Machine Learning Approaches for Multivariate Process Control on Count Data

Trouwen, Tristan

Award date:
2021

Eindhoven University of Technology
Department of Applied Mathematics

Machine Learning Approaches for Multivariate Process Control on Count Data

2WH40 - Bachelor Final Project

Tristan Trouwen
1322591

Supervisors:
Edwin van den Heuvel
Mona Emampour

Eindhoven, August 19, 2021

Abstract

Industrial processes collect large amounts of data for the purpose of quality control. Often this data is high-dimensional and highly correlated. Statistical process monitoring methods are widely used to ensure product quality and to identify processing mistakes. They should raise an alarm when a process is not in control anymore. The most common techniques are control charts, but recently (distribution-free) machine learning methods have proved to be successful in process monitoring. Here we will provide details of the Gaussian mixture model, a neural network called the self-organizing map, a robust version of the one-class support vector machine, a nearest neighbor approach, and the isolation forest for process monitoring. Their performance on multinomial count data will be compared with the generalized p-chart using simulation studies. Furthermore, a real case study is provided to compare these methods in less ideal situations where the data is not from a multinomial distribution.

1 Introduction

Statistical process control (SPC) is a technique to monitor and control a process. Upon detection of a special cause variation, corrective action can be taken such that the process operates within normal operating conditions. A control chart is a tool for SPC which calculates and plots a statistic together with control limits (CLs) over time. In combination with run rules it is able to detect nonnormal behavior. Thus, it can indicate out-of-control situations [1]. The performance of control charts is often evaluated using the average run length (ARL). The in-control ARL is defined as the expected number of observations before the control chart incorrectly signals out-of-control while the process is still in-control. The out-of-control ARL is defined as the expected number of observations before the control chart recognizes that it is out-of-control given that it is out-of-control. Hence, a well performing control chart has large in-control ARL and small out-of-control ARL. The set-up of control charts is divided into two phases. In phase I, preliminary control limits are determined using data from a processing period for which the process was in-control. In phase II, the control chart is applied in routine processing.

The type of control chart which is used depends on multiple factors such as the distribution of the data. For multivariate count data which follows a multinomial distribution, the generalized p-chart, introduced in [2], is often used. Let p_0 be a probabilities vector of a multinomial distribution of dimension m. Then, for each observation $x_t = (x_t(1), \ldots, x_t(m))$, we can use the Pearson goodness-of-fit statistic

$$Y_t^2 = \sum_{i=1}^{m} \frac{(x_t(i) - n_t p_0(i))^2}{n_t p_0(i)} \tag{1}$$

where $n_t = \sum_{i=1}^{m} x_t(i)$. This statistic is distributed as a χ^2 distribution with $m-1$ degrees of freedom for multinomial data, so we can use the following upper control limit (UCL) for a specific significance level α:

$$\text{UCL} = \chi^2_{\alpha,(m-1)}. \tag{2}$$

If $Y_i^2 > \text{UCL}$, then an alarm is raised for observation t.

However, in industrial processes, data is often high-dimensional, highly correlated, and therefore it is difficult to fit a distribution. Machine learning (ML) methods have proved successful on high dimensional data without a known distribution [3]. This is the reason that, in recent years, research on ML approaches for SPC has gained in popularity. Weese et al. created an overview of different machine learning methods used for multivariate SPC in which the methods proved successful with highly non-normal data and high dimensional data. Zimek et al. described the relation of process monitoring with machine learning in [4]. They concluded that process monitoring is highly related to anomaly detection in the field of machine learning. Two types of anomaly detection exist: novelty- and outlier detection. Novelty detection models are trained

using solely in-control data and are, therefore, called semi-supervised methods. Notice the parallel with phase I of statistical control charts where the model uses in-control data to determine the CLs. The model can then indicate whether new data is similar to the training data, thus signalling out-of-control. If outliers are also present in the unlabelled training data, outlier detection is used instead. Outlier detection methods use two properties of outliers. Namely, that outliers occur rarely and that they are different from in-control observations. Without these two properties, outlier detection becomes impossible. Although outlier detection is only done in one phase to separate a dataset in a set of outliers and a set of normal observations, it can also be applied to process monitoring. For every observation in phase II, the outlier detection method can check whether it is an outlier with respect to the set of all phase I observations.

There have been several attempts at classifying different anomaly detection algorithms. We will use the following classification made in [5]:

1. Probabilistic-based
2. Distance-based
3. Neighbor-based
4. Neural networks
5. Domain-based
6. Isolation-based
7. Information theory

One method from each class will be explained except for the information theory-based methods. These information theoretic methods such as the ones in the paper by Filippone et al. [6] are not related to ML. In [4], the probabilistic-, distance-, and neighbor-based methods are classified as a single density-based category. The methods in these three categories have a different algorithmic design dependent on probabilities, distances, and neighbors. However, all three are based on density estimates. For example distances between the nearest neighbor indicates are higher density estimates. However, we think that the methods within this one class are too diverse. Hence, we will analyze a neighbor-based model and a probabilistic-based model.

The probabilistic-based method analyzed is the Gaussian mixture model (GMMs) [7–9]. This method was chosen because it has been around for a very long time and, therefore, it is well researched and known which makes it a good comparison to other approaches. It is the most closely related to SPC and works by estimating the density as a mixture of multiple Gaussian densities. The neighbor-based method analyzed is the Local Outlier Factor (LOF). It is applied to process monitoring in [10–12]. It is neighbor-based since it creates an outlier score based on the value of only its nearest neighbors. The third method from the neural network category is the Kohonen Self-Organizing Map (SOM) which fits a map to the input data [13–15]. It is similar to the adaptive resonance theory network in the sense that it outputs the distance to the neuron closest by. We decided to investigate the SOM since it has been researched more extensively in the field of process monitoring and is more popular in general. The fourth method is from the domain-based category because it tries to demarcate a domain where observations are classified as normal. It is called the One-Class Support Vector Machine (OC-SVM). We will investigate a robust version of the OC-SVM which also works for outlier detection instead of only to novelty detection as discussed in [16]. The OC-SVM was chosen because it is the most prevalent, if not only, method in this category. The last method which will be explained is the Isolation Forest (iForest) [17] which dominates the isolation-based category. It is a state-of-the-art method for anomaly detection which separates itself from the other methods by isolating outliers instead of attempting to profile normal points. A case study on a real industry dataset was done in [18] and had promising results. The other category with information-theoretic approaches, such as

the one discussed in [6], fall outside the scope of this research since these methods do not have any relation with machine learning.

In Section 2, we start by introducing the aforementioned methods. This is followed by a discussion and theoretical comparison of the methods. Then, a simulation will be performed on multinomial data in Section 3. The methods' performance will then be compared to the generalized p-chart in addition to a comparison between the ML methods themselves. Lastly, Section 4 contains a case study in which the methods are compared using an industry dataset. This will provide more information on how the methods perform.

2 Methods

In this section, the methods already introduced in the introduction will be discussed. Firstly, the model and algorithms are explained. Subsequently, the application of the method to process monitoring will be explained.

2.1 Gaussian mixture models

The first method discussed is the oldest, and also the one most similar to traditional methods. It tries to alleviate the problem that data might not be normally distributed by estimating the density. Estimating such nonnormal densities is done by taking a, possibly infinite, mixture of normal densities. This was done in [8] where the technique proved successful in certain cases. Specifically, preprocessing with PCA was done first. We start by explaining the model and the training procedure. Then we proceed to explain how this is used for process monitoring.

2.1.1 Theoretical model

We can create a nonnormal distribution

$$p(x) = \sum_{i=1}^{M} \alpha_i N(x \mid \mu_i, \Sigma_i) \text{ with } \sum_{i=1}^{M} \alpha_i = 1, \tag{3}$$

where M is a parameter which determines the number of Gaussian densities and each $\alpha_i \geq 0$. This model is called the Gaussian mixture model. For each density i, α_i is its weight, and μ_i, Σ_i are the mean and covariance of the Gaussian density. From this, it becomes clear why this model is called the Gaussian mixture model (GMM).

The following step is to find appropriate values for $\alpha_i, \mu_i, \Sigma_i$. These values should be such that a set of training data X could be sampled from this new density. Hence, we try to maximize the log-likelihood function

$$\ell(\alpha, \mu, \Sigma; X) = \sum_{x \in X} \log(p(x)) \tag{4}$$

over the variables $\alpha_i, \mu_i, \Sigma_i$. Careful consideration must be made when choosing M such that the objective can be large but no overfitting occurs. Overfitting is a common problem in machine learning where your model works only on the data it is trained on but does not generalize well. So, although Equation (4) can be optimized better for large M, this should not be set too high. We proceed with using an expectation-maximization (EM) algorithm to maximize the log-likelihood over the parameters. The EM algorithm will iteratively estimate latent variables, and use that estimate to maximize the log-likelihood. In this case, the latent variables determine which Gaussian density each $x \in X$ belongs to. The algorithm is performed as shown in Algorithm 1. It starts by an initial guess for the parameters $\alpha_i, \mu_i, \Sigma_i$. Each mean is set to the value of some observation and all variances are set to the variance of all phase I data. Furthermore, $\alpha_i = \frac{1}{M}$. An alternative method performs k-means clustering to find initial values for the means

and Moody and Darken's rule to estimate the covariance [19]. Choi et al. showed that this provided faster convergence for their dataset [8]. This is mostly interesting for extremely large, or high-dimensional datasets, when training is slow and will not be discussed further. After the initialization, the expectation and maximization steps will be repeated until a constant number of iterations T is performed.

In each expectation step, we have some $\alpha_i, \mu_i, \Sigma_i$ given. For each mixture i and each observation x, we compute the probability that x belongs to mixture i:

$$r_i(x) = \frac{\alpha_i N(x; \mu_i, \Sigma_i)}{\sum_j \alpha_j N(x; \mu_j, \Sigma_j)}. \tag{5}$$

Then, in the maximization step, we update $\alpha_i, \mu_i, \Sigma_i$ as follows

$$\begin{aligned} m_i &= \sum_i r_i(x), \\ \alpha_i &= \frac{m_i}{n} \text{ with } n \text{ the total number of observations}, \\ \mu_i &= \frac{1}{m_i} \sum_i r_i(x) x, \\ \Sigma_i &= \frac{1}{m_i} \sum_i r_i(x)(x - \mu_i)^T (x - \mu_i). \end{aligned}$$

The complete procedure in shown in Algorithm 1.

Algorithm 1 Expectation maximization algorithm for Multivariate GMMs

Input: $X = \{x_1, \ldots, x_n\}$ - training data, M - number of mixture components, T - iteration limit

Make initial guess for $\alpha_i, \mu_i, \Sigma_i$ with $1 \leq i \leq M$
for T iterations **do**
  for each mixture component i **do**
    for each observation $x \in X$ **do**

$$r_i(x) \leftarrow \frac{\alpha_i N(x; \mu_i, \Sigma_i)}{\sum_j \alpha_j N(x; \mu_j, \Sigma_j)}$$

    end for
    $m_i \leftarrow \sum_{x \in X} r_i(x)$
    $\alpha_i \leftarrow \frac{m_i}{n}$
    $\mu_i \leftarrow \frac{1}{m_i} \sum_{x \in X} r_i(x) x$
    $\Sigma_i \leftarrow \frac{1}{m_i} \sum_{x \in X} r_i(x)(x - \mu_i)^T (x - \mu_i)$
  end for
end for

2.1.2 Process monitoring

In the process monitoring situation, this EM algorithm will be applied on a training data set to create a mixture model which fits the data. The training data should be in-control since the method is not robust to outliers in the data. From this, we will create a control chart. This is done by looking at which of the mixtures is most likely responsible for an observation. Then we choose that mixture and apply the Hotelling's T^2 chart for the decision of being out-of-control. This approach is also explained in the paper by Choi et al. [8].

For each mixture component i, and observation x the test statistic is

$$T^2 = (x - \mu_i)' \Sigma_i^{-1} (x - \mu_i). \tag{6}$$

Observe that m_i, as defined above, is the probability that an observation was sampled from mixture component i. So, for an observation x, the T^2 test statistic will be the smallest for mixture component $i = \text{argmax}_i\, r_i(x)$. Only the smallest test statistic will be used in the control chart. Then, given m quality characteristics and n observations, the UCL is given by

$$\text{UCL} = \frac{m(n+1)(n-1)}{n^2 - nm} F_{\alpha,m,n-m} \tag{7}$$

where α denotes the probability of a type I error. When $m > 100$, this is well approximated by the UCL given by

$$\text{UCL} = \frac{m(n-1)}{m-p} F_{\alpha,m,n-m}, \tag{8}$$

as explained in [20]. The approach taken gets more complex if we include preprocessing using PCA as explained in Section B. In addition to the control chart monitoring the T^2 statistic, an additional control chart based on the Q statistic will be introduced to monitor the residual, the part not modelled by PCA. Notice that PCA is only useful if data is high-dimensional or highly correlated.

Statistic T^2. We will first compute the loading matrix $P_s \in \mathbb{R}^{n\times s}$ of the training data X, where s is the number of dimensions to reduce to. It is not specified how s should be chosen by Choi et al. If out-of-control data is available, experiments can be done to find the best value. s can also be set equal to m which will not perform any dimension reduction, but will remove correlations. The columns of P_s are called loading vectors. Define Λ^{-1} as the diagonal matrix with on its diagonal, the inverse of the eigenvalues obtained in P_s. Then we have the test statistic defined as

$$T^2 = xP_s\Lambda^{-1}P_s^T x^T. \tag{9}$$

The UCL is the same as above except that there are now only s quality characteristics instead of m.

Statistic Q. The portion not explained by the PCA model, the residual, will have a separate control chart based on the Q statistic. The residual $(I - P_sP_s^T)x$ portion is not explained by the PCA model. The test statistic will be

$$Q = x^T(I - P_sP_s^T)x. \tag{10}$$

The UCL of this is

$$\text{UCL} = \theta_1 \left(\frac{z_\alpha \sqrt{2\theta_2 h_0^2}}{\theta_1} + 1 + \frac{\theta_2 h_0(h_0-1)}{\theta_1^2} \right)^{1/h_0}. \tag{11}$$

where z_α is the normal distribution value with significance level α. In addition, we have

$$\theta_j = \sum_{i=s+1}^{p} (\lambda_i)^j, \tag{12}$$

$$h_0 = 1 - 2\theta_1\theta_3/3\theta_2^2 \tag{13}$$

where λ_i is the eigenvalue corresponding to the ith loading vector.

This method is extended to the GMM approach easily. The Q statistic remains identical and will not be influenced by the GMM model. The GMM model will be trained using the preprocessed data and the Hotelling's T^2 chart will be based on the statistic from the mixture component from which the observation most likely originates.

2.2 Self-organizing map

The self-organizing map (SOM) is a neural network developed by Kohonen in 1982 for dimensionality reduction of data [21]. However, the network can also be used for classification and outlier detection. The network consists of a 2-dimensional grid of neurons, which is fitted to the input data and provides a representation of the input data. Each neuron corresponds to a weight with dimension m equal to the dimension of the input data. In a training procedure, these weights are updated such that the training data has values close to the values of the weights. We start by explaining the principles of the self-organizing map and some of its variations. After this, some extensive preprocessing will be discussed, as well as how to apply this to process monitoring as explained by Yu et al. in [15].

2.2.1 Theoretical model

A lot of variations of the SOM exist. Here, the variant used during training in [15] is used. Note that the SOM used on the training dataset was not explained fully, only a simpler variant. For the complete variant the author refers to the paper [22]. We will explain the complete model and we will assume settings that are not reported are kept at the default values. The method works by creating k neurons. These neurons should be organized in a low-dimensional, often 2-dimensional, grid. This allows us to compute a discrete neighborhood. This grid can be applied to a shape such as a toroid or cylinder. An example grid is shown in Figure 1 where the most simple 'sheet' shape is used. This grid should then be fit to an in-control dataset $X = (x_1, \ldots, x_n)$. Prior to

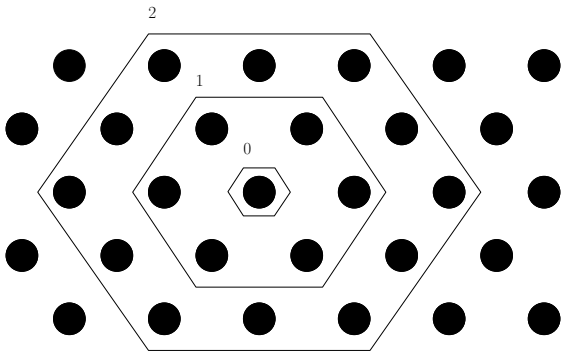


Figure 1: Hexagonal lattice structure with discrete 0, 1, and 2-neighborhoods

this fitting process, normalization should be done on X since larger features would dominate the process due to the use of the Euclidian distance. After this preprocessing step, each neuron i is assigned a random weight w_i with the same dimension as the training data. Other methods to intialize the weights are discussed more in-depth in the weight intialization paragraph below since multiple approaches are possible. The weights are then iteratively updated in the next lines. This is done by selecting a sample x from the training data randomly. The best matching unit (BMU) for x is then computed. This is the neuron with corresponding weight closest to x. We will use c to denote the index of the BMU. The distance between the BMU and x is called the minimum quantization error (MQE) of x. In the next steps, the MQE will be minimized by updating the weights.

For all neurons in the neighborhood of neuron c, the weight will be updated using an update rule which makes the weights closer to x. In this way, the weights will drift towards points where most of the training data is. Therefore, the MQE of each observation of the training data will reduce. This iterative process stops once an iteration limit T is reached. Each iteration is also called an epoch.

The update rule uses a learning rate function $\alpha(t)$ and a neighborhood factor function h_{ci}. The learning rate function is unclear from the paper. The paper states that a rough and a fine-tune

phase is performed. This first phase, also called the ordering phase, uses a large α. In the fine-tune phase this will be significantly lower. Concretely, one has to define the number of epochs in the ordering phase T_1 and α_0, α_1 to get

$$\alpha(t) = \begin{cases} \alpha_0/(1 + 100t/T_1), \\ \alpha_1/(1 + 100(t - T_1)/T_2), \end{cases} \tag{14}$$

where $T_2 = T - T_1$ is the number of iterations in the fine-tune phase. The $h_{ci}(t)$ is defined as

$$h_{ci}(t) = e^{-d_{ci}^2/2\sigma_t^2} \tag{15}$$

with d_{ci} is the Euclidian distance between neuron c and i.

Algorithm 2 Simple SOM Training algorithm

Input: $X = (x_1, \ldots, x_n)$ - training data, k - number of neurons, T - iteration limit

1: w_i randomly initialized for $1 \leq i \leq k$
2: **for** t from 1 to T **do**
3: select $x \in X$ randomly
4: $c \leftarrow \text{argmin}_i\, d(w_i, x)$
5: **for** $i = 1$ to k **do**
6: $w_i \leftarrow w_i + \alpha(t)h_{ci}(x - w_i))$
7: **end for**
8: **end for**

The training process shown in pseudocode in Algorithm 2 uses input data $X = \{x_1, \ldots, x_n\}$ where $x_i = (x_i(1), \ldots x_i(m))$ is an m-dimensional data point. After randomly initializing weights and iterative process starts. Each time an input vector x from the training data is randomly selected for which the best matching unit with index c is computed. So, c is the index of the weight vector closest to x.

Weight initialization. Having a good initialization of the neuron weights speeds up the training since fewer iterations are needed to converge. There are two popular methods possible for the initialization of weights. The simplest one is to estimate the mean and variance of the dataset using the maximum likelihood estimators giving $\hat{\mu}, \hat{\sigma}^2$. The weights can then be sampled from the normal distribution $N(\hat{\mu}, \hat{\sigma}^2)$. Another option is to initialize the weights with samples from the subspace spanned by the first n principal components of the training data. These two methods are compared by Akinduko et al. [23], which concluded that for nonlinear data the random initialization may work better. The approach we will use and which was also used in the paper of Yu et al. is linear initialization which is the same as the PCA approach with 1 principal component.

Batch training. The simple training algorithm trains sequentially by updating the neuron weights to be closer to randomly selected observations. An alternative approach uses a batch training algorithm which takes into account all training data before making an update. Algorithm 2 can be altered to perform the batch training as explained in [22] by removing line 3 and substituting lines 5,6, and 7 by the following:

for neuron $i = 1$ to k **do**
$w_i \leftarrow \frac{\sum_{i=1}^{n} h_{ic}(t)x_j}{\sum_{j=1}^{n} h_{ic}(t)}$
end for

2.2.2 Process monitoring

Before being fed as input data to the algorithm, some preprocessing on the time series data $X = (x_1, \ldots, x_n)$ is needed. Since using the past results as well as the current result can improve

performance, the algorithm takes into account the history. Traditionally, this is achieved by run rules which often have the following form: raise an alarm if at least k of the last n statistics are in an interval (a, b) [24]. Using these run rules the history of a statistic is taken into account. For the SOM this is done by windowing the data. Let ω be the window size. Then we can define a new data point x_i^* as

$$(x_i', \ldots, x_{i+\omega-1}')'$$

for $1 \leq i \leq m - \omega + 1$. Notice that this reduces the data by $\omega - 1$ observations. A study from Hassan et al. [25] showed that better performance can be reached for SPC charts by using statistical features instead of raw data. So, after windowing, we can compute several statistical features (mean, standard deviation, skewness, mean-square, autocorrelation, and Cusum) as well as the T^2 statistic of the last observation in the window. These statistical features were chosen in the paper of Hassan et al. to represent the data as well as possible. Autocorrelation was computed with lag 1. The Cusum statistic is the tabular Cusum as defined by Montgomery [20] as

$$\begin{aligned} C_i^+ &= \max(0, x_i - \hat{\mu} + C_{i-1}^+), \\ C_i^- &= \max(0, \hat{\mu}_0 - x_i + C_{i-1}^-). \end{aligned} \tag{16}$$

with $C_0^- = C_0^+ = 0$. Only the last positive and negative Cusum values are taken as input features. This preprocessing was also done in [15], except that they used the χ^2 statistic which approximates the T^2 statistic. However, for smaller datasets, the T^2 statistic is more appropriate.

Normalization of the features of the data should be done afterward to ensure there is no bias towards certain features. For multivariate data, each sample has m quality characteristics. Each one can have a different mean and variance. Hence, a difference from the mean of a certain size can be negligible for some characteristics and very important for others. To combat this, data can be normalized to have the mean 0 and variance 1. By estimating the mean by $\bar{x}$ and covariance matrix by S and then transforming each data point to $(x - \bar{x}) \oslash \text{diag}(S)$ where $\oslash$ denotes element-wise division.

Simulations were done using the Matlab SOM Toolbox, version 5 [22]. The parameters were left on the default values which are the following:

$$\begin{aligned} \sigma_0 &= k/4 \\ \sigma_T &= 1 \\ \alpha_0 &= 0.5 \\ \alpha_1 &= 0.05 \\ T_1 &= 4k/n \\ T_2 &= 16k/n \\ T &= 20k/n \end{aligned} \tag{17}$$

The number of neurons k and the height (k_1) and width (k_2) of the grid are chosen such that $k_1/k_2 \approx \lambda_1/\lambda_2$ where λ_1, λ_2 are the largest eigenvalues of the data's covariance matrix and the number of neurons is set using a heuristic formula $k = k_1 k_2 \approx 5n^{0.54321}$. This can only be done approximately since k_1, k_2, k have to be integers. This method make sure sensible default values are used.

Finally, when the algorithm is done, phase II can start. A control chart can be made by plotting the $MQE = |x - w_{bmu}|$. The control limit should be set manually. In the paper by Yu et al. this was done by generating 1000 in-control observations and setting the UCL to the largest statistic such that at most 1000α observations have an MQE that exceeds this statistic.

2.3 Local outlier factor

The local outlier factor (LOF) algorithm was developed in 2000 by Breunig et al. [26] to assign a degree of being an outlier to an observation instead of a binary classification. To perform process monitoring the LOF is integrated with independent component analysis (ICA). This is a method that transforms mixed signals into independent signals, minimizing Gaussianity. ICA can also be performed without the LOF algorithm, but this assumes that the data is a mixture of solely non-Gaussian distributions whereas the LOF integrated with ICA works well on data that is a mixture of both non-Gaussian and Gaussian data.

2.3.1 Theoretical model

To compute the LOF we introduce some definitions. Let $x \in X$, then the k-distance of x is the distance to the kth nearest observation in $X \setminus \{x\}$ for parameter $k \in \mathbb{N}$. Let $\text{kNN}(x)$ be the set of all observations in the k-distance neighborhood of x. So,

$$\text{kNN}(x) = \{y \in X \mid d(x,y) \leq k\text{-distance}(x)\} \tag{18}$$

with $d(x,y)$ the Euclidian distance from x to y. Hence, $|\text{kNN}(x)| \geq k$. Then we define the reachability distance of x to an observation $y \in X$,

$$\text{reach-dist}_k(x,y) = \max\{k\text{-distance}(y), d(x,y)\}. \tag{19}$$

Although named as a distance, it is not symmetric. After this, we introduce the definition of the local reachability density (lrd). This is defined as

$$\text{lrd}_k(x) = \frac{|\text{kNN}(x)|}{\sum_{y \in \text{kNN}(x)} \text{reach-dist}_k(x,y)}. \tag{20}$$

Observe that the lrd is large if the observations in $\text{kNN}(x)$ are close to the points in their KNN set. Since, if either the distance to x or the $k-$distance of an observation $y \in \text{kNN}(x)$ is small, then the reachability distance will also be small. Finally, the LOF is computed as

$$\text{LOF}(x) = \frac{\frac{1}{|\text{kNN}(x)|} \sum_{y \in \text{kNN}(x)} \text{lrd}_k(y)}{\text{lrd}_k(x)}. \tag{21}$$

This is the ratio of the average density of the observations in $\text{kNN}(x)$ to the density of x. Notice that if $\text{LOF}(x) \sim 1$ or $\text{LOF}(x) < 1$, the observation is not an outlier. But if the lrd of x is small, so $\text{LOF}(x) > 1$, then x can be considered as an outlier.

2.3.2 Process monitoring

To use this in process monitoring, we will follow the paper of Lee et al. [10]. The data will be preprocessed using independent component analysis (ICA). Furthermore, a method will be explained to remove outliers. A method to determine the control limits will be explained as well.

ICA is a preprocessing method explained in Appendix C and is performed without reducing the dimensionality. After this, the LOF of each of the training data X will be computed. These LOF values are from an unknown distribution. A kernel density estimator (KDE) is made to define an upper 99.3% limit. To find a $1-\alpha$ limit, we need to solve for u the equation

$$\int_{-\infty}^{u} p(x)dx = 1 - \alpha, \tag{22}$$

where $p(x)$ is the kernel density estimator

$$p(x) = \frac{1}{nh} \sum_i K\left(\frac{x - x_i}{h}\right), \tag{23}$$

with smoothing parameter h. The kernel function chosen is the Gaussian kernel function since this is common. In most literature, it is unspecified how this is solved. We will assume that we are able to do this, and will provide more details in Section 3. The data points with LOF above this limit are excluded and again ICA is performed and LOFs are computed. Then, a new limit using the KDE can be made; this will be the UCL. Finally, in phase II, the LOF can be computed for new observations after being processed by the ICA whitening and after applying the demixing matrix. Observations with LOF $>$ UCL will raise an alarm. The LOF's parameter k is set to 20 in the paper's experiments where training was done with 200 data points. Notice that a larger k might make the algorithm oversensitive. In addition, the training data set with size n will determine how large k can be. Hence, experimentation to find suitable k should be done.

The attentive reader might notice that the paper by Lee et al. has a different definition of the LOF and the lrd than the definition in the original paper by Breunig. In both definitions, the cardinality of kNN(x) is replaced by k. This will yield strange results when there are a lot of duplicates such that this cardinality is larger than k. Hence, the definition from the paper of Breunig is preferred. However, if the kNN function would return precisely k numbers, this would make no difference. We will see in Section 3 that this is actually the case for the implementation of the kNN we use.

2.4 Robust one-class SVM

This method is based on the one-class support vector machine (OC-SVM) proposed by Schölpkof et al. in 1999 [27] which uses a hyperplane to separate normal data from outliers. It is often referred to as the ν-SVM since ν is an important parameter, this will be clarified later on. An alternative OC-SVM by Tax and Duin [28] exists which uses a hypersphere instead of a hyperplane. This method has equivalent results and will not be analyzed in this work. The robust OC-SVM was analyzed by Ma et al. in [11] for fault diagnosis on time series data.

Since the method is sensitive to outliers, Yin et al. proposed an adaptation that makes it robust for outlier detection in [29]. The paper is specific for fault detection, so no analysis on ARLs was done. However, the method can still be used for the computation of ARLs on time-series data by using the found decision boundary on new data. The η OC-SVM developed in [16] outperformed the robust OC-SVM with multiple datasets. It provides a method such that outliers are not incorporated in computing the decision boundary. Although promising, the paper lacks the connection with SPC, hence, the method will not be discussed here. Firstly, we will start by explaining the original OC-SVM method. After that, the adaptation to make the method more robust will be explained.

This method uses PCA just like the GMM method explained in Section 2.1 but only for decorrelation. Hence, no dimensionality reduction is performed. Therefore, the residual part unexplained by PCA and analyzed using the Q statistic does not exist.

2.4.1 Theoretical model

To understand the OC-SVM, we will use knowledge from the field of linear optimization. Although concepts such as the dual, slack variables, and the Karush-Kuhn-Tucker (KKT) conditions will be explained, some introductory knowledge will help, as well als knowledge of the original basic SVM.

The OC-SVM uses a decision boundary, which separates the origin, being the zero vector in the data's feature space, from the data points $X = (x_1, \ldots, x_n)$. Data points residing on the same side of the boundary as the origin are classified as outliers. The decision boundary will be given by the function

$$g(x) = w^T \phi(x) - \rho \tag{24}$$

where ρ is the bias term and w a vector perpendicular to the decision boundary. $\phi(\cdot)$ is a feature mapping which maps the data into a higher-dimensional space which allows for a nonlinear decision boundary. Hence, when $g(x) < 0$, x is classified as an outlier. We will discuss later how we can get a test statistic instead of a binary result from the model.

The goal is to maximize separation between the data and the origin by choosing appropriate w, ρ. Therefore, we want to maximize $d(x) = \frac{|g(x)|}{||w||}$, the distance from a point x to the decision boundary defined by $g(x) = 0$. Observe that the distance to the origin is $\frac{|\rho|}{||w||}$. Notice that this is equivalent to minimizing $\frac{||w||^2}{2} - \rho$. This is has to be done under the constraint that for all $x \in X$, we have that

$$g(x) \geq 0. \tag{25}$$

This would mean all training points lay on the side which doe snot contain the origin. This model is extended by using slack variables $\xi_i \geq 0$ which make it possible for some points of the training data to reside on the same side of the decision boundary as the origin. The original constraint in Equation (25) is then replaced by $g(x) \geq -x_i$, and this is called a soft margin. Using a regularization parameter $\nu \in (0, 1)$, we can control how much these outliers are penalized by adding the term $\frac{1}{n\nu}\sum_{i=1}^{n}\xi_i$ to the original minimization objective, where n is the number of data points. From now on, when i is used in a constraint, we assume the constraint should hold for all i such that $1 \leq i \leq n$. Observe that when $\nu = 0$, the problem is the hard-margin problem where no outliers are allowed. So, we get as the optimization problem

$$\begin{aligned} &\text{objective: } \min_{w,\xi,\rho} \frac{||w||^2}{2} - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \\ &\text{subject to: } w^T\phi(x_i) \geq \rho - \xi_i, \quad \xi_i \geq 0. \end{aligned} \tag{26}$$

The goal of the method should now be clear. What follows is a derivation to get a quadratic optimization problem which is equivalent to the current problem but easier to solve algorithmically. This is done by computing the dual of the problem which is a version of the problem where the objective becomes a constraint, each constraint becomes part of an objective and minimization becomes maximization or vice versa. The solution to this problem is equivalent if the solution meets the KKT conditions which we will show to be the case. Notice that we can write the constraints as

$$-w^T\phi(x_i) - \xi_i + \rho \leq 0, \quad -\xi_i \leq 0.$$

Firstly, we define the kernel function $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ and the kernel matrix H with each $h_{ij} = K(x_i, x_j)$. To compute the dual, introduce Lagrange multipliers $\alpha_i \geq 0, \beta_i \geq 0$ with $1 \leq i \leq n$. Then we can create the Lagrange equation

$$L(w, \xi, \rho, \alpha, \beta) = \frac{||w||^2}{2} - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \left(\sum_{i=1}^{n}\alpha_i(w^T\phi(x_i) - \rho + \xi_i)\right) - \sum_{i=1}^{n}\beta_i\xi_i. \tag{27}$$

A saddle point of the Lagrange equation is a solution to the original problem. We proceed by setting the partial derivative of L with respect to w, ξ, ρ equal to zero to find the saddle points. In this way, we get for w the following

$$\begin{aligned} w &= \sum_{i=1}^{n}\alpha_i\phi(x_i) \\ \alpha_i &= \frac{1}{n\nu} - \beta_i \implies 0 \leq \alpha_i \leq \frac{1}{\nu n} \\ \sum_{i=1}^{n}\alpha_i &= 1 \end{aligned} \tag{28}$$

The last two equations pose constraints, and substituting these three into the Lagrange equation allows us to simplify the results excessively. Multiple variables will cancel each other out and we get

$$L(w, \xi, \rho, \alpha, \beta) = \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i (\phi(x_j))^T \phi(x_i)) \alpha_j = \frac{1}{2} \alpha K \alpha^T. \tag{29}$$

From this we get the following optimization problem:

$$\begin{aligned} \text{objective: } & \min_{\alpha} \frac{\alpha^T H \alpha}{2} \\ \text{subject to: } & 0 \leq \alpha \leq \frac{1}{\nu n}, \quad \sum_{i=1}^{n} \alpha_i = 1 \end{aligned} \tag{30}$$

Now we need to verify that the KKT conditions are satisfied. Indeed, the following conditions are true for solution w, α, β:

$$\begin{aligned} \frac{\partial}{\partial w_i} L(w, \alpha, \beta) &= 0, i = 1, \dots m \\ \frac{\partial}{\partial \beta_i} L(w, \alpha, \beta) &= 0, i = 1, \dots m \\ \alpha_i g_i(w) &= 0, i = 1, \dots n \\ g_i(w) &\leq 0, i = 1, \dots n \\ \alpha &\geq 0 \end{aligned} \tag{31}$$

If H is positive definite, the KKT conditions are satisfied, as well as the conditions for the dual to be equal to the primal problem. To make sure H is positive definite, duplicates will have to be removed from the training data. After this process, the kernel function ensures positive definiteness and we come to a convex problem for which the solution of the dual is also the solution of the primal problem. Since the found problem is convex, it can be solved using quadratic programming (QP) software in polynomial time.

Notice that this allows us to calculate α, but we also need ρ to get a decision boundary. The paper proposes to do this by computing ρ as

$$\rho = \frac{1}{n_s} \sum_{i=1}^{n_s} \sum_{j=1}^{n} \alpha_j K(x_i, x_j). \tag{32}$$

with x_i the support vectors which have slack variables $\xi_i = 0$ and n_s the total number of such vectors. The original paper suggests that any of the support vectors can be taken to come to a ρ. However, taking the average is a good idea to get numerical stability, and this is also done in the well-known implementation in LIBSVM as explained in their design document [30].

2.4.2 Robust OC-SVM

The main modification for the robust OC-SVM is modifying the slack variable such that its penalization is proportional to the distance of the centroid of all data. Hence, points distant from the center can have a large slack variable without influencing the decision boundary. In the original OC-SVM, the variables ξ_i were used for this, together with the parameter ν which controlled the level of penalization. Since the slack variables ideally are small, we again add a term to the objective which penalizes large ξ_i. Only now, the constant parameter Θ is used to control the harshness of this penalization instead of ν such that we get

$$\begin{aligned} \text{objective: } & \min_{w, \xi, \rho} \frac{||w||^2}{2} - \rho + \Theta \sum_{i=1}^{n} \hat{d}_i \xi_i \\ \text{subject to: } & w^T \phi(x_i) \geq \rho - \xi_i, \quad \xi_i \geq 0. \end{aligned} \tag{33}$$

were $\hat{d}_i$, are the penalty factors determined by their distance to the centroid of the dataset. More precisely we define

$$\begin{aligned} d_i &= ||x_i - C||^2, \\ \hat{d}_i &= \frac{\max_j d_j}{d_i} \end{aligned} \tag{34}$$

with C the centroid of the dataset. The total square loss center is used as a centroid due to its robustness to outliers. This is given by

$$\begin{aligned} C &= \sum_{i=1}^{n} k_i \phi(x_i), \\ k_i &= \frac{1}{\sqrt{q + 4||x_i||^2}} \Big/ \sum_{j=1}^{n} \frac{1}{\sqrt{1 + 4||x_j||^2}} \end{aligned} \tag{35}$$

as proposed in the paper by Liu et al. [31]. Then we compute d_i using Equation (36).

$$d_i = H_{ii} - 2\sum_{j=1}^{n} k_j h_{ij} + k^T H k \tag{36}$$

Increasing Θ will result in the model penalizing large slack variables more. However, if Θ is taken too high, the algorithm will not generalize well. The optimization problem can then be formulized as follows:

$$\begin{aligned} \text{objective: } & \min_{w,\rho} \frac{||w||^2}{2} - \rho, \\ \text{subject to: } & w^T \phi(x_i) \geq \rho - \Theta \hat{d}_i. \end{aligned} \tag{37}$$

Computation of the dual is similar to the computation of the normal OC-SVM dual and results in

$$\begin{aligned} \text{objective: } & \min_{\alpha} \frac{\alpha^T H \alpha}{2}, \\ \text{subject to: } & 0 \leq \alpha_i \leq \Theta \hat{d}_i, \quad \sum_{i=1}^{n} \alpha_i = 1. \end{aligned} \tag{38}$$

2.4.3 Process monitoring

For using the robust OC-SVM in process monitoring, a distance metric will be introduced which can serve as a test statistic. Furthermore, a UCL for the metric needs to be given. In addition, the kernel function and the parameter Θ need to be chosen. PCA is performed only for transforming possibly correlated to new uncorrelated data. No dimensionality reduction is done. For the kernel function, the radial basis function is chosen, given by

$$K(x_i, x_j) = e^{-||x_i - x_j||^2 / 2\sigma^2}. \tag{39}$$

Sometimes the radial basis function has a parameter γ instead of σ which is then equal to $1/2\sigma^2$. It was noted that this kernel can approximate most kernels. Furthermore, with only one parameter it was stated that the kernel can be tuned to an appropriate value with relative ease although no concrete method was given. In the paper, $\sigma = 2$ was chosen. However, implementations such as the one in [32] use $\gamma = \frac{1}{2\sigma^2} = \frac{1}{m}$ as the default. Lastly, the for the parameter Θ, 0.004 resulted in good results. Hence, this was chosen. To have a continuous test statistic instead of a binary classification, we compute the distance metric

$$F(x) = -\sum_{i=1}^{n} \alpha_i K(x_i, x) + \rho. \tag{40}$$

Normally, x is classified as an outlier when $F(x) < 0$. In the paper, it was suggested that a manually tuned threshold performs better since it is difficult to set the correct parameter values. Due to this difficulty, the optimal threshold is often slightly above 0.

2.5 Isolation forest

The Isolation Forest (iForest) is an unsupervised outlier detection algorithm developed in 2008 in [17]. It is fundamentally different from existing outlier detection methods because it does not try to profile normal points. Instead, it tries to isolate the outliers. This is done by creating an ensemble of trees, which is called a forest. Each tree is a binary decision tree trained on a subset of features and a subset of data. Now each tree should be made such that it has one of the training observations as a leaf. Notice that, therefore, data points in dense regions correspond to a leaf node of large height since lots of decisions have to be made to isolate that point from all others. On the other hand, outliers will lay far away from the normal data and can be isolated by few binary decisions. Hence, outliers will be at a leaf node with a small height. The method has low memory usage and linear time complexity which makes it an excellent method for large datasets.

2.5.1 Theoretical model

Algorithm 3 describes how an iForest is created. As input, the algorithm receives the training set, the number of trees in a forest t and the size of a sample ψ. Firstly, an empty set called *Forest* is made, and the height limit of an iTree is computed as $\lceil \log_2 \psi \rceil$. It then proceeds creating an iTree t times for a random data sample of size ψ and adding that to the *Forest* set. This set is then returned and can be used for classification. The iTree previously mentioned is made

Algorithm 3 iForest(X, t, ψ)

Input: X - training data, t - number of trees, ψ - subsampling size

```
Forest ← {}
l ← ⌈log_2 ψ⌉
for i = 1 to t do
    X' ←sample(X, ψ)
    Forest ← Forest ∪{ iTree(X', 0, l)}
end for
return Forest
```

by Algorithm 4 which splits the data based on a random feature, and a random value. It then recursively calls itself on each part of the split. The algorithm stops such that a leaf node is generated when only one node is left or when the maximum tree height would be exceeded by continuing. At each node, we save the feature on which to split in a variable *SplitFeature* and the value where to split in a variable called *SplitValue*. At the leaf nodes, we save the number of observations still in the node. Notice that this is a proper binary tree (i.e. all non-leaf nodes have exactly two children). In addition, the tree is given a height limit of $\lceil \log_2 \psi \rceil$. Hence, since the number of leaves is equal to ψ, the height limit is approximately the average tree height. Furthermore, this structure is very similar to the structure of a binary search tree (BST). Notice that the height for external node termination $h(x)$ is the path length from the root to a leaf node which is the same as the length of an unsuccessful search in a BST. The average path length of unsuccessful search in a BST is given by

$$c(n) = 2H(n-1) - (2(n-1)/n), \tag{41}$$

where the BST is fitted on a dataset of n instances and where $H(i)$ denotes the harmonic number. An estimate of the harmonic number is given by

$$H(i) = \ln i + \varepsilon \text{ where } \varepsilon \text{ is Euler's constant.} \tag{42}$$

Algorithm 4 iTree(X, e, l)

Input: X - training data, e - current tree height, l - height limit

if $e \geq l$ or $|X| \leq 1$ **then**
  return exNode{Size $\leftarrow |X|$}
else
  Select feature i with $1 \leq i \leq m$ randomly
  Select $p \in$ range$\{x(i) \mid x \in X\}$ randomly
  $X_l \leftarrow \{x \mid x \in X, x(i) < p\}$
  $X_r \leftarrow \{x \mid x \in X, x(i) \geq p\}$
  return inNode{Left $\leftarrow$ iTree$(X_l, e+1, l)$, Right $\leftarrow$ iTree$(X_r, e+1, l)$, SplitFeature $\leftarrow i$, SplitValue $\leftarrow p$}
end if

Since $c(n)$ is the average of $h(x)$, we can derive an anomaly score for an instance x by normalizing $h(x)$ by $c(n)$. The score s is then defined as

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}. \tag{43}$$

Here, $E(h(x))$ is the average of all $h(x)$ for a set of isolation trees. Path length $h(x)$ is computed by Algorithm 5. This is not an exact computation. Since the tree height is limited, the final part is estimated by taking the average height of a tree of certain size. To interpret the anomaly

Algorithm 5 PathLength(x, T, e)

Input: x - observation, T - an iTree, e - current path length (0 when first called)

if T is an external node **then**
  return $e + c(T.size)$
end if
$q \leftarrow T$.splitFeature
if $x_q < T$.splitValue **then**
  return Pathlength$(x, T.\text{left}, e+1)$
else
  return Pathlength$(x, T.\text{rigth}, e+1)$
end if

score, we distinguish the following cases:

- $E(h(x)) \implies c(n), s \implies 0.5$,
- $E(h(x)) \implies 0, s \implies 1$,
- $E(h(x)) \implies n-1, s \implies 0$.

Notice that longer path lengths lead to a lower anomaly score. Thus, scores tending to 0 indicate normal observations, while scores tending to 1 indicate outliers.

2.5.2 Process monitoring

To use the method in process monitoring, we follow the paper of Susto et al. [18]. a control chart is made by plotting $s(x, \psi)$. The paper uses some preprocessing methods, but they are not explained. Furthermore, they are very specialized for the precise data used. Hence, these will not be adopted in this work. The paper uses a cross-validation approach to ensure an optimal trade-off of type I and type II errors. However, this assumes you have labeled data and is, therefore, not interesting for this work. Hence, the UCL should be determined differently such as discussed earlier in Section 2.3 where the inverse cumulative density function of the KDE of

the test statistics was used. The parameters ψ and t were chosen to be 256 and 100 respectively as the guidelines in the original paper provided. Liu et al. found empirically that these values perform well on a wide range of data.

In the original paper of the method, special attention is given to running times. We will not discuss this since it falls outside the scope of this paper. A comparison was done with a univariate Shewhart chart and with an angle-based outlier detection (ABOD) method [33]. The latter is surprising since no research on process monitoring had been done with ABOD yet to our knowledge.

2.6 Discussion

	GMM	SOM	LOF	SVM	iForest
ARL reported	Yes	Yes	No	No	No
Robustness to outliers	No	No	Yes	Yes	Yes
Comparison method	T^2, Q	T^2, MEMWA	I^2, AO	OC-SVM	ABOD, Shewhart chart
Preprocessing methods	PCA	Windowing, feature extraction, normalization	ICA	PCA	-
Parameters	M, s	ω	k	UCL, Θ, σ	UCL

Table 1: Summary table with information about the GMM, SOM, LOF, Robust OC-SVM, and the iForest methods described in Section 2. For the GMM, preprocessing with PCA does both a dimensionality reduction and decorrelation, although dimensionality reduction is only done when needed. For the Robust OC-SVM, only decorrelation is done by the preprocessing method.

A comparison of the methods can be found in Table 1. Firstly, we note that only the GMM and SOM papers report an ARL. The other papers are more focused on detecting a single fault. Efficiency is then often reported with a confusion matrix or other metrics based on the confusion matrix such as precision and recall which are defined as

$$\begin{aligned} precision &= \frac{TP}{TP+FP}, \\ recall &= \frac{TP}{TP+FN}. \end{aligned} \tag{44}$$

Notice that the papers of the GMM and SOM are also the oldest, from 2004 and 2009 respectively. In addition to this first difference, the methods, which did not report the ARL, are precisely those that are robust to outliers in the training data. These are the LOF, robust OC-SVM, and the iForest. Methods reporting an ARL and which are robust to outliers are not researched. This is sensible because robustness to outliers can only be the case when faults are 'few' with respect to the in-control observations present. The out-of-control ARL is most interesting when the fault is persistent and, therefore, not 'few'.

The GMM and SOM are compared with the Hotelling's T^2 chart. This was the most popular comparison. The Q statistic was used to test the part not explained by PCA. The SOM also compares to the MEMWA chart, which is interesting since this method takes into account history. The robust OC-SVM was compared to the non-robust version from [34]. It performed better,

although it was only compared with training data containing outliers. The non-robust version was compared to the T^2 and Q charts with the data preprocessed by PCA. Both the LOF and the iForest were compared to a rectangular method, adjusted outlierness for the LOF, and univariate Shewhart chart for the iForest. The LOF used the I^2 statistic, common for data preprocessed with ICA. The iForest used ABOD, a method not yet used for process monitoring.

For preprocessing, GMM, LOF, SVM, iForest keep it simple. PCA and ICA are used. Noticeable is that the GMM is the only method that also creates the Q control chart. This is the case since the LOF and SVM methods do not perform any dimensionality reduction when applying these methods. Hence, no Q statistic is necessary. Contrary to these methods, the SOM does a lot of preprocessing. It firstly windows the data which makes it the only method taking into account history. Furthermore, a rather extensive feature extraction is done, which even includes calculating the T^2 statistic of the last sample. After that, normalization is done.

Finally, we will discuss the parameters. Notice that the Robust OC-SVM, and iForest have a UCL that must be set manually. The LOF has a clever trick using the kernel density estimator to get a good UCL automatically. It would certainly be interesting to use this also for the Robust OC-SVM, and iForest. The GMM must be given the number of Gaussians M and the dimension to reduce to using PCA. The SOM usually has a lot of parameters but these were given defaults based on the amount of input data. The LOF has only one paramter k. The Robust OC-SVM needs Θ, a parameter controlling robustness, and σ, the parameter of the RBF kernel function. The iForest has two parameters, the subsampling size ψ and the number of trees t. However, these parameters have default values and do not have to be chosen. For none of the methods it is explained how these parameters can be tuned without using out-of-control data.

All methods proved interesting in their respective paper. This is the case since the methods were catered to their input data. For example, the Gaussian mixture model was tested on data originating from a mixture of 5 Gaussians.

3 Simulation

The previously described methods will be tested for effectiveness on generated multinomial data. Firstly, the data generation simulation setup will be explained. This is followed by Section 3.2 in which the implementation details for all methods are discussed. After this the training process is explained which includes choosing appropriate parameters and an appropriate UCL.

3.1 Simulation description

For generating the data we draw N from a negative binomial distribution $\text{NegBin}(\mu_{nb}, p_{nb})$. Then we draw N i.i.d. samples $Y_1, \dots Y_N$ from a lognormal distribution $\text{LogNormal}(\mu_l, \sigma_l^2)$. We count for m intervals the number of samples Y_i inside the interval. This results in the counts $X_1, \dots X_m$. This will be repeated n times to get the phase I data.

For, the simulation we will generate $n = 1000$ observations with each combination of $\mu_{nb} = 100$ or $\mu_{nb} = 5000$ and $p_{nb} = 0.25$ or $p_{nb} = 0.75$ for the $m = 6$ intervals

$$[0.5, 0.7), [0.7, 1.0), [1.0, 3.0), [3.0, 5.0), [5.0, 10.0), [10.0, \infty).$$

The phase I data will be generated with $\mu_l = 0.5$ and $\sigma_l^2 = 0.75$. This dataset will be the only dataset used to train the model and to set all the parameters and the UCL. This is different from most machine learning papers where out-of-control data is used to tune the model parameters. It is also different from the SOM where a separate dataset was used to set the UCL. After the training, we will investigate the ARL for the following situations:

1. $\mu_l = 0.5$, $\sigma_l^2 = 0.75$ (in control)
2. $\mu_l = 0.45$, $\sigma_l^2 = 0.75$

3. $\mu_l = 0.55$, $\sigma_l^2 = 0.75$
4. $\mu_l = 0.5$, $\sigma_l^2 = 0.5$
5. $\mu_l = 0.5$, $\sigma_l^2 = 1.0$
6. $\mu_l = 0.45$, $\sigma_l^2 = 1.0$

The distributions according to these parameters are also shown in Figure 2.

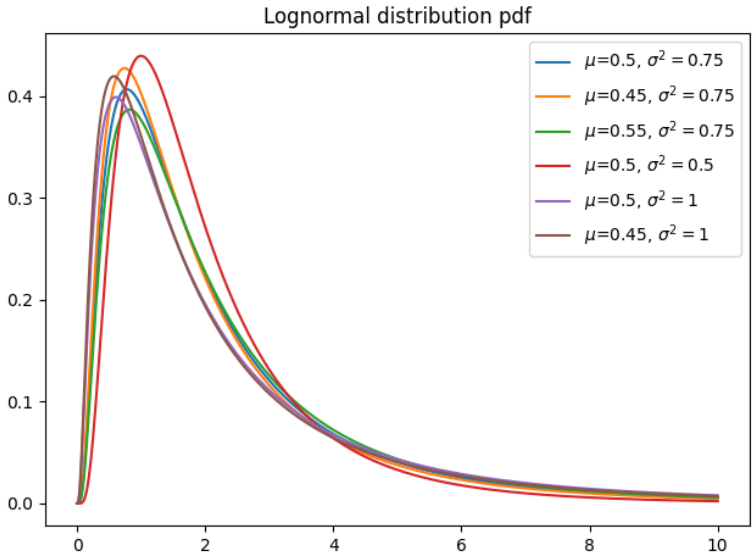


Figure 2: Lognormal distribution for each shift.

3.2 Implementation details

The GMM was simulated using the Scikit-learn library [32] for the GMM algorithms. The PCA module of Scikit-learn was not used since it does not have the features for easily computing the Q statistic. Hence, our own implementation of PCA was used.

The SOM used the Matlab SOM Toolbox [22]. Implementations in other languages are scarce and less feature-rich. To calculate the ARL for parameters μ_l, σ_l^2, we first create $\omega - 1$ observations according to the distribution of the phase I data. Only then can this be appended by data generated with the parameters μ_l, σ_l^2. Due to the windowing preprocessing step, the first preprocessed observation will be from a window encompassing $\omega - 1$ in-control observations and 1 observation generated with parameters μ_l, σ_l^2.

The LOF was simulated using the Scikit-learn library for both the LOF and the ICA algorithms. Notice that the scores returned by this library are the opposite of the LOF scores. In addition, the kNN(x) method will return a set of cardinality k. Hence, this is slightly different from the earlier definition were the cardinality could be larger if there were multiple points at the k-distance of x. The LOF method explained included in the training procedure a way to remove outliers first, after which the method was retrained. This procedure was skipped since by removing the outliers, the method became more sensitive which made the in-control ARL always significantly shorter than wanted. Furthermore, ICA does not work when the data has zero eigenvalues. This occurs when there are a lot of observations linearly dependent and the rank of the data matrix is smaller than m. In this case, a dimensionality reduction is done. This happens more often if the data is discrete with low variance. Tolerance and the maximum number of iterations used in the FastICA algorithm also had to be increased to a tolerance of 0.01 and 5000 iterations

for it to converge consequently. The computation of the inverse CDF of a Gaussian KDE was conveniently left out by Lee et al. [10]. We implemented this by computing the CDF on a discrete set of points. Then, we translate the CDF horizontally by $1-\alpha$ where α is the required significance level. On these points, we use the Scipy package to get a smoothing spline. The Scipy package can then automatically find the roots of this spline using a root-finding algorithm. Notice that since the approximation of the CDF is translated down by $1-\alpha$, the root found will be the inverse of the CDF in $1-\alpha$.

The Robust OC-SVM only has an implemention for RapidMiner, a commercial application. This application lacks the features for computation of ARL. Hence, it was chosen to implement this in python using the module Quadprog to solve the quadratic programming problem. For the computation of the RBF kernel, the Scikit-learn library was used. Duplicate input data was removed such that the RBF kernel matrix is always positive definite. After this, due to floating-point errors, it was necessary to add a small perturbation to the diagonal of H. This was done by adding a diagonal matrix with as diagonal elements $0.01 \cdot \min_{i,j} h_{ij}$. This ensured stability of the algorithm while having minimal impact on the method.

The iForest was simulated using the Scikit-learn library. Again, the scores reported by Scikit-learn are the opposite of the scores defined earlier in this work. We take the number of trees $t = 100$ and subsampling size $\psi = 256$ according to the guidelines of the Isolation Forest paper.

After implementing the methods all the parameters were optimized to deliver good performance. Preferably each method has the same in-control ARL since this allows a better comparison. This proved difficult to achieve through tuning all parameters.

3.3 Training procedure

In the training procedure all models need to be trained and their parameters need to be set. Notice that the UCL of the OC-SVM and iForest need to be set manually as well. The generalized p-chart, GMM, and LOF have a parameter α to set the UCL, the SOM also has a method for setting the UCL for some α. Notice that the amount of times no alarm is raised in the in-control situation follows a geometric distribution with parameter α. So we expect an ARL of $\frac{1}{\alpha}$. In this work, we will try to get an ARL of approximately 100. Hence, we take $\alpha = 0.01$ for these methods. The method for the SOM requires a new dataset to set the UCL according to its procedure. The training set could be used but from experience this provides a far shorter ARL, most likely because the model is slighty overfitted, and performs disproportionally well on the exact training data. All these parameters could be tweaked best if out-of-control data was available. Since this is typically not the case in process monitoring, we try to get the parameters such that the model performs consistently. Unfortunately there is no method to get the optimal results without out-of-control data. The methods do describe procedures which include experimentation and tuning which requires out-of-control data. The SVM paper does mention parameter tuning is difficult, but not too important since the UCL can be shifted to get good results. Algorithm 6 describes the training procedure. *model* is the model used (e.g. LOF). Model settings is a set containing all possible combinations of settings for the model. α is the preferred false positive probability (e.g. 0.027 to get 350 ARL). Firstly, a set of 2-tuples *data_splits* of train and test sets is made. This is done similarly to the procedure for 10-fold cross validation. Each fold is used once as the test set and the other 9 folds as the train set. Then $F_{s,l}$ is made, this is an array in which we store the estimated false positive probabilities for each tuple in *data_splits* using UCL l and settings s. After all these arrays are made, the s, l are returned which had the most false positive probabilities in the range $[0.0075, 0.0125]$ which is close to $\alpha = 0.01$. In practice it can be cumbersome to find a set with appropriate UCLs for the algorithm to try. In this work the *ucls* set consisted of the UCLs which resulted in the desired false positive rate for each of the test sets for each of the model settings. Hence, for each method with a certain parameter configuration 10 UCLs were found.

Notice that this training procedure is rather extensive and time consuming. Although changing

Algorithm 6 Model training algorithm

Input: *model* - the model to train, X - training data, *ucls* - set of ucls, *model_settings* - set with tuples containing different model settings

data_splits is the set of 2-tuples containing training and test data according to 10-fold cross validation
for $s \in model\_settings$ **do**
  for $l \in ucls$ **do**
    $F_{s,l} \leftarrow []$
    for $X_{train}, X_{test} \in data\_splits$ **do**
      $model.train(X_{train}, model\_settings)$
      $F_{s,l}.append(|\{x \in X_{test} \mid model.out\_control(x, l)\}|/|X_{test}|)$
    end for
  end for
end for
return $\text{argmax}_{s,l} |\{x \in F_{s,l} \mid x \in [0.0075, 0.0125]\}|$

the UCL does not require retraining the model, changing other parameters does require retraining. In addition, for each fold the method is trained. So using this procedure each method will perform training $10 \cdot |model\_settings|$ times. To keep training times reasonable, we limited the model settings for each method to the following combinations:

For the GMM:

- $M = 2, s = 5$
- $M = 4, s = 5$
- $M = 2, s = 3$
- $M = 4, s = 3$

For the SOM: $\omega = 3, 5, 10$.

For the LOF: $k = 8, 16$.

For the OC-SVM:

- $\Theta = 0.04, \sigma = 2$
- $\Theta = 0.004, \sigma = 2$
- $\Theta = 0.04, \sigma = 1$
- $\Theta = 0.004, \sigma = 1$

3.4 Results

The results can be found in Tables 2, 3, 4, and 5. Each method was trained with the same dataset of size $n = 1000$ using the previously described procedure. After the training, 1000 run lengths were simulated to find each ARL. The parameters chosen by the training procedure are given under each table as well as the precise CLs. Prior to analyzing the results we will discuss these parameters. For the GMM, most often $M = 4$ and $s = 5$ was chosen except for $\mu_{nb} = 200, p_{nb} = 0.75$ where $M = 2, s = 3$. For the SOM, all models used the smallest option for the window size $\omega = 3$. Since each fold contains 1000 observations, the testing data consists of 100 observations. Preferably, exactly 1 of these is an outlier. A large window size would likely result in no outliers or multiple outliers which is not desired. Hence, it is likely that the algorithm prefers the smaller window size. The LOF used $k = 16$ for the simulations with $\mu_{nb} = 200$. For

the other simulations $k = 8$ and $k = 4$ were used. The SVM used $\Theta = 0.004$ and $\sigma = 2$ in all simulations, which are the same values as chosen in [29].

μ_l, σ_l^2	Gen. p	GMM	SOM	LOF	SVM	iForest
0.5, 0.75	75.84 (71.21, 80.47)	121.79 (114.39, 129.19)	72.99 (68.42, 77.55)	218.12 (204.85, 231.39)	154.68 (145.42, 163.94)	131.30 (122.47, 140.13)
0.45, 0.75	61.38 (57.49, 65.26)	72.22 (67.65, 76.80)	45.65 (42.84, 48.46)	184.16 (172.78, 195.54)	132.93 (124.93, 140.93)	138.38 (130.06, 146.70)
0.55, 0.75	28.57 (26.79, 30.35)	76.21 (70.98, 81.43)	26.35 (24.75, 27.95)	81.16 (75.94, 86.37)	71.41 (67.00, 75.83)	71.09 (66.65, 75.53)
0.5, 0.5	0.03 (0.02, 0.04)	0.09 (0.07, 0.11)	0.62 (0.59, 0.65)	0.12 (0.09, 0.14)	0.20 (0.17, 0.23)	0.84 (0.76, 0.93)
0.5, 1	0.13 (0.11, 0.16)	2.00 (1.84, 2.16)	0.45 (0.41, 0.48)	0.46 (0.41, 0.51)	0.42 (0.37, 0.46)	1.64 (1.52, 1.77)
0.45, 1	0.20 (0.17, 0.23)	2.57 (2.38, 2.76)	0.60 (0.56, 0.64)	0.79 (0.72, 0.87)	0.65 (0.59, 0.71)	2.51 (2.31, 2.70)
CLs	15.09	15.25	6.20	1.63	0.11	0.59
Parameters	α=0.01	Q_{cl}=1117.80 $\alpha = 0.01$ M=4 s=5	ω=3	k=16 $\alpha = 0.01$	Θ=0.004 σ=2	-

Table 2: Table with performance results rounded to 2 decimals, 95% confidence intervals are given. Training data of size $n = 1000$ is generated with $\mu_{nb} = 200$, $p_{nb} = 0.25$. Robust OC-SVM is shortened to SVM. The intervals used are $[0.5, 0.7), [0.7, 0.1), [0.7, 1.0), [1.0, 3.0), [3.0, 5.0), [5.0, 10.0), [10.0, \infty)$, smaller than 0.5 is dropped.

Firstly, it is necessary to examine the in-control average run lengths. Preferably these would be as close to 100 as possible. For the generalized p, this is not the case when $\mu_{nb} = 200$. This is probably because there is a poor fit. Notice that the rule of thumb proposed by Cochran for applying the χ^2 statistic is not met [35]. The rule states that at least twenty percent of the expected frequencies $n_i p_j$ must be above five and none under one. For the simulation we had $n_0 = 187.77$ and $p_0 = (0.074, 0.132, 0.567, 0.151, 0.066, 0.008)^T$. Hence, the expected frequencies are $(13.895, 24.786, 106.466, 28.353, 12.393, 1.502)^T$. The last component has a very low expected frequency and is likely to be below one occasionally, which clashes with the rule of thumb. The GMM has in-control ARLs closest to 100 overall. The other distributionless methods have difficulties getting an accurate in-control run length. The OC-SVM performs best of these methods although it has one high value in the first simulation. The SOM has an extremely low in-control ARL in the fourth simulation. The iForest has an extremely high ARL in the third simulation. The LOF has an extremely high ARL in the first simulation. It is interesting that all methods behave very differently although they were trained on the same dataset.

We will also redo the simulation with $\mu_{nb} = 200, p_{nb} = 0.25$ but with a phase I data set of size $n = 5000$ to investigate whether a larger phase I dataset will allow for better UCLs which give in-control ARLs closer to the desired value. Prior to this, we will discuss the out-of-control results by investigating Table 3 since this table has the in-control ARLs closest to 100. For the change in mean, the SOM performs best and the GMM and SVM perform the worst. The SOM has a window which is an advantage over the other methods which possibly explains the result. However, the windowing also has a downside. This can be seen in the last three rows where the SOM is one of the poorer methods. Only the GMM performs worse. This can be expected since the window of the SOM contains only one outlier and two normal points when the first outlier is presented. Only after three points, the SOM gets a window with solely outliers. When a

μ_l, σ_l^2	Gen. p	GMM	SOM	LOF	SVM	iForest
0.5, 0.75	73.68 (69.35, 78.01)	112.84 (105.75, 119.93)	83.50 (78.15, 88.86)	84.49 (79.29, 89.70)	126.40 (118.35, 134.46)	96.63 (90.71, 102.55)
0.45, 0.75	57.52 (53.95, 61.09)	94.88 (89.40, 100.35)	37.17 (34.95, 39.40)	52.66 (49.26, 56.06)	94.02 (88.25, 99.79)	80.47 (75.42, 85.51)
0.55, 0.75	33.34 (31.25, 35.43)	62.44 (58.47, 66.40)	27.93 (26.12, 29.73)	43.23 (40.60, 45.86)	51.60 (48.33, 54.86)	44.70 (41.79, 47.61)
0.5, 0.5	0.02 (0.01, 0.03)	0.25 (0.22, 0.28)	0.37 (0.34, 0.40)	0.01 (0.00, 0.01)	0.05 (0.04, 0.07)	0.18 (0.15, 0.21)
0.5, 1	0.12 (0.10, 0.15)	5.65 (5.27, 6.03)	0.41 (0.37, 0.44)	0.29 (0.25, 0.33)	0.18 (0.15, 0.21)	0.23 (0.20, 0.26)
0.45, 1	0.20 (0.17, 0.23)	6.44 (6.03, 6.85)	0.62 (0.58, 0.66)	0.52 (0.47, 0.58)	0.32 (0.28, 0.36)	0.32 (0.28, 0.36)
CLs	15.09	11.44	6.32	1.50	0.10	0.57
		Q_{cl}=271.01				
Parameters	α=0.01	α=0.01	ω=3	k=16	Θ=0.004	-
		M=2		α=0.01	σ=2	
		s=3				

Table 3: Table with performance results rounded to 2 decimals, 95% confidence intervals are given. Training data of size $n = 1000$ is generated with $\mu_{nb} = 200$, $p_{nb} = 0.75$. Robust OC-SVM is shortened to SVM. The intervals used are $[0.5, 0.7), [0.7, 0.1), [0.7, 1.0), [1.0, 3.0), [3.0, 5.0), [5.0, 10.0), [10.0, \infty)$, smaller than 0.5 is dropped.

μ_l, σ_l^2	Gen. p	GMM	SOM	LOF	SVM	iForest
0.5, 0.75	96.96 (91.17, 102.75)	110.75 (103.76, 117.75)	57.18 (53.62, 60.74)	69.53 (65.24, 73.82)	112.22 (105.27, 119.16)	274.11 (257.63, 290.59)
0.45, 0.75	0.51 (0.46, 0.56)	0.84 (0.77, 0.92)	0.78 (0.75, 0.82)	0.95 (0.86, 1.03)	0.95 (0.87, 1.04)	3.09 (2.87, 3.31)
0.55, 0.75	0.49 (0.44, 0.55)	0.55 (0.49, 0.61)	0.75 (0.72, 0.79)	0.58 (0.52, 0.64)	0.63 (0.57, 0.69)	2.19 (2.02, 2.36)
0.5, 0.5	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)
0.5, 1	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)
0.45, 1	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)
CLs	15.09	15.25	5.90	1.53	0.10	0.60
		Q_{cl}=1291.27				
Parameters	α=0.01	α=0.01	ω=3	k=8	$\Theta = 0.004$	-
		M=4		$\alpha = 0.01$	$\sigma = 2$	
		s=5				

Table 4: Table with performance results rounded to 2 decimals, 95% confidence intervals are given. Training data of size $n = 1000$ is generated with $\mu_{nb} = 5000$, $p_{nb} = 0.25$. Robust OC-SVM is shortened to SVM. The intervals used are $[0.5, 0.7), [0.7, 0.1), [0.7, 1.0), [1.0, 3.0), [3.0, 5.0), [5.0, 10.0), [10.0, \infty)$, smaller than 0.5 is dropped.

μ_l, σ_l^2	Gen. p	GMM	SOM	LOF	SVM	iForest
0.5, 0.75	100.29 (93.82, 106.77)	80.51 (75.49, 85.53)	29.14 (27.24, 31.04)	65.09 (60.90, 69.28)	86.15 (81.00, 91.30)	103.81 (97.68, 109.94)
0.45, 0.75	0.48 (0.42, 0.53)	0.58 (0.52, 0.64)	0.62 (0.58, 0.66)	1.70 (1.58, 1.83)	0.68 (0.61, 0.74)	1.14 (1.03, 1.24)
0.55, 0.75	0.52 (0.46, 0.57)	0.62 (0.56, 0.68)	0.57 (0.53, 0.60)	0.49 (0.44, 0.54)	0.44 (0.39, 0.49)	0.56 (0.50, 0.61)
0.5, 0.5	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)
0.5, 1	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)
0.45, 1	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)	0.00 (0.00, 0.00)
CLs	15.09	15.25	5.97	1.58	0.10	0.57
Parameters	α=0.01	Q_{cl}=1293.52 α=0.01 M=4 s=5	ω=3	k=4 α=0.01	Θ=0.004 σ=2	-

Table 5: Table with performance results rounded to 2 decimals, 95% confidence intervals are given. Training data of size $n = 1000$ is generated with $\mu_{nb} = 5000$, $p_{nb} = 0.75$. Robust OC-SVM is shortened to SVM. The intervals used are $[0.5, 0.7), [0.7, 0.1), [0.7, 1.0), [1.0, 3.0), [3.0, 5.0), [5.0, 10.0), [10.0, \infty)$, smaller than 0.5 is dropped.

large change happens and the ARL is rather small, methods without the window have a benefit since they are handling one outlier while the SOM is handling several features of two normal observations and one outlier. On the other hand, when the change is small, and ARLs are large, the window only benefits the method.

Now, when redoing the simulation with a larger training set, we get the results in Table 6. Again, the generalized p has in-control ARL which is too small. The other methods have ARLs significantly closer to the desired value of 100, although the in-control ARL of the OC-SVM is still rather small. Either the generalized p and the OC-SVM have smallest out-of-control ARL in all situations except for the mean shift to $\mu_{nb} = 0.55$ when the SOM performs better. Before concluding that these are the best methods, one should consider that they also have the smallest in-control ARL. From the other methods, the SOM and LOF performs best. Similar as earlier, the SOM works best for the slightly larger ARLs and the LOF when the ARLs are very small. The iForest and GMM are not competitive options, although the iForest is still better than the SOM when ARLs small.

μ_l, σ_l^2	Gen. p	GMM	SOM	LOF	SVM	iForest
0.5, 0.75	76.77 (72.09, 81.45)	105.37 (98.57, 112.17)	111.05 (103.98, 118.12)	87.16 (81.69, 92.62)	72.03 (67.42, 76.63)	91.39 (85.75, 97.02)
0.45, 0.75	56.16 (52.73, 59.60)	67.26 (63.03, 71.50)	48.42 (45.47, 51.37)	71.93 (67.47, 76.39)	44.03 (41.31, 46.75)	111.86 (104.63, 119.08)
0.55, 0.75	34.84 (32.66, 37.02)	60.17 (56.45, 63.88)	28.53 (26.70, 30.36)	42.55 (39.87, 45.23)	37.49 (35.16, 39.83)	47.11 (44.12, 50.10)
0.5, 0.5	0.03 (0.02, 0.04)	0.09 (0.07, 0.11)	0.66 (0.63, 0.69)	0.06 (0.04, 0.07)	0.02 (0.01, 0.03)	0.45 (0.40, 0.50)
0.5, 1	0.14 (0.12, 0.17)	1.51 (1.39, 1.63)	0.52 (0.48, 0.55)	0.22 (0.19, 0.25)	0.26 (0.22, 0.29)	0.50 (0.45, 0.56)
0.45, 1	0.23 (0.20, 0.27)	1.93 (1.79, 2.07)	0.72 (0.68, 0.77)	0.42 (0.38, 0.47)	0.40 (0.36, 0.45)	0.89 (0.81, 0.97)
CL	15.09	15.12	6.05	1.46	0.03	0.59
Parameters	$\alpha = 0.01$	Q_{cl}=593.48 $\alpha = 0.01$ M=4 s=5	ω=3	k=16 α=0.01	Θ=0.04 σ=2	-

Table 6: Table with performance results rounded to 2 decimals, 95% confidence intervals are given. Training data of size $n = 5000$ is generated with $\mu_{nb} = 200$, $p_{nb} = 0.25$. Robust OC-SVM is shortened to SVM. The intervals used are $[0.5, 0.7), [0.7, 0.1), [0.7, 1.0), [1.0, 3.0), [3.0, 5.0), [5.0, 10.0), [10.0, \infty)$, smaller than 0.5 is dropped.

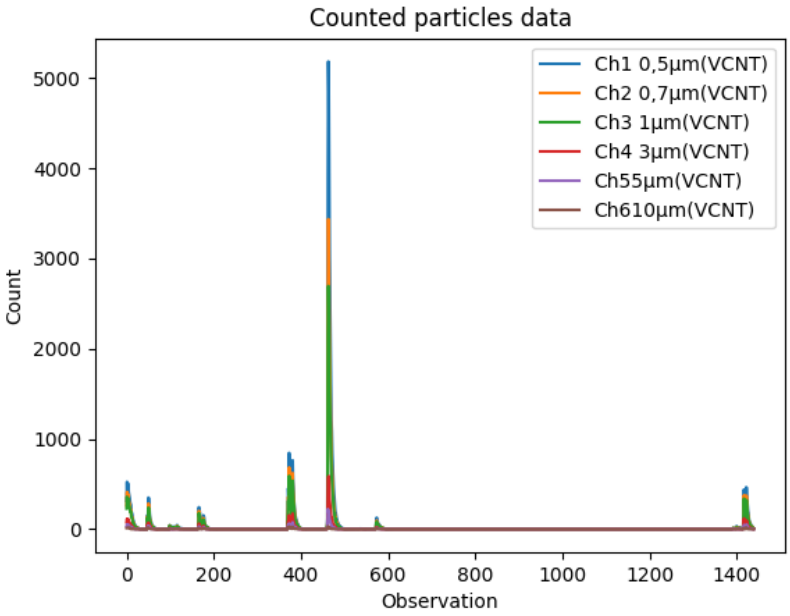
4 Case study

In this section, a case study will be done on count data. More specifically particles are counted and grouped by size in six classes creating a 6 dimensional vector for each observation. 1440 samples are given. Unfortunately, no out-of-control data is provided. Hence, we cannot compare the performance of each method directly. We will focus on how methods differ between each other.

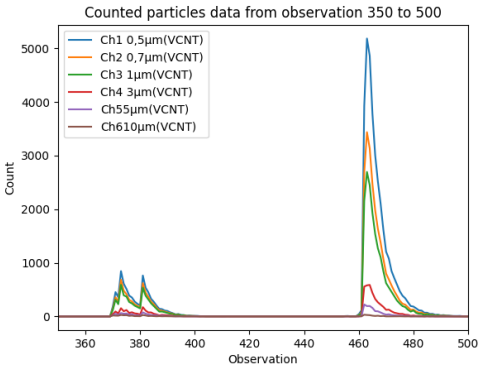
The data, plotted in Figure 3, shows a lot of near-zero data. In total, we observe a few spikes. If we take a closer look at the larger spikes between observations 350 and 500 depicted in Figure 3b, we see that all counts rise quickly, after which they fade out. The peak at observation 370 is followed by a peak at 380. We will test the methods on observations 1400-1440. These are depicted in Figure 3c. Notice that it is similar to the observations between 370 and 400 although the peaks are slightly less steep.

We will use the exact same training procedure as in Section 3. The training set will consist of the first 1000 observations. In Figure 4, the control charts are depicted for the methods with settings shown in Table 7. Observe that the GMM and SVM use parameter settings that were not chosen in any of the previous simulations.

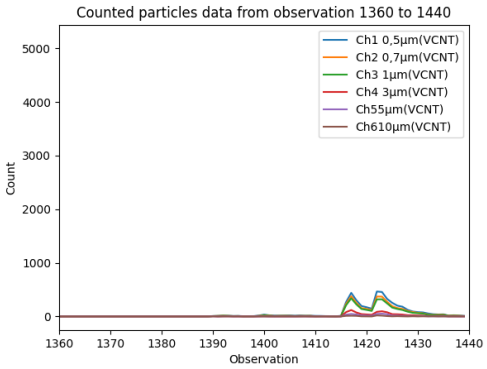
Notice that the SOM only starts after 2 observations due to its window size of 3. The training data starts with a peak at observations 1415. For this peak, we see that all the distribution-free methods stay under the UCL except for the SOM. Considering the window of the SOM, one would expect a smooth curve. Although the window size is small, this is still the method with the most variation. This suggests that a slightly larger window size would be beneficial. The LOF has $k = 4$, so the score is only determined by looking at the 4 neighbors closest to the LOF. Since the training data also had a peak of a similar size, the LOF remains rather low. If the peak consisted of less than k observations, the method would likely raise an alarm. In this way, we have an



(a) Complete data



(b) Observations 350 to 500



(c) Observations 1360 to 1440

Figure 3: Data of particle sensors measuring particles in 6 size classes.

	Gen. p	GMM	SOM	LOF	SVM	iForest
CL	15.09	15.25	5.73	3.30	0.08	0.77
		Q_{cl}=186.64				
Parameters	α=0.01	α=0.01	ω=3	k=4	Θ=0.04	-
		M=2		α=0.01	σ=2	
		s=5				

Table 7: Simulation settings for case study.

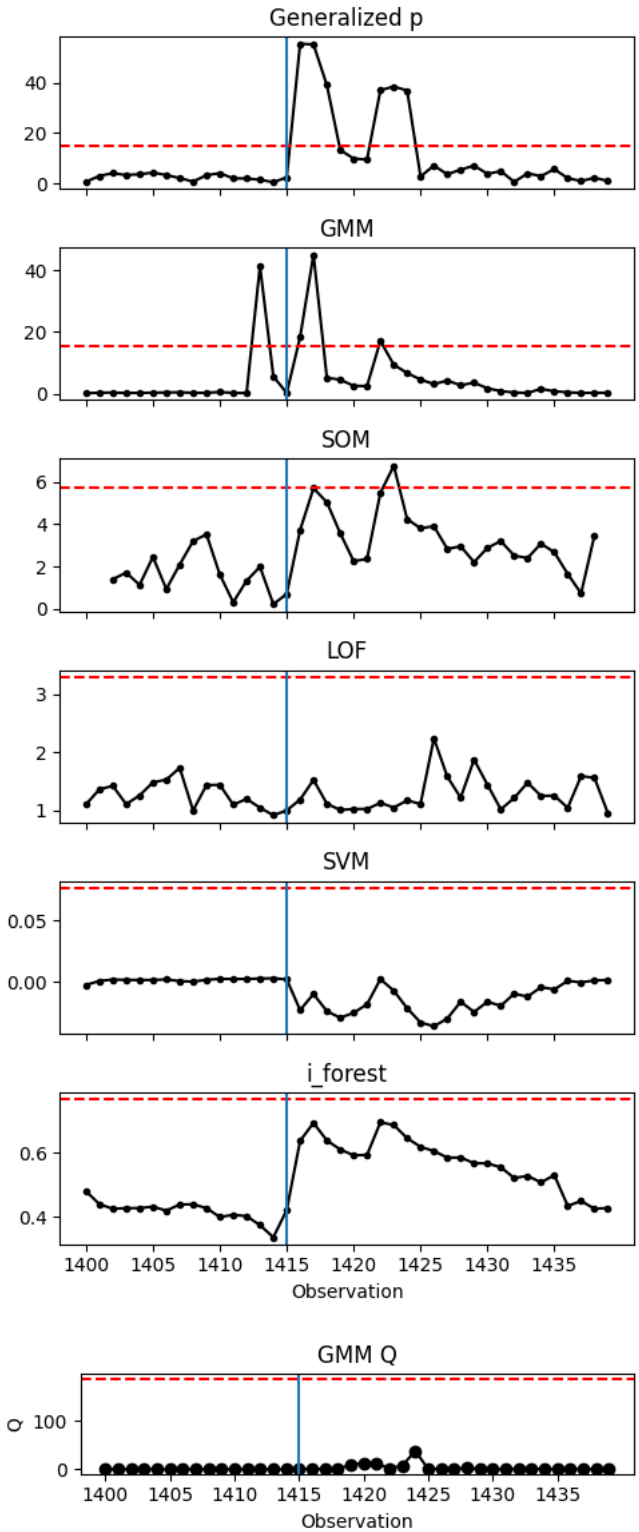


Figure 4: Case study control charts for observations 1400 to 1440. Each model trained on observations 1 to 1000. The UCL is given by the dotted red line. The blue vertical line at observation 1415 indicates the place of the first peak in the test data.

intuitive understanding of how the parameter k can control the method's sensitivity to outliers. Interestingly, the statistic of the Robust OC-SVM becomes smaller starting at observation 1415. Notice that in the training set, there was a large peak. In demarcating the area, the boundary can lay very close to 0 since no observations have values below 0. Hence, these points still get a relatively high score compared to the observations which reside in the middle of the demarcated area. The generalized p-chart seems to perform rather poorly. The training data which consisted of a large amount of near-zero data obviously did not correspond to the assumption needed for the generalized p-chart. The GMM performs similarly, although it is interesting to see it has an out-of-control value before the peak in the training data. Lastly, the iForest has two peaks which correspond to the peaks in the test data. However, they do stay under the UCL.

5 Conclusion and future work

In this work, the goal was to identify which machine learning approaches work best for count data and whether that method performs better than a traditional method. Several methods were explained and analysed in a simulation and in a case study. In this process, we came up with a method for parameter tuning and UCL setting using only an in-control dataset. We chose a wide range of ML models from different categories. This fills in a gap in previous literature, where ML overviews are incomplete and the models are not compared against by means of a simulation on the same dataset.

In the simulation, count data was generated and ARLs were computed for each of the methods. From the results, we see that it is difficult to set the UCL such that the desired in-control ARL is attained. None of the ML methods provided a way to tune its parameters other than to use out-of-control data. In the simulation section, we also presented an algorithm which picks parameters and sets a UCL such that consistent results are obtained. The generalized p also had issues since the requirement for its usage was not met resulting in a small in-control ARL.

The GMM is the most statistical approach. It performed rather poorly since one of its assumptions, the data fitting a mixture of normal distributions, was not met. It did provide a good UCL in almost all cases. The SOM method uses an extensive preprocessing method by computing 7 characteristics. Its preprocessing includes windowing. It did make the method insensitive which resulted in larger ARLs when large variations occured. The LOF is a rather simple method, which performed rather well. It proved interesting in the case study where the parameter k of the LOF provided an intuitive way to reduce sensitivity for the large peaks in the training data. The Robust one-class SVM did perform very well. However, it has a tendency to either have very long or very short ARLs resulting in an undesired in-control ARL. This might be solved by tuning the models parameters well but no information was given on how to do this. The final method is the iForest, which had unremarkable performance. It can still be interesting in other settings with larger datasets and more features since this method's main benefit is its linear time complexity and low memory requirement.

In the case study, rather different results were. The generalized p-chart became very sensitive and raised alarms for almost all points. This shows the strength of the machine learning methods, which are distributionless and generalize better. The LOF and SVM did not raise alarms even for the peeks. The SOM and GMM raised only few alarms. The LOF can be convenient since its parameters are more intuitive to change in comparison to the SVM's or GMM's parameters.

We conclude that when the data follows an multinomial distribution and we can use the method according to the rule of thumb from Cochran, the gereralized p is better. However, when a slight deviation from the multinomial distribution is present or the rule of thumb does not allow for using the method, one of the ML approaches is likely better. If a reliable in-control ARL is desired, the GMM can work although this comes at a sacrifice of performance. If the in-control

ARL can deviate, the Robust OC-SVM performs well. When it is important to detect small changes, the SOM is prefered.

For future work, analyzing the methods with the same preprocessing methods can be of interest. Especially the SOM's preprocessing by windowing the data seems to have a large impact. Comparing these to methods without windowing says very little about the method and more about the preprocessing. In addition, a phase I with some outliers would be interesting to analyze. The LOF, iForest, and Robust OC-SVM should perform better in those cases. Finally, future research could focus on analyzing the impact of the training data's size. In the simulation, we set $n = 1000$ and $n = 5000$. However, for $n = 10, 100, 10000$ the results could be drastically different. In the case that $n = 10000$, it would also be relevant to analyze the running times of the algorithms to see if they can be run in a reasonable time.

References

[1] F. Aparisi, C.W. Champ, and J.C. García-Díaz. A performance analysis of hotelling's χ^2 control chart with supplementary runs rules. *Quality Engineering*, 16(3):359–368, 1 2004.

[2] M. Marcucci. Monitoring Multinomial Processes. *Journal of Quality Technology*, 17(2):86–91, 4 1985.

[3] M. Weese, W. Martinez, F.M. Megahed, and L.A. Jones-Farmer. Statistical learning methods applied to process monitoring: An overview and perspective. *Journal of Quality Technology*, 48(1):4–27, 1 2016.

[4] A. Zimek and P. Filzmoser. There and back again: Outlier detection between statistical reasoning and data mining algorithms, 11 2018.

[5] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, 2 2018.

[6] M. Filippone and G. Sanguinetti. Information theoretic novelty detection. *Pattern Recognition*, 43(3):805–814, 3 2010.

[7] F. Zorriassatine, J. D.T. Tannock, and C. O'Brien. Using novelty detection to identify abnormalities caused by mean shifts in bivariate processes. *Computers and Industrial Engineering*, 44(3):385–408, 3 2003.

[8] S.W. Choi, J.H. Park, and I.B. Lee. Process monitoring using a Gaussian mixture model via principal component analysis and discriminant analysis. *Computers and Chemical Engineering*, 28(8):1377–1387, 7 2004.

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

[10] J. Lee, B. Kang, and S.H. Kang. Integrating independent component analysis and local outlier factor for plant-wide process monitoring. *Journal of Process Control*, 21(7):1011–1021, 8 2011.

[11] H. Ma, Y. Hu, and H. Shi. Fault detection and identification based on the neighborhood standardized local outlier factor method. *Industrial and Engineering Chemistry Research*, 52(6):2389–2402, 2 2013.

[12] B. Song, H. Shi, Y. Ma, and J. Wang. Multisubspace principal component analysis with local outlier factor for multimode process monitoring. *Industrial and Engineering Chemistry Research*, 53(42):16453–16464, 10 2014.

[13] S. L. Jämsä-Jounela, M. Vermasvuori, P. Endén, and S. Haavisto. A process monitoring system based on the Kohonen self-organizing maps. *Control Engineering Practice*, 11(1):83–92, 1 2003.

[14] C.W. Frey. Diagnosis and monitoring of complex industrial processes based on self-organizing maps and watershed transformations. In *CIMSA 2008 - IEEE Conference on Computational Intelligence for Measurement Systems and Applications Proceedings*, pages 87–92, 2008.

[15] J.B. Yu and S. Wang. Using Minimum Quantization Error chart for the monitoring of process states in multivariate manufacturing processes. *Computers and Industrial Engineering*, 57(4):1300–1312, 11 2009.

[16] M. Amer, M. Goldstein, and S. Abdennadher. Enhancing one-class Support Vector Machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description, ODD 2013*, pages 8–15, 2013.

[17] F.T. Liu, K.M. Ting, and Z.H. Zhou. Isolation forest. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 413–422, 2008.

[18] G.A. Susto, A. Beghi, and S. McLoone. Anomaly detection through on-line isolation Forest: An application to plasma etching. pages 89–94. Institute of Electrical and Electronics Engineers (IEEE), 7 2017.

[19] J. Moody and C.J. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1(2):281–294, 6 1989.

[20] D.C. Montgomery. *Introduction to statistical quality control*. John Wiley & Sons, Ltd, 2007.

[21] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1 1982.

[22] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. SOM toolbox for Matlab 5. Technical report, Citeseer, 2000.

[23] A.A. Akinduko, E.M. Mirkes, and A.N. Gorban. SOM: Stochastic initialization versus principal components. *Information Sciences*, 364-365:213–221, 10 2016.

[24] C.W. Champ and W.H. Woodall. Exact Results for Shewhart Control Charts with Supplementary Runs Rules. *Technometrics*, 29(4):393, 11 1987.

[25] A. Hassan, M. Shariff, N. Baksh, A.M. Shaharoun, H. Jamaluddin, A. Hassany, N. Bakshy, A.M. Shaharouny, and H. Jamaluddiny. Improved SPC chart pattern recognition using statistical features. *International Journal of Production Research*, 41(7):1587–1603, 2003.

[26] M.M. Breunig, H. Kriegel, R.T. Ng, and J. Sander. LOF. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data - SIGMOD '00*, pages 93–104, New York, New York, USA, 2000. ACM Press.

[27] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, J. Platt, and R. Holloway. Support Vector Method for Novelty Detection. Technical report, 2000.

[28] D.M.J. Tax and R.P.W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20(11-13):1191–1199, 11 1999.

[29] S. Yin, X. Zhu, and C. Jing. Fault detection based on a robust one class support vector machine. *Neurocomputing*, 145:263–268, 12 2014.

[30] C.C. Chang and C.J. Lin. LIBSVM: A Library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 4 2011.

[31] M. Liu, B.C. Vemuri, S.I. Amari, and F. Nielsen. Total Bregman divergence and its applications to shape retrieval. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3463–3468. NIH Public Access, 2010.

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

[33] H.P. Kriegel, M. Schubert, and A. Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–452, 2008.

[34] S. Mahadevan and S.L. Shah. Fault detection and diagnosis in process data using one-class support vector machines. *Journal of Process Control*, 19(10):1627–1639, 12 2009.

[35] W.G. Cochran. The χ^2 Test of Goodness of Fit. *https://doi.org/10.1214/aoms/1177729380*, 23(3):315–345, 9 1952.

[36] G.A. Carpenter and S. Grossberg. The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. *Computer*, 21(3):77–88, 1988.

[37] G.A. Carpenter, S. Grossberg, and D.B. Rosen. Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4(6):759–771, 1 1991.

[38] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3):37–52, 8 1987.

[39] A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 6 2000.

Appendices

A Fuzzy adaptive resonance theory

Grossberg and Carpenter developed adaptive resonance theory (ART), a theory that suggests an adaptive, yet stable solution for learning systems [36]. In 1998, they developed the system called Fuzzy ART which extends their original system ART1, which only allows binary inputs [37]. By applying fuzzy set theory Fuzzy ART functions also with analog input variables.

In this section we will adopt fuzzy set theory notation. So the $\wedge$ and $\vee$ operators denote minimum and maximum respectively.

The algorithm to train the Fuzzy ART network is shown in Algorithm 7. The input data should be normalized such that each $x \in X$ has parts $x_i \in [0, 1]$. This normalization is done by a coding scheme dependent on a tunable parameter $l > 0$. It maps each x_i to

$$\begin{cases} 0 & x_i < \mu_i - l \\ \frac{1}{2}(1 + \frac{x_i - \mu}{l} & \mu_i - l \leq x_i \leq \mu_i + l \\ 1 & x_i > \mu_i + l \end{cases}$$

Notice that extreme values will be mapped to 0, and 1. If l is smaller, more values will be mapped to 0, 1. After this coding, another form of coding called complement coding is performed. Here $x = (x_1, \ldots, x_M)$ is mapped to $x^c = (x_1, \ldots, x_n, 1 - x_1, \ldots, 1 - x_m)$. Notice that $|x^c| = n$ for

each $x \in X$ with norm $|\cdot|$ defined as the l_1-norm. Hence, no further normalization needs to be done.

Then N categories are made. Each category j has a weight $w_j = (w_{j1}, \ldots, w_{jm})$ with $1 \leq j \leq N$ initialized as 1. Each category starts out uncommitted. For each input x and category j, a choice function can be defined as follows:

$$T_j(x_i) = \frac{x \wedge w_j}{\alpha + |x|}.$$

The algorithm will start iterating until a stopping condition is reached. Each iterations will select an observation from the input data, say $x \in X$. Then resonance can occur or mismatch reset can occur. Consider $J = \text{argmax}_j\, T_j(x)$. Resonance occurs when the match function meets the vigilance criterion

$$\frac{|x \wedge w_J|}{|x|} \geq \rho.$$

If the criterion is not met, mismatch reset will occur, in which case category J will not be selected anymore. This can be achieved by temporarily setting $T_J(x)$ to -1. After this a new J is chosen until the vigilance criteria is satisfied.

If the criterion is met, the algorithm can continue and weight vector w_J shall be updated as follows:

$$w_J = \beta(x \wedge w_J) + (1 - \beta)w_J$$

Notice that the algorithm pseudocode works slightly differently. It does not have a constant number of categories but it creates a category and a weight when needed by appending to w. So first w contains only the category of the x_1 and if needed new categories are appended. This alleviates the problem which occurs when no categories are uncommitted but the vigilance test is not met for any of the categories.

Algorithm 7 Simple Fuzzy ART Training algorithm

Input: $X = \{x_1, \ldots, x_n\}$ - training data, α - choice parameter, β - learning rate, ρ - vigilance parameter

$x_i \leftarrow x_i^c = (x_i, 1 - x_i)$ for each i
$w \leftarrow x_1$
while Stop criteria does not hold **do**
    for x in X **do**
        if $\frac{|x \wedge w_i|}{\alpha + |x|} < \rho$ for all i **then**
            Append w with x
        else
            $J \leftarrow \text{argmax}_i \frac{|x \wedge w_i|}{\alpha + |x|}$
            $w_J \leftarrow \beta(x \wedge w) + (1 - \beta)w$
        end if
    end for
end while

After the training of the model, new data should be normalized. A control signal should be raised whenever w_J does not pass the vigilance test. Optionally, a control chart can be created by plotting the value for w_J with UCL ρ.

B PCA for process monitoring

Principal component analysis is a technique which performs a change of basis and can be used for dimensionality reduction of data. This, and the explanation on how to compute principal

components, a subject which we will not discuss here is explained in [38]. A principal component is a unit vector which maximizes the variance of the projected data onto the component. Furthermore, the ith principal component must be orthogonal to all the j principal components with $j < i$. Notice that the principal components with larger indices contribute less to variance in the data. Hence, they are less important, since they do not describe a lot about the observations.

Let s be the dimensionality to which we reduce. Given n data points of dimension p, we can create an $n \times p$ matrix X containing the data. We assume it is scaled to zero mean.

Now, we start with the model $X = TP_s' + E$. The columns of P_s, the loading matrix, are actually the eigenvectors of the covariance matrix of X corresponding to the s eigenvalues in descending order $(\lambda_1, \ldots, \lambda_s)$. T is called the score matrix. E comprises of the residuals made using this projection.

C ICA

Independent component analysis (ICA) is a statistical method which separates a multivariate signal into independent sub-parts, so called latent variables which are assumed to be non-Gaussian. For the variables x, with latent variables s, there exists a matrix A such that $x = As$. So, $s = A^{-1}x$. We call $W = A^{-1}$, the demixing matrix. Before the computation of W, whitening needs to be performed. This is a preprocessing method which removes correlations in the data. This is done by finding V such that for $y = Vx$ we have $\text{corr}(y_i, y_i) = 1$ and $\text{corr}(y_i, y_j) = 0$ for $i \neq j$. This is done by choosing $V = C^{-1/2}$ with C the correlation matrix of x. Computation of W is then commonly performed by the FastICA algorithm [39].