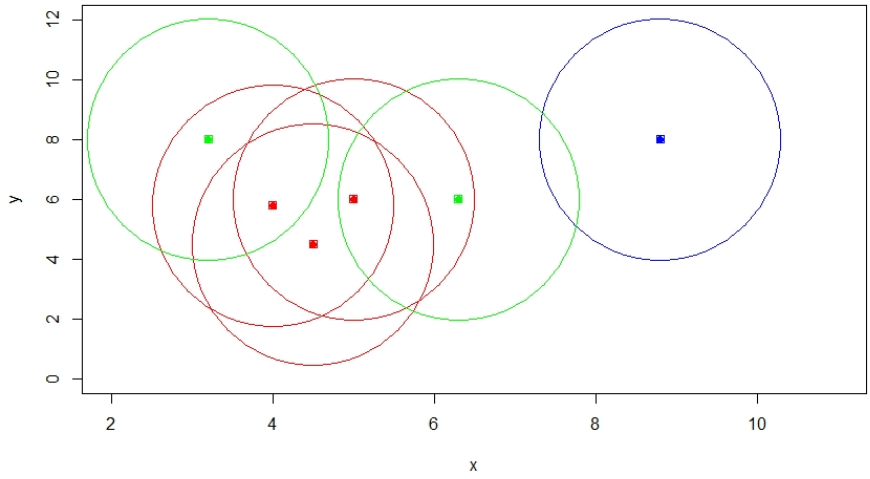

Figure 5: Example of clustering using DBSCAN in 2D, with $minPts = 3$ and $\epsilon = 1.5$. The red points are *core points*, meaning they have at least minPts number of neighbors lying within a radius of ϵ. The green points are *directly reachable points* from the red cluster, meaning they have a core point placed within a radius of ϵ. They are also said to be *reachable points* to each other (as they are connetcted through the core points). The blue point is an *outlier point*, as it does not fulfill any of the criteria above.

The other approach to use clustering for anomaly detection proceeds from the assumption that anomalous points lie far away from the center of the clusters they belong to, their so called centroids. The idea is to form clusters, calculate the anomaly score depending on the distance from every point to its centroid and flag the points far away as anomalies. K-means clustering is a well-known machine learning method which is mainly used for clustering but can be extended to apply anomaly detection as well. Self-organizing map (SOM), is a method for clustering as well as dimensionality reduction. It gathers high-dimensional data into a 2D map, where groups can be revealed [1][6]. After the clustering is done the anomaly score and classification follows the same principle as K-means. These two will be described further in the theory section.

2.6.4 Classification-based anomaly detection

Classification methods are trained to learn a model classifying data. Usually the training process consists of supervised learning, meaning labeled data is necessary. There are however a few versions of these trained with unlabeled data. Two examples of classification methods are support vector machines and artificial neural networks [1].

Support vector machines (SVM) is mainly used for classification in a supervised mode, trained with labeled data. After providing enough training data, the support vector machine will hopefully learn to classify future data into two or more categories. The categorization of data is achieved by defining an optimal hyperplane between the groups, separating the data. In order to find and learn more complex boundaries to separate data it enlarges the number of dimensions using something called kernel functions. In this way, a boundary that is non-linear in the original feature space can become linear in the new enlarged space and thus separated by the hyperplane. A very simple example of this principle is illustrated in figure 10. This will be explained and derived in the theory section. Support vector machines can also be applied in one-class setting, which is often the case in anomaly detection. In this case, training data from only one class (mostly consisting of normal data) is provided, meaning semisupervised anomaly detection. From that the support vector machine learns to recognize and distinguish normal and abnormal data from each other [3, chapter 9].

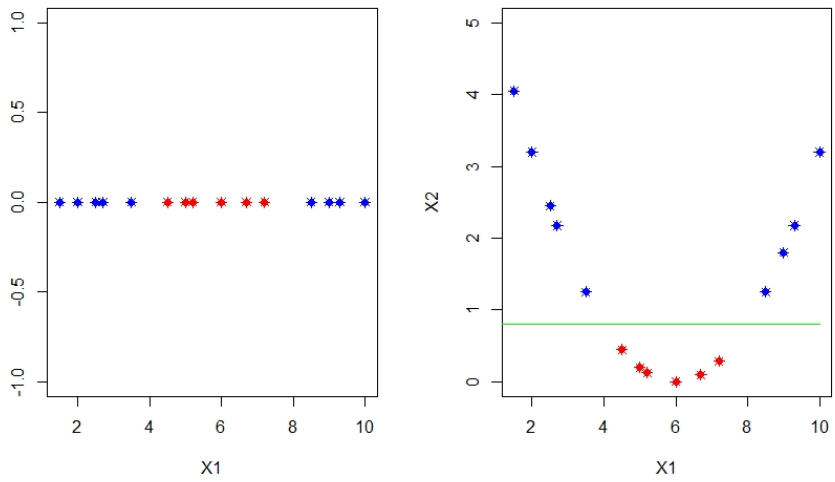

Figure 6: An example of enlarging the feature space to find a linear boundary between two classes. The left figure shows the original data of one dimension. In the right figure the data has been enlarged to two dimensions, enabling a linear boundary to correctly separate the classes.

Artificial neural networks (ANN) are often used in the field of machine learning and have a wide application area. The structure reminds of the brain, with neurons/nodes connected together in a way that makes it possible to learn. In feedforward neural networks the nodes are placed in layers where the input comes in from one side, goes through the layers and comes out on the other side. Every node has a so-called activation function which receives data from all the nodes in the previous layer before sending it forward. When the signals are transmitted between layers they are multiplied by certain weights. Usually the training of a neural network is based on supervised learning with input-output pairs, where the weights are adjusted to minimize the difference between the given output and the output received from the network. When trained properly the network can be used on new data to predict a certain class (classification problem) or a numerical value (regression problem) based on the input data [7].

2.7 Time series analysis

A time series is a collection of data points ordered in time, with a certain sampling interval, called sampling time. Time-series analysis is a field of analyzing time-series to gain meaningful knowledge about the data, its characterizations and patterns. In anomaly detection this approach focus on finding unusual behaviour in time series of the data. Time series analysis can be used to find contextual anomalies too, as time is a context [1].

In time series analysis the dataset is often split up into different elements. All these elements added together form the complete time series. This division is described further in the theory section. When having the dataset split up like this it is easier to analyze and find abnormal behaviour. There are many approaches to detect anomalies in time series. An anomaly could be defined as a data point outside a defined interval, a trend line outside a defined interval, data points being a certain distance away from a trend line, variance of data points outside a defined interval etc. By learning the patterns of the time series one can make forecasts/estimations. If successful, data points deviating much from the estimations can be flagged as anomalies. The estimations can be done in different ways, linear regression and neural network are two possibilities.

3 Theory and Implementation

The very first part of this chapter intends to give an overview of some main concepts in the field of machine and statistical learning. After that follows a subsection about the scaling procedure of the data. The remaining subsections go through the methods used in the thesis in detail and how they were implemented. The main work was done in the programming language R, using Rstudio.

3.1 General concepts

Anomaly detection often falls under machine- and/or statistical learning. This section provides some general terms from the two topics, which will be used in the report.

- **Bias and variance trade off**. The trade off between bias and variance arises in many problems. When setting up a model trying to estimate data, a simplification of the real situation is usually made. This approximation causes an error called bias. In regression problems the bias corresponds to the numerical error between real and estimated values. In classification problems the bias is the number of samples being incorrectly categorized. Variance on the other hand is a measure on to which degree the model would change if using another training set. Having complex functions describing a model tends to increase the variance. In a classification problem this corresponds to having complex boundaries between the clusters, which may decrease the bias but makes the model unreliable on other datasets. Changing the training set should ideally not change the model too much, as the goal is to find one optimized model for future data. Thus it is a trade-off between bias and variance. A high bias and low variance often indicates underfitting, meaning a too simple model has been chosen. Having a low bias but high variance might indicate overfitting - the model performs very good at the training data but is too complex and will perform poorly on new datasets it is applied on [3, chapter 5]. This is illustrated in figure 7.

- **Features**. The features are the input data in a method, i.e. the input variables. Other terms used for features are variables, independent variables or predictors. A feature vector is the whole input vector of a specific variable, for example a vector of CPU values. One can use original features such as CPU usage and memory usage. New features can also be created, an example could be the ratio between two features, CPU usage/memory usage [3].

- **Cost function**. Cost function (also called loss function) is a term in machine learning describing a function the algorithms are trying to minimize. In general the cost function describes the distance between the data and the model. One of the simplest examples is the linear regression problem where a straight line is fit to a dataset. The cost function is describing the error between the line and the data points. By minimizing the cost function the line is fitted to the data [7]. A cost function can also depend on the accuracy of classification, which is further described in the support-vector machine section.

- **Training- validation- and test dataset**. When creating a machine learning model the data are often split up in two or three parts: training, (validation,) and testing. The training dataset is used to train the model, i.e to fit the parameters. In supervised learning the training set is labeled and consists of input-output pairs. During training of a neural network for example, these input-output pairs are used to adjust the weights. When the training is finished a testing dataset is used to evaluate how well the model performs. These are the main two ones. A third dataset can also be used before the method is tested, the validation set. This set intends to prevent overfitting as it tries to give an unbiased evaluation of the training set while tuning the parameters [3]. In unsupervised learning the division of the dataset is not as obvious, or even non existent. For example in the KNN method where the distances to every point's K nearest neighbors are calculated and summed up. After performing the algorithm on a dataset the results can be analyzed directly. Thus no division between training and testing. Some unsupervised methods (for example KNN) are modified in this thesis and are divided into training and testing. The reason is to perform real time anomaly detection computationally effective instead of running KNN on a big set of old and irrelevant data. The idea is to use old data when calculating the distances to neighbors of new data points, without performing the algorithm on the old points. Thus the distance to the K nearest neighbors is only calculated for the new observation points but using the distances

to old points in the training set. This will be explained further in the implementation of the methods.

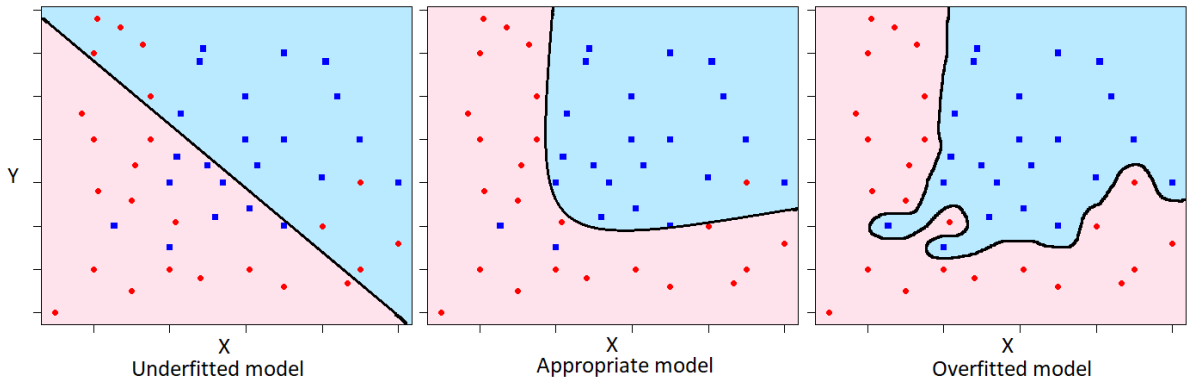

Figure 7: An example illustrating the trade off between bias and variance. The plots show training data of two classes from which a model is trained to classify future data correctly. The boundary separating the two classes is marked in black. In the left plot an underfitted model with high bias and low variance is shown. It is too simple and classifies many samples incorrectly. The right plot shows an overfitted plot with zero bias (all points are classified correctly), however with a big variance. This model will perform poorly on new datasets, as the suggested boundary is unrealistic and strongly shaped after this specific training set. The middle plot shows a model with a good balance between bias and variance.

3.2 Scaling of the data

Before applying any method with more than one feature vector the data was scaled. In this way the different parameters get evenly weighted in the algorithms. The scaling was done using the following formula on every feature:

$$\mathbf{x'} = \frac{\mathbf{x} - \overline{x}}{\sigma} \tag{3}$$

where $\mathbf{x}$ is the original feature vector (input data), $\overline{x}$ is the mean value of $\mathbf{x}$, σ is the standard deviation of $\mathbf{x}$ and $\mathbf{x'}$ is the new scaled vector. In this way $\mathbf{x'}$ will get a mean value of 0 and standard deviation 1.

For the methods where the data were split up in a training set and a testing set the mean value and standard deviation were calculated using only the training set. New observation points (test data) used the old mean values and standard deviations. Having a big training set it is not efficient to calculate a new mean value and standard deviation for every new observation point, and the difference will be negligible. Furthermore as this report aims to investigate the possibility of analyzing performance data continuously, scaling the whole test vector at once is unrealistic. This would mean that "future points" are used, meaning data which have neither been used in the training set nor been tested yet are used for scaling current data. Therefore the scaling procedure was made this way.

3.3 K-Nearest Neighbor

K-nearest neighbor methods (KNN) calculates the sum of distances from every data point to its K nearest neighbors. This distance is the basis used to decide whether the point is normal or not. The amount of neighbors, K, and the length of distance required to be classified as an anomaly has to be decided. There are also other variants of nearest neighbor methods, for example counting the amount of data points within a certain area. In that case the amount of neighbors is used as anomaly score instead of the distance itself.

A problem with nearest neighbor based methods like KNN is the computational work, which can be reduced by using space-partitioning data structures. One example is $k-d$ trees. A $k-d$ tree is a binary tree used to create a data structure organizing the dataset as a tree. In this way

all the points in the data fall into different so called leaf nodes. Every leaf in the tree is a point in a k-dimensional space, depending on the number of dimensions [9]. $k-d$ trees are very useful when searching for nearest neighbors. The computational work for applying a K-nearest neighbor method would usually grow very fast as the distances from every point to all other points in the dataset has to be calculated. More precisely the computational work will grow as n^k where n is the number of points in the dataset and k the number of dimensions. By using a $k-d$ tree on the other hand, the only distances calculated are the ones within every leaf node. Thus the computational work is reduced heavily, however on the expense on potentially missing close neighbors lying outside the current leaf node.

3.3.1 Implementation of K-Nearest Neighbor

Unsupervised KNN is usually performed without dividing the dataset into training and testing, but running it on the whole set at once. Applying anomaly detection, the sum of the distances to the K nearest neighbors at every point would be calculated and the anomaly score based on that. In this thesis KNN was implemented in a modified way where the KNN was divided into a training set and a testing set. The reason for this modification was real time anomaly detection. Implementing it the classical way means the whole dataset needs to be recalculated every time a new sample is added. Using a training set however makes it possible to investigate new samples, coming one at the time, more effectively. The "training" simply consisted of taking all the coordinates from training samples. The new points in the testing set were then compared to the coordinates from the training set with the anomaly score based on the distance to the K-nearest neighbors. K was set to 7. The implementation was made in R using the packages 'dbscan', 'sp', and 'Searchtrees'.

3.4 Support-vector Machines

The support vector machine is a generalization of another more simple classifier, the maximal margin classifier. The maximal margin classifier has lots of limitations but is intuitive and a good start before explaining the more complicated support vector machine. The principle is simple. Imagine having a p-dimensional dataset of observations, $x_1, x_2, ...x_n \in \mathbb{R}^p$ labeled into two categories as $y_1, y_2, ...y_n \in [-1, 1]$. Supposing the two categories are separated in two different clusters, one way to distinguish the categories from each other would be defining a hyperplane between them. If the dataset is p-dimensional, the hyperplane has $p-1$ dimensions [3, chapter 9]. Thus having three dimensions in the dataset results in a two dimensional hyperplane. Mathematically the general equation defining the p-dimensional hyperplane, H is given by:

$$H = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0 \tag{4}$$

where β are the parameters defining the hyperplane and X is the data. After defining the hyperplane (setting the β parameters), classification of observation points is simply made by checking whether H is bigger or smaller than 0 for every X, meaning which side of the hyperplane the point lies in. For example all observations with $H > 0$ are associated with $y = 1$ and for $H < 0$, $y = -1$.

Having the dataset separated like this means that an infinite amount of hyperplanes can be defined in between. One way to find the (hopefully) optimal hyperplane is to define it in such a way that it maximizes the distance between the hyperplane and the closest datapoints (see figure 8). This is called the maximal margin hyperplane [3, chapter 9]. The margin refers to the perpendicular distance between the hyperplane and closest data points in the training set, which is maximized. Thus the maximal margin hyperplane is the solution to the optimization problem of maximizing M:

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_p X_{ip}) \geqslant M, \quad \forall i = 1, ..., n \tag{5}$$

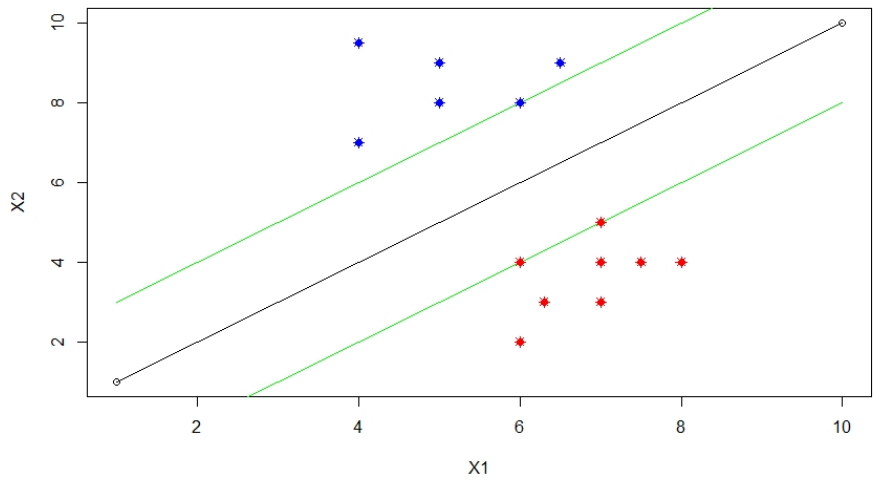

Figure 8: An example of a maximal margin hyperplane, separating two classes (red and blue) in a dataset of two dimensions. The two groups are separated and the hyperplane (black line) is adjusted so that the margin is maximized. The margin is defined as the distance from the hyperplane to the green lines. The points lying on the green lines are support vectors. They "support" the hyperplane, meaning they determine the equation of it.

One obvious drawback with the maximal margin classifier is the non-separable case, meaning the dataset cannot be separated by a hyperplane. Instead the clusters are overlapping, with some data points on the wrong side of the hyperplane regardless the equation of it. Another drawback is the sensitivity to overfitting, as one single point could have a big impact on the hyperplane seperating two large clusters [3]. A solution to these drawbacks is to use the *support vector classifier* which uses a so called soft margin. Instead of using a hyperplane trying to separate the classes perfectly, this method is willing to miss-classify data points in favour of robustness (i.e. higher bias but lower variance). It takes all points within a certain margin from the hyperplane into account, not only the very closest ones as the maximal margin classifier. Increasing the number of data points taken into account when defining the hyperplane makes it more soft [3, chapter 9]. Mathematically the optimization problem can be expressed as:

$$y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_p X_{ip}) \geqslant M(1 - \epsilon_i) \quad \forall i = 1, ..., n \tag{6}$$

$$\epsilon_i \geqslant 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{7}$$

where ϵ are slack variables allowing data points to be on the wrong side of the hyperplane and margin and C is a tuning parameter. For a datapoint X_i, $\epsilon_i = 0$ if the data point is on the correct side of the margin, $\epsilon_i > 0$, if it is on the wrong side of the margin, and $\epsilon_i > 1$ if on the wrong side of the hyperplane. M is the margin, which is being optimized/maximized. The tuning parameter, C determines the budget over the number of violations tolerated (the maximum allowed sum of ϵ, seen in equation 7). One could say C regulates the softness. A large C means a wider margin with more support vectors included with the algorithm more tolerant to violations. This increases the bias in favor for the variance which becomes lower. A small C on the other hand decreases the bias but with increased variance, potentially causing overfitting [3]. The support vector classifier is illustrated in figure 9 below:

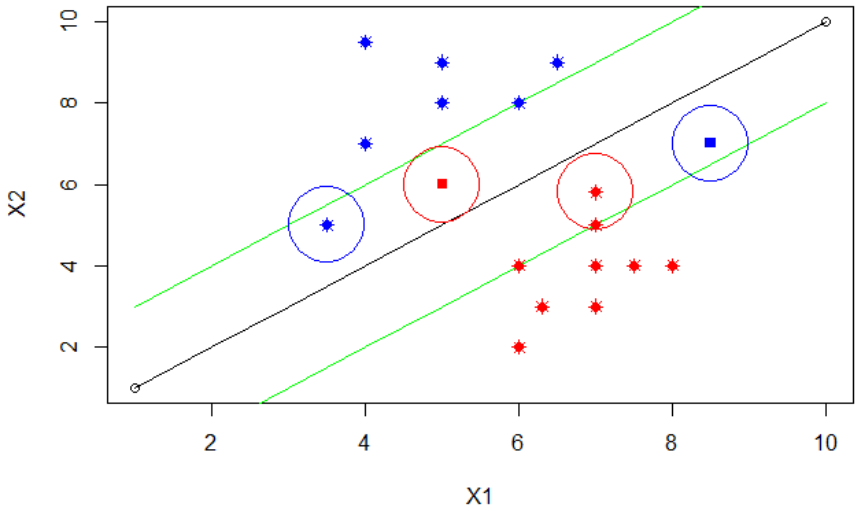

Figure 9: An example of a support vector classifier, fit to a small dataset of two classes (red and blue) in two dimensions. The hyperplane is shown in black and the margins in green. The normal points marked with circles have violated the margin ($\epsilon_i > 0$). The square shaped points marked with circles have also violated the hyperplane ($\epsilon_i > 1$). The other points are lying on the correct side of their respective margin ($\epsilon_i = 0$) and do not affect the hyperplane as they are not support vectors.

The support vector classifier solved the problem of overlapping clusters. However it has a big limitation: it is a linear classifier, meaning it only works on datasets being linearly separable. For cases of non-linear class boundaries the method has to be extended further which leads to the final approach: the support vector machine. The idea is to enlarge the number of dimensions of the feature space. It could be extended to quadratic, cubic, or higher polynomial features [3]. Thus instead of using the support vector classifier in p dimensions (where p is the original number of features/dimensions), the support vector classifier could use $2p$ dimensions for example, meaning extending the feature space with quadratic terms. The optimization problem of maximizing M now becomes:

$$X_1, X_1^2, X_2, X_2^2, ..., X_p, X_p^2 \tag{8}$$

$$y_i(\beta_0 + \sum_{j=1}^{p} \beta_{j1} X_{ji} + \sum_{j=1}^{p} \beta_{j2} X_{ji}^2) \geqslant M(1 - \epsilon_i) \quad \forall i = 1, ..., n \tag{9}$$

$$\epsilon_i \geqslant 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \quad \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1 \tag{10}$$

The reason to extend the feature space to higher dimensions is finding a linear boundary separating the data. A boundary that is non-linear in the original space can become linear in the new enlarged feature space (see figure 10). Enlarging the feature space can be done in different ways. The support vector machine use kernels [3, chapter 9].

3.4.1 Kernel methods

Kernel functions measure the similarity between two points, meaning producing a score depending on the distance. Thus the output of the kernel function will depend on the Euclidean distance between a given data point, x and so called landmarks, l. There are different types of kernel functions, the radial kernel calculates the score in the following way:

$$f = K(x, l^{(i)}) = exp(-\frac{||x - l^{(i)}||^2}{2\sigma^2}) = exp(-\gamma||x - l^{(i)}||^2) \tag{11}$$

where γ can simply replace the $1/(2\sigma^2)$ term. Note that $||x - l^{(i)}||$ is simply the Euclidean distance, thus the coordinates are not of interest for the kernel score, but the distance between them. This distance is calculated by calculating the dot product (see the section below). Now imagine the feature space gets enlarged, so x and l becomes $f(x)$ and $f(l)$ instead. In general increasing the number of dimensions like this is computationally heavy. However the kernel functions solve this with the so called kernel trick [8]. As mentioned the distance is calculated by the dot product, $K(< f(x), f(l) >)$, which usually means computing $f(x)$ and $f(l)$ before calculating the dot product of the two. However with the right function (kernel) it turns out that the result of the dot product becomes a scalar. Thus the dot product can be calculated implicitly without the computational work of extending the number of dimensions. This makes support vector machines powerful and computationally cheap [3]. The mathematical reason behind this is explained in the subsection 3.4.2 below, 'Derivation of support vector machines'.

3.4.2 Derivation of support vector machines

This section provides an explanation of support vector machines, from a more mathematical point of view. The section below is explained through figure 10. The figure is in 2D but the formulas and derivation are general for any number of dimensions. We start by defining a weight vector, ω perpendicular to the hyperplane. The dot product of two vectors will be the projection of one of the vector onto the other one, which is utilized when calculating the distance between observation points to the hyperplane. In figure 10 the dot product of ω and an observation point x is marked as red. If we denote the closest distance from the origin to the hyperplane as b, the dot product of ω and a point lying to the right of the hyperplane (marked as '-') will be bigger than b, while the dot product of ω to points left of the hyperplane (marked as '+') is smaller than b. Requiring a margin of at least 1 from the hyperplane to the observation points for classification, this can be expressed as:

$$\boldsymbol{\omega} \cdot \boldsymbol{x}_+ + b \geqslant 1, \quad \boldsymbol{\omega} \cdot \boldsymbol{x}_- + b \leq -1 \tag{12}$$

where $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ are observation points to the left and right of the hyperplane respectively. If the two groups are also associated with a y-value, $y_i = 1$ for '+' samples and $y_i = -1$ for '−' samples, the two equations can be merged into one and written as:

$$y_i(\boldsymbol{\omega} \cdot \boldsymbol{x_i} + b) - 1 \geqslant 0 \tag{13}$$

The width, W in figure 10 can be expressed as:

$$W = (\boldsymbol{x}_+ - \boldsymbol{x}_-) \cdot \frac{\boldsymbol{\omega}}{||\boldsymbol{\omega}||} \tag{14}$$

where $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ are the blue observation points '+' and '−' lying on the margin in figure 10. Since the two points lie on the margin we know the expressions in equation 12 equate to 1 and -1 respectively and can therefore be rewritten as: $\boldsymbol{x}_+ \cdot \boldsymbol{\omega} = 1 - b$ and $\boldsymbol{x}_- \cdot \boldsymbol{\omega} = 1 + b$. Putting these expressions into equation 14 we get a new expression of the width:

$$W = \frac{2}{||\boldsymbol{\omega}||} \tag{15}$$

The hyperplane should be defined in a way that maximizes the width, meaning minimizing $||\boldsymbol{\omega}||$. This is equivalent to minimizing $\frac{1}{2}||\omega||^2$ (writing it this way will make the math easier). Before minimizing $||\omega||$, the constraints needs to be taken into account. Solving the optimization problem with the constraints can be done using Lagrange multipliers. This means taking the original minimization problem and subtracting the constraints multiplied by a constant, α. On this form the new loss function, L can be minimized as usual, by finding where the gradient is zero. Solving this optimization problem using Lagrange multipliers gives:

$$L = \frac{1}{2}||\boldsymbol{\omega}||^2 - \sum_{i=1}^{n} \alpha_i y_i[(\boldsymbol{\omega} \cdot \boldsymbol{x_i} + b) - 1] \tag{16}$$

$$\frac{\partial L}{\partial \boldsymbol{\omega}} = \boldsymbol{\omega} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x_i} = 0 \implies \boldsymbol{\omega} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x_i} \tag{17}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{18}$$

Using equation 17 and 18 where the derivatives are set to zero, two expressions can be received. Putting these expressions into the equation for L (equation 16) the final expression looks like:

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j}^{n} \sum_{i}^{n} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \tag{19}$$

Now it can be seen that the maximization of the width depends on the dot products of the sample vectors, $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$. Thus maximizing the margin only requires the dot product. Suppose now the original feature space is enlarged from $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ to $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. Defining the optimal hyperplane in the enlarged space (maximizing the width), can now be done only by calculating the dot product of the two transformations, $f(\boldsymbol{x_i}) \cdot f(\boldsymbol{x_j})$. Thus the kernel function, $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = f(\boldsymbol{x_i}) \cdot f(\boldsymbol{x_j})$, calculates the dot products of the enlarged space transformations without explicitly calculating the transformations, $f(\boldsymbol{x_i})$ and $f(\boldsymbol{x_j})$. This is what makes it computationally cheap to apply support vector machines.

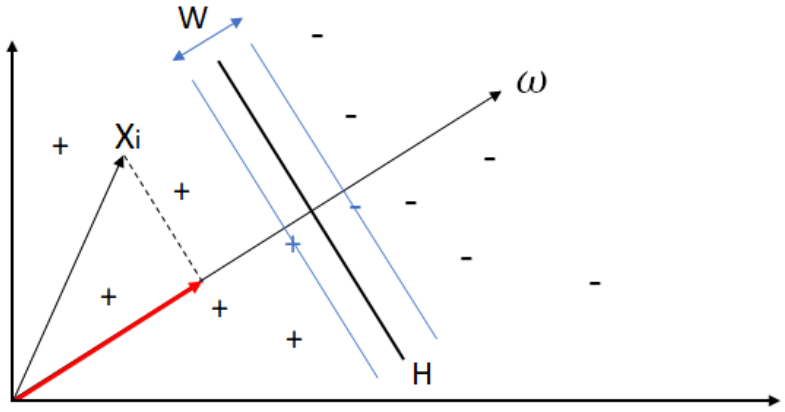

Figure 10: An illustration of support vector machines and the utilization of dot products. ω is the weight vector, perpendicular to the hyperplane, H. The dot product between ω and the observation point, X_i is marked as red. Observation points to the right of the margin (colored in blue) are marked as '-', while points to the left of the margin are '+'.

3.4.3 Implementation of support-vector machine

The SVM was implemented using one-class SVM in R with the 'e1070' and 'rpart' package, using radial Kernel. The cost parameter was set to 1. It did not have any effect on the results. The parameter γ on the other hand did have an effect on the number of anomalies (see equation 11). N is another tuning parameter. The parameters N and γ were adjusted by trial and error to regulate the ratio between anomalies and normal data. Setting the parameters $N = 0.00009456$ and $\gamma = 0.00001$ resulted in that about 0.023% of the samples in the test set were flagged anomalous (14 in total). Setting the parameters $N = 0.0000962665$ and $\gamma = 0.00002$ resulted in a 0.53 % of the samples in the test set were flagged anomalous (323 in total). These were the two sets of parameters used in this thesis.

3.5 K-means clustering

K-means clustering is a widely used clustering method. It splits the dataset into K different clusters. Each data point belongs to exactly one of the clusters, thus they are not overlapping. With these constraints the next step is to minimize the variation within each cluster to cluster points lying close to each other together. There are different ways of defining the variation in the clusters but most common is to use squared Euclidean distance [3, section 10.3.1]. Having a dataset $x_1, x_2, \ldots, x_i, \ldots, x_n$ with a clusters $C_1, C_2, \ldots, C_K$, we define the variance within one cluster as:

$$V(C_K) = \frac{1}{|C_K|} \sum_{i,i' \in C_K} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \tag{20}$$

where X_i and X_i' are data points with indexes belonging to cluster C_K. $|C_K|$ is the number of points within the cluster, and p the number of features/dimensions. Having a dataset of K clusters, denoted as $C_1, C_2, ..., C_K$, each with variation $V(C_1), V(C_2), ..., V(C_K)$ the goal is to minimize the sum of the variations by assigning every point to the appropriate cluster [3, section 10.3.1]. The final optimization problem for K-means clustering consist of minimizing the summed within variance, S_v:

$$S_v = \sum_{k=1}^{K} \frac{1}{|C_K|} \sum_{i,i' \in C_K} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \tag{21}$$

Now the question is how to solve this. There is an extremely high amount of combinations ($\sim K^n$) to cluster n data points into K clusters, unless n and K are very small. The way the K-means algorithm solves this can be divided into 3 steps:

1. Assign every data point with a random number of 1 to K. In this way it will form initial clusters with random centroids.
2. Compute the centroids for every cluster. The centroids are the centers of the clusters, meaning the mean value of the data points belonging to that cluster.
3. Assign the data points to the cluster having the closest centroid. This means that data points belonging to a cluster C_{K1}, with another centroid C_{K2} lying closer become assigned to the cluster C_{K2} instead of C_{K1}.

Steps 2 and 3 are iterative and will keep going until the centroids converge to certain positions. When the centroids do not change along the updates anymore the algorithm stops. Since the first step is random the final clusters for K-means clustering may differ from time to time for the same dataset. Thus the method is only guaranteed to find the local minimum of equation 21 not the global one [3]. To find the global minimum or at least a good local minimum K-means can be run multiple times (with different starting centroids) before choosing the one with lowest variation in the final clusters, meaning the lowest S_v (see equation 21). Except the problem of finding a local rather than the global minimum K-means also has the drawback of having a fixed number of clusters. One possibility to find the best amount of clusters is to run K-means several times with different number of clusters and compare the S_v in the end. By plotting the results one can compare and hopefully find a number of K where convergence has been reached. K-means can be used in different ways for anomaly detection, one way is to select the data points furthest away from their clusters as anomalies.

3.5.1 Implementation of K-means clustering

K-means clustering was implemented in R, using the 'Kmeans' function. The number of clusters, K was chosen by running K-means several times with different values of K, plotting the sum of the final within-cluster variances from every run. The number of K where the curve flattened out was chosen as an increased number of clusters would not contribute as much. In R the 'Kmeans' function has a built-in argument, 'nstart', to choose the best initial configurations. Nstart was set to 50 meaning the K-means algorithm uses 50 different configurations (see step one in the K-means algorithm explanation above), finally reporting the best one, meaning the one with the least total within-cluster variance. This is a way to avoid undesired local minimums in the cost function, as a bad starting configuration might not converge well, but stop in a local minimum instead.

The anomaly score was based on the Euclidean distance to the closest centroid, meaning the points furthest away from their belonging cluster center were flagged as anomalies. Usually the K-means algorithm is run on one set, meaning there is no division between training and testing, but simply one big set. In this thesis however the set was split into training and testing. The 'Kmeans' method was run only on the training set, to get K different centroids. The coordinates of the centroids were saved and the anomaly detection on the test set consisted of calculating the distance to the nearest centroid, with the anomaly score based on the Euclidean distance between the data points in the test set and the centroids.

3.6 Self-organizing maps

The self-organizing map (SOM) is a type of unsupervised artificial neural network, using so called competitive learning. It is used to classify and discover patterns in high dimensional data, meaning a type of dimensional reduction and clustering method. It creates a 2D-map of the data, which might reveal structures and clusters of the multidimensional input data in a legible way. The 2D-map consists of nodes. Data with the most similar characteristics end up in the same nodes and are thus clustered together. Nodes lying close to each other contain data of similar characteristics too [6]. One part of applying SOM consists of training, by building the map. The map can then either be analyzed itself, meaning analyzing the training data, or used to classify new data. Classifying new data is called mapping.

In contrast to ordinary ANN methods (which are explained further under in section 3.8.2), SOM is an unsupervised algorithm making the fundamentals a bit different. Some concepts in SOM have the same names as in ANN but with different meaning.

The SOM is built of nodes arranged in a grid. The grid is either rectangular or hexagonal with the size predetermined. Each node has weights representing a coordinate in input space. Thus if the number of dimensions in the training data is three for example, the nodes have weights on the form (W_1, W_2, W_3), i.e. a coordinate in 3D. The learning consist of matching every sample vector in the training set to the most similar node. More exactly to find the node whose coordinates have the shortest Euclidean distance to the current sample vector. The weights of the winning node will be slightly modified, to lie closer to that sample vector. Every sample vector is introduced to the map and the nodes are slightly adjusted every time [6].

Every sample vector is introduced several times each in order for the map to converge properly. It is not only the very best matching node whose weights are adjusted. The neighbors to that node, meaning nodes lying close to it are also updated but to a lesser extent. In the beginning the number of neighbors are high, i.e. every node within a large radius of the best matching node is modified. As the map converges this radius decreases.

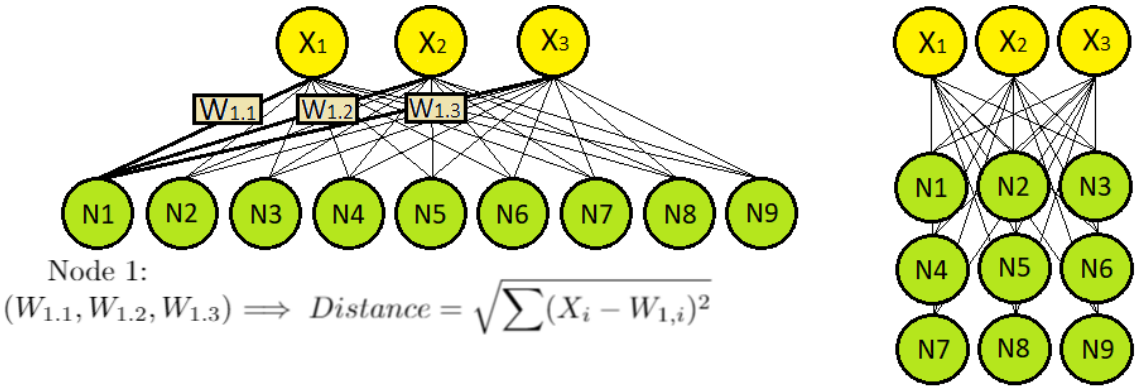

Figure 11: Illustration of the training process of a self-organizing map. The yellow circles are input features, the green ones are nodes. The black lines mark the connection between them. To the right a self-organizing map consisting of nine nodes is seen. To illustrate the connection between the input features and the nodes better, the nodes are lined up in a row to the left. This example have three features and thus every node's weight consist of three components. In the figure to the left the weights to node 1 is marked. These are used when calculating the Euclidean distance between node 1 and the sample vector. Supposing node one is the best matching node, its weights will be slightly modified to the sample vector. Looking at the right figure it can be seen that node number 4, 5 and 2 are neighbors to node 1. Thus these might also be updated. Note that SOMs are usually much bigger, consisting of many more nodes.

The training is performed in different steps.

1. Initialize the weights of each node.
2. A sample vector from the training data is extracted. The length of the vector depends on the number of dimensions in the data.
3. Find the node whose weights match the sample vector best. The sample vector is compared with every node's weights by calculating the Euclidean distance between them. The best matching node is marked as the BMU, best matching unit.
4. Calculate the radius and update the weights. After finding the BMU the weights are adjusted to become more like the sample vector. This is done on the BMU node and its neighbors, meaning nodes that lie within a certain radius from the BMU. Nodes within that radius are adjusted, the closer they lie the more adjusted. Usually the radius starts off with a big value and decreases as the the map converges.

After the weights have been adjusted the cycle is repeated from step 2, meaning the next sample vector from the dataset is selected and undergoes the same procedure. As more input vectors are inserted the weight's in the nodes become more accurate and a map will form, potentially revealing patterns in the data [6].

3.6.1 Implementation of Self-organizing maps

SOM was implemented in R with the 'kohonen' package. The grid for the codebook, i.e. the actual map was 10x10 of type hexagonal. The number of times in which the whole dataset was presented to the network was set to 150. This is equivalent to the number of iterations, i.e. step 2 to step 4 described above was performed 150 times per sample vector. A graph of the mean distance to the BMU as a function of the number of iterations was used to confirm the SOM converged within 150 iterations. The learning rate of the algorithm, meaning the degree of which the weights were modified every update started at 0.05 and declined linearly to 0.01 over the number of iterations. The radius also decreased with the number of iterations. Here the default value was used, meaning it started with a radius adjusted to cover 2/3 of all the nodes. The radius decreased linearly with the number of iterations to zero.

A training set was used to create a self-organizing map, described above. The map was plotted and studied in different ways, showing different properties. Nodes with abnormal properties, such as high CPU disk and usage, or large distances between the neighbors in a node, were picked as 'anomaly nodes'. All sample vectors from the testing set were then mapped into the SOM, paired up with their best matching nodes. The samples ending up in the 'anomaly nodes' were flagged anomalous. Depending on the number of anomalies desired, more or less 'anomaly nodes' could be picked. The nodes with large distances between neighbors were always chosen first.

3.7 Memory-CPU usage ratio

When applying anomaly detection new features can be created. This method takes the logarithm of the quotient between memory usage and CPU. In this way a new feature is received. If a feature is generated from a certain stochastic model/distribution a simple anomaly detection approach can be applied by flagging all points outside a certain interval as anomalies, lying in low probability areas.

This approach was implemented in R, without no use of any packages. The feature was normally distributed but no probability density function (PDF) was used. Instead the anomaly detection were based on the ratios directly. Thus the normal distribution was not directly used but the principle is still the same however, flagging extreme values in low probability areas as anomalies. By taking the logarithm of the quotient as the feature the results were easier to study. A change in ratio could be then be studied in the plot, regardless if CPU usage or memory usage was bigger. Using the original memory usage/CPU as the feature makes it easy to spot abnormal high values of the ratio but harder to see the smaller ones as they go towards zero. Two limits were set up regulating the maximum and minimum ratio between memory usage and CPU. The samples exceeding the ratio limits were flagged anomalous. The limits can be adjusted after the desirable number of anomalies. Different limits were tried, as they were very easy to adjust. Setting the

minimum limit to -1.44 and the maximum to 1.8 resulted in 14 anomalies, i.e. one detected anomaly every third day by average. Limits of -0.9795 and 1.0296 resulted in 323 points landing outside the allowed interval being flagged as anomalies.

3.8 Time series analysis

There are different possible features to extract and make use of when applying anomaly detection on time series. Three examples are listed below:

- **Original data.** Original data. The most straight forward approach is to extract and look at the time series of the original data, for example CPU Usage. This is the most important feature and is often the only one used.
- **Difference between sample points.** By making a time series of the absolute value of the difference between every sample point and its neighbor, $|S(t) - S(t-1)|$, a new feature can be made and investigated for anomaly detection. This is a measure of how much the data varies, and thus a way to detect abnormal behaviour.
- **Local variance.** A drastic increase or decrease in variance might suggest that something strange is happening. Therefore using a local moving variance is a way to analyze the time series. The variance can also be used to determine whether a local extreme value should be classified as an anomaly or not. Some data points positioned far away from the moving average may or may not be normal depending on the variance in that area.

When having a feature in a time series the next step is to analyze the time series of that feature. There are three main parts a time series might consist of: a *trend component*, a *seasonal component* and a residual *stochastic component*. The trend component is the long-term trend of the data, a moving mean value. The seasonal component consist of the cyclic patterns in the dataset, meaning seasonal variations which repeat over a certain time period, for example daily or yearly. The residual stochastic part is simply the remaining part which cannot be predicted by the first two components. This division of a time series is illustrated in figure 12.

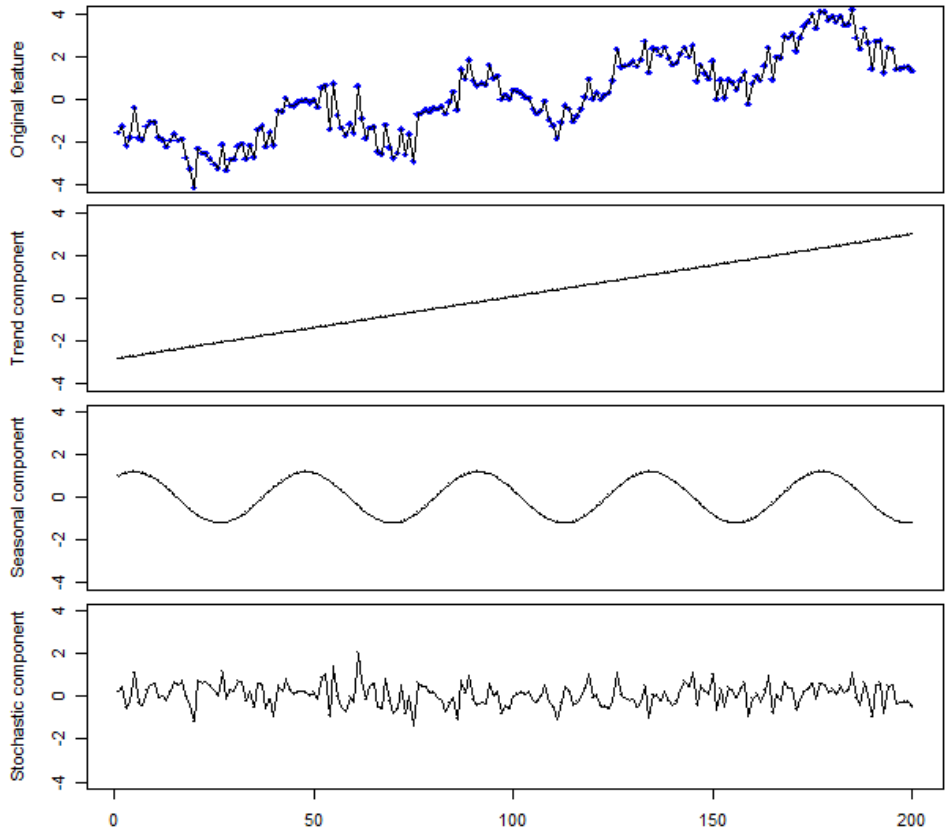

Figure 12: Division of a time series. The original feature in the top is divided into three components: the trend component, the seasonal component and the stochastic component. Added together they constitute the complete time series of the original feature. In this figure the trend component is linear and the seasonal component constitutes one perfect sine wave. This is the simplest case but the components can take different forms.

There are different ways to apply anomaly detection in time series, largely depending on the sampling data and the components. Having only a stochastic part, a simple interval could be set where sample points outside of it are flagged as anomalies. It could be a set of sample points normally distributed around a constant mean value, where points more than for example 3 standard deviations away are classified as anomalies.

A long-term trend of the data makes this approach useless as the mean value is not constant anymore. In this case a moving average could be used instead where the anomaly classification is based on the difference between the trend line and the sample points. A moving average is simply a moving mean value, taking only a certain number of sample points into account when calculating the mean value. This gives a smoothed version of the original data with less variance and fluctuations. A similar alternative approach is the so called weighted moving average. The idea is to take the n latest data points and give them certain weights before calculating the mean value. In this way the closer sample points can have a bigger impact than the earlier ones, and the trend line will shift faster. The previous sample point can for example be multiplied by n, the second previous by $n-1$ etc, until the last value in the n-points moving window is multiplied by 1. Except for using a trend line for comparing to sample points, one can also study the trend line itself, by setting up an interval in which the trend line must lie to be classified as normal.

Having seasonal components makes it a bit less trivial to pick up the patterns of the time series. The trend line will not catch the seasonal movements unless they are very long. Or the trend line has to be based on a very short time interval and/or be strongly weighted. In this case however it will fail to pick up the long-term trend and thus loose that quality. Also it will not be able to find and flag unusual patterns in the seasonal components, meaning contextual anomalies. A better way to catch seasonal components is to use linear regression or neural networks, depending on its complexity.

3.8.1 Linear regression

Applying linear regression on time series is a way to catch patterns in the data. Similar to the trend line picking up the long-term trend of a dataset, linear regression can be applied to catch the seasonal components. The principle is to estimate the current sample point by using previous sample points multiplied by certain weights. This corresponds to creating a column vector of earlier sample points, and multiply the transpose of it by a column vector containing the weights. The output becomes a scalar and the value estimates of the current sample point. This can be expressed mathematically as:

$$Y_{est}(t) = Y(t-1)w_1 + Y(t-2)w_2 + ... + Y(t-n)w_n = \mathbf{Y}^\top \mathbf{W} \tag{22}$$

where $Y_{est}(t)$ is the estimated sample point of $Y(t)$. The earlier sample points are denoted $Y(t-1)$,..., $Y(t-n)$, n steps back in time. w_1,...,w_n are the elements of the weight vector. $\mathbf{Y}$ is a column vector of earlier sample points and $\mathbf{W}$ the weight column vector, both vectors of length n. Finding the optimal weights becomes an optimization problem which seeks to minimize the sum of all the squared errors, E between the estimated sample points and the real ones. The cost function, which is minimized, can be expressed as:

$$E = \sum_{t=1}^{m} |Y(t) - Y_{est}(t)| \tag{23}$$

where E is the sum of the squared errors between all estimated values (defined in equation 22) and real values, over an interval of m time samples. The variable m can be viewed as the length of the training interval. Thus the training part of the algorithm consist of optimizing the weight vector such that the sum of the squared errors between the estimated and real values is minimized. When trained properly it can be applied on new sample points to make estimations/predictions, finding patterns in data.

In anomaly detection the estimated/predicted values can be compared to the real values, and the anomaly score is again based on the difference between them, following the same principle as with the trend line. The fact that linear regression is linear is an advantage as the cost function will get only one minimum, meaning a guarantee of finding the global minimum. Non linear problems on the other hand will get multiple minimums and the optimization algorithm might find and get stuck in a local one. This results in a weight vector which is not optimal. However linear regression has limitations due to its linearity. It cannot learn as complex things as a non linear neural network for example.

3.8.2 Artificial neural network

An artificial neural network (ANN) is a network consisting of nodes/neurons connected in layers (see figure 13). It starts with an input layer where the signals/data comes in, one or several hidden layers and finally an output layer. In a feed forward neural network the signals are going in only one direction, meaning the nodes do not form a cycle. Every node in the hidden layers has an activation function [7].

The actual learning of a neural network consists of adjusting the weights. All the black lines connecting the nodes in figure 13 do not only transmit the output value to the next layer but also multiply them by a certain weight, w. Thus every connection has a certain weight, controlling the dampening/amplification of the signal. During training these weights are adjusted, similar to how the weights were tuned in the linear regression case. The training data consists of input-output pairs and the weights in the ANN are adjusted through back-propagation to minimize the error between the given output and the output produced by the ANN [7]. Neural networks can be applied both for classification and regression problems. Having a time series analysis (regression problem) the inputs could be values of earlier time samples and the output the value of the current sample. Through training the weights are adjusted so that earlier values can be used to predict/estimate the current one [7].

In the hidden and the output layer the nodes have several inputs, i.e. the outputs from every node in the previous layer multiplied by its respective weight. The received input in every node is summed up and put into its activation function before it is sent to the next layer. To every layer (except for the input layer) there is also a bias added of value one. The reason for this is to shift the function vertically if necessary. This can be understood by an example of a simple linear regression

case in one dimension such as $y = kx + m$. The "normal" node will then decide the slope of the line, meaning the term kx. However the bias will shift the line vertically and thus constitutes the m value of the equation. The output of the bias nodes are like the rest of the outputs multiplied with a weight before received by the nodes. Thus the bias value of one is not put straight into any node [7].

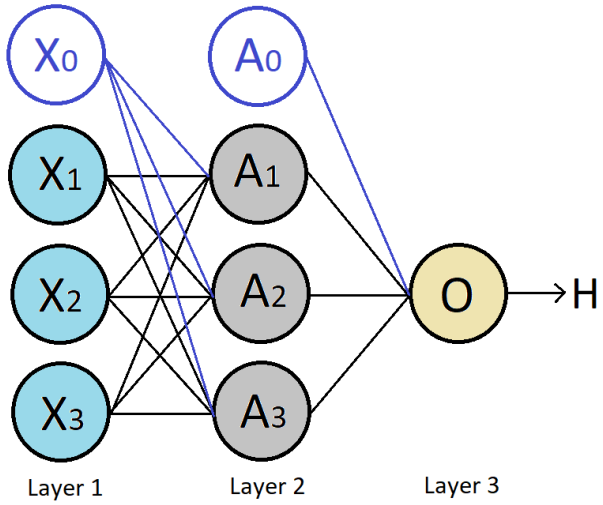

Figure 13: Illustration of a feed forward neural network, consisting of three layers. Layer 1 is the input layer with input units. Layer 2 is the hidden layer with activation functions and layer 3 is the output layer, giving the hypothesis, H. The top blue nodes are bias units, which always outputs a value of one. This is a feed forward neural network and thus all the signals go from left to right, never backwards or in loops. The black lines connecting the nodes constitute both the transmission lines of the signals and also include the weights, regulating the dampening/amplification of the signals.

There are different activation functions, sigmoid (also known as logistic) and tanh (a hyperbolic function) are two common ones in feed forward neural networks. The sigmoid and tanh functions are defined by equation 24 and 25 respectively, with their output shown in figure 14.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{24}$$

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{25}$$

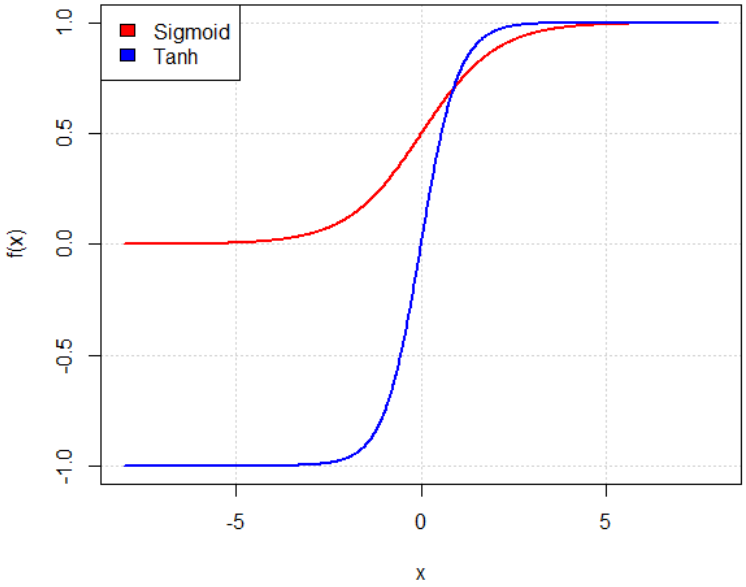

Figure 14: The output of the activation functions sigmoid and tanh as a function of x.

As can be seen in figure 14 the output of the sigmoid function ranges between the interval (0 to 1), while the tanh function ranges between the interval (-1 to 1). The activation function can also be linear, which will result in a linear regression problem. There are other activation functions too, commonly used in deep learning, like ReLU (rectified linear unit) for example. These are not described in this thesis. In time series analysis it is often enough with one hidden layer in the neural network, as the problems are usually not too complex.

3.8.3 Implementation of Time series analysis

Applying time series analysis for anomaly detection was done in three different ways, using linear regression, neural network, and one simple script looking for abnormal trends in the data. Linear regression and neural networks aimed to pick up the seasonal variance in the CPU usage, meaning the daily patterns. They were implemented to make predictions/estimations of the current sample point using earlier sample points. The anomaly score was based on the difference between the estimated CPU value and the real CPU value.

In order to find a clear pattern of the daily cycles the weight and sample vector needs to be long enough. To have a minute interval and catch daily patterns requires a very long sampling and weight vector. Therefore the time sampling interval was increased to one hour (using mean values from the minute samples). This was done before applying linear regression and neural networks.

- **Linear regression.** The linear regression script was written in R from scratch, following the principle described in the earlier section 3.8.1. CPU usage was the only feature investigated as it had a clearly seasonal pattern (mostly daily). The minimum of the optimization problem was found with the 'optim' function, from the 'ucminf' package in R. In optim the function 'L-BFGS-B' was used, which is the limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm [10]. The interval was set to 100, meaning the last 100 hours/samples were used to estimate the next one. Thus the weight vector had 100 elements being optimized.

- **Neural network.** A neural network was implemented in R, using the 'caret' package (short for Classification And Regression Training), which utilizes other R packages, for example 'nnet'. The function 'train' was used with the default method 'RF'. As input nodes earlier sample points were used. Only one hidden layer was used as time series analysis is usually not that complex that more layers are required. Maximal number of iterations were set to 300. The number of nodes in the hidden layer was set to 50. The activation function was not set and thus the default function (sigmoid) was used.

- **Trend analysis** This script consisted of two simple methods to find anomalies. The first one aimed to check for abnormal high variance. The extracted feature was the difference between sample points of CPU. By calculating the moving mean value of this new feature vector a measure of the variance was received. The moving mean value vector included 500 samples (with a sampling interval of one minute), i.e. values going slightly over 8 hours back in time.

  The other method aimed to catch abnormal changes in the CPU usage. Having a high variance in the CPU usage, single samples of high CPU usage occur on a regular basis without indicating errors. This method however focused on finding drastic changes in the short-term trend. Three trend lines were created of the original feature, CPU. Two long-term trend lines, consisting of a moving mean value of 3000 samples. One trend line was added by 10 and the other subtracted by 10. In this way there were two long-term trend lines lying above and below the mean value. In between those two long-term trend lines a short-term trend line was placed, using a weighted moving average consisting of only 300 samples. If this middle line brakes through the bottom or top long-term trend lines, the latest sample points are flagged as anomalies.

3.9 Evaluation methods

Different anomaly detection methods have been described in this section. After testing them on a testing interval an important aspect is to evaluate their capability. Comparison and evaluation of methods is preferably accomplished by rating/scoring them according to certain predefined criteria. The scoring procedure in this thesis was performed in two different ways, described below.

After each method, all the dates and times of their flagged anomalies were saved and put into an own written simple script. The script compared the time points of the detected anomalies/errors with the time points of the reported errors from the event log. There were two types of reported errors from the event log, system errors and application errors (these are described further in section 4.1). The score that the methods received was based on the matching level between detected and reported errors. More specifically, the closer in time a detected anomaly was to a reported error the higher score to the method. Every detected anomaly point was examined by the script, adding a scoring point of between 0-15 depending on distance to an application error and 0-40 based on the distance to a system error. Points more than 30 minutes away from a reported error received 0 scoring points. The exact scoring approach is shown in table 1 below.

The methods were scored using two different approaches. For simplicity they can be denoted 'Scoring method 1' and 'Scoring method 2'. 'Scoring method 1' was based on the absolute difference in time, meaning an error before or afterwards were both receiving equal number of scoring points. 'Scoring method 2' was based only on detected anomalies lying *before* reported anomalies. Thus an anomaly detected by the method directly after the error was reported received 0 points.

To make the scoring fair all methods were tuned to flag the exact same number of anomalies. The first round the number of anomalies were set to 323, which was the total number of reported anomalies during the test interval. The second round the number of anomalies were set to 14. This equaled one detected anomaly every third day on average.

The scoring methods however could not be used to compare methods with different sampling interval for obvious reasons. As two of the time series analysis methods (linear regression and neural networks) used one hour samples they could not be compared to the rest of the algorithms.

Table 1: Scoring procedure of the methods. The top row, 'Interval' shows the time intervals between a detected anomaly and a reported error. The scoring points received for each interval and type of error is seen below. 'Scoring method 1' used the absolute value of the time interval when scoring. 'Scoring method 2' scored methods according to the table only if the anomaly was detected before the reported error.

Interval (minutes)	0 - 5	6 - 10	11 - 15	16 - 20	21 - 30
Scoring points from application error	10	4	3	2	1
Scoring points from system error	50	30	15	8	4

4 Data collection, exploration and cleaning

This section describes the gathering and exploration of the performance data. The data was extracted manually via VMware, in CSV-format. To get clean useful data, without text, Excel was used to split the text into columns where the actual numbers of performance data could be extracted. The extracted performance data were CPU, memory usage, memory overhead, network usage, disk usage and disk highest latency. The sampling rate was 1/minute. The data in VMware with this sampling rate is only available for 5 days, after that a mean value of the performance data is taken and the rate decreases. Thus the performance data of 5-days periods were extracted at a time. A description of the performance data extracted in this project is listed below, taken from VMware's website:
https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.bsa.doc_40/vc_admin_guide/performance_metrics/r_memory_counters.html

- CPU. CPU usage as percentage during that interval. Unit: %
- Memory Usage. Memory usage as a percentage of total configured or available memory. Unit: %.
- Memory Overhead. Amount of additional machine memory allocated to a virtual machine for overhead. The overhead amount is beyond the reserved amount. Unit: Kilobytes.
- Network Usage. Sum of the data transmitted and received during the collection interval. Unit: Megabits/second.
- Highest Disk Latency. Highest latency value across all disks used by the host. Latency measures the time taken to process a SCSI command issued by the guest OS to the virtual machine. The kernel latency is the time VMkernel takes to process an IO request. The device latency is the time it takes the hardware to handle the request. Unit: Milliseconds.
- Disk Usage. Aggregated disk I/O rate. For hosts, this metric includes the rates for all virtual machines running on the host during the collection interval. Unit: Kilobytes/second.

There were other types of performance data available as well. However they had basically no patterns and were therefore ignored as they did not seem optimal when looking for anomalies. Exploration of the data were done in Octave and Rstudio. Plotting the data, both in time-series and different parameters against each other helped to get a good view of the data. It also gave an insight of the distribution necessary to have when choosing methods and algorithms. Some servers had sample points with performance data of value -1 (meaning nonsensical). Usually there were 1 to 5 sample vectors in a row with all or some of the performance data parameters with values of -1. These were cleared in the beginning of every script, with their indexes marked and stored. The results in this thesis are based on a server with no nonsensical data.

4.1 Event logs

The original plan was to collect data from 7 different virtual production servers with high activity, apply anomaly detection and finally compare the detected anomalies with the errors reported in the servers' event logs. Event logs are the virtual servers' logs showing historical events like error-, warning-, and information messages with the exact time they were registered. However it was only one of the servers investigated that had a reasonable event log that could be compared with. The rest had either too many errors, or too few (basically none). For this reason it was only one server that was investigated, as the results could be evaluated with the event log. This server did not have nonsensical points, with values of -1.

The event log was filtered and the only events extracted were errors. There were two categories of the errors: application errors and system errors. Application errors are application software conflicts or other bugs. System errors are errors in the operating system (OS). Figure 15 shows the schematics of a virtual server architecture to understand the difference between application and system errors.

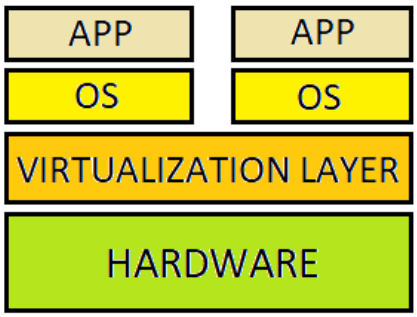

Figure 15: Schematics of the virtual server architecture. The system errors occurs in the operating system layer (OS), and the application errors are located in the application layer (APP) in the top.

The sample points with a reported error within its time range were marked. Thus if an error was reported 08:23:48, the performance data sample point at 08:23 was marked and the time 08:23 was saved. Every error in the event log comes with a message. This message was ignored. However the application errors were separated from the system errors, thus there were only two classes of event errors extracted: application errors and system errors. Application errors occur much more frequent than system errors in the investigated server.

5 Results

This section presents all the results from the methods used in this thesis: KNN, SVM, SOM, memory-CPU usage ratio, K-means and time series analysis. The time series analysis were implemented in three different ways: using linear regression, neural networks and 'trend analysis'. They are all described in section 3.

The results from each method are be presented in individual subsections, showing plots of the performance data. In the plots the reported errors along with the detected anomalies are marked differently. The scoring methods described in 3.9, were used to score all the methods. The scoring points were put together into tables and can be seen under 'Compilation of methods' below. Time series analysis using linear regression and neural network however used another sampling interval and are not included in the compilation. A discussion of the results is presented in section 6.5.

5.1 Compilation of methods

This subsection provides an overview of the results in form of scoring points. The methods were scored using 'Scoring method 1' and 'Scoring method 2' (described under 'Evaluation methods' in the theory section) and the results are presented in three tables below. There is also one last table showing the matching rate of the methods to each and every reported system error. This gives an overview of the number of system errors that were detected by each method. Note that the tables below only show the methods using a sampling interval of once per minute, as the scoring procedure could not compare methods of different sampling intervals. Thus time series analysis with linear regression and neural networks are not in the tables.

All the methods presented in this subsection were run on the same testing interval. The investigated interval was 6 weeks long, from 2019-04-10 to 2019-05-22, consisting of 60281 sample points in total.

Table 2 shows the scoring points from all methods using 'scoring method 1', meaning scoring based on detected errors both before and after a reported error. The number of detected anomalies were predetermined to 323, the same amount as the number of reported errors during the test interval. Table 3 also shows the scoring points using 'scoring method 1', but with only 14 detected anomalies. Table 6 shows the scoring points of the methods using 'scoring method 2', i.e. scoring based only on detected errors occurring before a reported error. The number of anomalies were set to 14. 'Scoring method 2' was not run in the case when having 323 detected anomalies as it was not as relevant as the case with 14 anomalies. If implementing an anomaly detection algorithm on virtual servers it should not report too many errors. As 14 detected anomalies is more reasonable the comparison between detected errors before versus after reported errors was made only in that case.

Application error score is based on the matching accuracy between application errors and detected errors. Similarly the system score is based on system errors. "Random numbers" consists of random time points in the testing interval to examine the scoring points received from guessing anomalies. Three sets of random numbers were generated and a mean value is presented in the tables. The number of random time points were the same as the detected anomalies in every table, i.e. 323 and 14.

Table 5 shows which of the reported system errors that were detected and by which method. Every method was set to flag 14 anomalies in total. M-C-R stands for Memory CPU usage Ratio and detected 6 out of 9 system errors within a 30 minute time range. However, it only detected 4 system errors in a 15 minute time range and all where detected 3-5 minutes after the error was reported. KNN, K-means and SVM detected 3 out of 9 errors, flagging them before the system error was reported at the exact same times.

Table 2: Scoring points of the methods, received from 'scoring method 1'. The score is based on 323 detected anomalies. The higher score, the better.

	Application error score	**System error score**	**Total score**
K-nearest neighbor	720	1646	2366
Self-organizing maps	810	1407	2217
K-means clustering	472	1242	1714
Memory-CPU usage ratio	609	956	1565
Trend analysis	106	1050	1156
Support-vector machine	465	603	1068
Random numbers	207	62	269

Table 3: Scoring points of the methods, received from 'scoring method 1'. The score is based on 14 detected anomalies. The higher score, the better.

	Application error score	**System error score**	**Total score**
Memory-CPU usage ratio	131	330	461
Support-vector machine	178	183	361
K-means clustering	72	65	137
K-nearest neighbor	70	65	135
Trend analysis	30	0	30
Self-organizing map	3	0	3
Random numbers	8	0	8

Table 4: Scoring points of the methods, received from 'scoring method 2'. The score is based on 14 detected anomalies. The higher score, the better.

	Application error score	**System error score**	**Total score**
Support-vector machine	80	65	145
K-means clustering	39	65	104
K-nearest neighbor	37	65	102
Memory-CPU usage ratio	44	0	44
Self-organizing map	3	0	3
Trend analysis	0	0	0
Random numbers	1	0	1

Table 5: This table shows which reported system errors that were detected by which method, having the number of detected anomalies pre set to 14. The first column shows all reported system errors during the test period (9 in total) and the time they were reported. The middle column shows a list of methods that flagged at least one anomaly within a 30 minutes time range from the time the system error was reported (maximum 30 minutes before or after the reported error). The column to the right follows the same principle but with 15 minutes time range. The last column has numbers in parenthesis after the method names. This is the time difference in minutes of their best matching detected anomaly (lying closest to the reported anomaly). Plus sign means the detected error was flagged after the reported error, minus sign means the error was detected before it was reported. M-C-R is an abbreviation for the method 'Memory-CPU usage ratio'.

Time of system error	**30 minutes**	**15 minutes**
04-14-09:59	M-C-R	M-C-R(+3)
04-14-14:00	-	-
04-14-18:00	-	-
04-14-19:20	M-C-R	M-C-R(+5)
04-29-07:34	M-C-R	M-C-R(+5)
05-04-17:10	M-C-R	M-C-R(+5)
05-19-09:13	KNN, Kmeans, SVM	KNN(-3), Kmeans(-3), SVM(-3)
05-20-15:54	KNN, Kmeans, SVM, M-C-R	KNN(-11), Kmeans(-11), SVM(-11)
05-20-15:57	KNN, Kmeans, SVM, M-C-R	KNN(-14), Kmeans(-14), SVM(-14)

5.2 K-Nearest Neighbor

The results from the KNN method are presented below. The figures show plots of the six original features, i.e. the performance data extracted from the virtual server during the test interval (2019-04-10 to 2019-05-22). Having a sampling interval of one minute this is equivalent to 60281 sample points in total.

In each plot there are results from two runs of KNN, one with 323 detected anomalies and one with 14. The detected anomalies from the first run (323 anomalies) are marked as small red dots. The detected anomalies from the second run are slightly bigger and marked as red squares with a black thin outline.

In the first figure below (figure 16) the features are plotted against time. In the second one (figure 17) they are plotted against CPU usage.

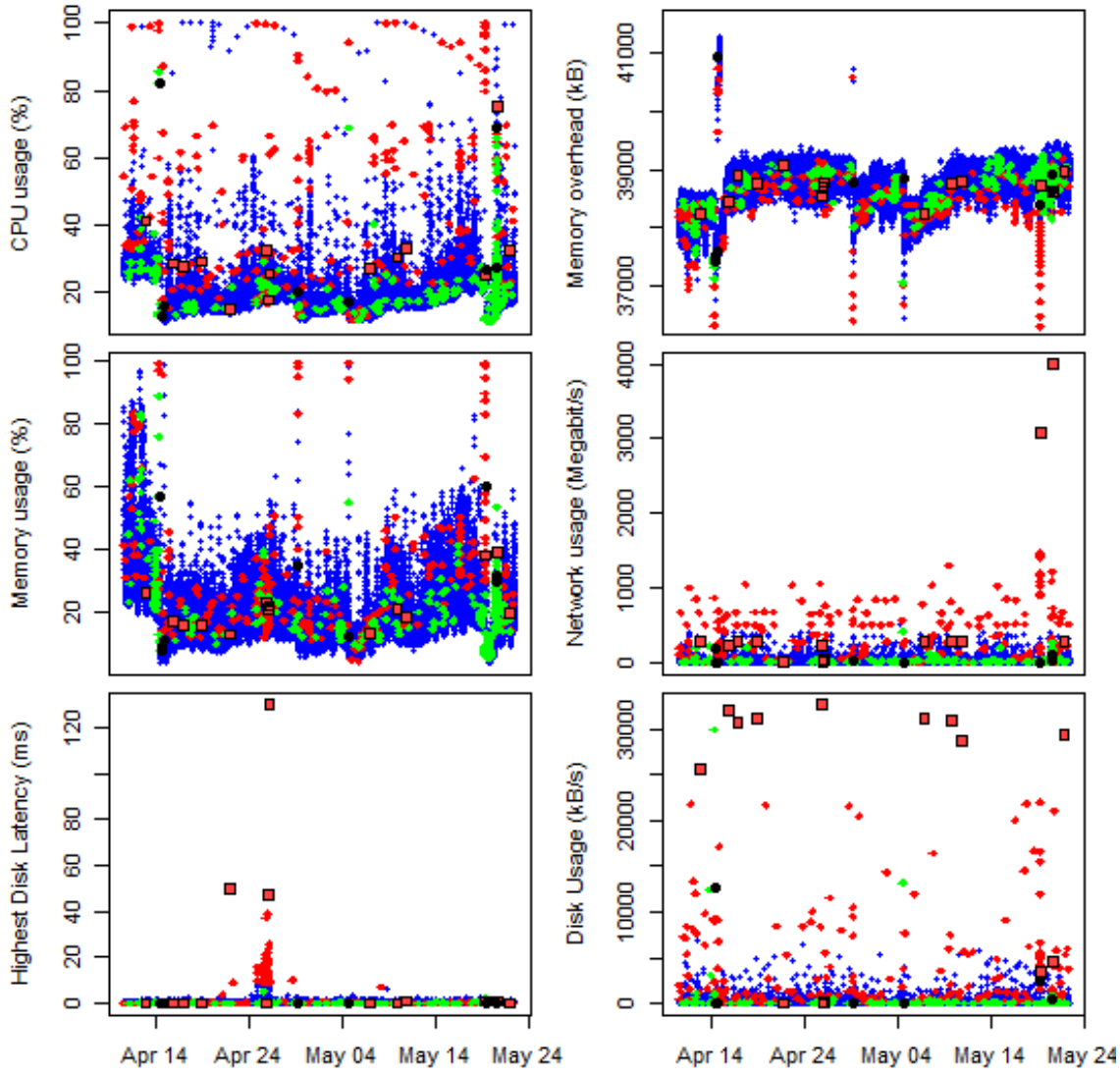

Figure 16: Results from the KNN method. Performance data plotted in time series, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or 2 respectively.

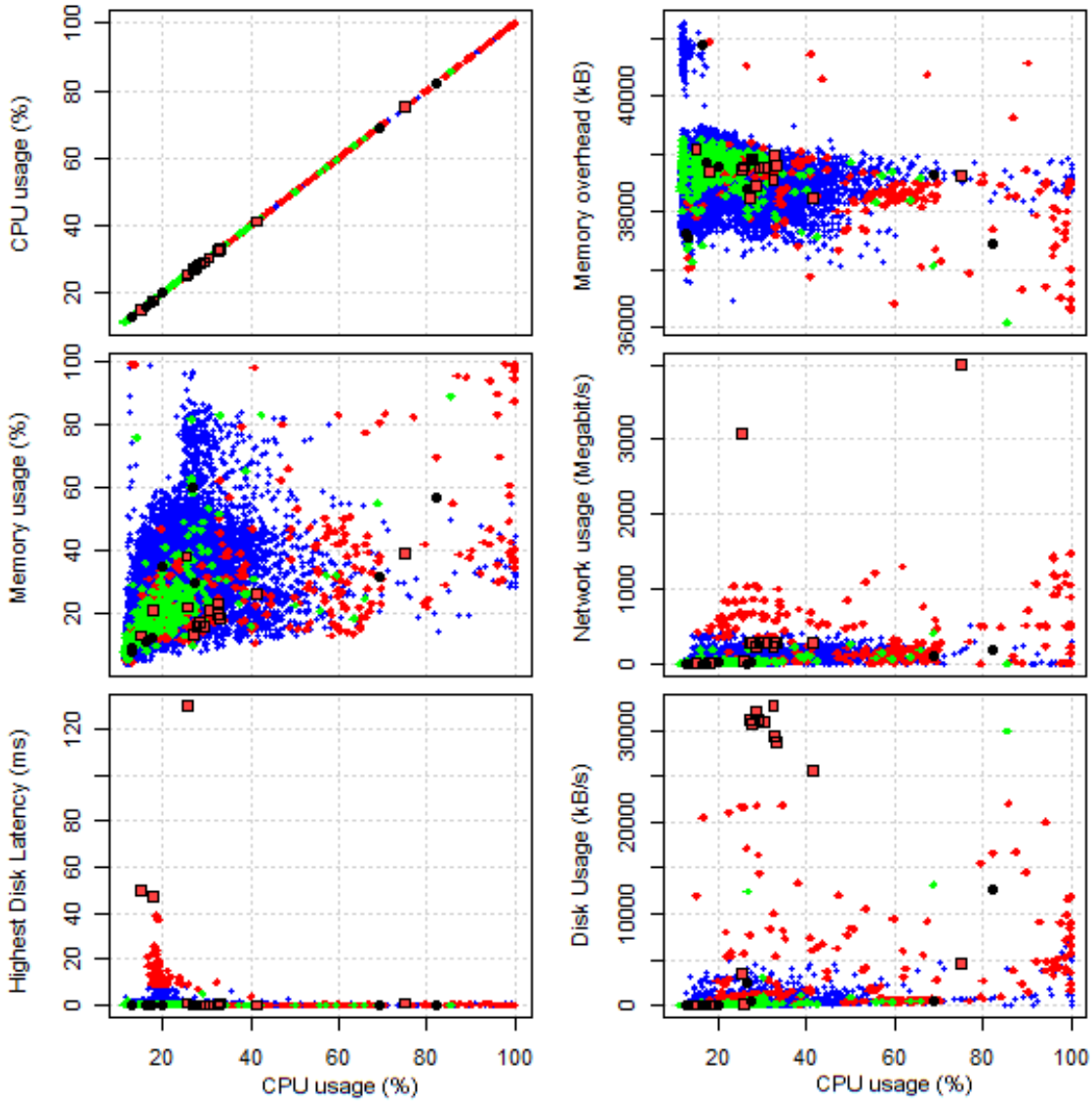

Figure 17: Results from the KNN method. Performance data plotted against CPU usage, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

5.3 Support-vector machines

The results from the SVM are presented below. The figures show plots of the six original features, i.e. the performance data extracted from the virtual server during the test interval (2019-04-10 to 2019-05-22). Having a sampling interval of one minute this is equivalent to 60281 sample points in total.

In each plot there is results from two runs of SVM, one with 323 detected anomalies and one with 14. The detected anomalies from the first run (323 anomalies) are marked as small red dots. The ones from the second run are slightly bigger and marked as red squares with a black thin outline.

In the first figure below (figure 18) the features are plotted against time. In the second one (figure 19) they are plotted against CPU usage.

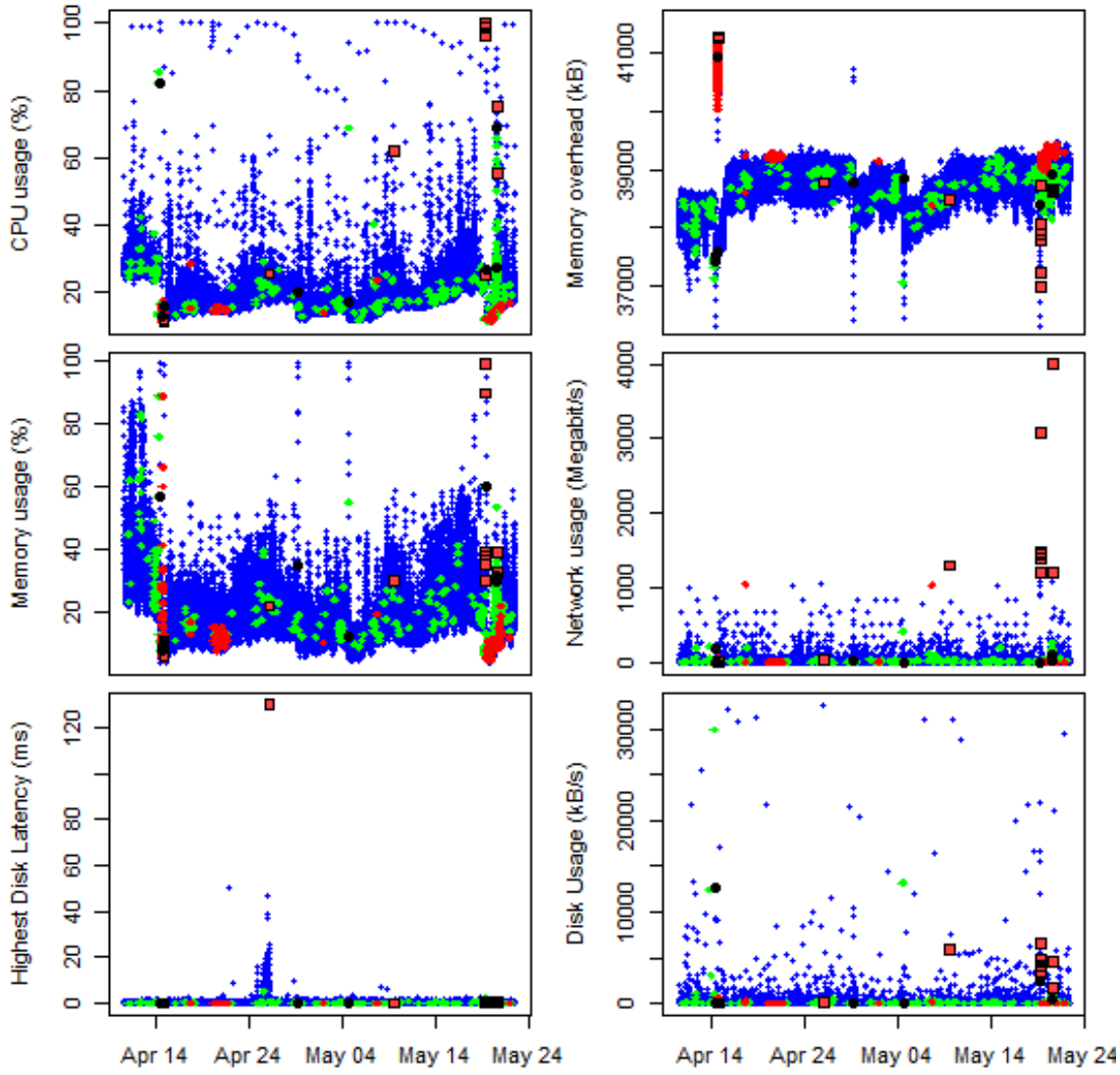

Figure 18: Results from the SVM. Performance data plotted in time series, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

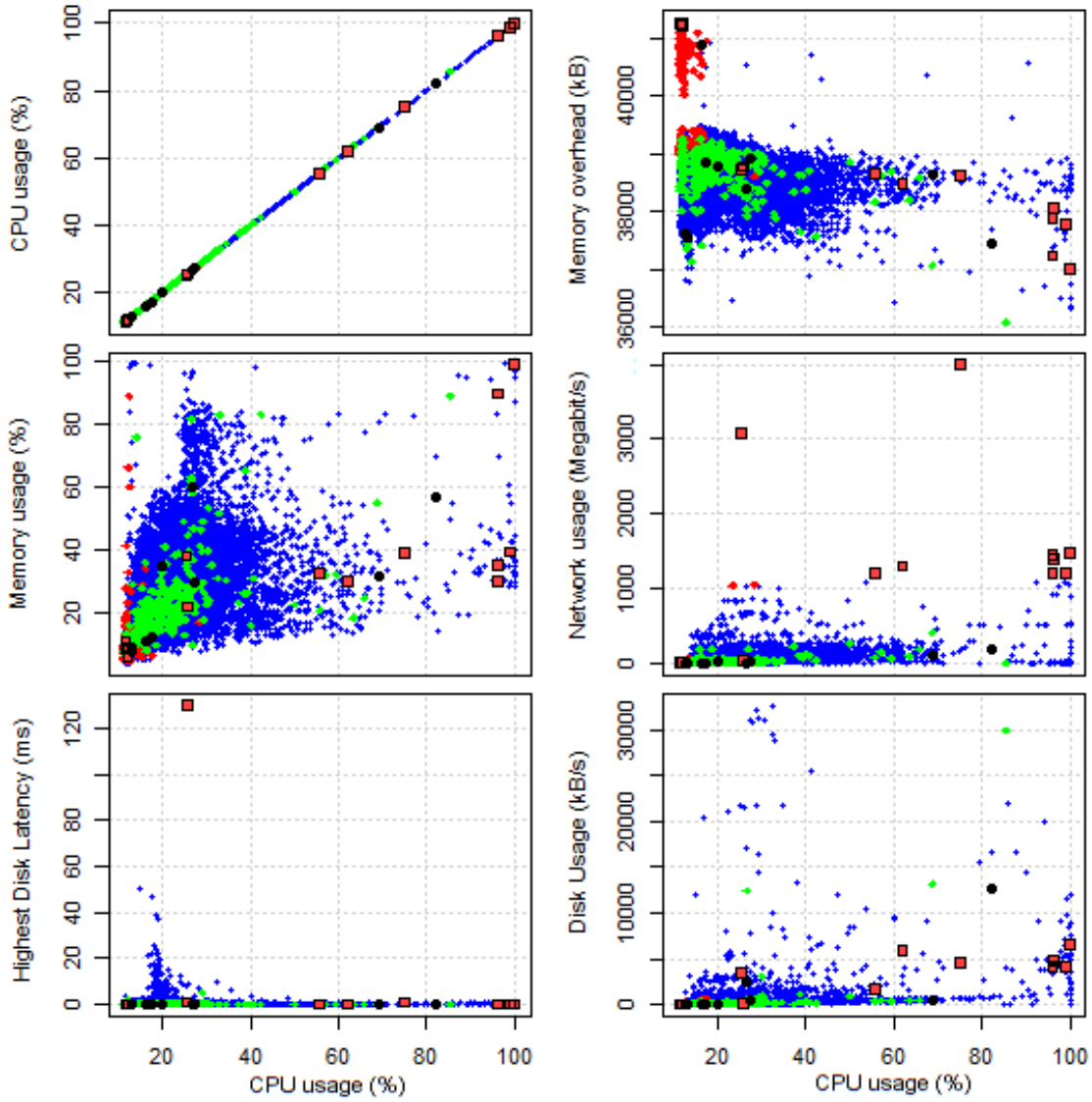

Figure 19: Results from the SVM. Performance data plotted against CPU usage, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

5.4 K-means clustering

The results from the K-means clustering method are presented below. The figures show plots of the six original features, i.e. the performance data extracted from the virtual server during the test interval (2019-04-10 to 2019-05-22). Having a sampling interval of one minute this is equivalent to 60281 sample points in total.

In each plot there are results from two runs of the K-means method, one with 323 detected anomalies and one with 14. The detected anomalies from the first run (323 anomalies) are marked as small red dots. The ones from the second run are slightly bigger and marked as red squares with a black thin outline.

In figure 20 the features are plotted against time, while in figure 21 they are plotted against CPU usage. Figure 22 shows the total within-cluster variance as a function of the number of clusters. Based on this plot the number of clusters were set to 20, as the curve began to flatten out.

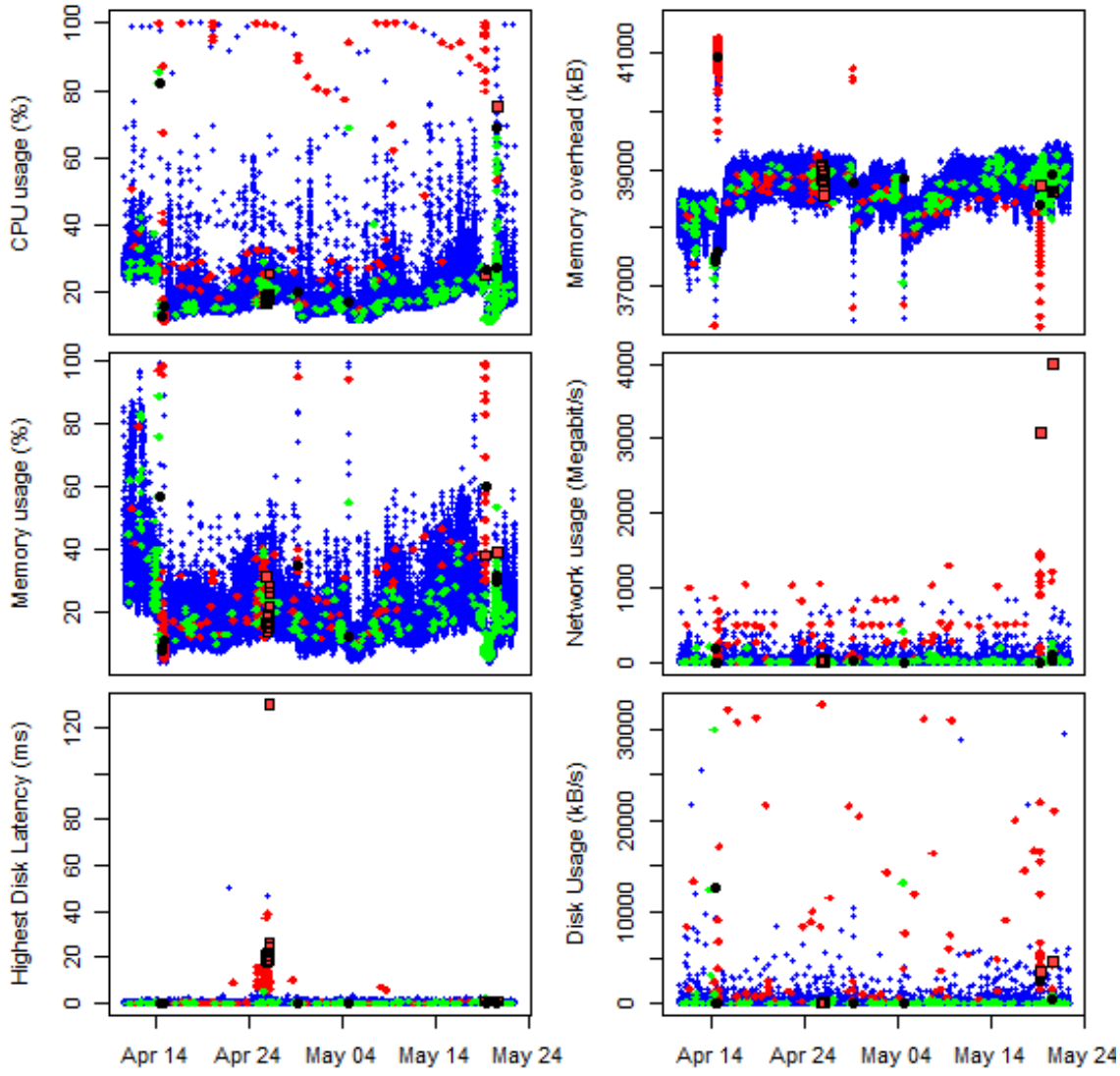

Figure 20: Results from the K-means method. Performance data plotted against time, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

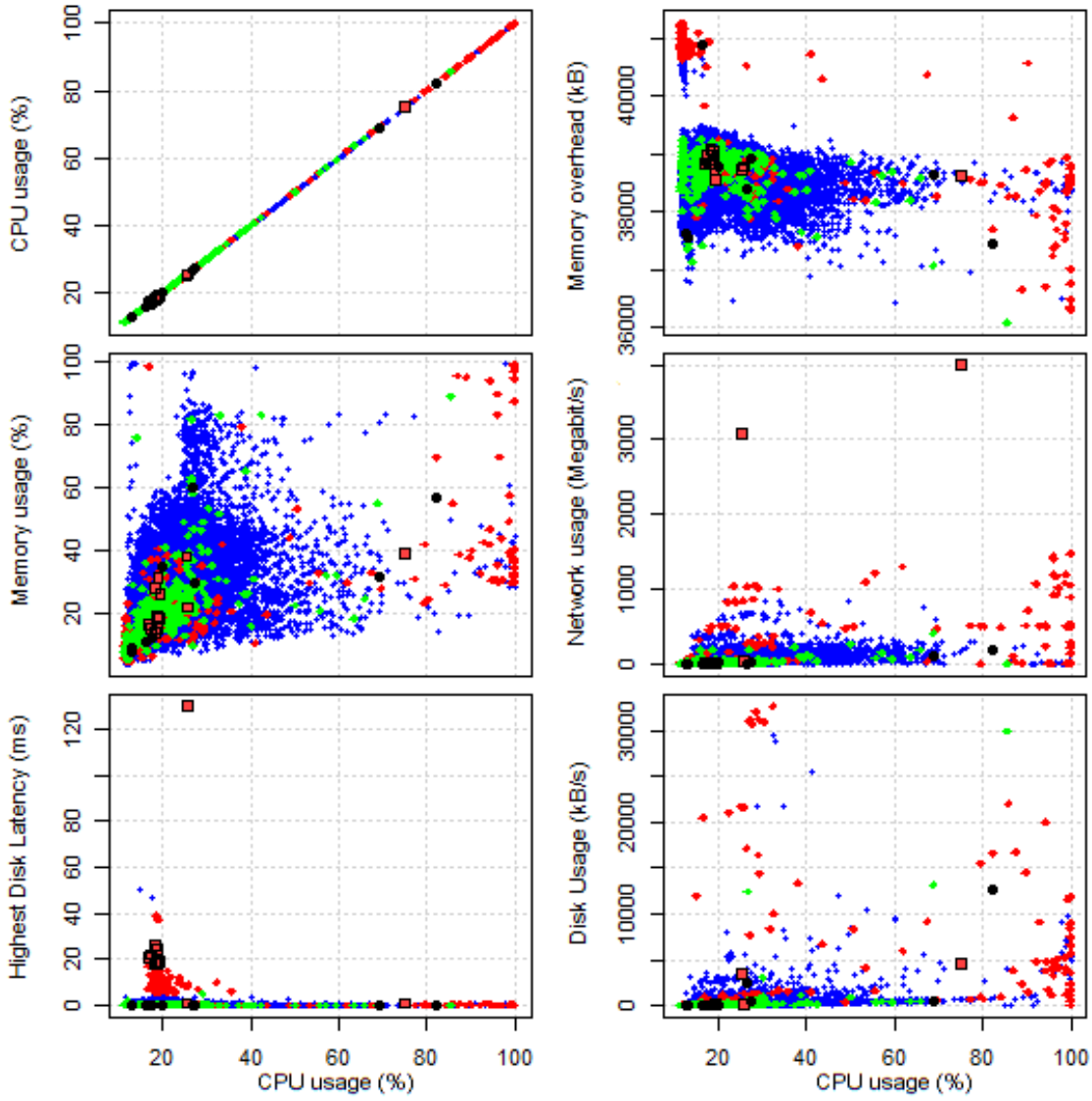

Figure 21: Results from the K-means method. Performance data plotted against CPU usage, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

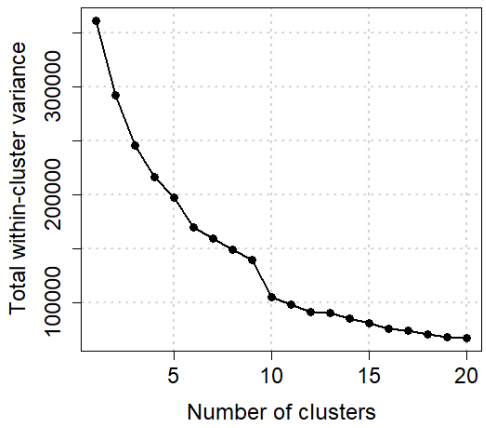

Figure 22: Convergence plot of K-means clustering for choosing the number of clusters. It shows the total within-cluster variance as a function of the number of clusters.

5.5 Self-organizing maps

The results from the SOM are presented below. The first two figures show plots of the six original features, i.e. the performance data extracted from the virtual server during the test interval (2019-04-10 to 2019-05-22). Having a sampling interval of one minute this is equivalent to 60281 sample points in total.

There are results from two runs of SOM, one with 323 detected anomalies and one with 14. The detected anomalies from the first run (323 anomalies) are marked as small red dots. The ones from the second run are slightly bigger and marked as red squares with a black thin outline. The same self-organizing map were used on both runs but using different numbers of 'anomaly nodes'.

In the first figure below (figure 23) the features are plotted against time. In the second one (figure 24) they are plotted against CPU usage. Figure 25 shows a standard representation of a self-organizing map with 100 nodes, based on the training set. The pies in the map shows the properties of the performance data in every cluster. The radius of a wedge corresponds to the magnitude of that feature. This map constituted the basis for the anomaly detection of the testing set. After creating a self-organizing map from training data, new samples (from the test set) were paired to the most relevant nodes. All the marked nodes were used as 'anomalous nodes', thus data points from testing data, referred to these nodes where classified as anomalous. All marked grey nodes were used for the first run, giving 323 anomalies. The three nodes to the top-right, coloured in light grey were the only nodes used in the second run, resulting in 14 anomalies. The right plot in figure 26 shows the mean distance to the 'best matching unit' as a function of number of iterations. It confirms that the SOM converged within 150 iterations. The left plot shows the within-node distance between neighbors in every node which was used to choose 'anomaly nodes'.

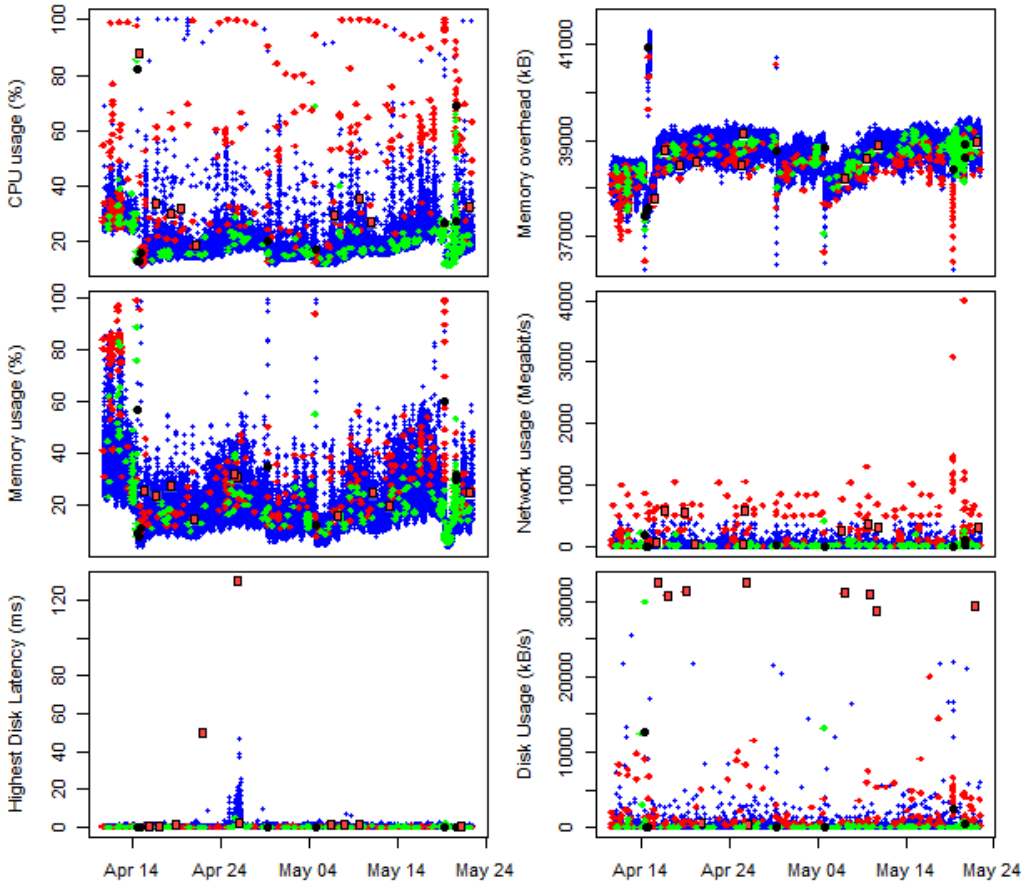

Figure 23: Results from the SOM method. Performance data plotted against time, normal data in blue, application errors in green, system errors in black. The sample points detected as anomalous by the K-means method method are marked in red.

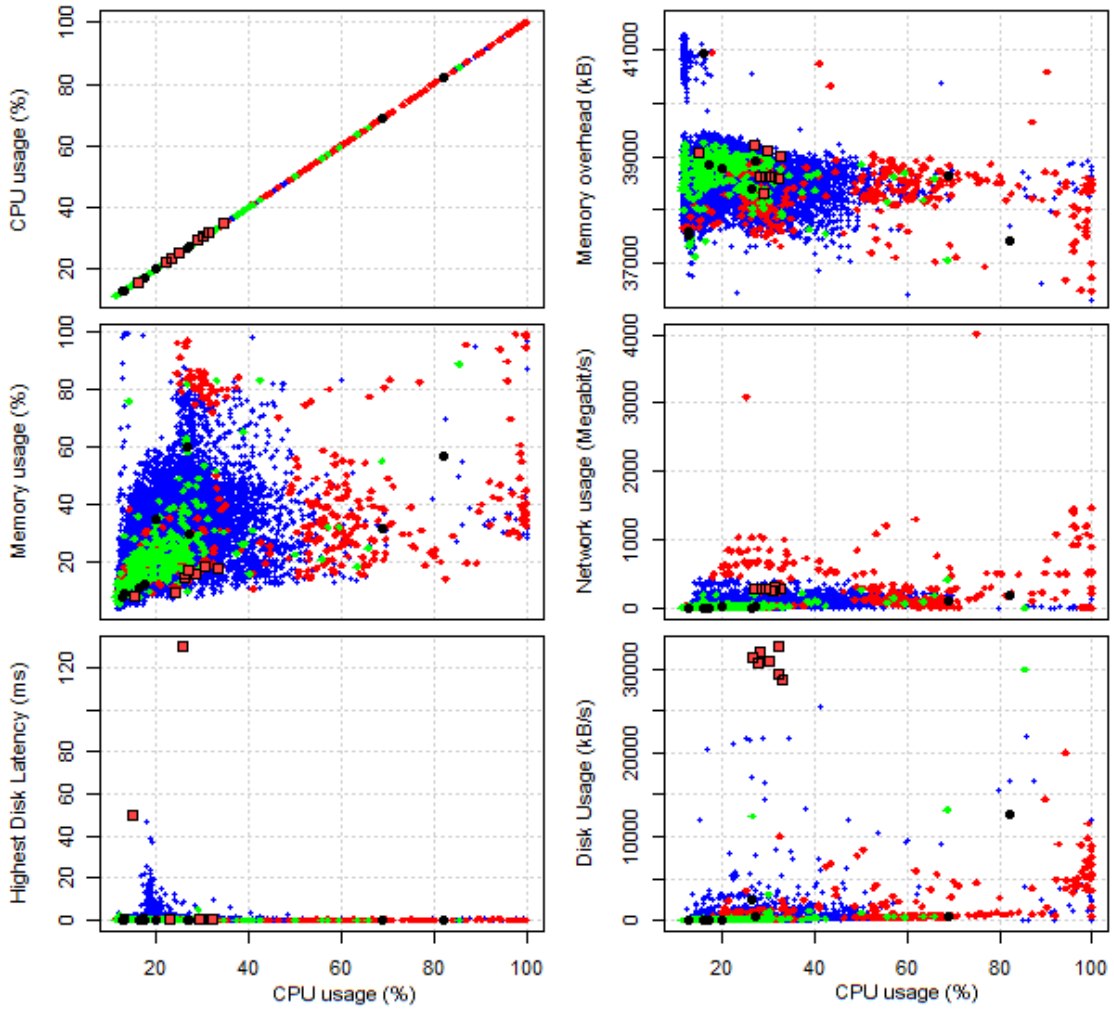

Figure 24: Results from the SOM method. Performance data plotted against CPU usage, normal data in blue, application errors in green, system errors in black. The sample points detected as anomalous by the K-means method method are marked in red.

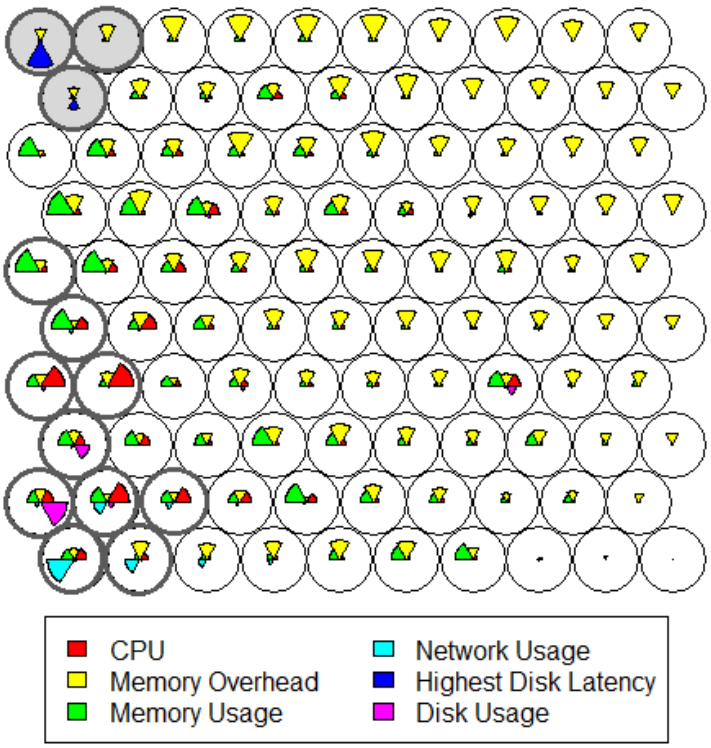

Figure 25: Plot of the self-organizing map, based on the training set and used to apply anomaly detection on the test set. The clusters in which the data have been grouped are represented as pies, with the radius of the wedges showing the properties of each feature. The marked grey circles were used as 'anomalous nodes' for the 323 anomaly run. The three marked nodes in the top left corner were the only nodes used for the second run with 14 anomalies.

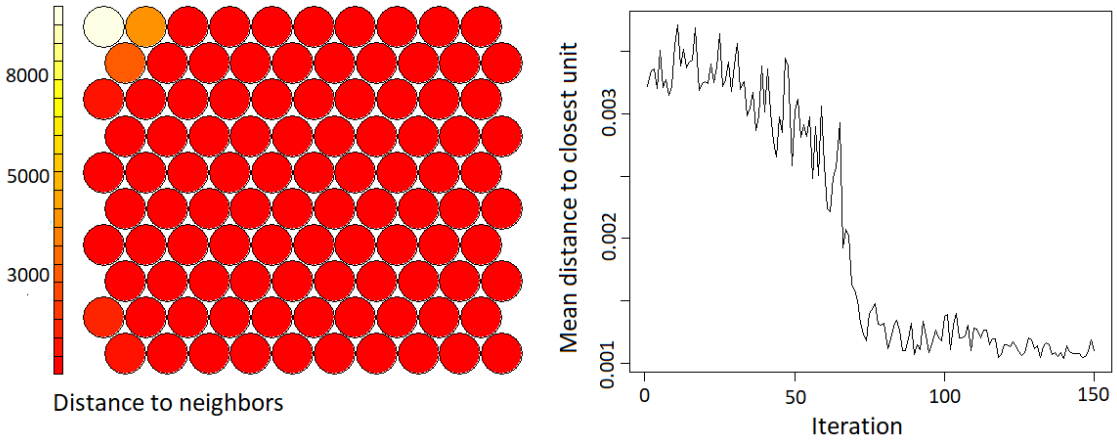

Figure 26: Property plots of the self-organizing map. The left plot shows the distance between neighbors in every node on the map. The top left nodes have the highest within-node variance. The right plot shows the convergence of the self-organizing map. The mean distance to closest unit decreases with the number of iterations and stabilizes at around 80.

5.6 Memory-CPU usage ratio

The results from the 'memory-CPU usage ratio' method are presented below. This method uses only one feature, which is the logarithm of the quotient between memory and CPU usage (memory usage divided by CPU usage). The limits, meaning the values of the feature separating the normal values from the abnormal ones, are shown as grey lines. Figure 27 shows the feature plotted against time with detected and reported anomalies marked. There are two runs presented in the figure. The top plot shows the results of having the limits closer to the mean value detecting 323 anomalies during the testing interval. The bottom plot has higher limits, detecting 14 anomalies. Figure 28 shows the same results as the bottom plot in figure 27 (same 14 detected anomalies) but plotted with two features, CPU usage/memory usage and memory usage/CPU and without taking the logarithm. The corresponding limits without using logarithmic numbers are 4.2 and 6 for the CPU/memory feature and memory/CPU feature respectively, shown in grey.

Figure 29 shows the distributions of the whole testing set in form of histograms. The quotients memory usage/CPU usage and CPU usage/memory usage are not perfectly normally distributed. The x-axis is limited to 3.5 on the two top histograms which is why the biggest values are not seen there, but only in figure 28. In figure 29 it is also shown that taking the logarithm results in a mean value close to zero and a distribution of the samples that is almost normally distributed.

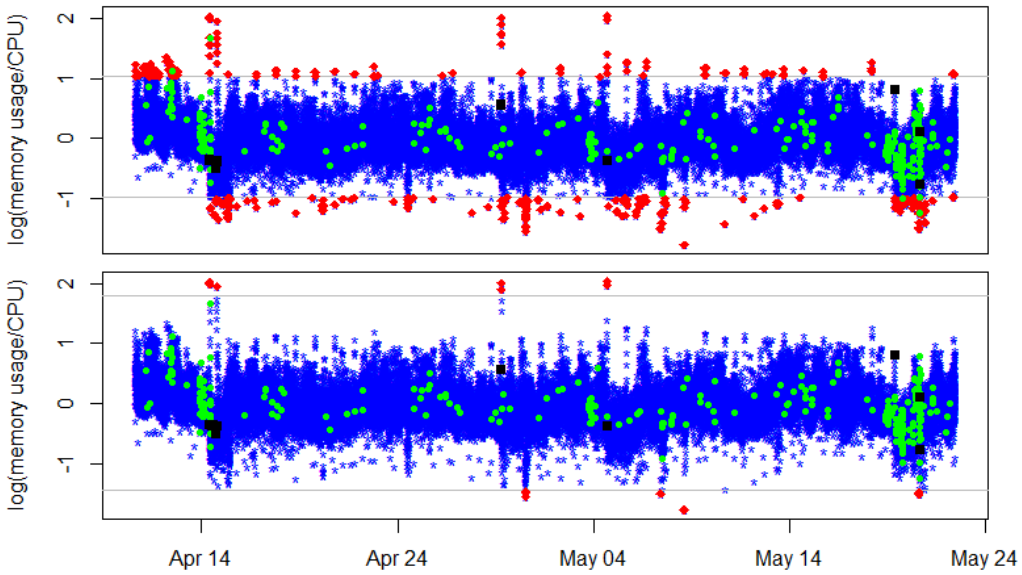

Figure 27: Results from two runs using the memory-CPU usage ratio method. The top plot shows the first run, detecting 323 anomalies. The bottom plot shows the second run, with 14 detected anomalies. Both plots show the only feature used in this method, the logarithm of memory usage/CPU. The normal data in blue, application errors in green, system errors in black and detected anomalies in red. The grey lines show the values of the ratios separating the normal and abnormal data. In the top plot they are 1.0296 and -0.9795 and in the bottom plot -1.44 and 1.8.

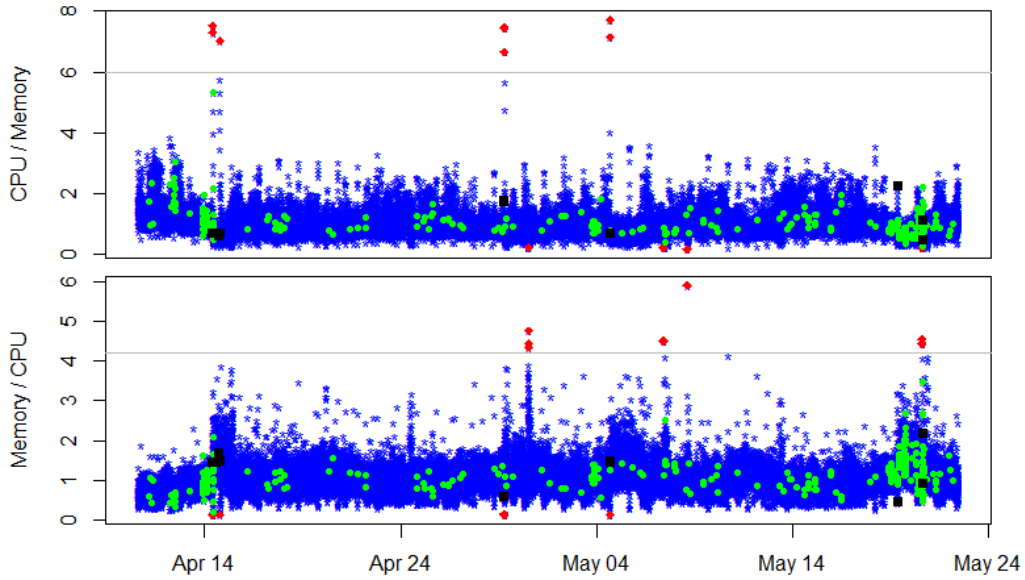

Figure 28: Results from the memory-CPU usage ratio method, from the second run with 14 detected anomalies. This figure show both ratios between CPU and memory usage and without taking the logarithm. The normal data are shown in blue, application errors in green, system errors in black and detected anomalies in red. The grey lines show the values of the corresponding ratios separating the normal and abnormal data, giving the same result as in figure 27.

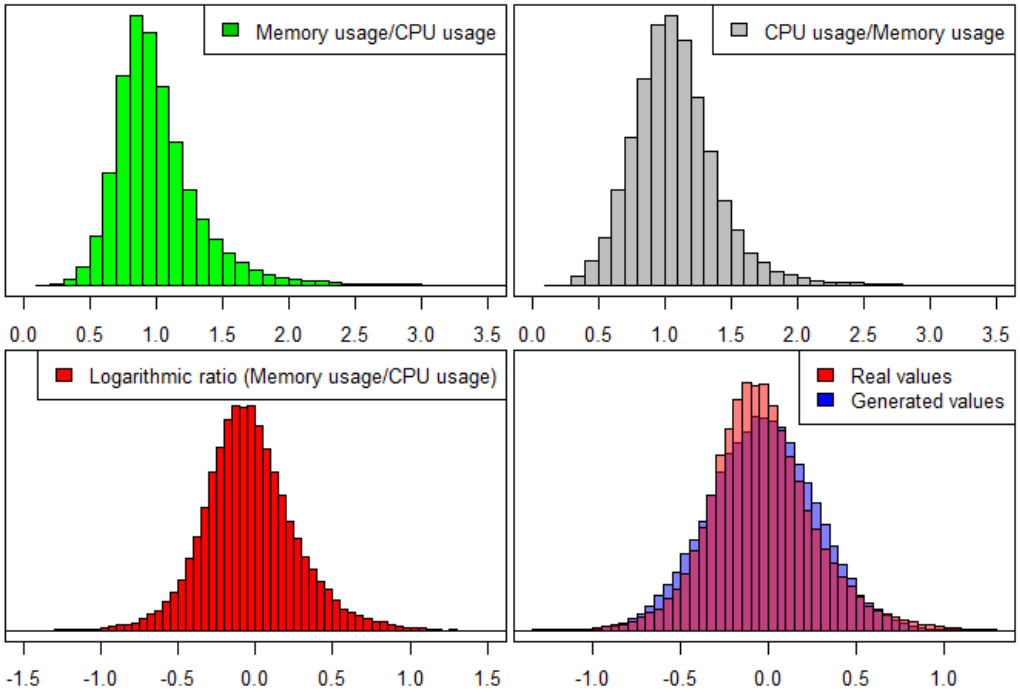

Figure 29: Histograms showing the distributions of different ratios. The histogram in green shows the distribution of memory usage/CPU usage. The histogram in grey shows the distribution of the opposite feature, CPU usage/memory usage. The red histogram to the left shows the distribution memory usage/CPU usage after taking the logarithm of the feature vector. The lower histograms to the right also shows the logarithm of memory usage/CPU usage again (in red) but together with normally distributed numbers with the same mean value and variance.

5.7 Trend analysis

The results from the 'trend analysis' are presented below. The only feature used was CPU usage and difference vector of CPU usage. The top plot shows the trend analysis of CPU usage. The normal samples are shown in blue, application errors in green, system errors in black and detected anomalies in red. The weighted moving average in black is a short-term trend line of CPU usage, meaning it shifts fast when the CPU varies. The grey long-term lines of CPU lying above and below moves slower. When the black line breaks through the bottom or top grey line those samples are flagged as anomalies. This happened at four occasions, during the time interval 2019-04-10 to 2019-05-22. During those occasions many sample points were reported as anomalous as the short-term trend might be outside the allowed interval for some time. The number of anomalies were regulated by reducing the total number of anomalies to only the first ones after a break-through.

The very first part was not used though as the trend lines had to get in position first. There are detected anomalies within the grey trend lines too. Since the samples are flagged anomalous when the trend lines are crossed it does not implicate the individual sample points must be outside the interval. The same script and trend lines were used both when detecting 14 and 323 anomalies. The only difference were the number of anomalies flagged after the trend lines were crossed. The bottom plot shows the trend line of the difference vector, meaning a form of variance. The trend line did not break through the limited lines and thus no flagged anomalies.

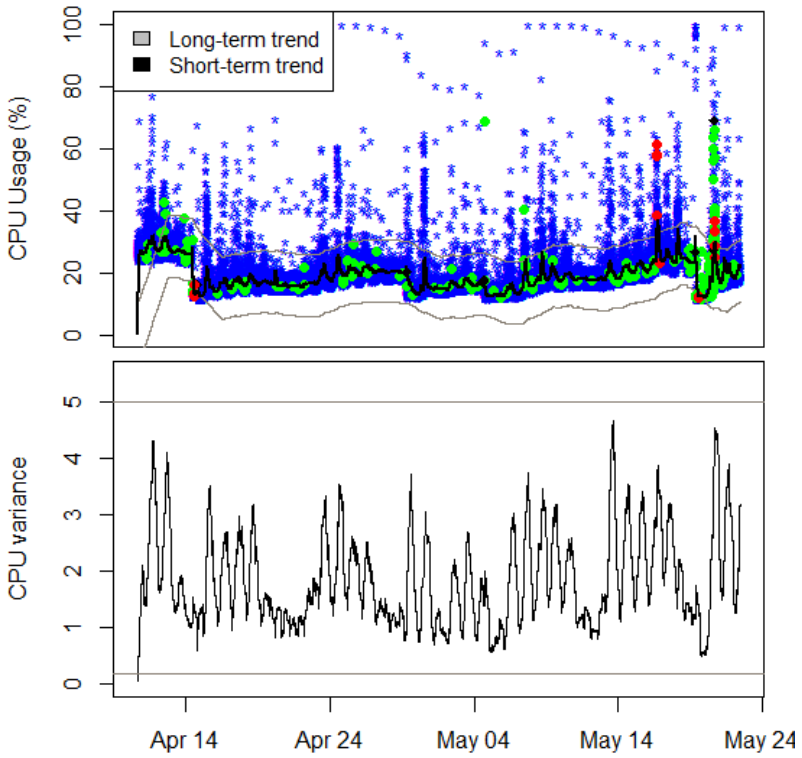

Figure 30: Results from trend analysis, using CPU usage. The top plot shows the trend analysis of CPU, with normal samples in blue, application errors in green, system errors in black and detected anomaly samples in red. The black line is a short-term trend and the grey lines are long term trends. The bottom plot shows the trend line of the difference vector. In both cases a transient period is taken into account at the start of the time series before reporting anomalies. Therefore there are no flagged anomalies in the beginning before the variance and short-term trend of CPU is within the allowed intervals.

5.8 Time series analysis using linear regression

The results from time series analysis using linear regression are presented below, using a sampling interval of one hour. The investigated time interval reached from 2019-05-09 to 2019-05-22, meaning the time period for the testing interval was significantly shorter compared to the earlier methods. The reason was lack of continuous data. In contrast to the other methods, the training data must be continuous. The bottom plot in figure 31 compares the real CPU usage with the predicted/estimated CPU usage (computed with earlier sample points). The top plot shows the absolute difference between the real and predicted CPU values and constitutes the feature used for anomaly detection.

Setting the maximum limit of this feature to 15, the three highest peaks in the plot are flagged anomalous. These three peaks has four sample points exceeding the limit (two in the middle peak). Two of these three peaks contain system errors. Thus on this short time interval it seems to work fairly well as it detects both system errors but with one false positive. However the test interval should be longer.

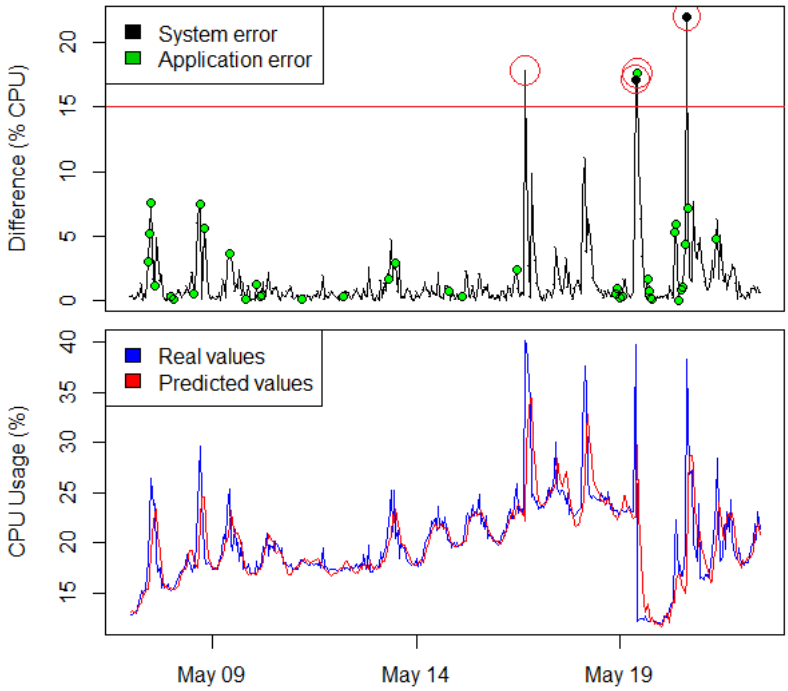

Figure 31: Results from time series analysis using linear regression. The bottom plot shows the real CPU usage in blue and the predicted/estimated CPU values in red. The plot above shows the difference (in absolute value) of the two curves with the system errors and application errors marked in black and green respectively. The red line shows the maximum allowed value and thus separates normal data from abnormal data. The red circles mark sample points exceeding the limit, the detected anomalies.

5.9 Time series analysis using a neural network

The results from time series analysis using a neural network are presented below. The investigated time interval reached from 2019-05-09 to 2019-05-22 with a sampling interval of one hour, just like in the linear regression method. The bottom plot in figure 32 compares the real CPU usage with the predicted/estimated CPU usage (computed with earlier values). The top plot shows the absolute difference between the real and predicted CPU values and constitutes the feature used for anomaly detection.

Setting the maximum limit of this feature to 15, the three highest peaks in the plot are flagged anomalous, just like in the linear regression case. Two of these three peaks contain system errors. Thus on this short time interval it seems to work fairly well as it detects both system errors but with one false positive. However the test interval should be longer.

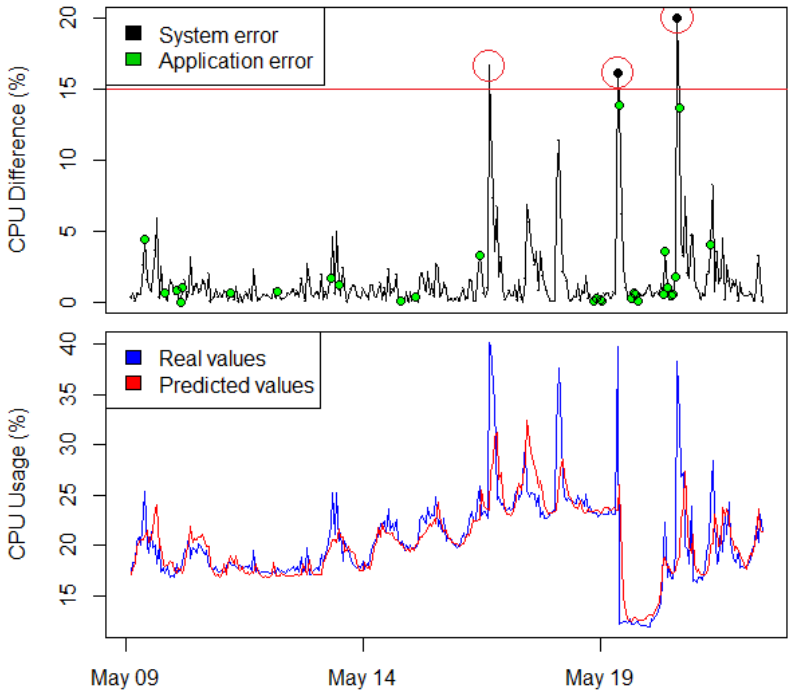

Figure 32: Results from time series analysis using a neural network. The bottom plot shows the real CPU usage in blue and the predicted/estimated CPU values in red. The plot above shows the difference (in absolute value) of the two curves, with the system errors and application errors marked in black and green respectively. The red line shows the maximum allowed value and thus separates normal data from abnormal data. The red circles mark sample points exceeding the limit, the detected anomalies.

6 Discussion

This discussion will be divided into different subsections. Section 6.1 discusses the possibility of a malfunctioning server that is not reported in the event log. The second part (section 6.2) elaborates on why some methods introduced in 'Background on anomaly detection' were not optimal for this task. Section 6.3 discuss the advantages and disadvantages with longer/shorter sampling times. Some methods were implemented in a supervised mode, trained with labeled data to apply anomaly detection. This did not work out but is discussed in section 6.4. Lastly the subsection 6.5 discusses the methods' capabilities.

In general it can be said that application errors were hard to detect. Sometimes many application errors were reported during a short period of time, often in connection with one or several system errors. During these periods all methods detected anomalies. Occasional application errors on the other hand were harder as they seemed to have minimal or no effect at all on the performance data. System errors on the other hand did affect the performance data and were therefore possible to detect.

6.1 Non reported errors

In this thesis the comparison of the methods is mainly based on the matching rate between detected anomalies and reported errors. However not all errors are reported but the algorithms could probably detect malfunctioning servers which are not reporting errors. Evaluating the methods' abilities to detect reported errors is way to evaluate their overall capabilities on detecting errors.

As was mentioned in the 'Scope', the focus in this thesis lies in anomaly detection and not the underlying causes to the errors in a server. However two examples of problems will be very briefly explained here, *infinite loops* and *memory leak*. Errors like these might not be reported in the event log but can inhibit a server.

Infinite loops describes the situation of an instruction sequence looping endlessly. The reason can be a computer program having a terminating condition which is never met or causing the loop to restart before ending.

A memory leak occurs when memory is incorrectly allocated. Usually memory is allocated temporarily to perform a task and afterwards it is released again. During a memory leak however the memory is not released but continues to be allocated and therefore reduces the performance of the server/computer as the available memory decreases.

Both these problems cause increased response time and poor performance by the system. An unchecked memory leak can eventually cause the system to crash. Errors like these can be hard to notice as they slowly reduce the performance and may not be reported in the event log. They can however be detected with anomaly detection. Infinite loops makes the CPU load increase and memory leak causes the memory usage to increase. However investigating the values of CPU and memory usage, flagging the highest values as anomalies is misleading. High values of CPU load and memory usage may just indicate that the server is performing some demanding task. Looking at the ratio between them is a better approach as memory leak and infinite loops only makes one of the two variables increasing, not both of them as in most normal cases. The method looking at memory-CPU usage ratio is accordingly perfect for this task.

It is hard to evaluate the methods capabilities on detecting non reported errors like memory leaks and infinite loops as they can not be validated. However their capabilities on detecting reported errors could be a measure on their ability to detect errors in general, including non reported errors.

6.2 Methods not used

Not all the methods introduced in the background were used. This section discuss why these methods (namely Multivariate Gaussian distribution, PCA, local outlier factor, and DBSCAN) were not used.

Multivariate Gaussian distribution for anomaly detection was intended to be used with at least the two features, CPU usage and memory usage as they were believed to be linearly correlated and normally distributed. This turned out to be an incorrect assumption. They were not normally distributed and only correlated to some extent (see figure 33). Thus multivariate Gaussian distribution could not be used. Instead non parametric methods were used.

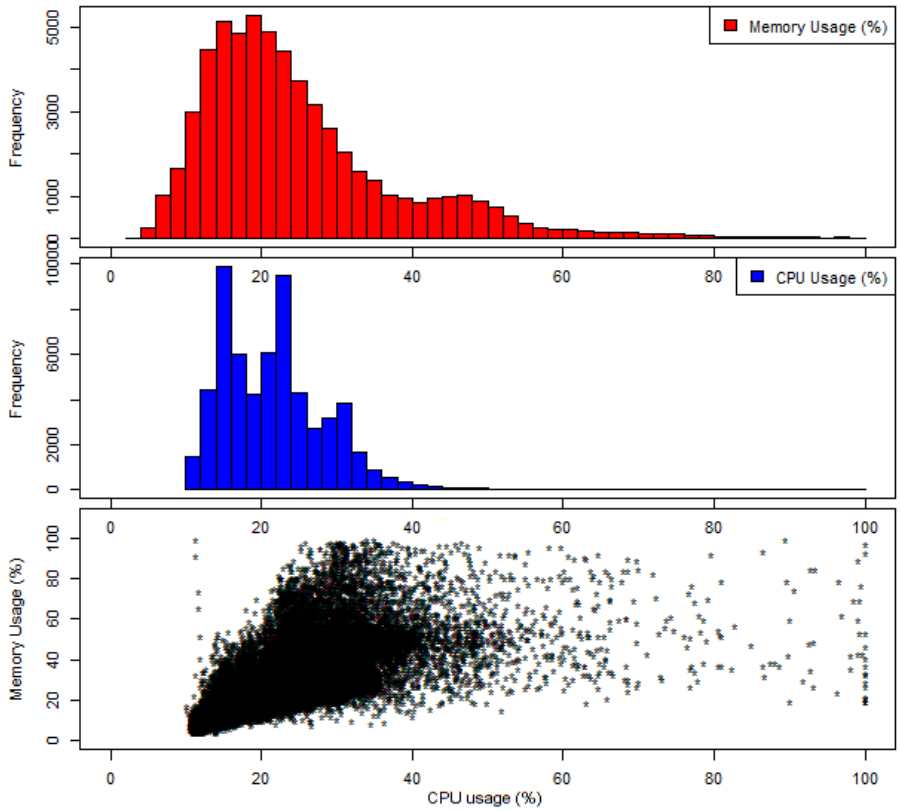

Figure 33: Distribution of CPU usage and memory usage. The top red histogram shows the distribution of memory usage and the blue one the distribution of CPU usage. In the bottom plot memory usage is plotted against CPU usage.

The dimension reduction technique, PCA is mainly used when having many dimensions and was not necessary in this case, having only 6 features. Moreover the PCA also requires normally distributed and correlated data. As this was not the case, PCA was not used in the thesis.

The local density based method, Local outlier factor (LOF) was not optimal in this case. There was not much of a need for local density, as the more simple KNN detected outliers well enough. One main problem with LOF was the very thick concentration of points. This made the LOF algorithm flag many points at the boundary of the high density area, instead of the secluded points. This is shown in figure 34 below. The radius of the red circles are proportional to the anomaly score, i.e. how deviating they are based on local density.

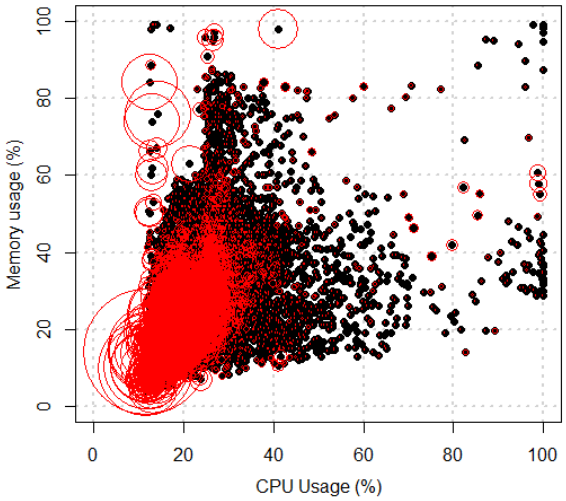

Figure 34: Results from LOF. CPU and memory usage plotted against each other. The radius of the red circles are proportional to the anomaly score, and are largest at the boundary of the high density area.

Density-based spatial clustering of applications with noise (DBSCAN) was not used either. This method seemed relatively successful. However it was slow and the computer could not handle the large amount of data when making predictions. The training data worked well (slightly over 60 000 samples). In this case DBSCAN was run the standard way, going through the dataset creating clusters. However predicting the test set did not work out when having a big dataset. Predicting the test set means to use the training set to predict which (if any) of the clusters every sample vector in the test set belongs to. The maximum size of the testing set it could handle were between 1000-3000 sample points. When predicting smaller sets the results seemed promising. DBSCAN could also be run on the whole set and afterwards investigate and detect outliers, with approved success (see appendix). However this thesis investigates the possibility of continuously checking performance data, meaning real-time anomaly detection. Investigating a set of historical data and analyzing it is thus not an option.

6.3 Sampling frequency

In the beginning of the project the only sampling rate used was 1 sample/minute, which was the highest rate available to extract from VMware. The motive of choosing the highest frequency was partly to get a big set of training data but especially to detect anomalies quickly. The bigger sampling interval the longer time before detecting an error. The two methods applying time series analysis with linear regression and neural network used hourly intervals, which is a drawback. An error occurring at 08:24:23 could potentially be detected the next minute, 08:25:00 using one-minute intervals. Using hourly intervals the error cannot be detected until 09:00:00.

The problem with using a high sample rate however is the noise. Looking at point anomalies, one single outlier may not be of interest if the performance data goes back to normal the minute after. However using a lower sampling frequency, a point anomaly might be a stronger indication that the server is malfunctioning and should be checked.

6.4 Attempt on supervised classification

All the classification methods in this thesis have been based on unsupervised learning. It was tried to implement anomaly detection in a supervised mode as well. This was tried using SVM and SOM. Supervised learning needs labeled training data. Consequently the data were labeled, marking the sample points in the training set as 0 or 1 depending if they were normal/abnormal. The abnormal samples were based on the event log history from the server, using all reported errors both application and system errors.

Different approaches were tried. One way was to mark the sample points with a reported error within their time range as abnormal, and the rest to normal. This was tried using different sample intervals. Another approach was to include sample points around an error as abnormal

too. Meaning if an error was reported at 09:34:28, not only the sample at 09:34 was labeled as an anomaly but the samples before and after as well (for example 09:33 and 09:35). Different number of neighbor points were used, up to 15 minutes away. It was also tried to make the ratio between the two classes more even. The anomaly class were therefore weighted, by copying the vector of anomalies in the training dataset. In this way the ratio got more even. However non of these attempts worked as every predicted value fell into the "normal" class.

Figure 35 shows the convergence of the self-organizing map using supervised learning. The independent variable seen in black converges. The red line shows the dependent variable which constitutes the base for supervised learning. As can be seen the dependent variable does not converge but keeps oscillating, indicating the supervised learning has failed. It was tried to increase the number of iterations but with no success of convergence.

Most of the application errors did not have an impact on the performance data. Looking at figure 24 for example, it can be seen that most reported errors are in the middle of the clusters, mixed with the normal data. Using labeled data for binary supervised learning, the data from the two classes must differ and have different properties. In this way the method can learn to categorize correctly. In this case however the normal samples could not be distinguished from the abnormal ones in the training set, thus the supervised approach failed. Also it is a disadvantage of having an uneven ratio between the classes. Usually anomaly detection is trained in an unsupervised mode and the supervised learning approaches were more or less expected to fail. One third possibility to why the supervised approach failed is that anomalies do not fall into one single category. Maybe there are many different kinds of system and application errors which needs to be labeled differently.

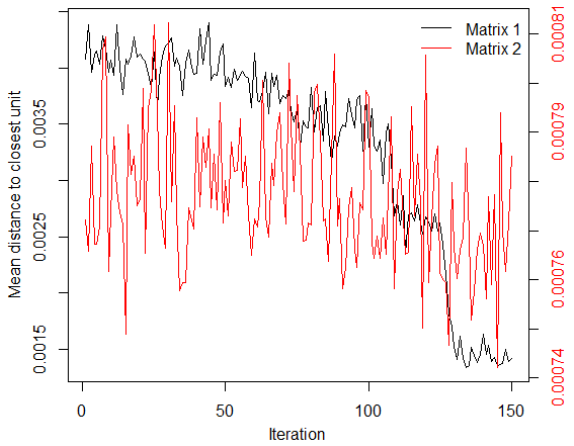

Figure 35: Convergence plot of the self-organizing map using supervised learning. Matrix 1 shows the convergence of the independent variable (black line) and matrix 2 the convergence of the dependent variable (red line).

6.5 Comparison of methods

Looking at the tables in the results, compiling the scores received by the two scoring methods gives an indication of their capacities. In table 2 (having 323 anomalies), KNN received the highest score. Having less detected anomalies (namely 14) the memory-CPU usage ratio method was the most successful one according to table 3. It also detected most of the system errors according to table 5. The drawback however was that it detected all the system errors *after* they had been reported. The optimal case would be to have a method detecting anomalies before they are reported. With 'scoring method 2' only anomalies being detected before the error is reported. According to the results in table 6, SVM, KNN and K-means were the best methods from that aspect. They detected many identical samples resulting in the exact number of scoring points. In table 5 it is also seen that the detected anomalies included there were identical for KNN, K-means, and SVM. It is an advantage to detect errors as quickly as possible, however the most important thing is to detect errors at all. As the number of detected errors is of greater importance the method, memory-CPU usage ratio is probably the best one.

When evaluating the methods future potential and capability for anomaly detection on virtual servers, more weight should probably lie on the scoring points from the runs with 14 detected

anomalies. Having 323 anomalies during 6 weeks means 6-7 errors per day on average. Such high anomaly-flagging rate would probably not be useful as people might start to ignore all the detected anomalies.

The fact that there were 323 reported errors in the event log during the period does not mean there were 323 errors on the server that needed to be detected and checked. Only nine of the errors were system errors in the operating system while the rest were application errors. Application errors are caused by many things. It can be minor errors with no interest for detection, sometimes these errors do not even originate from the server under study. The virtual server could for example fail to establish contact with another server which could result in a reported application error, despite the error was not caused by the investigated server.

14 anomalies during the testing interval means by average one detected anomaly every third day which seems more reasonable.

Time series analysis using linear regression and neural network have the drawback of one hour sampling interval and are thus not optimal. They both detected 2 out of 3 system errors during the test period. However they were tested during a short period of time (2 weeks) compared to the other methods (6 weeks). This makes the results unreliable. The reason to the short testing set was lack of continuous data. In contrast to the other methods they need to be trained with absolute continuous data which was only available from the 3rd of April. As the two methods needed as much training data as possible to perform well, it became a trade off between the length of the testing interval and training interval.

The investigated server presented in this thesis had no missing samples. However other severs investigated in the beginning of the project had sample points of value -1 on all features. Missing data points can be a disadvantage when applying time series analysis, as previous samples are used and can have a negative impact on the method. If any of these two methods are implemented in the future, linear regression is probably the best. Time series analysis is usually not that complex and the more simple linear regression is therefore preferable, as the neural network using other non linear activation functions are simply not necessary.

7 Conclusion

Several methods have been tried on a virtual server, to evaluate their capabilities of detecting anomalies. There were two types of reported errors: application errors and system errors. Application errors occurs often and might not necessarily imply an error of the server. They were in general hard to detect, as they had minimal or no effect on the performance data, except for when they occurred in groups with very many application errors reported in a short period of time. These groups were always connected to one or several system errors and were detected. System errors did affect the performance data and could be detected.

Time series analysis using neural network and linear regression used one hour sampling intervals. The reason was partly to decrease the number of weights in the algorithm and also to pick up the daily patterns in the CPU usage better. As the variance was very high this was hard to do using a minute interval. Using a longer time interval is a drawback however as it takes much longer to detect anomalies (up to one hour). The test period was short which makes it hard to evaluate them. During the short test interval however they both manage to detect the only two system errors and flagged one false positive each.

All the other methods were run two times, tuned to detect 323 and 14 anomalies. These numbers were selected since the total number of reported errors during the test interval was 323, and 14 anomalies corresponds to 1 anomaly every third day on average which seemed like a reasonable detecting rate. KNN were the best method when having 323 detected anomalies, meaning a high anomaly detecting rate. Assuming such a high rate of detected anomalies is not desirable (resulting in 7-8 detected anomalies per day on average), 14 detected anomalies is a better comparison when evaluating the methods. In this case the 'memory-CPU usage ratio' was the most successful method. It detected most of the system errors, 6 out of 9 were detected within a time range of 30 minutes. This method was also the very simplest and the anomaly detection rate is very easy to regulate by simply adjusting the limit of maximum allowed ratio between memory and CPU usage. It is probably the best method detecting memory leaks and infinite loops. This however has not been established in practice. The drawback with this method is the fact that the anomalies were detected after the errors were reported. KNN, SVM and K-means were more successful on detecting anomalies before the errors were reported in the event log. However they did not detect as many errors as the memory-CPU usage ratio method.

To conclude the whole thesis it can be said that there is a connection between anomalies in performance data and errors in the virtual server, at least system errors. This enables the possibility of using anomaly detection for detecting malfunctioning virtual servers. How well the methods perform on other types of errors (non reported errors) is hard to say but the fact that it was possible to detect system errors is a good sign. The most simple and successful method was the memory-CPU usage ratio.

7.1 Future work

If applying real-time anomaly detection on virtual servers the performance data should be extracted and analyzed automatically. One possibility to extract performance data automatically on these servers is to use VMware PowerCLI. This is a command-line and scripting tool built on Windows PowerShell for managing and automating, inter alia, vSphere. Some kind of monitoring program needs to be implemented too.

A combination of methods could be used to detect anomalies. There are different ways to combine them. One example is to flag anomalies only if all or at least several methods detect anomalies simultaneously.

The time series analysis could potentially be improved by using recurrent neural networks instead of feedforward neural networks. LSTM Models (Long Short-Term Memory networks) is an example that can be implemented using Keras and Tensorflow. In contrast to feedforward neural networks, recurrent neural networks have internal states (memory) which can improve the forecasting in time series analysis, as they learn more complicated and long-term patterns.

References

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: ACM Comput. Surv. 41.3, 15:1–15:58. issn: 0360- 0300. doi: 10.1145/1541880.1541882, 2009.

[2] Victoria J. Hodge and Jim Austin. "A Survey of Outlier Detection Methodologies". In: Artificial Intelligence Review 22.2, pp. 85–126, 2004.

[3] Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. "An introduction to statistical learning", chapter 2, 8, and 9. Springer, 2013.

[4] Peter J. Rousseeuw and Mia Hubert. "Robust statistics for outlier detection". WIRE's data mining and knowledge discovery, pp. 73-79, 2011.

[5] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". KDD'96, Proceedings of the second international conference on knowledge discovery and data mining, pp. 226-231, 1996.

[6] Teuvo Kohonen. "The Self-Organizing Map". Proceedings of the IEEE, 78(9), pp. 1464-1480, 1990.

[7] David J.C. MacKay "Information Theory, Inference, and Learning Algorithms. Cambridge University Press, pp. 471-480, 2003.

[8] Ha Quang Minh, Partha Niyogi and Yuan Yao. Mercer's Theorem, Feature Maps, and Smoothing. In: Lugosi G., Simon H.U. (eds) Learning Theory. COLT 2006. Lecture Notes in Computer Science, vol 4005. Springer, Berlin, Heidelberg.

[9] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". Stanford University. pp. 509-510 1973.

[10] Neculai Andrei. "Continuous Nonlinear Optimization for Engineering Applications in GAMS Technology". Springer, pp. 160-169, 2010.

8 Appendix

DBSCAN was not optimal when using training data to make predictions on new data. In other words to train the model and then add one value at the time to make predictions and see if the sample vector got into any cluster or not. It could not handle the around 60 000 predictions in the test set. This is essential in real time anomaly detection, as one new value at the time has to be analyzed. DBSCAN could only analyze big datasets when evaluating all values at once, meaning a set of historical data. The pictures below shows a run of DBSCAN, when the whole dataset was analyzed at once. In that case the results looks promising.

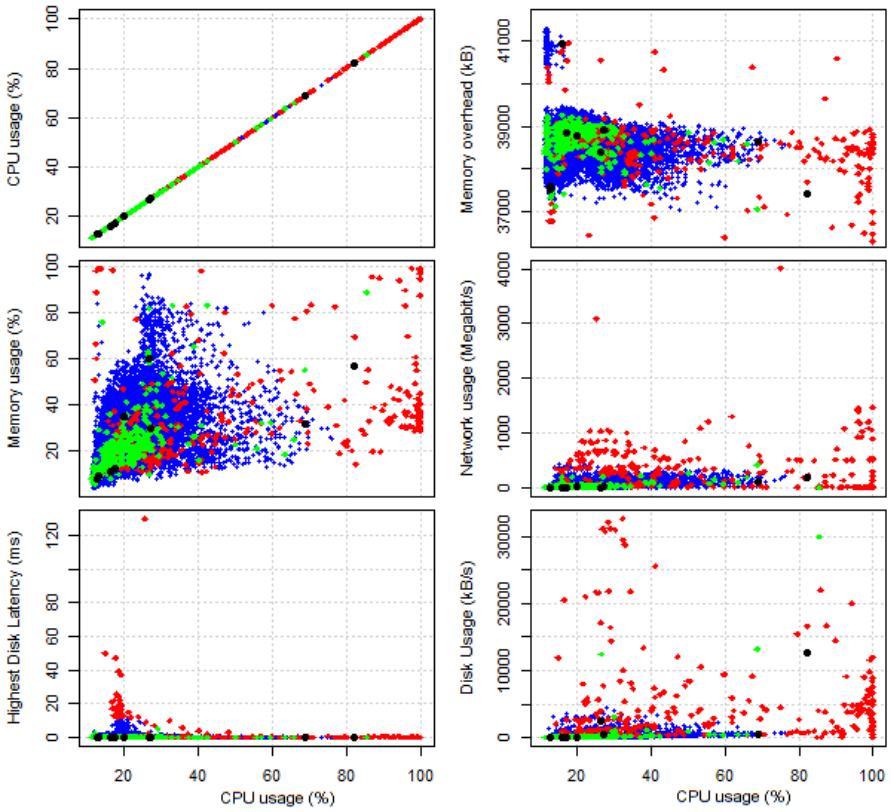

Figure 36: Results from the DBSCAN method. Performance data plotted against CPU usage, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

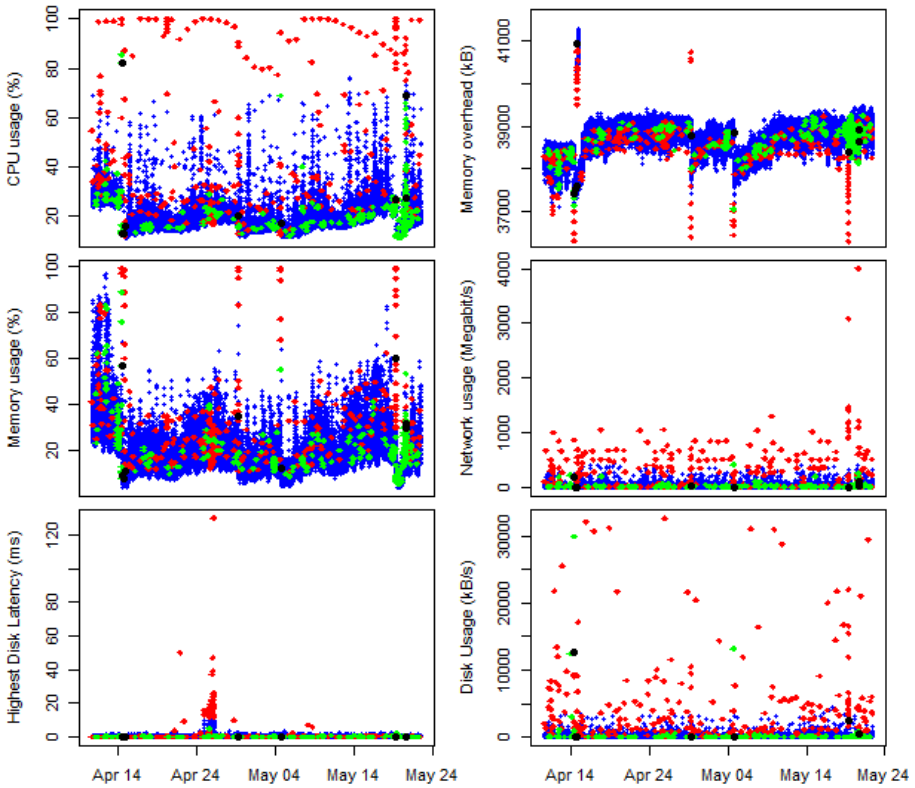

Figure 37: Results from the DBSCAN method. Performance data plotted against time, normal data in blue, application errors in green, system errors in black. Detected anomalies are marked in red, as small dots or squares with black outline belonging to run 1 or run 2 respectively.

Table 6: The exact time of the 14 detected anomalies for all methods. The nine reported system errors were reported: 04-14-09:59, 04-14-14:00, 04-14-18:00, 04-14-19:20, 04-29-07:34, 05-04-17:10, 05-19-09:13, 05-20-15:54, 05-20-15:57.

KNN	**K-means**	**SVM**	**SOM**	**Trend analysis**	**memory-CPU usage ratio**
04-12-22:07	04-25-19:10	04-14-21:21	04-21-21:10	04-14-12:55	04-25-22:02
04-15-22:00	04-25-22:02	04-14-21:23	04-25-19:10	04-14-12:56	04-14-10:04
04-16-22:02	04-25-23:34	04-14-21:25	04-25-22:02	04-14-12:57	04-14-19:25
04-18-21:53	04-25-23:49	04-14-21:28	04-25-22:59	05-16-17:36	04-29-07:39
04-21-21:10	04-26-00:14	04-26-04:35	04-25-23:49	05-16-17:37	04-29-07:40
04-25-22:27	04-26-01:22	05-09-13:34	04-26-01:03	05-16-17:38	04-30-13:26
04-26-01:03	04-26-02:22	05-19-09:10	04-26-01:10	05-19-13:36	04-30-13:34
04-26-04:35	04-26-02:52	05-19-09:18	04-26-01:22	05-19-13:37	04-30-13:35
05-06-21:58	04-26-04:24	05-19-09:29	04-26-02:22	05-19-13:38	05-04-17:14
05-09-22:18	04-26-04:35	05-19-09:25	04-26-02:52	05-19-13:39	05-04-17:15
05-10-22:24	04-26-04:43	05-19-09:26	04-26-04:24	05-20-16:47	05-07-10:39
05-19-09:10	04-26-04:44	05-19-09:29	04-26-04:35	05-20-16:48	05-08-15:40
05-20-15:43	05-19-09:10	05-20-15:43	04-26-04:43	05-20-16:49	05-20-15:14
05-21-22:18	05-20-15:43	05-20-16:28	04-26-04:44	05-20-16:50	05-20-15:15