DTU Process and System Engineering Center

Department of Chemical and Biochemical Engineering

Framework for data-driven fault detection models in fermentation processes

Master Thesis

Framework for data-driven fault detection models in fermentation processes

Master Thesis

July, 2024

By

Maximilian Klein

Cover photo: Vibeke Hempler, 2012

Published by: DTU, Department of Chemical and Biochemical Engineering, Søltofts Plads, Building 228A, 2800 Kgs. Lyngby Denmark

https://www.kt.dtu.dk/research/prosys

Preface

This Master's thesis was written at the Department of Chemical and Biochemical Engineering in the Process and System Engineering Center from January until July 2024. Prof. Krist Gernaey, Carina Gargalo and Mads Kaiser-Albæk supervised the work.

I declare that this document is an original work of my authorship and that it fulfills the requirements of DTU's code of honour.

Abstract

The thesis presents the development and application of a framework for creating data-driven fault detection models using synthetic data. The framework is applied in the context of fermentation processes, which are becoming increasingly more complex, making fault detection and diagnosis more challenging. While traditional model-based approaches to fault detection are resource-intensive and require considerable process knowledge, data-driven models can be developed rapidly but require large labeled datasets, which are often unavailable. This work offers a solution to this challenge by utilizing large synthetic datasets.

The framework consists of two parts: data generation and fault detection model development. In the first part, a fed-batch fermentation model is modified to simulate fault events and biological variance. With that, large synthetic datasets are generated through Monte Carlo simulations. In the second part, various fault detection models are developed and optimized based on multiway principal component analysis (MPCA), partial least squares discriminant analysis (PLS-DA), support vector machines (SVM), and artificial neural networks (ANN). The ANN model is further developed for fault diagnosis, including additional feature engineering.

The model evaluation shows that simpler linear models like MPCA and PLS-DA are limited in detecting certain faults but show better generalization capabilities. While the SVM model outperforms the ANN model in terms of fault detection, it requires significantly more computational power. The ANN models demonstrate the lowest false alarm rates and prove to be effective in both fault detection and diagnosis. However, their generalization capabilities need to be further improved.

This work provides a foundation for developing robust data-driven fault detection models for fermentation processes based on synthetic datasets. By validating the models with real process data and improving them based on the results, the framework can be extended to provide robust fault detection models for practical application.

Acknowledgements

I would like to thank a few people who were an important part of this journey. To Carina, for her continuous guidance throughout my thesis. Her mentoring abilities, paired with her modeling expertise, were essential for this work, especially in times of doubt. To Mads, whose hands-on process expertise made this project come alive. Our discussions helped me tremendously in giving this work its practical relevance. To Krist, for his insightful feedback and for guiding the broader context of the thesis. And finally, to my partner Mona, to whom I am deeply grateful for supporting me throughout this journey as well as for her many hours spent reading and giving feedback on my work.

Abbrevations

ACF	Auto-Correlation Function
ANN	Artificial Neural Network
DO	Dissolved Oxygen
$\boldsymbol{dV/dt}$	Rate of Change of Volume
F	Feed Flow Rate
FAR	False Alarm Rate
FDR	Fault Detection Rate
$F_{\mathbf{evap}}$	Water Evaporation Rate
MPCA	Multiway Principal Components Analysis
MPLS	Multiway Partial Least Squares
m_s	Maintenance Coefficient for Substrate
n	Moving time window size
NIPALS	Nonlinear Iterative Partial Least Squares
OOC	Out of Calibration
ODE	Ordinary Differential Equations
OOD	Out of Domain
OUR	Oxygen Uptake Rate
PCA	Principal Component Analysis
PLS-DA	Partial Least Squares Discriminant Analysis
PLS-R	Partial Least Squares Regression
RBF	Radial Basis Function
RMSE	Root Mean Squared Error
$\rho_{\mathbf{broth}}$	Density of Broth
ρ_f	Density of Feed
ρ_r	Rate Parameter Vector
$\rho_{\mathbf{water}}$	Density of Water
S_m	Stoichiometric Matrix
SVM	Support Vector Machine
$Y_{XS}^{\mathbf{true}}$	True Stoichiometric Coefficient from Substrate to Biomass

Contents

1 Introduction

1.1 Motivation

Fermentation processes have become increasingly complex over the last decades. While this complexity brings some advantages, detecting faults also becomes more complicated, requiring more accurate and specific models for fault detection and diagnosis. Fault detection is concerned with the presence of a fault, while fault diagnosis determines the specific type and location of the fault. For simplicity, both “fault detection” and “fault diagnosis” are referred to as fault detection in this work. Developing fault detection models based on fundamental knowledge about a process is often resource-intensive. Data-driven approaches can alleviate this by inferring the relationships between the process and faults from historical data. However, this requires large datasets with a substantial number of labeled fault events, which is not given in many cases. One way to solve this problem is by synthesizing balanced and labeled datasets from process models.

Consequently, this work presents a framework for developing data-driven fault detection models based on synthetic data and applies the framework to a fed-batch fermentation process.

1.2 Approach

The framework developed in this work provides a systematic approach for the development of real-time fault detection models. It provides a basis for developing a variety of data-driven fault detection models based on large synthetic datasets. These models should be able to raise real-time fault alarms so that operators can react accordingly, avoiding sub-optimal process conditions and failures.

A fed-batch fermentation model provided by Albæk et al. was used for the application of the framework [1]. Based on expert knowledge, the five most common fault types of this process were identified and implemented in the fermentation model. These faults are listed in the following.

- Steam entering the tank through the feed stream.
- Steam entering the tank through defective steam barriers.
- Off-gas spectrometer being out of calibration, leading to deviations in oxygen-uptake rate (OUR) measurements.
- Blocked spargers, reducing airflow into the tank.
- Airflow meter being out of calibration.

In the first step of the framework, large datasets with both regular and faulty batches are synthesized based on Monte Carlo simulations. Realistic biological variance is also introduced into the datasets. In the second step, the goal is to develop and optimize a selection of data-driven fault detection models. In the final step, a fault detection model is selected after a thorough evaluation of all models.

The four machine learning methods used in this work are multiway principal component analysis (MPCA), partial least squares discriminatory analysis (PLS-DA), support vector machines (SVMs), and artificial neural networks (ANNs).

MPCA was chosen since it is the most commonly used method in fault detection [2]–[5]. PLS-DA is also widely used but offers different advantages than MPCA as it is a supervised learning method [6], [7]. SVMs and ANNs offer the ability to capture more complex patterns and relationships. SVMs can create highly nonlinear decision boundaries for fault detection [8], [9]. ANNs on the other hand are proven to have complex pattern recognition abilities, making them a relevant method for fault detection [8], [10].

The framework presented here is intended to be used as a flexible open-source code base, contributing to efforts for more reproducible open science. By generating synthetic datasets with different process parameters, these fault detection models can be adjusted to various new conditions, making them flexible and relevant to the wide operating space of the fermentation process. Furthermore, this work is meant to be expanded upon. This could involve adding new faults, refining current faults, or applying the framework to a different process.

The codebase with all developed models and additional resources for this project can be found on GitHub at https://github.com/datameerkat/fermentation-fault-detection.

1.3 Outline

The thesis is structured into four chapters. The general structure of the fermentation model is explained in Chapter 2, focusing on aspects relevant to generating large datasets for fault detection. Furthermore, introductions to fault detection for fermentation processes and the machine learning techniques used in this work are given.

The framework for developing data-driven fault detection models is described in detail in Chapter 3, followed by a detailed explanation of how this approach was applied to the fermentation model of Albæk et al.

Chapter 4 covers the results and discussion of developing fault detection models with this approach. It begins with insights into the datasets, followed by showing how the performance and optimization of the different types of models. Finally, all models are evaluated and compared. The thesis finishes with conclusions about this work and a glimpse into future perspectives.

2 Background

2.1 Fed-batch fermentation model

This work is based on a fermentation model developed by Albæk et al. [1], [11]. The following section gives an introduction and overview of this model, focusing on aspects relevant to modeling and detecting fault events. For further theory on this model, please refer to Albæk et al. [1], [11].

The model aims to simulate a submerged fed-batch fermentation process of a filamentous fungus for the production of enzymes. It was previously used to study various aspects of this process. Albæk et al. [1] investigate different agitation, aeration, and agitator types. In a subsequent study, [11], the focus was on evaluating the energy efficiency of the process depending on the agitation, aeration, and headspace pressure. In this work, the model will be used to simulate the effects of fault events and biological variance to generate large datasets, which are used to train fault detection models.

The fermentation model is constructed as a state-space model. A state-space model uses a set of differential equations to evolve a collection of state variables over time, based on initial conditions. In the case of this model, the state variables are the biomass concentration X, the substrate concentration S, the enzyme concentration E, the dissolved oxygen concentration DO, and the submerged volume V. For each time step, the system is solved for the state variables using a set of ordinary differential equations based on the mass balances of the state variables (see Eqs. 4-9 in [1]). To solve this set of ODEs, several physical and empirical equations are used to determine the parameters of the system such as the driving force for the oxygen transfer or the volumetric mass transfer coefficient (kLa), which are crucial to solve for the concentration DO. Calculating kLa is based on an empirical equation, which also accounts for the effects of viscosity since fermentations with filamentous fungi exhibit an increase in viscosity over time. A simple flow chart showing the state-space structure of the model is depicted in Figure 2.1.

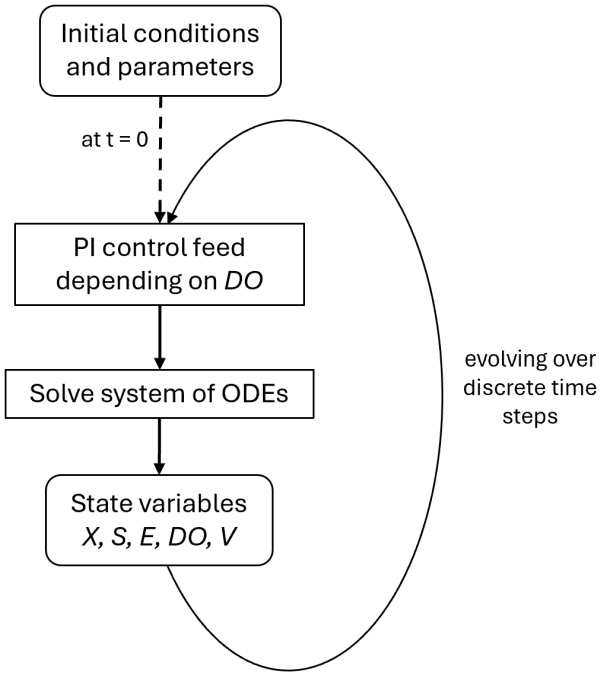


Figure 2.1: Flowchart of the state-space fermentation model. Starting with a set of initial conditions and parameters, the model controls the feed rate through a PI controller and solves a system of ODEs for the state variables. This is reiterated for every time step, evolving the state variables.

Another essential part of the mass balances is the specific growth rate μ. The assumption of an immediate uptake of the substrate by the fungus combined with very low substrate levels in the broth leads to a quasi-steady state of the substrate concentration. In this way, μ can be determined from the mass balances as follows:

$$\mu = \left(\frac{c_f \cdot F - S \cdot \mathrm{dVdt}}{V \cdot X} - m_s \right) \cdot \frac{1}{Y_{XS}^{true}} \tag{2.1}$$

where c_f is the substrate concentration in the feed, F is the feed, m_s is the maintenance coefficient for the substrate, and $Y_{\mathrm{XS}}^{\mathrm{true}}$ the "true" stoichiometric coefficient from the substrate to the biomass. The calculation of μ, as well as the substrate and oxygen uptake calculation, require the true yield coefficients Y_{XS}^{true} and $Y_{\mathrm{XO}}^{\mathrm{true}}$, which represent the theoretical maximum yield under optimal conditions. The rate of change of the volume dVdt follows from the volume balance of the reactor

$$\mathrm{dVdt} = \frac{\rho_f \cdot F + \rho_{\mathrm{water}} - F_{\mathrm{evap}} + Y_{SO} \cdot F \cdot c_f - Y_{SC} \cdot F \cdot c_f}{\rho_{broth}} \tag{2.2}$$

where ρ_{f}, ρ_{water} and ρ_{broth} are the densities of the feed, water and broth, F_{evap} is the water

evaporation rate, and Y_{SO} as well as Y_{SC} are the "observed" yield coefficients.

The rate of change of X, S, E, and DO is determined by the stoichiometric matrix S_m and the rate parameter vector $\boldsymbol{\rho_r}$. Each column in S_m corresponds to one of the four state variables, while the rows correspond to biomass growth (μ) and the maintenance metabolism of the fungus, divided into the substrate (m_s) and oxygen consumption (m_o).

Rate parameter	**Stoichiometric Matrix** (S_m)			
Vector ($\boldsymbol{\rho_r}$)	Biomass (X)	Substrate (S)	Enzyme (E)	Dissolved Oxygen (DO)
μX	1	Y_{XS}^{true}	$\frac{Y_{SE}}{Y_{SX}}$	$-Y_{XO}^{true}$
$m_s X$	0	-1	0	0
$m_o X$	0	0	0	-1

The rate of change vector for X, S, E, and DO is then given by:

$$r = S_m \cdot \boldsymbol{\rho_r} \tag{2.3}$$

The biokinetic parameters, namely the "observed" and "true" yield coefficients as well as the maintenance coefficients, are part of a configurable set of parameters. Additionally, the model allows for different configurations of initial volume and biomass. As mentioned earlier, the model is flexible with regard to the reactor and impeller configuration. The empirical parameters for the viscosity and kLa can be adjusted to reflect different scenarios. The aeration of the system is based on a pre-defined profile, achieved by controlling the DO through the feed rate. Upper and lower boundaries are defined for the feed rate.

Although the model allows flexibility in different parts of the fermentation system, it is fixed in some regards. The most relevant aspects here are the temperature and the pH. The model assumes that the pH is controlled through ammonia dosing and the temperature is kept constant by a process control system. All empirical and biokinetic parameters are therefore tied to a specific temperature and pH.

2.2 Fault detection in fermentation processes

As discussed in Chapter 1, the development of fault detection models began several decades ago due to their importance for the safety and reliability of processes. These earlier fault detection approaches were mostly model-based techniques and expert systems, using fundamental mathematical models and expert knowledge, respectively. The increasing computational power in the last three decades gave rise to data-driven approaches relying on process-historical data. The following paragraphs will provide an overview of all three types of models, focusing mainly on data-driven approaches since the models developed in this work belong to this third category.

Model-based approaches for fault detection use mechanistic representations of a process to detect abnormal process conditions. The process model is usually a state estimation method, e.g., a state-space model. For this, state variables are defined and recursively solved for using online measurements as input and a set of physical and empirical equations. State variables are usually non-measurable variables that are related to specific faults. Fault decisions can then be made based on the pattern of these state variables. [12]

Expert systems use the knowledge of operators and engineers to define precise algorithms to detect faults [4]. A common way to build a rule-based expert system is through probability theory or fuzzy rules. Fuzzy rules describe a system consisting of a set of IF-THEN statements based on knowledge about the process [4]. Therefore, such a system is transparent in its reasoning and it is accessible as it does not require advanced programming knowledge [13].

Data-driven approaches build empirical models based on historical process data [4]. This statistical approach does not require fundamental knowledge of the underlying process. Instead, the model "learns" the underlying relationships within the process through training on historical data. The first data-driven approaches for fault detection of batch processes were pioneered by Nomikos and MacGregor in the 1990s [4], [14]. They developed multiway principal component analysis (MPCA) and multiway partial least squares (MPLS) approaches to monitor such processes based on process historical data. Here, "multiway" refers to organizing the data batch-wise. These advances in using multivariate statistical techniques to monitor batch processes proved to be a breakthrough, as both MPCA and MPLS are still commonly applied today [3], [15], [16]. One key advantage of these multivariate approaches is their ability to handle highly correlated batch process data. MPCA models have an additional benefit: they do not require faulty batches for calibration, as they are trained purely on normal batch data

(more in Section 2.3.1). This alleviates the need for balanced datasets between normal and faulty batches, which would typically be required to train a data-driven model for fault detection [17].

One disadvantage of multivariate approaches such as MPCA and MPLS is their assumption of linear relationships. Later advancements in computational power enabled the application of techniques capable of capturing non-linear relationships. The two most common of these techniques are support vector machines (SVMs) and artificial neural networks (ANNs). Both of these as well as MPCA and MPLS will be explained in more detail in the following sections.

2.3 Machine Learning-based techniques for fault detection

In this work, four different machine learning approaches were applied to fault detection in the aforementioned fermentation process. The following section gives an introduction to each method with references for further theoretical background. Additionally, Table 2.1 gives a short comparison of the machine learning methods based on relevant categories, e.g. their primary purpose or their hyperparameters.

2.3.1 Multiway principal component analysis

The general goal of PCA is to project the original variables of a dataset into a reduced space defined by new coordinates, called principal components. While still explaining a large amount of the variance in the original variables, the PCA model describes the data in a simpler and more meaningful way. The principal components are arranged in descending order of variance explained: the first component accounts for the highest amount of variance, followed by the second, and so on. A PCA model is characterized by its loadings vectors, which are the directions in the original feature space along which the data varies the most. Each loadings vector corresponds to a principal component. In this way, new data points can be projected onto the PCA space through these loadings vectors [4].

A set of historical batch data can be seen as a three-dimensional array, with its dimensions being time, process variables, and batches. This dataset can be unfolded into a two-dimensional array by concatenating the batch-wise arrays along the time axis. Considering this, MPCA is equivalent to performing ordinary PCA on an unfolded 2D version of a 3D array, since the underlying concept of MPCA and PCA is the same [4], [18]. In general, PCA assumes the data to be Gaussian distributed [2]. Its computational complexity is low, as it only requires one singular value decomposition of the input data matrix [2].

As PCA is an unsupervised learning method, statistical tests are used to detect faults. In the

Table 2.1: Summary of the machine learning approaches used in this project.

Category	**MPCA**	**PLS-DA**	**SVM**	**ANN**
Learning type	Unsupervised	Supervised	Supervised	Supervised
Primary purpose	Dimensionality reduction	Regression	Classification	Classification Regression Clustering
Assumption on input data	Gaussian distributed	Gaussian distributed	-	-
Adaptability	Static	Static	Static	Adaptive
Hyperparameters	Principal components Detection threshold	Components	C (regularization parameter) Kernel function Kernel parameters	Hidden layers & nodes Batch size Learning rate Activation function Optimizer type
Relevant variants	Kernel PCA Recursive PCA Dynamic PCA	Kernel PLS Recursive PLS	One-class SVM Recursive SVM	CNN RNN

case of a (M)PCA model, these are usually Hotelling's T^2 statistic and the squared prediction error (SPE). Both serve different purposes and will be explained shortly.

Hotelling's T^2 statistic is based on the sum of the normalized squared scores. A score is obtained by projecting a new observation onto the PCA space. A low score in PCA indicates that the observation is close to the mean of the data on which the PCA model was calibrated. The sum of normalized squared scores therefore indicates the distance of a set of observations to the center of the model space. A fault can be detected by defining a threshold, above which a fault alarm is triggered. This threshold is usually based on the distribution of the Hotelling's T^2 statistic for normal operating conditions.

The SPE focuses on a different aspect when projecting a new observation onto the PCA space. Rather than evaluating the structured part of the PCA model, namely the scores, it uses an unstructured part, namely the residuals. The SPE is defined as the sum of squared residuals and answers the question of how much of the variation of a new observation is captured by the PCA model. Similar to the T^2 statistic, an empirically based threshold is used to detect faults. The application of both statistics in the context of the framework is further explained in Chapter 3.3.3.

The workflow of applying MPCA to fault detection is summarized in Figure 2.2. For further details on MPCA for fault detection and process monitoring refer to Nomikos and MacGregor [4].

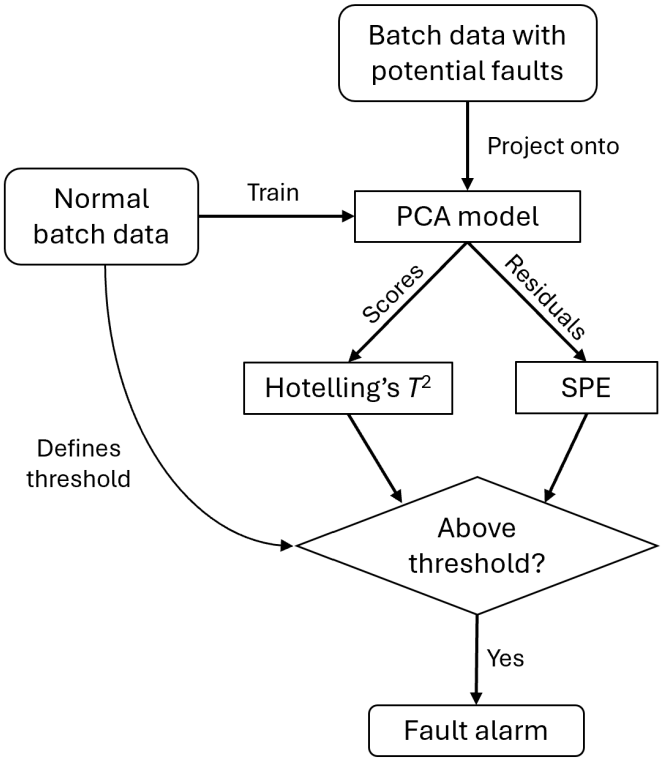


Figure 2.2: Workflow of applying MPCA to fault detection via Hotelling's T^2 statistic and squared prediction error (SPE). All input data needs to be standardized.

Besides the standard PCA approach, several variants of PCA exist. Recursive PCA can handle data in batches, enabling it to be incrementally trained. Dynamic PCA, on the other hand, can capture correlations in time series data. Another variant is kernel PCA, which eliminates the limitation of linear relationships, as its kernel function can be non-linear [19].

2.3.2 Partial least squares discriminant analysis

Partial least squares discriminant analysis (PLS-DA) is derived from partial least squares regression (PLS-R). While PLS-R aims to form a regression model to predict a continuous variable, PLS-DA is used for classification tasks. This is achieved by using discrete target variables. For a two-class problem, one target variable can be defined, where the two classes are assigned the labels 0 and 1. For problems with more than two classes, each class is assigned a one-hot-encoded target variable. PLS-DA uses slightly different algorithms for each case, namely PLS1 and PLS2. For further details on the PLS-DA algorithms refer to Brereton and Lloyd [20].

PLS-DA captures the most significant variation in the input variables while finding an optimal separation between the target classes. In this way, PLS-DA performs dimensionality reduction and classification at the same time. The output of the PLS-DA model for a new observation is a continuous value between 0 and 1. This value can be interpreted as the probability that a

fault is present for this set of input variables.

Similar to PCA, PLS assumes the input data to be Gaussian distributed. Additionally, it requires a well-defined input-output relationship. Its computational complexity is higher compared to PCA, as it carries out k single value decompositions, where k is the number of iteration steps needed to satisfy the tolerance requirement when using the NIPALS algorithms [2].

2.3.3 Support vector machines

Support vector machines (SVM) are a common machine learning technique for classification tasks in nonlinear systems. They were first developed by Vapnik and Chervonenkis in the 1990s. The theory of this section is based on Vapnik's book about statistical learning, which offers further explanations beyond the scope of this work [21].

The goal of a support vector machine (SVM) is to construct a decision boundary to perform classification in a potentially nonlinear system. It achieves this by mapping the input vector into a higher-dimensional space through a nonlinear mapping, based on a chosen kernel function. In this space, an optimal separating hyperplane is constructed as the decision boundary.

An essential part of an SVM is the "kernel trick". This trick allows for computing the similarity between two points in a higher dimensional space without ever explicitly transforming the data into this higher dimensional space. In this way, the computation required for training an SVM is reduced significantly. A SVM can operate with different kernel functions. Common ones are the linear kernel, the polynomial kernel and the radial basis function kernel (RBF kernel). Out of these three, the RBF kernel can create the most flexible and complex decision boundaries by measuring the Euclidean distance between points in the transformed feature space.

Although SVMs can construct highly nonlinear decision boundaries, they tend to generalize better than ANNs [9]. On the other hand, even when using the kernel trick, the computational complexity for training an SVM increases drastically for larger datasets. Depending on the algorithm used for training, the computational complexity for training an SVM is between $O(n^2)$ and $O(n^3)$, where n is the number of observations in the training set [22].

As with other machine learning techniques, there exist several variants of SVMs. One variant similar to the MPCA approach is the one-class SVM [9]. It is similar as it also only requires normal batch data to be trained. The goal of this variant is to find a decision boundary that encapsulates the normal data. Any point outside of this boundary is considered a fault. An-

other variant relevant to fault detection is the recursive SVM, which can be trained in batches instead of being trained all at once. This allows for re-training of the model once new data becomes available.

2.3.4 Artificial neural networks

Artificial neural networks (ANN) became essential to machine learning during the rapid increase in computational power in the last two decades. ANNs offer a high degree of flexibility and potential complexity, which also opens up many pitfalls. ANNs have several use cases like regression, classification, or clustering, but this work focuses on classification. The following explanations are based on Michael Nielsen's extensive web page about deep learning [23]. Refer to that page for information beyond the explanations given below.

The basic unit of an ANN is a neuron. The simplest form of connecting neurons to a network is a feed-forward neural network (FFNN). The basic structure of a FFNN is shown in Figure 2.3. Feed-forward means that all data moves from an input layer to an output layer in one direction. Each neuron in an input layer corresponds to one input variable, whereas each neuron in the output layer corresponds to a class. Between the input and output layers are hidden layers. The architecture of a FFNN is mainly defined by the number of hidden layers and the number of neurons in each hidden layer. The basic workflow of a FFNN in the case of one hidden layer is as follows:

1. Inputs are sent to each neuron in the hidden layer and multiplied by the connection weights.
2. A bias is added in each neuron.
3. The activation function is applied in each neuron.
4. Values are sent from neurons in the hidden layer to output layers.
5. The bias and the activation function are applied in output neurons.

When applying an ANN for classification, the softmax function is usually applied to the output of the network to ensure that all outputs sum to 1. These outputs can therefore be thought of as probabilities.

An ANN learns from a dataset by adapting its set of weights and biases. The algorithm for this is called backpropagation. In essence, it uses the partial derivative of the loss function with respect to the weights and biases in order to update these. The most commonly used loss function in ANNs for classification is the cross-entropy loss function. This function quantifies

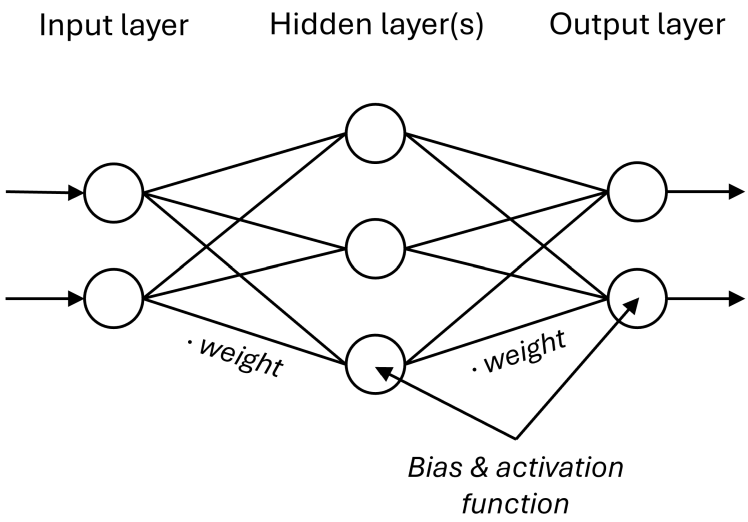


Figure 2.3: Structure of a basic feed-forward neural network.

the difference between the predicted probabilities and the actual classes. The objective is to minimize this difference, so the predicted probabilities are as close as possible to the actual labels.

With this setup, ANNs have proven to be a capable machine learning technique. Theoretically, according to the universal approximation theorem, ANNs can approximate every function given a complex enough network. Additionally, they are naturally capable of learning adaptively without any extensions to the model. This makes them suitable for large datasets and for learning incrementally from new data. They also offer multiple ways of customizing a network to a specific problem by changing its various hyperparameters, like the number of hidden layers/neurons, the activation function, the loss function, the learning algorithms, etc.

As neural networks can become rather extensive and complex due to their architecture and number of hyperparameters, several pitfalls exist when training ANNs. One common problem is the issue of vanishing gradients, where the network essentially stops learning after a few iterations. Additionally, as neural networks can theoretically fit any function, overfitting must be avoided. Numerous regularization techniques exist to ensure that a network generalizes well enough outside of its training dataset. The vast amount of hyperparameters of an ANN leads to a high amount of possible combinations, making the optimization of hyperparameters a challenging task.

3 Methodology

3.1 Methodological framework

The framework presented in this section provides structured practical guidance for developing fault detection models based on synthetic datasets by outlining the sequence of methods applied. A visual representation of the framework can be seen in Figure 3.1. The framework can be divided into two main parts: (I) data generation and (II) fault detection model development. Part I, data generation, encompasses all the steps needed for the creation of large synthetic datasets. To simulate the five different fault events, and replicate the biological variance of the system, modifications are made to the existing model developed by Albaek et al. A set of process parameters and initial conditions is defined according to the specific process setup. Using this modified fermentation model, datasets are generated through Monte Carlo simulations, which randomly sample parameters to simulate two kinds of variance in the fermentation model: biological variance and variance in fault events. Part II, the development of the fault detection models, starts by pre-processing the datasets. This always involves the standardization of the data and can be extended with feature engineering techniques. To develop a model, four different datasets need to be defined: a training, a validation, and two different test datasets. All models are trained based on the training dataset. Following this, each model's hyperparameters are optimized according to their performance on the validation dataset. Following this, the performance of all models is compared based on a separate test dataset. Additionally, the generalization ability of the models is evaluated using a second test set with out-of-domain data. Based on the metrics from these two test sets, a decision for an optimal model can be made. The metrics used for evaluating the performances of the models are further explained in Section 3.3.2.

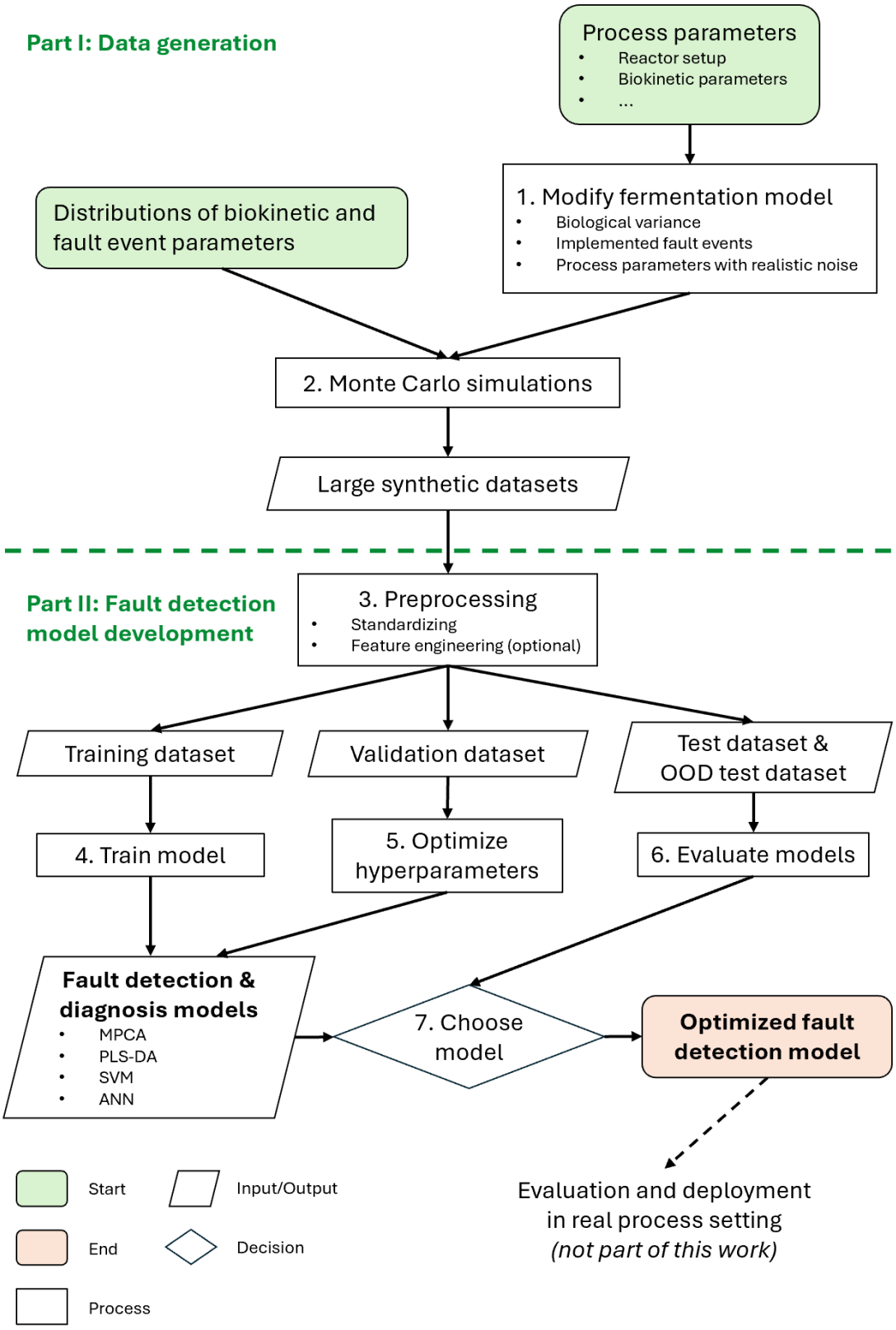


Figure 3.1: Methodological framework developed in this work. The framework is divided into two parts (green dotted line): (I) data generation and (II) fault detection model development.

3.2 Part I: Data generation

3.2.1 Step 1: Modifying the fermentation model

The basis for generating synthetic datasets is a modified version of the mechanistic fermentation model. This model is defined by a set of process parameters. In this work, the process parameters of the fermentation model were based on expert knowledge to replicate a realistic process behavior. A detailed list of all parameters can be found in Table A.1 in the appendix. This set of parameters is consistently used throughout this work.

The modifications to the model allow it to include biological variance, simulate fault events, and apply realistic noise to process parameters. These modifications will be explained in detail in the following sections.

Implementing biological variance

The biological system of the fermentation model is defined by a set of biokinetic parameters, previously explained in Chapter 2.1. A concise overview of all parameters can be seen in Table 3.1.

Table 3.1: Biokinetic parameters of the fed-batch fermentation model. The given median for each parameter is used consistently throughout this work. The parameters are set to resemble an exemplary case of *Aspergillus oryzae* based on expert knowledge.

Parameter	Median	Units	Description
X_0	5.0	g_{DW}/L	Initial biomass concentration
Y_{SX}	0.3	g_{DW}/g_S	Yield coefficient from substrate to biomass
Y_{SE}	0.1	g_E/g_S	Yield coefficient from substrate to enzyme
Y_{SO}	0.6	g_{DO}/g_S	Yield coefficient from substrate to dissolved oxygen
Y_{SC}	0.9	g_{CO_2}/g_S	Yield coefficient from substrate to CO_2
Y_{XS_true}	1.81	g_S/g_{DW}	"True" yield coefficient from biomass to substrate
Y_{XO_true}	0.0328	$moleO_2/g_{DW}$	True yield coefficient from biomass to oxygen
m_s	0.013	$g_S/(g_{DW} \cdot h)$	Substrate maintenance coefficient
m_o	0.0003	$moleO_2/(g_{DW} \cdot h)$	Oxygen maintenance coefficient

Based on expert knowledge, all biokinetic parameters are assumed to be normally distributed and to vary by 10 % around their median value. Since normal distributions have no defined upper and lower boundaries, truncated normal distributions were used. This distribution is based on the assumption that 95 % of the values are within its upper and lower boundaries. With that, the standard deviation s of the truncated normal distribution can be

calculated based on its upper and lower boundaries as

$$s = \left| \frac{par_{max} - par_{min}}{4} \right| \tag{3.1}$$

, where par_{max} and par_{min} are the upper and lower boundaries of the respective parameter.

Fault events implementation

As mentioned in the problem statement (see Chapter 1.1), the five most common faults according to expert knowledge were implemented in the fermentation model. An overview of the five fault types is shown in Table 3.2.

Table 3.2: Fault types implemented in the fermentation model. For each fault type, a description explains its effect on the process. Individual fault events are defined by their fault type and a set of parameters tied to this type.

Fault name	Description	Parameters
Steam in feed	Constant flow of steam entering with the feed stream, starting at t_{start}	• t_{start} [h] • Steamflow [L_{water}/h]
Defect steam barriers	Constant flow of steam entering the fermenter directly, starting at t_{start}	• t_{start} [h] • Steamflow [L_{water}/h]
Blocked spargers	Airflow decreasing linearly over a certain period until being constant at a certain setpoint	• t_{start} [h] • Decay period [h] • Offset [%]
Airflow meter OOC	Airflow meter being OOC by a certain percentage over the whole batch	• Offset [%]
OUR OOC	OUR measurement being OOC by a certain percentage over the whole batch	• Offset [%]

Fault events are introduced into the model by passing a fault configuration file to the model. This configuration file contains fault events as specified by their name and their set of parameters (see Table 3.2). When solving the model equations, a function evaluates if a fault event is present at each time step. When this is the case, certain parameters are changed according to the fault event. For both OOC-related faults, the measured airflow or OUR is changed by their specific offset percentage. In the case of blocked spargers, the airflow profile is adjusted according to the specifications in the fault event. For steam-related fault events, steam enter-

ing the tank is tracked. It is assumed that the steam immediately condensates, so the steam is tracked as water. If the steam is leaking through the feed stream, the density of the feed stream is adjusted. In the tank, the volume, weight and density of the broth are adjusted.

Applying noise to process parameters

For this fermentation model, airflow and weight data from a real batch was provided to extract and apply the underlying noise signal to the model. The raw data is presented in Figure 3.2. The goal was to extract the white noise signal for both measurements. White noise describes uncorrelated zero-mean random variables. The auto-correlation function (ACF) was used as a criterion for white noise. According to this function, white noise is present when the ACF is within its confidence interval except for lag zero [24]. Lag refers to the delay between an observation and previous data points. The white noise signal is quantified by determining its standard deviation, which is then applied to the process signal in the model.

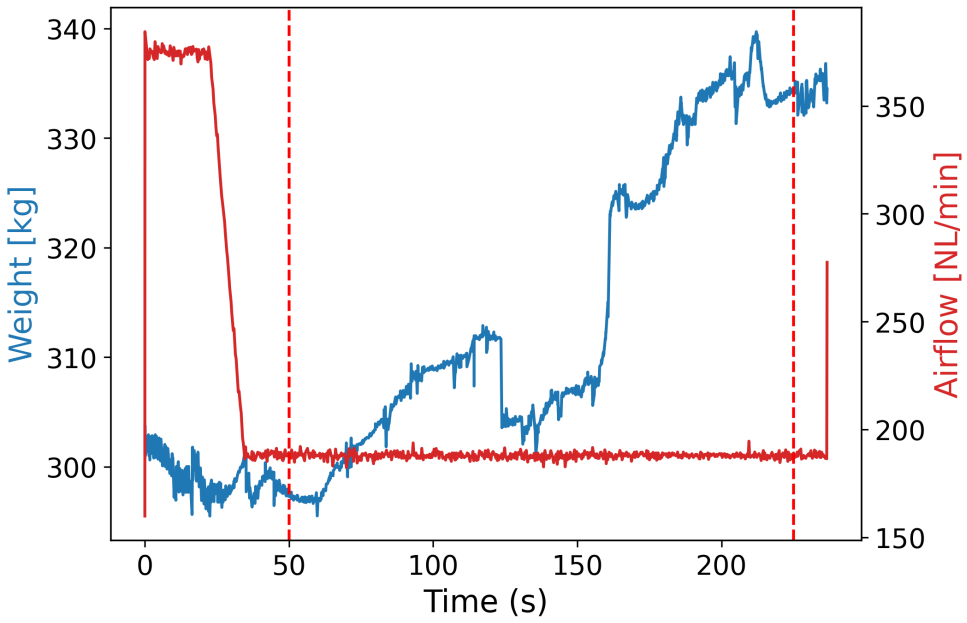


Figure 3.2: Airflow and weight data from an illustrative batch. The section of data used for extracting the white noise of the airflow data is indicated by red dotted lines.

The airflow signal does not need to be filtered before extracting the white noise signal, since for the signal between 50 and 225 hours, the ACF plot already fulfills the criterion for white noise (see Figure A.1 in the appendix). The standard deviation for the airflow noise signal is:

$$s_{airflow} = 1.42\,\text{NL/min}.$$

Contrary to the airflow signal, the weight signal does not offer a stationary signal imme-

diately. For this reason, two techniques were applied to the weight signal to extract the white noise components: differencing and trimming. Differencing computes the differences between consecutive points, removing non-stationary components of the time series (see Figure A.2 in the appendix). Trimming describes the process of deleting outliers and anomalies from the data. This is done by removing all points in the lowest and highest 5 % of the data (see Figure A.3 in the appendix). After applying these techniques, the ACF plot showed minor autocorrelative behavior but was overall still close to a white noise signal. The standard deviation of the weight signal after applying differencing and trimming is:

$$s_{weight} = 0.57\,\text{kg}.$$

3.2.2 Step 2: Generating datasets through Monte Carlo simulations

The large synthetic datasets in this framework are generated by Monte Carlo simulations. The goal is to create a dataset with batch data that replicates the biological variance of the fermentation process and contains a variety of fault events. For each Monte Carlo simulation, the number of total batches and the number of fault events of each type are defined beforehand. For each batch, a set of biokinetic parameters is sampled from their respective distribution (see Table 3.1). Similarly, for each fault event a set of fault event parameters of the defined fault type is sampled from their respective distributions (see Table 3.3).

Table 3.3: Distributions of fault event parameters. Each fault type is defined by a set of fault event parameters. To perform a Monte Carlo simulation, each fault event parameter is assigned a distribution, which the Monte Carlo method randomly samples from.

Defect steambarriers & steam in feed		
Steamflow	0.1 - 0.6 L/h	Truncated normal
t_{start}	0 - 100 h	Uniform
Airflow out-of-calibration		
Offset	10 to 20 %	Positive & negative normal
OUR out-of-calibration		
Offset	10 to 30 %	Positive & negative normal
Blocked spargers		
Duration (of decrease)	20 to 30 h	Truncated normal
Offset	-20 to -50 %	Truncated normal
t_{start}	0 to 100 h	Uniform

Out-of-calibration faults are modeled by two truncated normal distributions, a positive and a negative one. Both distributions do not include values close to zero to avoid smaller and therefore less meaningful out-of-calibration deviations. When a value is sampled for creating a fault event, the Monte Carlo method randomly picks a value from one of these two distributions.

Both lists of parameter sets are saved locally and subsequently used to perform the simulations to create the dataset.

Since the Monte Carlo method randomly samples from the respective parameter distributions, the training, validation, and test datasets can be independently sampled as long as their distributions are the same. Generating these datasets separately instead of splitting one large dataset simplifies the process of creating datasets with similar distributions of fault events. It also allows one to, for example, modify the test set to test the extrapolation abilities of a fault detection model by creating slightly different fault events compared to the training set.

Every random operation is seeded by using the function `random.seed` from the `NumPy` Python package to ensure reproducibility. If necessary, a local seed is introduced. Depending on the type of dataset, the following seeds were used.

Table 3.4: Seed values used for generating each type of dataset.

Dataset	Seed
Training	42
Validation	24
Test	44

The features in the dataset were chosen according to the online data available in the fermentation process. The feature matrix containing this data constitutes the input for the fault detection models. The target matrix on the other hand contains the information about the fault events and is to be predicted by the fault detection models. Models can be trained for fault detection or fault diagnosis. In fault detection, the occurrence of a fault has to be detected without classifying the fault type. In this case, the target matrix contains two target variables. However, if a model is trained for fault diagnosis, the target matrix is made of six target variables, one for each fault type and one for the normal state. The target variables are one-hot-encoded, meaning that for any observation one of the target variables is equal to 1, while the rest is 0. Table 3.5 shows an overview of the features and target variables.

Table 3.5: Features and target variables of batch-wise data. The two-class setup is used for training for fault detection, while the six classes are needed for training for fault diagnosis.

Features	Target Variables

• Time
• DO saturation
• Weight
• Feed setpoint
• Airflow setpoint
• Measured airflow
• kLa
• OUR
	• **Fault detection**
	  – Fault
	  – No fault
	• **Fault diagnosis**
	  – Defect steam barriers
	  – Steam in feed
	  – Airflow OOC
	  – OUR OOC
	  – Blocked spargers
	  – No-fault

3.3 Part II: Fault detection model development

3.3.1 Step 3: Pre-processing

The most important step in pre-processing the data is the standardization of all features. Specifically, this means subtracting the mean from each observation and scaling it by the standard deviation. A sample x is thereby transformed into a standardized sample z as:

$$z = \frac{(x - u)}{s} \tag{3.2}$$

where u is the mean and s is the standard deviation of all the samples with respect to this feature. This is an essential step, as it ensures an equal contribution of all features to the model by transforming all features to the same scale. The training, validation, and test sets are all scaled by the mean and standard deviation of the training set. This avoids information being spilled between datasets [25].

The pre-processing is handled by data loaders, with separate data loaders for the training, validation, and test set. The data loader maintains the time series structure and order inside each batch, while being able to shuffle the order of batches. Additionally, it can standardize the data inside the batches based on the whole dataset without changing the internal order of the batches.

Feature engineering is also part of the pre-processing step. However, this topic is discussed later in Section 4.3.2, as feature engineering is used as a tool to improve the performance of diagnosing specific fault types and therefore depends on the results obtained without it.

3.3.2 Model performance metrics

Developing and optimizing fault detection models requires an adequate set of metrics. Two commonly used metrics are the fault detection rate (FDR) and the false alarm rate (FAR), which are also used here. Both can be explained from a general confusion matrix (see Figure 3.3)

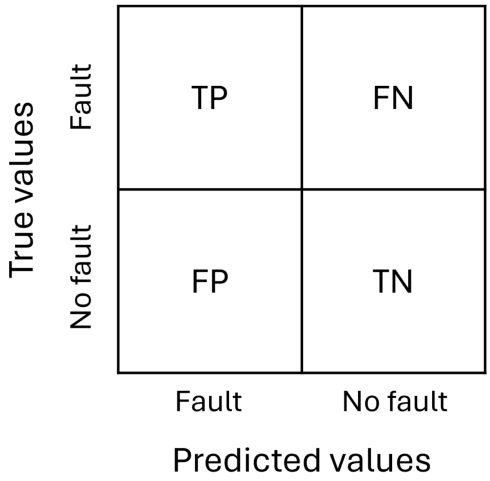


Figure 3.3: Confusion matrix for fault detection. TP = true positive, FP = false positive, FN = false negative, TN = true negative.

The FDR is defined as the ratio of observations that were correctly identified as being faulty to all observations with faults.

$$FDR = \frac{TP}{TP + FN} \tag{3.3}$$

The FAR describes the ratio of observations that were incorrectly identified as being faulty to all observations without faults.

$$FAR = \frac{FN}{TN + FP} \tag{3.4}$$

Generally, the goal is to maximize the FDR while minimizing the FAR. To be able to optimize and compare models based on a singular metric, the accuracy can be used as a performance metric. It is the ratio of the number of correctly classified observations to the total number of observations [3]. The accuracy can be derived from the FDR and FAR in the following way.

$$ACC = \frac{FDR + (1 - FAR)}{FDR + (1 - FDR) + FAR + (1 - FAR)} \tag{3.5}$$

3.3.3 Steps 4 & 5: Training and optimizing Machine Learning based models

MPCA model

The implementation of the MPCA model is based on the `PCA` class from the `scikit-learn` Python package. The feature matrix was constructed by concatenating the batch-wise data along the time axis. An important first decision for any PCA model is the number of principal components used. A common approach to this is defining the percentage of variance that should be explained by the principal components. In this way, just enough components are used to explain the defined amount of variance. For this case, the required explained variance was set to 95 %, as this is a common threshold used with this approach [26].

When calibrating the model with normal batch data, the thresholds for the Hotelling's T^2 and SPE statistics are calculated based on a given parameter α. The T^2 threshold is then based on sampling from a chi-distribution using α and the number of principal components (pc).

$$T^2_{limit} = \chi_{1-\alpha,pc} \tag{3.6}$$

The SPE threshold is based on the SPE of the training set, i.e., the squared difference between the original feature matrix and the reconstructed feature matrix using PCA. This results in a vector along the time axis. The SPE threshold is determined by identifying the value below which the observations of the $(1-\alpha)$ percentile fall.

$$SPE_{limit} = \text{percentile}(SPE, 100 \cdot (1-\alpha)) \tag{3.7}$$

The hyperparameter α can therefore also be seen as the tolerated false alarm rate of the MPCA model.

When predicting fault events, a fault alarm is triggered if both the SPE and T^2 are above their respective thresholds. Both statistics are required to be above their thresholds as this can lower the false alarm rate [3]. The predictions of this method are absolute (1 or 0) instead of probabilities.

An additional method to prevent false alarms is the implementation of a moving time window. By defining a certain window size w, a fault is only triggered if the previous w observations were also already classified as faulty. This prevents false alarms due to sporadic outliers.

With this setup, an MPCA fault detection model can be adjusted through three hyper-

parameters: the number of principal components, the limit α, and the size of the moving time window n. While the number of principal components is chosen by defining a required percentage of explained variance, both α and n are optimized by grid search, choosing the configuration leading to the highest accuracy.

PLS-DA model

The implementation of the PLS-DA model is based on the `PLRegression` class from the `scikit-learn` Python package. As with MPCA, choosing the number of components is a crucial decision. The approach used in this work is to train a PLS-DA model on the training set and then evaluate the RMSE on the validation set. This is performed for each possible number of components, choosing the number of components resulting in the lowest RMSE. The reason this method was chosen over the common approach of performing k-fold cross-validation with only the training set was to ensure a consistent balancing of target classes, as all datasets have the same proportions of each fault type. While stratified cross-validation could achieve the same, this approach also ensures data from the batch is not divided into different datasets. The fact that the performance of the model is evaluated using a separate test set justifies using the validation set for optimizing hyperparameters.

Since PLS is originally designed for regression, the output of the PLS-DA model is continuous. According to the PLS-DA model, the higher the prediction value, the more likely a fault event is. This allows one to implement a more fine-tuned moving time window compared to the MPCA model. For this moving time window, two hyperparameters need to be defined: the window size n and a threshold τ. A fault alarm is triggered if the average prediction value of the last n observations is higher than the threshold. Both hyperparameters are optimized by grid search, choosing the configuration with the highest accuracy.

SVM model

The implementation of the SVM model is based on the `SVC` class from the `scikit-learn` Python package. The SVM model is implemented as a two-class classifier, meaning there is one decision boundary between “fault” or “no fault”. The radial basis function (RBF) is used as a kernel function since it is proven to be effective in nonlinear mappings with a high generalization ability [27]. An SVM with an RBF kernel is defined by two hyperparameters: γ and C. The first parameter, γ, decides how large the influence of each individual observation on the decision boundary is allowed to be. C is a generalization parameter, controlling the trade-off between a lower prediction error and a low model complexity, hence improving the generalization ability. The third hyperparameter for the SVM fault detection model is the size

of the moving time window, equivalent to the one used for the MPCA model.

One challenge when training SVM models with large datasets is the computational complexity (see Section 2.3.3). To be able to perform grid search for hyperparameter optimization on a local machine within a reasonable timeframe, a dataset of 100 batches was used for training the SVM fault detection model instead of the usual 1000 batches used in the other models. The final SVM model used for model evaluation was trained on a dataset with 250 batches, allowing for a more complex model while still limiting the computational cost of training the model and performing predictions. Table A.2 in the appendix shows an overview of these different datasets.

ANN model

The ANN fault detection model is built using the PyTorch framework and is structured as a feed-forward neural network. The number of input nodes is equivalent to the number of features, while the number of output nodes can either be one or six, depending on whether the model is trained only for fault detection or also for fault diagnosis (see Table 3.5). The network depth is fixed to one hidden layer, as this offers sufficient complexity for most applications [28]. The softmax function is applied to the output layer to transform the raw output values (logits) into a probability distribution.

A general overview of the ANN model setup is shown in Table 3.6. Typical for classification problems, the cross-entropy loss was used. The optimizer of choice is the widely used *Adam* optimizer, as it offers gradient-based optimization that is computationally efficient, has low memory requirements, and is well suited for large data sets [29]. Since the *Adam* optimizer adapts the learning rate during the training process, a common initial learning rate offering stable but not too slow convergence is chosen. As an activation function, the ReLU function was used, as it is widely used and proven. It also suffers less from vanishing gradient problems compared to other activation functions [30].

In each forward loop, the data from one fermentation batch is used to calculate the prediction of the ANN for this batch and evaluate the loss function. After each forward loop, all weights and biases are updated with backpropagation based on the loss function. One epoch is completed when the ANN has been trained on the data of every fermentation batch in the training set. The order of batches is shuffled after each epoch. For each ANN model, 100 epochs are performed. After each epoch, the state of the network together with the current loss is saved. In this way, the set of weights and biases of the model with the lowest loss is picked.

Table 3.6

Model parameter	Choice
Framework	PyTorch 2.3
Loss function	Cross-entropy loss
Optimizer	Adam
Activation function	ReLU
Number of hidden layers	1
Initial learning rate	0.001
Batch size	1 fermentation batch
Epochs	100

The predictions of the ANN fault detection model can be interpreted as probabilities due to the application of the softmax function. The class with the highest probability is picked as the predicted class.

The ANN model has two hyperparameters that need to be optimized. The first one is the number of nodes in the hidden layer. This determines the complexity of the network. The second one is the weight decay of the network during training, which is a regularization parameter limiting the network's complexity by keeping weights low. An optimal configuration of these hyperparameters is determined through grid search.

One primary goal of training fault detection models is to achieve a low FAR. To facilitate this, a false alarm penalty factor was implemented in the loss function of the ANN. The cross-entropy loss is multiplied by this factor in the case of a false alarm.

$$L_{\text{FAP}} = L \times \begin{cases} \lambda & \text{if false alarm} \\ 1 & \text{otherwise} \end{cases} \tag{3.8}$$

where L_{FAP} is the loss function with a false alarm penalty factor applied, L is the regular cross-entropy loss function, and λ is the false alarm penalty factor. This specifically discourages the model from triggering false alarms. The penalty factor is tuned after the grid search that determines the number of nodes and the weight decay.

3.3.4 Step 6: Model evaluation

Finally, the optimized models for fault detection and diagnosis are evaluated on two different test sets. The in-domain test set is generated with the same distributions for the Monte

Carlo simulation as the training and validation set. This dataset enables determining the performance metrics of the models on data on which they were not optimized on. The second test set contains out-of-domain (OOD) data, meaning it contains fault events outside of the upper and lower bounds the models were trained on. This dataset allows for the evaluation of the generalization capabilities of the models. Based on the metrics from both datasets, a decision can be made for an optimal model.

All adjustments made to the out-of-domain test set are shown in Table 3.7. In the OOD test set, steam-related faults can occur later in the fermentation process than they did in the previous data sets. Additionally, the ranges for both OOC-related faults were set to be wider. Lastly, the decrease in airflow due to blocked spargers was now allowed to occur over a longer period and lead to a smaller offset.

Table 3.7: Adjustments made to the fault event parameters in the OOD test set compared to the regular datasets.

Fault Type	**Parameter**	**Normal Range**	**OOD Range**
Defect Steam Barrier	t_{start}	0 to 100	0 to 150
Steam in Feed	t_{start}	0 to 100	0 to 150
Airflow OOC	Offset	10 to 20	5 to 30
OUR OOC	Offset	10 to 30	5 to 40
Blocked Spargers	Offset	-20 to -50	-10 to -50
Blocked Spargers	Duration of decrease	20 to 30	20 to 50

4 Results and discussion

This chapter shows how the framework developed in this work (presented in Figure 3.1) was applied to the aforementioned fed-batch fermentation model. To begin with, the biological variance of the fermentation process was visualized and discussed, followed by exemplary simulations to illustrate the effect of the five fault types on the process. In the first step of model development, four different models were optimized for fault detection, thereby performing binary classification without diagnosing the fault type. Following this step, an ANN model was developed and fine-tuned for fault diagnosis, enabling it to identify a specific fault type. This model was further improved by feature engineering. In a final step, the performance of all models was evaluated with an in-domain test set. An additional test set containing out-of-domain data was utilized to compare the generalization capabilities of all models.

4.1 Validation of large synthetic datasets

4.1.1 Biological variance

For the case of this fed-batch fermentation process, fault detection is based on eight input features. The statistical distributions of six of these features throughout the fermentation are shown in Figure 4.1. The distributions are based on a Monte Carlo simulation with 1000 individual simulations (see Table 3.1 for the distributions of the biological parameters). One of the remaining two features is time itself. The other, the airflow setpoint, is not included in this visualization as without certain fault events the airflow setpoint resembles the measured airflow rate without its noise.

The highest variance is observed in the feed setpoint. This is expected, as the feed setpoint is the only controlled variable in this set of features. By being controlled through the DO saturation, it is closely tied to the growth kinetics of the biological system. In some cases, the feed setpoint reaches its maximum value of 2 g/L and consequently, the OUR decreases while the DO saturation of the broth increases as less substrate is metabolized.

The weight also exhibits some degree of variance, which can be mostly attributed to the variance in the feed flow rate. The kLa value differs just slightly between batches. In the fermentation model, the kLa depends mainly on the airflow rate, the power input, and the viscosity of the broth. As the impeller speed, linked to the power input, and the airflow rate are fixed, the minor variance of the kLa is likely caused by slight changes in the viscosity of

the broth due to the variance in the metabolism of *A. oryzae*.

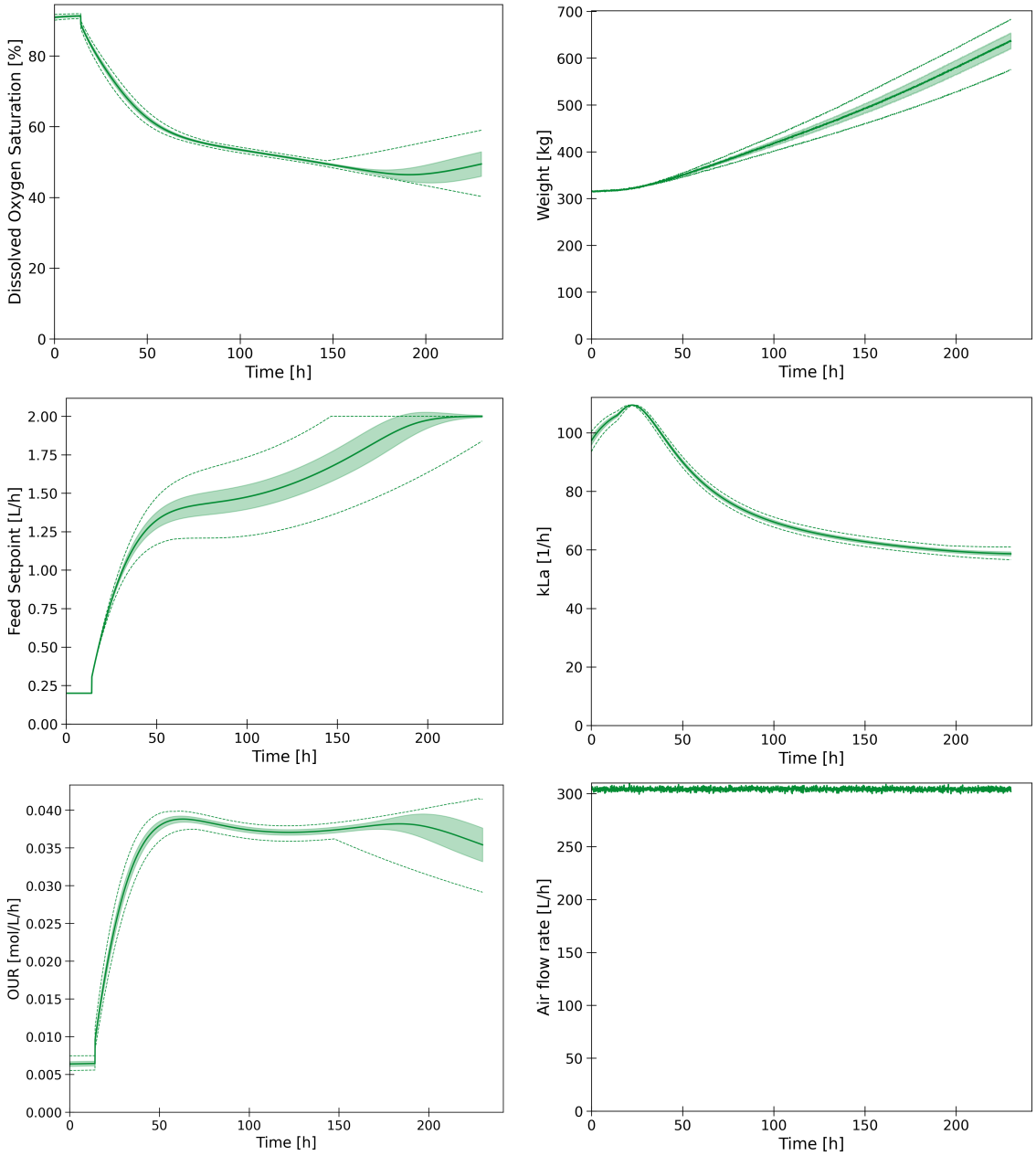


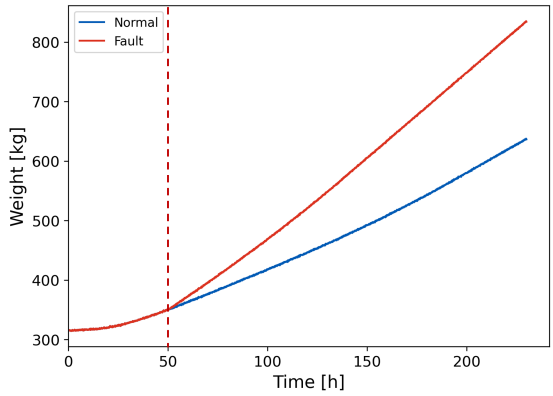
Figure 4.1: Impact of biological variance on input variables. The green line shows the average, the green area the standard deviation and the dotted line the minimum and maximum values from 1000 batches.

4.1.2 Validation of fault event implementation

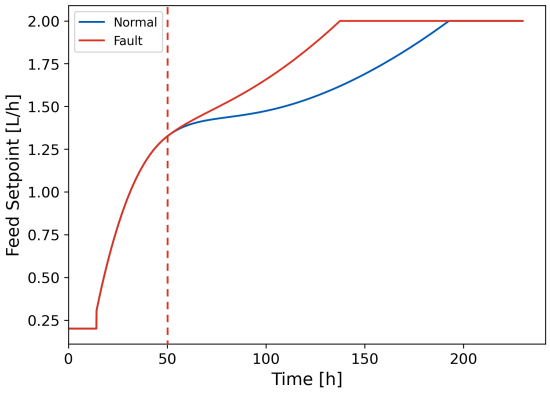
All five fault event types were integrated into the fermentation model. Figure 4.2 visualizes the maximum effect of the different fault types on the system by comparing the respective faulty batch to a normal one. For each fault type the parameters were chosen according to their respective maximum offset in their distribution (see fault event parameter distributions

in Table 3.3). The start times of faults are indicated with a red dotted line.

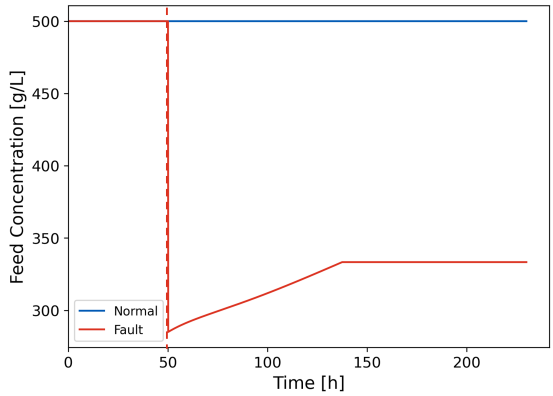
Defect steam barriers lead to an increase in weight due to water entering the tank. They also result in a higher feed setpoint, since the substrate concentration in the tank is lowered due to the water intake, which the PI controller is reacting to. Steam entering the tank through the feed has the same effect as defect steam barriers, but additionally lowers the substrate concentration in the feed as well as the feed stream's density. However, both variables are not part of the features used for fault detection as they are not measured in the real process. Both OOC-related fault types statically change the OUR or airflow by a certain percentage over the whole batch without influencing other variables. In the case of blocked spargers, the airflow decreases linearly until a certain point. Besides decreasing the airflow measured, this also affects the biological system by limiting the oxygen supply, leading to slightly decreased biomass and enzyme production.



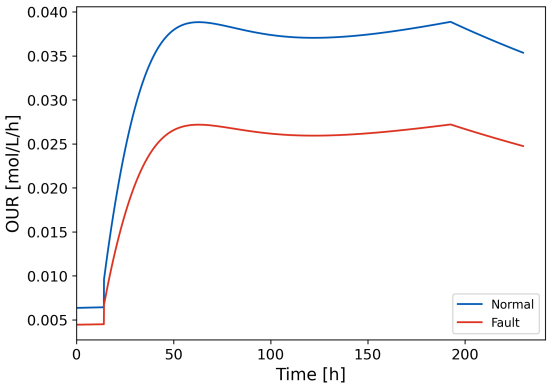
(a) Defect steam barriers



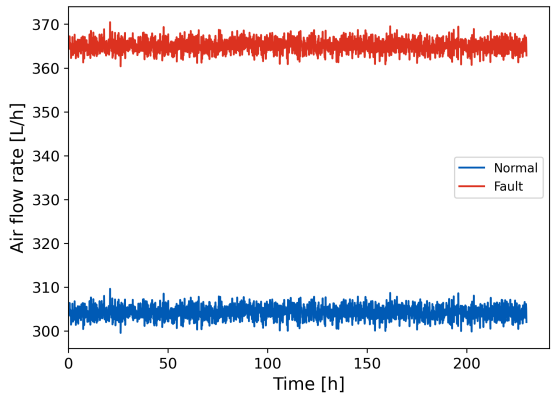
(b) Defect steam barriers



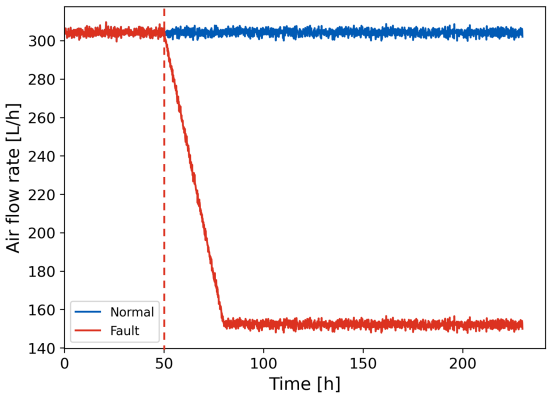
(c) Steam in feed



(d) OUR OOC



(e) Airflow meter OOC



(f) Blocked spargers

Figure 4.2: Validation of fault event types through exemplary comparison of simulations. Each fault event was generated with its respective maximum impact according to Table 3.3. The simulations were performed with the standard value of each biological parameter (see Table 3.1), without the introduction of variance.

4.2 Model development for fault detection

4.2.1 MPCA

Before training the MPCA model according to step 4 in the framework (see Figure 3.1), the number of principal components to be used by the model was determined. For this, the cumulative explained variance for each possible number of principal components was calculated. The principal components were required to explain at least 95% of the variance of the training set. While two components explained 88% of the variance, three components were able to explain 98% of the variance, so three principal components were chosen (see Figure A.5 in the appendix).

The threshold α and the size of the moving time window n were optimized using grid search with the validation dataset according to step 5 of the framework (see Figure 4.3).

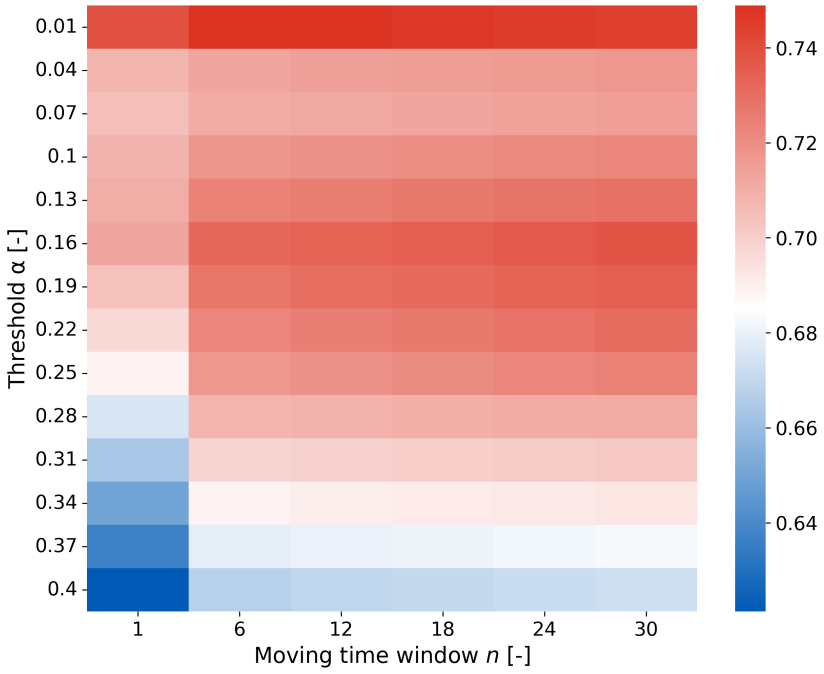


Figure 4.3: Accuracy of the MPCA model depending on the threshold α and the moving time window size n.

The grid search showed two local accuracy maxima. The global maximum of these two occurred with a configuration of $\alpha = 0.01$ and $n = 6$ with an accuracy of 0.749. The second local maximum was found at $\alpha = 0.16$ and $n = 30$ with an accuracy of 0.738. However, choosing n as low as possible is advantageous for a process operator to be able to act quickly

in case of a fault. A moving time window size of $n = 30$ would lead to a delay in fault detection of 3 h, which is why the configuration of $\alpha = 0.16$ and $n = 6$ was explored. Its accuracy is just 0.06 below the configuration with $n = 30$. Therefore, this configuration was further considered besides the global maximum. The performance metrics of the MPCA with these two configurations are shown in Table 4.1.

Table 4.1: Performance metrics of to different configurations of the MPCA model. Both models use three principal components and $n = 6$. Metrics show performance on detecting faults, but not on diagnosing faults.

Metric	Fault type	MPCA ($\alpha = 0.01$)	MPCA ($\alpha = 0.16$)
FDR	Defect steam barriers	0.129	0.320
	Steam in feed	0.149	0.353
	Blocked spargers	0.987	0.992
	Airflow OOC	1.0	1.0
	OUR OOC	**0.188**	**0.730**
FAR		0.006	0.240
Accuracy		0.749	0.732

Both MPCA models performed similarly in detecting blocked spargers and out-of-calibration airflow meters. This indicates that these faults were detected below the threshold of $\alpha = 0.01$. The MPCA with $\alpha = 0.16$ had a significantly higher FDR regarding an out-of-calibration OUR value compared to the model with a threshold of $\alpha = 0.01$. This could imply that the local accuracy maximum around $\alpha = 0.16$ corresponds to its threshold for detecting the OUR OOC fault. As expected, increasing α also leads to a higher FAR. Both models were unable to robustly detect steam-related faults. Since a low FAR is crucial for the application of fault detection models, the configuration of the MPCA with $\alpha = 0.01$ was further considered.

Table 4.2: Configuration of the final MPCA model.

Hyperparameter	Value
Principal components	3
α	0.01
n	6

One issue the MPCA fault detection model had was its tendency for early fault alarms. Regardless of the fault type, the model tended to classify the first few time intervals of a batch as faults. Two examples of this behavior can be seen in Figures A.6 and A.7 in the appendix.

4.2.2 PLS-DA

The performance of the PLS-DA model is tied to three hyperparameters: the number of components, the detection threshold τ, and the moving time window size n. In PLS-DA, the number of components does not only determine the explained variance of the model with regard to its training data, but it is also relevant for the correlation between the input and target variables. As training the PLS-DA model is computationally inexpensive, an optimal set of hyperparameters was determined by a three-dimensional grid search. This resulted in a set of hyperparameters with an accuracy of 0.781.

Table 4.3: Optimal configuration of PLS-DA model for fault detection after grid search.

Hyperparameter	Value
nr. of components	6
τ	0.45
n	24

Again, choosing n as low as possible is advantageous for a process operator to be able to act quickly in case of a fault. Changing the window size n from 24 time steps (2.4 h of delay) to 6 time steps (36 min of delay) decreased the accuracy from 0.781 to 0.772, which was seen as a reasonable tradeoff for the reduction in delay from 2.4 h to 36 min. See Figure A.8 in the appendix for a heatmap showing the accuracy of the PLS-DA model depending on τ and n for 6 components.

The overall metrics of the PLS-DA model with $n = 6$ are shown in Table 4.4. The PLS-DA model could detect a majority of the steam-related faults and robustly detect blocked spargers. The model underperformed at detecting out-of-calibration (OOC) faults. It operated with a FAR of 0.058.

4.2.3 SVM

The performance of an SVM model with a radial basis function kernel is mainly dependent on two hyperparameters: the kernel coefficient γ and the regularization parameter C. Similar to the previous models, an optimal configuration was determined by grid search. A logarithmic scale was used for adjusting γ and C, as is common practice for tuning the hyperparameters of SVM models. As mentioned in Section 3.3.3, the computational complexity of training an SVM model increases polynomially with the number of observations the model is trained on. To train a variety of SVM models with the available computational power, the size of the data set was reduced to 100 batches for grid search. This data set still obeyed the same class

Table 4.4: Performance metrics of the chosen configuration of the PLS-DA model on the validation set. Configuration according to Table 4.3. Metric shows performance on the task of fault detection, not fault diagnosis.

Metric	Fault type	PLS-DA
FDR	Defect steambarrier	0.853
	Steam in feed	0.813
	Blocked spargers	0.961
	Airflow OOC	0.330
	OUR OOC	0.352
FAR		0.058
Accuracy		0.781

distribution as the standard training data set.

With an accuracy of 0.980, the SVM model with the configuration of $\gamma = 10$ and $C = 1$ was the best-performing model in the grid search (see Figure 4.4). The other models also showed a high accuracy, as the worst model with $\gamma = C = 0.1$ still had an accuracy of 0.933.

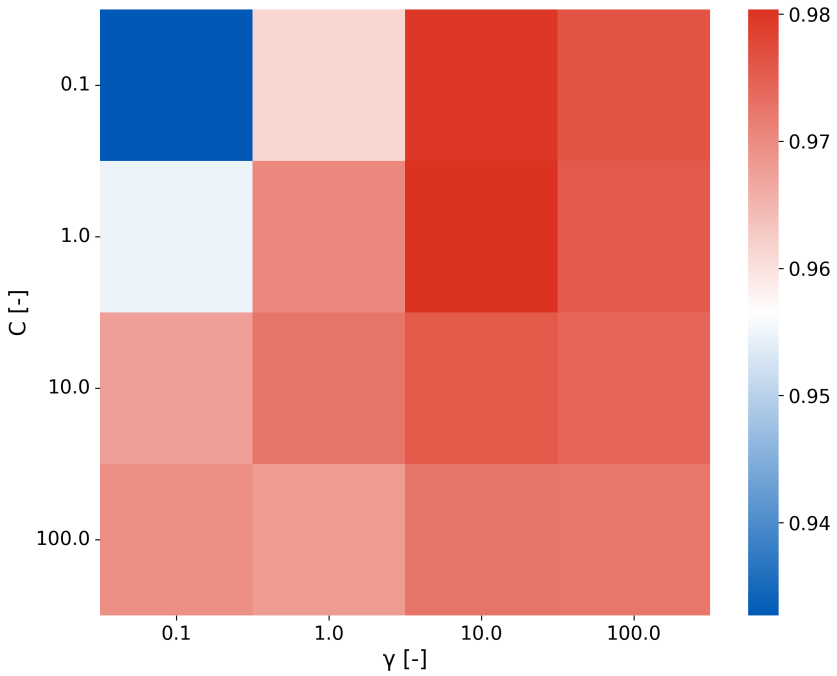


Figure 4.4: Accuracy of the SVM model with an RBF kernel depending on its regularization parameter C and its kernel coefficient γ.

A moving time window was applied to the best-performing model from the grid search. However, this decreased the accuracy, offering no improvement to the model (see Figure A.9 in the appendix).

The number of batches in the training data not only affected the duration of the training but also increased the time it took to make predictions on new data since the SVM uses more support vectors when being trained on more observations. Therefore, the final model was trained with a training set of 250 instead of 1000 batches to allow predictions in a reasonable time frame. The performance metrics of this model on the validation set can be seen in Table 4.5 as well as its configuration in Table 4.6.

Table 4.5: Performance metrics of the optimal configuration of the SVM model on the validation set. Configuration: $C = 1$, $\gamma = 10$. Metric shows performance on the task of fault detection, not fault diagnosis.

Metric	Fault type	SVM
FDR	Defect steam barrier	0.923
	Steam in feed	0.935
	Blocked spargers	0.989
	Airflow OOC	1.0
	OUR OOC	0.973
FAR		0.006
Accuracy		0.980

Table 4.6: Optimal configuration of the SVM model for fault detection after grid search.

Hyperparameter	Value
Kernel	radial basis function
C	1
γ	10

4.2.4 ANN

The goal when optimizing an ANN for a specific problem is finding a trade-off between closely fitting the model to the data and ensuring its ability to generalize. This was approached here in two ways: via the structure of the ANN and via a regularization parameter. For nearly all problems, one hidden layer is sufficient [28]. The structure of this ANN is therefore solely defined by the number of nodes in the hidden layer. The regularization parameter to be tuned was the weight decay, which avoids overly complex models by keeping weights low.

An optimal configuration of both parameters was found with grid search (see Figure 4.5). The highest accuracy was observed at a weight decay of $1 \cdot 10^{-8}$. Increasing the number of nodes above 40 did not yield a relevant improvement in accuracy (see Figure A.10). Therefore, to avoid overfitting, the number of nodes in the hidden layer was set to 40.

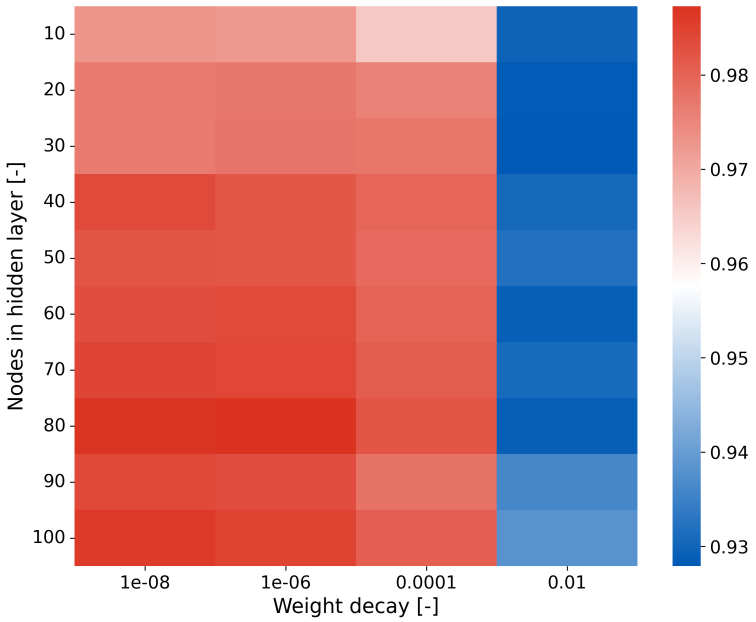


Figure 4.5: Accuracy of the ANN model depending on the number of nodes in the hidden layer and the weight decay during training. The models were trained for 100 epochs with callback to the model state with the highest accuracy. The learning rate was $1 \cdot 10^{-3}$.

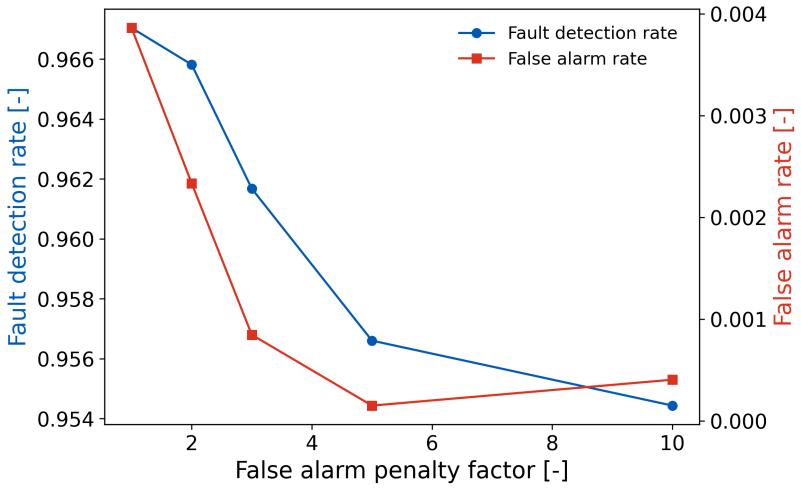


Figure 4.6: Effect of applying a false alarm penalty factor in the cross-entropy loss function on the ANN model for fault detection. The model has 40 nodes in the hidden layer and was trained with a weight decay of $1 \cdot 10^{-8}$.

The ANN model with 40 nodes that was trained with a weight decay of $1 \cdot 10^{-8}$ already showed a FAR of under 0.004. To further decrease this, the ANN model was trained with different false alarm penalty factors. This factor amplifies the effect of false alarms on the loss function, incentivizing the network to avoid false alarms. Figure 4.6 shows the FDR and FAR depending on the penalty factor applied. Applying the penalty factor seemed to both lower the FDR and FAR. However, with a penalty factor of 5, the decrease in FAR was nearly 40-fold, while the FDR only decreased by about 0.10. Since a low FAR is crucial for a fault detection model, the ANN trained with a false alarm penalty of 5 was chosen.

The performance metrics of the final ANN model for fault detection can be seen in Table 4.7. For reproducing this model, it is important to mention that training an ANN model with the same parameters can lead to slightly different models. Even though the random number generators in `PyTorch` and `NumPy` are seeded in the codebase, `PyTorch` still uses non-deterministic algorithms.

Table 4.7: Performance metrics of the optimal configuration of the ANN model on the validation set. Configuration according to Table 4.8. Metric shows performance on the task of fault detection, not fault diagnosis.

Metric	Fault type	ANN
FDR	Defect steam barrier	0.896
	Steam in feed	0.900
	Blocked spargers	0.986
	Airflow OOC	1.0
	OUR OOC	0.966
FAR		0.0001
Accuracy		0.954

Table 4.8: Optimal configuration of the ANN model for fault detection.

Hyperparameter	Value
Nodes	40
Weight decay	$1 \cdot 10^{-8}$
False alarm penalty factor	5

4.3 Model development for fault diagnosis

While a model for fault detection can detect if a fault is present, a model for fault diagnosis can also classify which fault is happening. This changes the classification from a binary to a multiclass task. In this chapter, an approach to optimizing an ANN for fault diagnosis is shown. An ANN was used as the model instead of an SVM for several reasons. To begin with, SVMs do not natively support multiclass classification. Furthermore, the ANN model showed a significantly lower FAR for fault detection, with 0.0001 compared to 0.006 for the SVM model. Lastly, the computational complexity of the SVM already limited its hyperparameter optimization for fault detection. This would be an even greater challenge when developing a model for the more complex task of fault diagnosis.

4.3.1 ANN for diagnosis

The most significant difference between the ANN model for fault detection and the ANN model for fault diagnosis is the number of output nodes. While a model for fault detection differentiates between two classes (fault and no fault), a model for fault diagnosis differentiates between six classes (five fault types and no-fault).

To optimize the ANN fault diagnosis model, the same approach as in Section 4.2.4 was used. Grid search revealed the highest accuracy to be at a weight decay of $1 \cdot 10^{-6}$ (see Figure A.11 in the appendix). With this weight decay, no relevant increase in accuracy could be achieved with more than 60 nodes. The ANN model for fault diagnosis therefore had 60 nodes in the hidden layer and was trained with a weight decay of $1 \cdot 10^{-6}$, which yielded an accuracy of 0.984.

This model's FAR on the validation set was already very low at $2.99 \cdot 10^{-5}$. This rate means that out of the 133615 observations without a fault, only four were misclassified as having a fault. Regardless, the model was trained with different false alarm penalty factors to analyze its effect on the model performance. As seen in Figure 4.7, the penalty factor does not seem to have a clear effect on the training of the model. A reason for this could be that the changes in FDR and (already low) FAR are obscured by the differences between trained models introduced by the randomness of the non-deterministic algorithms used. The model trained with the penalty factor of three was chosen since it offered the lowest FAR with the second-highest FDR.

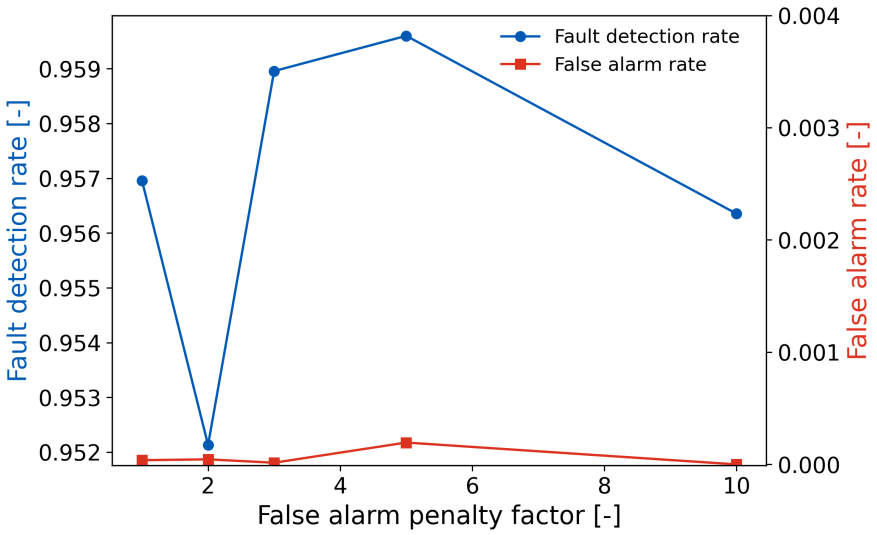


Figure 4.7: Effect of applying a false alarm penalty factor in the cross-entropy loss function on the ANN model for fault diagnosis. The model has 60 nodes in the hidden layer and was trained with a weight decay of $1 \cdot 10^{-6}$.

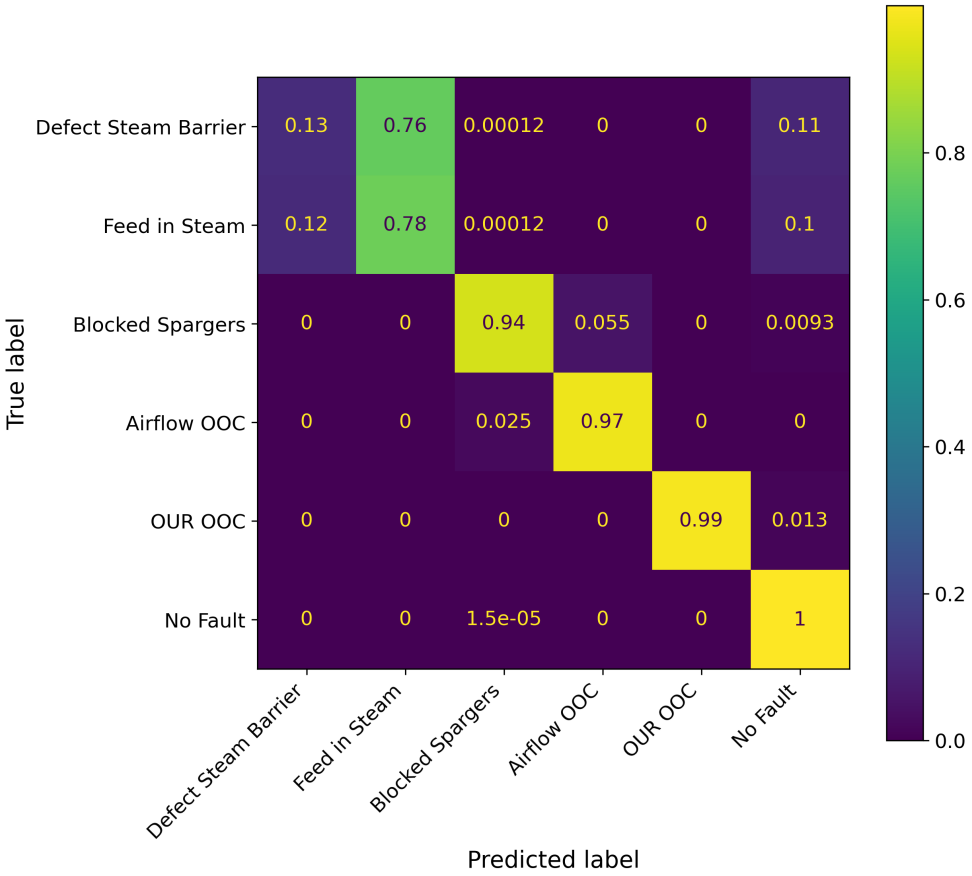


Figure 4.8: Confusion matrix of the ANN model for fault diagnosis. Configuration according to Table 4.9.

A confusion matrix was used to gain insights into the performance of this model on the validation set (see Figure 4.8). For a perfect model, the diagonal entries of the matrix are equal to 1 while the other entries are equal to 0. The model shown here was nearly perfect in the case of the two out-of-calibration (OOC) faults. There appeared to be some confusion when differentiating between blocked spargers and an OOC airflow meter. It misclassified 5.5 % of the observations with a blocked sparger as an OOC airflow meter and 2.5 % of the observations with an OOC airflow meter as being due to a blocked sparger. The model did not detect around 10 % of the observations with steam-related faults. Additionally, the model did not seem to be able to differentiate between the two steam-related fault types.

Consequently, the final configuration for the ANN model for fault diagnosis was as follows.

Table 4.9: Optimal configuration of the ANN model for fault diagnosis.

Hyperparameter	Value
Nodes	60
Weight decay	$1 \cdot 10^{-6}$
False alarm penalty factor	3

The analysis shows that the input data did not contain enough information to locate whether a steam-related fault is due to the steam barriers or the feed stream. Additional flow sensors in the feed stream could enable the model to differentiate these fault types, as steam entering through the feed would alter its flow rate. The confusion matrix also shows that the model seemed to confuse the two airflow-related fault types, namely blocked spargers and an OOC airflow meter, in a small percentage of cases. The next section discusses how these issues could be improved with feature engineering.

4.3.2 Feature engineering for improved diagnosis

There were two changes made to the features. First, the two target variables indicating steam-related fault types were combined into one. Second, the airflow rate setpoint and the measured airflow were replaced by two new features derived from these measurements.

Even though combining the steam-related faults into one is a reasonable step to avoid the implication that the model could discern between the two, it should be noted that it does not improve predictive capabilities.

The newly introduced features related to airflow should support the model in distinguishing between blocked spargers and an OOC airflow meter. Airflow-related faults always result in a difference between the airflow setpoint Q_{set} and the measured airflow Q_{measured}, since

only Q_{measured} changes in the case of a fault event. Consequently, the first feature added was the difference between these two values.

$$\Delta Q = Q_{\text{set}} - Q_{\text{measured}} \tag{4.1}$$

One hypothesis as to why the model misclassified airflow-related fault is a lack of information about the time-dependent dynamics of the airflow. When spargers get blocked, the airflow decreases throughout a certain period until it reaches a certain setpoint. An out-of-calibration airflow meter on the other hand statically changes the airflow value. As the fault detection model classifies each observation individually, it does not consider the trajectory of a variable over time. To provide the model with this information about the measured airflow, the average of ΔQ over the last 150 time steps was calculated. For the first 150 time steps, the average was calculated over all previous values.

$$\overline{\Delta Q} = \frac{1}{n}\sum_{i=1}^{n} \Delta Q_i, \quad \text{where } n = \min(150, \text{current time step}) \tag{4.2}$$

A new ANN model was trained on the training data set with the adjustments mentioned above and with the same configuration as in Table 4.9. The confusion matrix of this model can be seen in Figure 4.9. As expected, the overall FDR for steam-related faults did not improve, since combining these steam faults does not provide additional information. More importantly, introducing the two new airflow-related features resolved the model's confusion between blocked sparger and OOC airflow fault events, as both fault types are now clearly identified with an FDR of 1 or nearly 1.

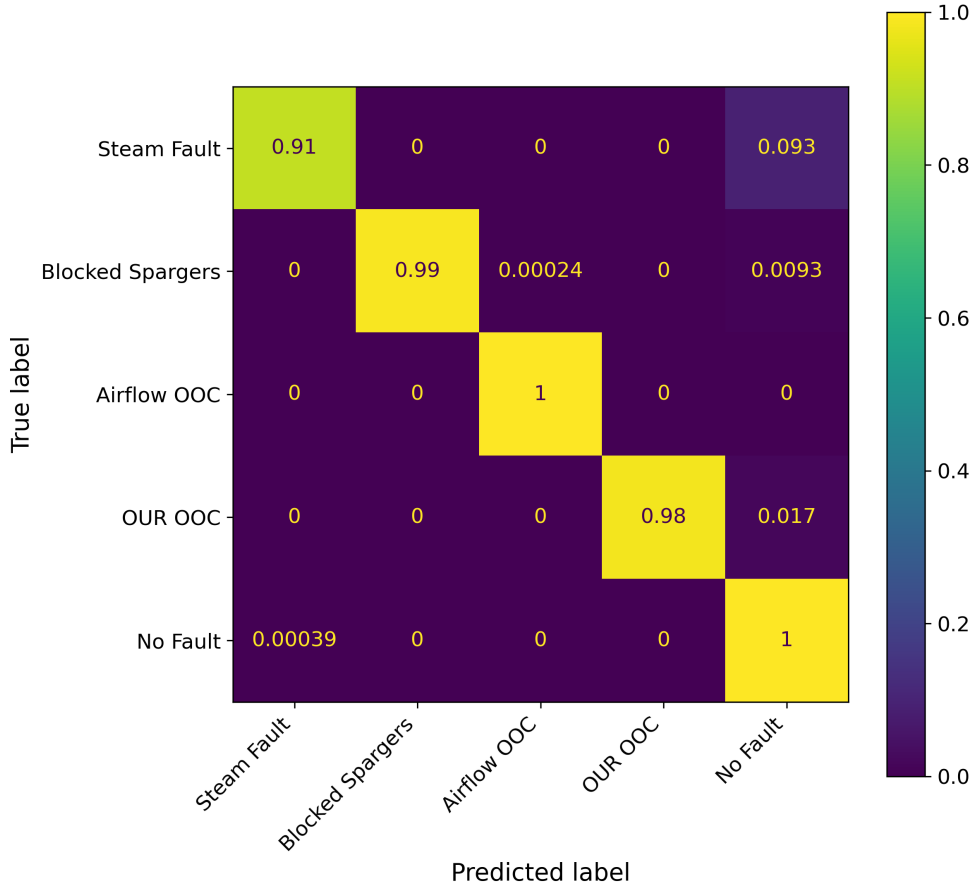


Figure 4.9: Confusion matrix of the ANN model for fault diagnosis with feature engineering. Steam-related faults were combined. The airflow set point and airflow meter features were replaced by the difference between these two features and a moving past average of this difference (see Equations 4.1 and 4.2). Configuration according to Table 4.9.

4.4 Model evaluation

4.4.1 Model performance on test set

The performance of all models for fault detection and diagnosis was evaluated using the test data set. This data set consists of 1000 batches and was generated with the same setup as the training and validation set, but with a different seed for the random number generators, resulting in different biological parameters and fault events sampled by the Monte Carlo method.

For the task of fault detection, the FDR of each model depending on the fault type can be seen in Figure 4.10. Note that these models detect if a fault is happening without classifying the fault type. Out of the four models, the SVM and ANN models were the best-performing, with the SVM model being slightly better over the entire range of fault types. This is remarkable since the SVM model was just trained on a quarter of the dataset the ANN model was trained

on due to the computational cost of training an SVM. This shows the strong capabilities of SVMs to form highly nonlinear decision boundaries for binary classification. However, this model type is also by far the most expensive to operate in terms of computational resources.

Both the MPCA and PLS-DA models struggled to detect certain fault types. Especially detecting that the OUR is out-of-calibration seemed to be the most difficult task for these two linear models. One reason for this could be that the OUR is both influenced by the biological variance and the offset caused by faults, which could make detection harder for these two simpler models.

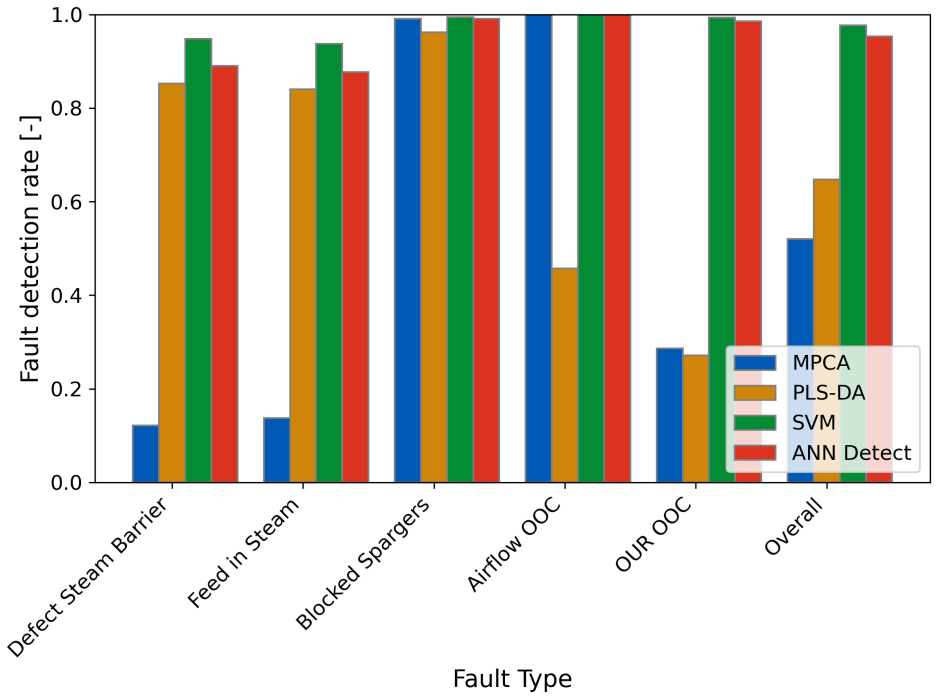


Figure 4.10: Fault detection rates for each fault type of all models trained and optimized for fault detection.

The ANN models for fault diagnosis could reliably detect OOC-related faults, regardless of feature engineering (see Figure 4.11). The feature engineering, however, significantly improved the diagnosis of blocked spargers. Both models did not detect about 10 % of observations with steam-related faults. Investigating the model predictions more closely, these missed observations were mostly shortly after the steam-related fault began. As this fault type leads to an accumulation of water, the effect of this fault increases over time. This makes this fault type difficult to detect instantly, as its effect is minor in the beginning. By diagnosing the fault with a delay, the model can still help avoid the majority of this fault's negative effects.

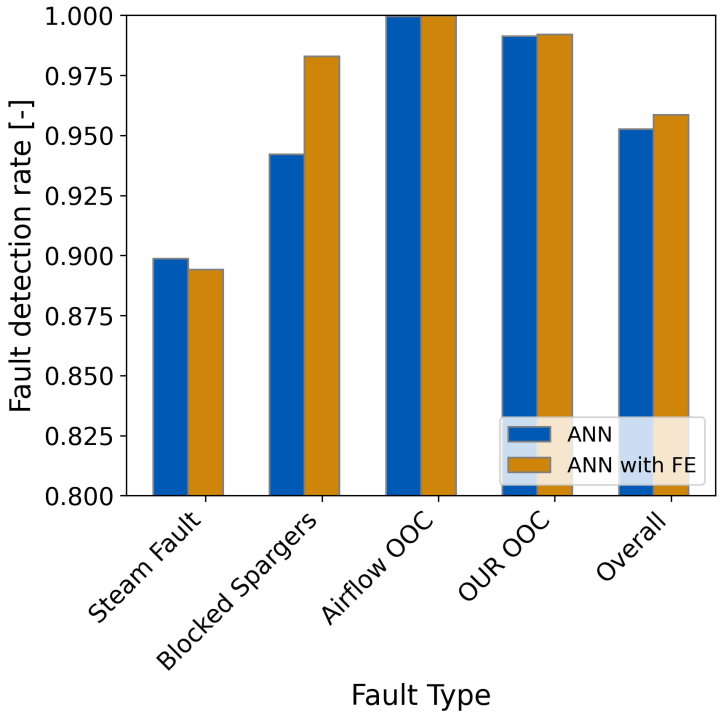


Figure 4.11: Fault detection rates for each fault type of the ANN model trained and optimized for fault diagnosis with and without feature engineering.

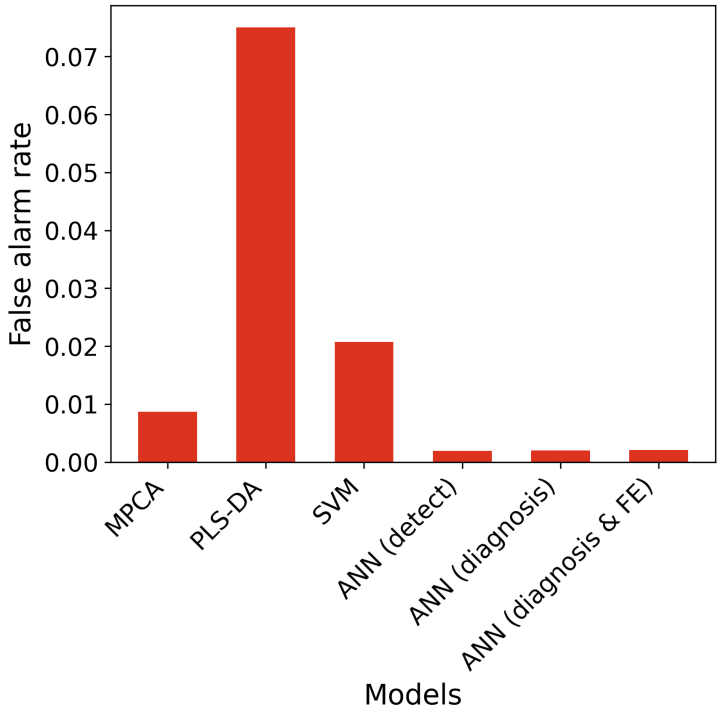


Figure 4.12: False alarm rates of models. The MPCA, PLS-DA, SVM and ANN (detect) modes were trained and optimized for fault detection, while the two most right ANN models were trained and optimized for fault diagnosis.

ANN models offered by far the lowest FAR, as they only raised a false alarm for about 0.2 % of the observations without fault events. The PLS-DA model has by far the highest FAR with more than 7 %. As mentioned in Section 4.2.2, the PLS-DA model requires a trade-off between FDR and FAR by adjusting the threshold τ. The FAR could therefore be lowered, but at the cost of a decreased FDR.

4.4.2 Evaluating model robustness with out-of-domain data

Applying a model to a separate test set is an important step for evaluating the performance of the model. Still, the test set shared the same domain as the validation and training data set, meaning they had the same underlying statistical distributions. To further evaluate the robustness and extrapolation abilities of the fault detection and diagnosis models, a second test set with out-of-domain (OOD) data was created. This was implemented by adjusting the distributions from which the Monte Carlo method samples the fault events as explained in Section 3.3.4.

The generalization ability of the models can be evaluated by comparing the FDR on the test set and the out-of-domain (OOD) test set. This difference in performance is shown in Figure 4.13 for the fault detection models. A higher increase in FDR when evaluated using the OOD test set implies a worse generalization ability of the model.

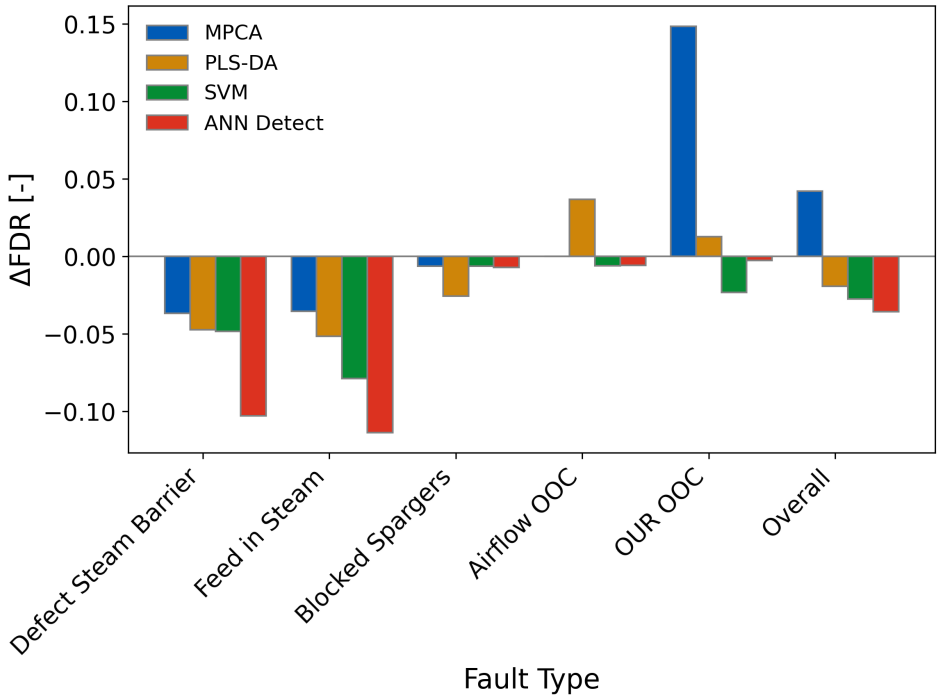


Figure 4.13: Difference in fault detection rates when fault models trained for fault detection are applied on the out-of-domain test set instead of the regular test set.

The MPCA and PLS-DA models showed stronger generalization capabilities than the more complex SVM and ANN models. An explanation for this could be that both MPCA and PLS-DA perform dimensionality reduction, which reduces the risk of overfitting [31]. Additionally, during its calibration, PLS-DA focuses on the most relevant components of the data regarding the target variable, further reducing potential overfitting.

Out of the more complex and nonlinear models, namely SVM and ANN, the SVM model generalized better regarding the steam-related faults. Indeed, SVMs show better extrapolation capabilities than ANNs in different application areas in general [32], [33]. ANNs can converge to local minima while SVMs converge to a global optimum of their decision boundary, leading to models that generalize more robustly. To improve the generalization capabilities of the ANN model, several regularization techniques besides the weight decay exist. One approach is Bayesian regularization, which favors smaller weights through a prior distribution and thus penalizes more complex models [34]. Additionally, the weights of an ANN trained with Bayesian regularization can be interpreted in a probabilistic way, making the network more explainable. Another approach could be an ensemble method. Instead of using one ANN, an ensemble of ANNs could be used to reduce individual biases in each model. Shao et al. improved the generalization capabilities of their ANN-based fault detection model for a fuel cell system by introducing this approach [35].

The generalization capability of the two ANN-based fault diagnosis models can be evaluated with Figure 4.14. For steam-related faults and blocked spargers, no difference between the two models could be observed. However, the ANN model with feature engineering showed a better FDR on the OOD test set than the model without feature engineering. Therefore, introducing the two new airflow-related variables did not just help the model in differentiating between blocked spargers and an OOC airflow meter, it also strengthened the model's generalization capability.

The FAR of no model changed meaningfully when evaluated on the OOD test set (see Figure A.14). This was expected, as only the distributions related to the fault events were changed, while the statistical characteristics of the biological parameters were kept the same. Consequently, the batches without faults in the OOD test set had the same underlying statistical distributions as in the other datasets.

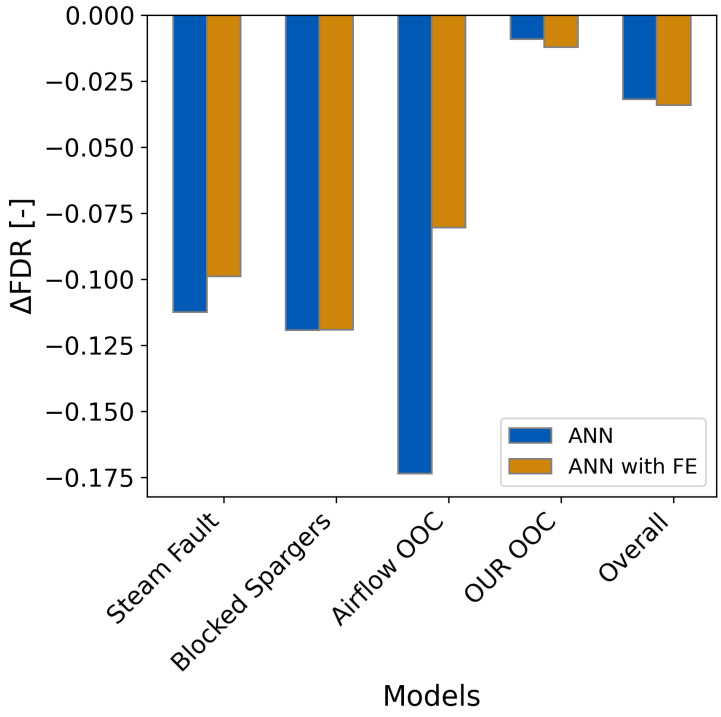


Figure 4.14: Difference in fault detection rates when fault models trained for fault diagnosis are applied on the out-of-domain test set instead of the regular test set.

5 Conclusions and future perspectives

5.1 Conclusions

The primary objective of this thesis was to create and apply a framework for developing data-driven fault detection models based on synthetic data. This framework was developed and then tested on a fed-batch fermentation process, resulting in models capable of fault detection and diagnosis.

In the first part of the thesis, the fermentation model was modified to simulate fault events and realistic biological variance. Large synthetic datasets were generated through Monte Carlo simulations, providing the basis for developing data-driven fault detection models. The second part of the thesis focused on optimizing four different types of fault detection models: MPCA, PLS-DA, SVM, and ANN. From these, the ANN model was selected for further development and fine-tuning as a fault diagnosis model. Finally, all models were evaluated for their performance and generalization ability.

Developing the fault detection models for this fermentation process yielded a few key insights. Steam-related faults proved to be the most challenging to detect without delay due to their cumulative effect over time. The MPCA and PLS-DA models were limited in their ability to detect certain fault types, while the SVM model slightly outperformed the ANN model in fault detection. The capabilities of the ANN model translated well from fault detection to fault diagnosis. Feature engineering of the airflow-related features further improved its fault diagnosis accuracy by providing the model with additional information. The PLS-DA model exhibited the highest FAR, which could be lowered by adjusting its detection threshold at the cost of a decreased FDR. Regardless of the task being fault detection or diagnosis, the ANN models consistently showed the lowest FAR, making them promising for real-world applications. However, simpler linear models showed better generalization capabilities, indicating a potential weakness of the ANN with the present setup. The computational cost of training and operating the models varied, with the MPCA and PLS-DA models being the least demanding, and the SVM model requiring by far the most resources.

This work and framework provide a starting point for further efforts in developing robust fault detection models for this particular fermentation process as well as other process models given some adaptations. In practice, the framework enables the efficient development of fault

detection models. In the case of an ANN, the model can even be incrementally re-trained with new batch data. However, some limitations should be considered. The fault detection models developed with the framework have not been validated with real process data, which is a crucial step towards their application in practice. Additionally, noise has only been added to weight and airflow, while other features could also exhibit noisy signals. Fault detection was limited to five specific fault types, not covering the whole range of potential fault types.

In summary, this thesis successfully created and applied a framework for developing data-driven fault detection models to a fermentation process, contributing to the efforts for rapid deployment of robust fault detection models.

5.2 Future perspectives

Looking forward, this research could be extended in multiple ways. A crucial next step would be to validate the framework with real process data to confirm the performance and generalization capabilities of the fault detection models. Further improvements to the models trained with the framework can be achieved by exploring different variants such as kernel PCA and kernel PLS. In this way, simpler models with better generalization capabilities could be applied to nonlinear problems. Using recurrent neural networks could also be beneficial to improve the detection of faults with effects accumulating over time. Another promising extension of the framework could be the introduction of a hybrid modeling approach. This could strengthen the robustness and generalization capabilities of the fault detection models [36].

In terms of its practical application, the framework could be implemented in the real fermentation process by validating it with real process data and adapting it based on the validation results. The effects of further fine-tuning and re-training with new process data on the performance and robustness of the models could be explored. The open-source code base can be used to iteratively improve the framework and adapt it to new processes.

Bibliography

[1] M. O. Albaek, K. V. Gernaey, M. S. Hansen, and S. M. Stocks, "Modeling enzyme production with *Aspergillus oryzae* in pilot scale vessels with different agitation, aeration, and agitator types," *Biotechnology and Bioengineering*, vol. 108, no. 8, pp. 1828–1840, Aug. 2011.

[2] S. Yin, S. X. Ding, A. Haghani, H. Hao, and P. Zhang, "A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark tennessee eastman process," *Journal of Process Control*, vol. 22, no. 9, pp. 1567–1581, Oct. 2012.

[3] I. Jul-Jørgensen, P. Facco, K. Gernaey, M. Barolo, and C. Hundahl, "Data fusion of raman spectra in MSPC for fault detection and diagnosis in pharmaceutical manufacturing," *Computers & Chemical Engineering*, vol. 184, p. 108 647, May 2024.

[4] P. Nomikos and J. F. MacGregor, "Monitoring batch processes using multiway principal component analysis," *AIChE Journal*, vol. 40, no. 8, pp. 1361–1375, Aug. 1994.

[5] J. C. Gunther, J. S. Conner, and D. E. Seborg, "Fault detection and diagnosis in an industrial fed-batch cell culture process," *Biotechnology Progress*, vol. 23, no. 4, pp. 851–857, 2007.

[6] P. Nomikos and J. F. MacGregor, "Multi-way partial least squares in monitoring batch processes," *Chemometrics and Intelligent Laboratory Systems*, vol. 30, no. 1, pp. 97–108, Nov. 1995.

[7] B. M. Wise and N. B. Gallagher, "The process chemometrics approach to process monitoring and fault detection," 1996.

[8] I. Monroy, K. Villez, M. Graells, and V. Venkatasubramanian, "Fault diagnosis of a benchmark fermentation process: A comparative study of feature extraction and classification techniques," *Bioprocess and Biosystems Engineering*, vol. 35, no. 5, pp. 689–704, Jun. 2012.

[9] S. Yin, X. Zhu, and C. Jing, "Fault detection based on a robust one class support vector machine," *Neurocomputing*, vol. 145, pp. 263–268, Dec. 2014.

[10] D. Ruiz, J. Nougués, Z. Calderón, A. Espuña, and L. Puigjaner, "Neural network based framework for fault diagnosis in batch chemical plants," *Computers & Chemical Engineering*, vol. 24, no. 2, pp. 777–784, Jul. 2000.

[11] M. O. Albaek, K. V. Gernaey, M. S. Hansen, and S. M. Stocks, "Evaluation of the energy efficiency of enzyme fermentation by mechanistic modeling," *Biotechnology and Bioengineering*, vol. 109, no. 4, pp. 950–961, Apr. 2012.

[12] R. Isermann, "Process fault detection based on modeling and estimation methodsa survey," *Automatica*, vol. 20, no. 4, pp. 387–404, Jul. 1984.

[13] V. Venkatasubramanian, R. Rengaswamy, S. N. Ka, and K. Yin, "A review of process fault detection and diagnosis part III: Process history based methods," *Computers and Chemical Engineering*, 2003.

[14] P. Nomikos and J. F. MacGregor, "Multivariate SPC charts for monitoring batch processes," *Technometrics*, vol. 37, no. 1, pp. 41–59, Feb. 1995.

[15] F. A. P. Peres, T. N. Peres, F. S. Fogliatto, and M. J. Anzanello, "Fault detection in batch processes through variable selection integrated to multiway principal component analysis," *Journal of Process Control*, vol. 80, pp. 223–234, Aug. 2019.

[16] J. Zhou, F. Huang, W. Shen, Z. Liu, J.-P. Corriou, and P. Seferlis, "Sub-period division strategies combined with multiway principle component analysis for fault diagnosis on sequence batch reactor of wastewater treatment process in paper mill," *Process Safety and Environmental Protection*, vol. 146, pp. 9–19, Feb. 2021.

[17] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: Special issue on learning from imbalanced data sets," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 1–6, Jun. 2004.

[18] M. Ruiz, L. E. Mujica, J. Sierra, F. Pozo, and J. Rodellar, "Multiway principal component analysis contributions for structural damage localization," *Structural Health Monitoring*, vol. 17, no. 5, pp. 1151–1165, Sep. 2018.

[19] Y.-J. Park, S.-K. S. Fan, and C.-Y. Hsu, "A review on fault detection and process diagnostics in industrial processes," *Processes*, vol. 8, no. 9, p. 1123, Sep. 9, 2020.

[20] R. G. Brereton and G. R. Lloyd, "Partial least squares discriminant analysis: Taking the magic away," *Journal of Chemometrics*, vol. 28, no. 4, pp. 213–225, Apr. 2014.

[21] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY: Springer New York, 1995.

[22] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large-Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds., The MIT Press, Aug. 17, 2007, pp. 1–28.

[23] M. Nielsen. (). "Neural networks and deep learning," [Online]. Available: http://neuralnetworksanddeeplearning.com/ (visited on 06/19/2024).

[24] H. Madsen, “Introduction,” in *Time series analysis*, OCLC: 1260363861, Boca Raton: Chapman & Hall/CRC, 2008.

[25] A. Géron, “End-to-end machine learning project,” in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Second edition, Beijing [China] ; Sebastopol, CA: O’Reilly Media, Inc, 2019.

[26] I. T. Jolliffe, “Principal components in regression analysis,” in *Principal Component Analysis*. New York, NY: Springer New York, 1986, pp. 129–155, Series Title: Springer Series in Statistics.

[27] “RBF kernel based support vector machine with universal approximation and its application,” F. Yin, J. Wang, and C. Guo, Eds., ser. Lecture Notes in Computer Science, Meeting Name: International Symposium on Neural Networks, Dalian (China): Springer, 2004.

[28] G. Panchal, A. Ganatra, Y. P. Kosta, and D. Panchal, “Behaviour analysis of multilayer perceptronswith multiple hidden neurons and hidden layers,” *International Journal of Computer Theory and Engineering*, pp. 332–337, 2011.

[29] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, Jan. 29, 2017. arXiv: 1412.6980[cs].

[30] S. Sharma, S. Sharma, and A. Athaiya, “Activation functions in neural networks,” *International Journal of Engineering Applied Sciences and Technology*, vol. 04, no. 12, pp. 310–316, May 10, 2020.

[31] P. Geladi and B. R. Kowalski, “Partial least-squares regression: A tutorial,” *Analytica Chimica Acta*, vol. 185, pp. 1–17, 1986.

[32] M. Behzad, K. Asghari, M. Eazi, and M. Palhang, “Generalization performance of support vector machines and neural networks in runoff modeling,” *Expert Systems with Applications*, vol. 36, no. 4, pp. 7624–7629, May 2009.

[33] E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider, “Comparison of support vector machine and artificial neural network systems for drug/nondrug classification,” *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 6, pp. 1882–1889, Nov. 1, 2003.

[34] D. Livingstone, Ed., *Artificial neural networks: methods and applications*, Methods in molecular biology 458, Totowa, NJ: Humana Press, 2008, 254 pp.

[35] M. Shao, X.-J. Zhu, H.-F. Cao, and H.-F. Shen, “An artificial neural network ensemble method for fault diagnosis of proton exchange membrane fuel cell system,” *Energy*, vol. 67, pp. 268–275, Apr. 2014.

[36] J.-A. Jiang, C.-L. Chuang, Y.-C. Wang, C.-H. Hung, J.-Y. Wang, C.-H. Lee, and Y.-T. Hsiao, “A hybrid framework for fault detection, classification, and locationpart i: Concept, structure, and methodology,” *IEEE Transactions on Power Delivery*, vol. 26, no. 3, pp. 1988–1998, Jul. 2011.

A Appendix

A.1 Process parameters

Table A.1: Setup of process parameters. This set of parameters is consistently used throughout this work.

Process conditions		
Parameter	Value	Unit
N (agitation)	250/60	rps
Headspace pressure	1.01325	bar
Substrate feed concentration	500	g/L
Feed density	1100	g/L
Total power input	3	kW/m^3
Inlet air temperature	18	°C
Temperature	28	°C
Humidity	0.90	%/100
Initial conditions (after batch phase)		
Initial substrate concentration	0.05	g/L
Initial enzyme concentration	0.05	g/L
Initial dissolved oxygen concentration	0.000486	Moles O_2/L
Initial volume	300	L
Impeller and reactor properties		
Metzner constant	11	-
Impeller diameter (B2-30 impeller)	0.298	m
Mean circulation time	0.5	s
Radius of fermenter	0.34	m
Cylindrical area of fermenter	0.363	m^2
Height of volume associated with bottom impeller	0.332	m
Bottom impeller width/diameter	0.505	-
Bottom impeller number of vanes/blades	4	-
Top impeller width/diameter	0.685	-

Top impeller number of vanes/blades	4	-
Unaerated impeller power number for Rushton disc turbine	5	-
Relative power draw upon aeration for Rushton disc turbine	0.5	-
Number of impellers	2	-
kLa empirical parameters		
C	28.46	-
a	0.4544	-
b	0.0538	-
c	-0.4516	-
Flow/Viscosity behavior		
C_1	0.0053	-
α	1.56	-
β	0.060	-
C_2	2.69	-
δ	-0.568	-
Physical constants		
Molar mass of oxygen	32	g/mole
Molar mass of CO_2	44	g/mole
Atmospheric pressure	1.01325	bar
Oxygen saturation	0.00057317	Moles O_2/L
Temperature of off-gas	27	°C
Density of dry air	0.001292496	kg/NL
Gas constant-air	2.870028305	L Bar/ K kg
Gas constant-water	4.614024417	L Bar/ K kg
Gas constant	0.08134472	L Bar/ K mol
Density of broth	1050	g/L
Gravitational constant	9.80665	m/s^2
Density of water at 28 °C	996.31	g/L
Feed rate PI controller		
Proportional gain	-40	-
Integral gain	-400	-

Maximum time vector	0, 14, 24, 28	h
Maximum value vector	0.2, 0.9, 1.5, 2.0	L/h
Minimum value	0.1	L/h
Ramp for maximum flow initially	0.00818181	$L/(h^2)$
Ramp for minimum flow initially	-0.004545	$L/(h^2)$
Flow controller bias during initial limitation	0.2	L/h
Max flow after initial limitation	2	L/h
Min flow after initial limitation	0.1	L/h
Feed bias	0.3	L/h
Batch time and sampling rate		
Sampling rate	0.1	h
Simulation end time	230	h
Aeration ramp		
Start aeration	300	NL/min
Final aeration	300	NL/min
Initial DO set point value	0.60 * DO_star	Moles O_2/L
Final DO set point value	0.40 * DO_star	Moles O_2/L
Duration of aeration ramp	12.5	h
Flow rate ramp duration	180	h

A.2 Noise analysis

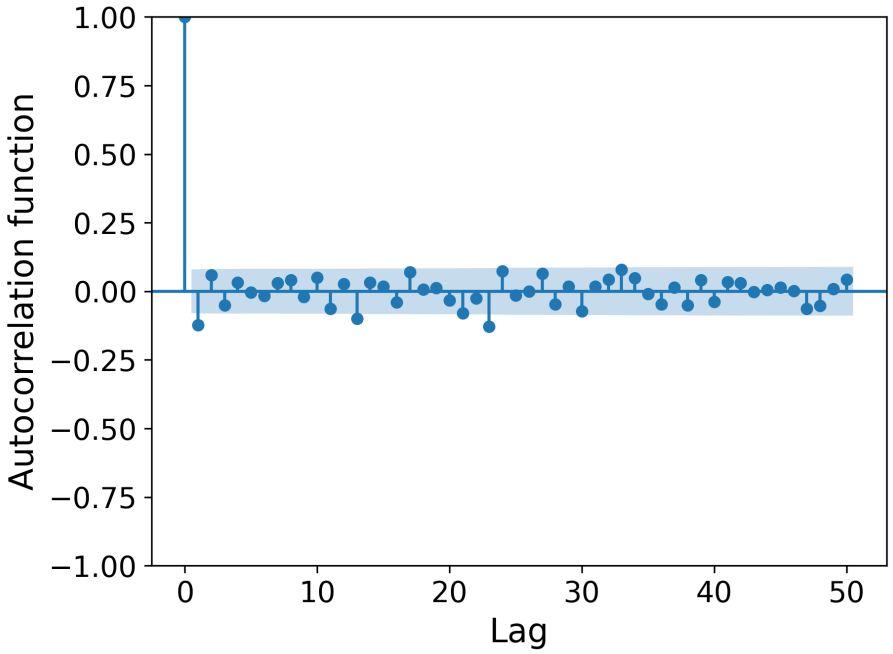


Figure A.1: Autocorrelation function (ACF) of the airflow signal between 50 and 225 hours. The raw data can be seen in Figure 3.2. The ACF was calculated and plotted using the `statsmodels` package in Python. Since the ACF is inside or very close to its confidence bars except for lag 0, a white noise signal can be assumed.

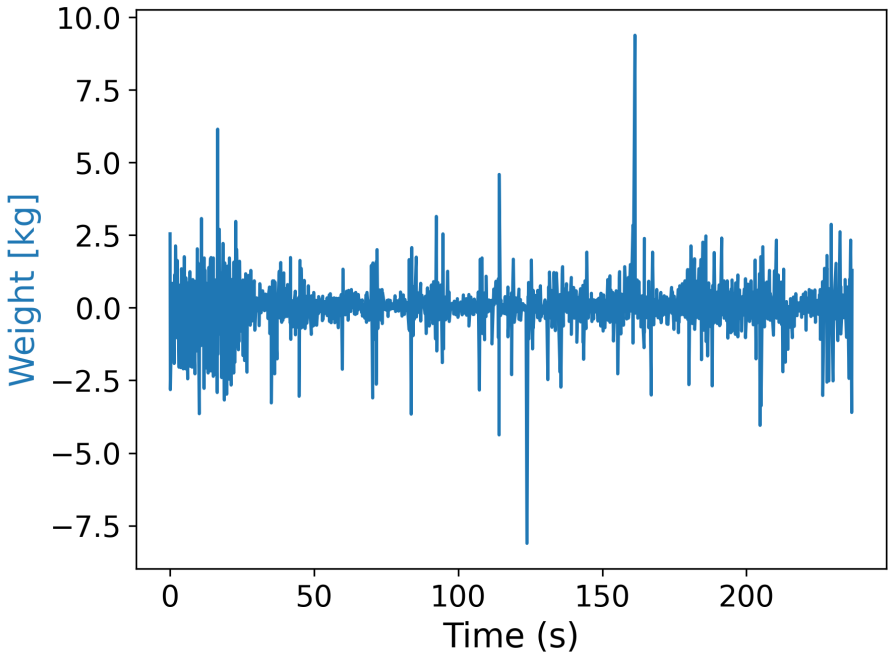


Figure A.2: Weight signal after differencing. The raw data can be seen in Figure 3.2. Differencing leads to a stationary time series with mean zero.

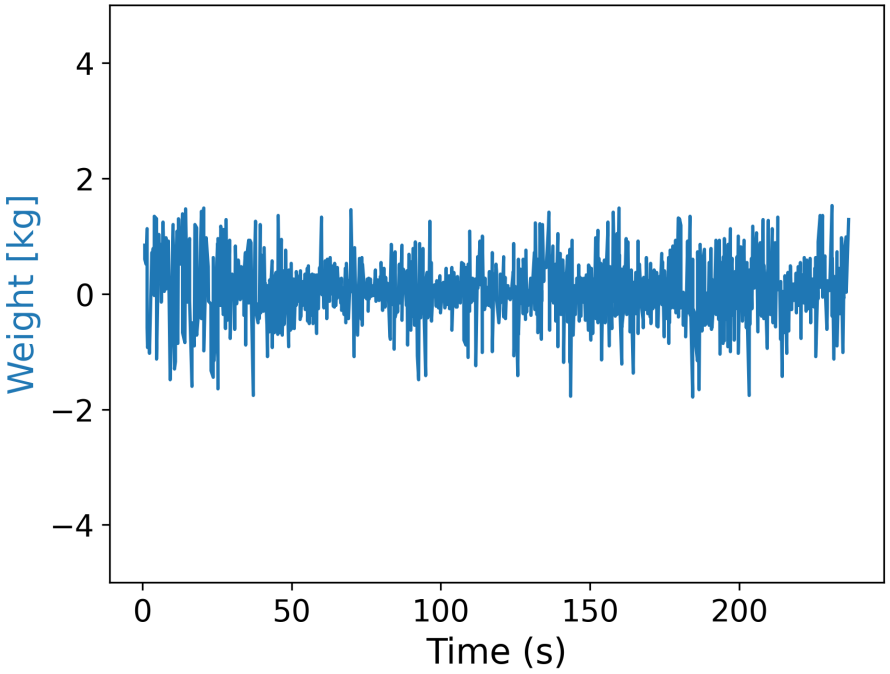


Figure A.3: Weight signal after differencing and trimming. The raw data can be seen in Figure 3.2. Compared to Figure A.2, one can see how trimming reduced the amount of outliers in the time series.

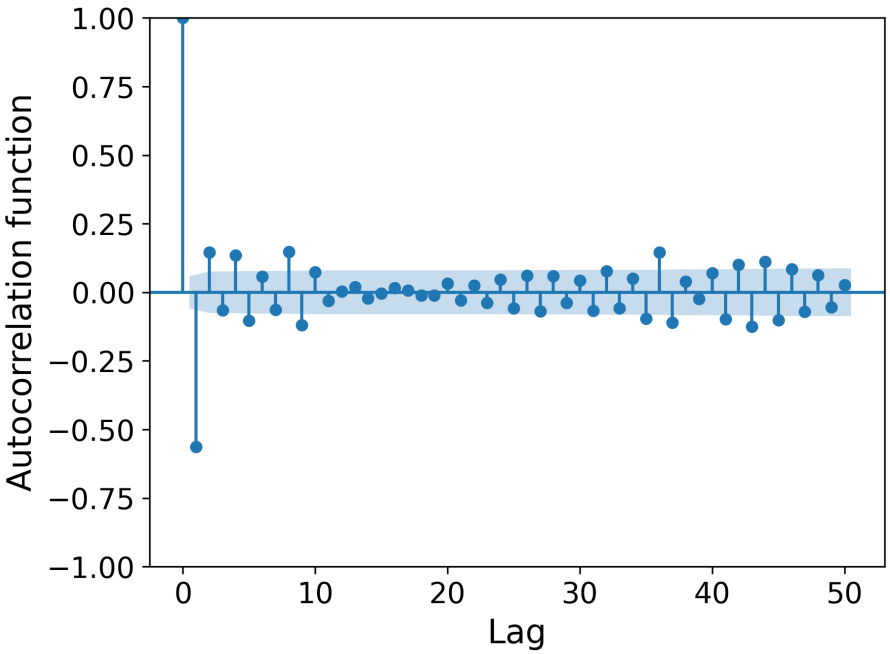


Figure A.4: Autocorrelation function (ACF) of the weight signal after differencing and trimming. The raw data can be seen in Figure 3.2. The ACF was calculated and plotted using the statsmodels package in Python. It still shows minor autocorrelative behaviour at lag 1, but overall behaviour close to a white noise signal.

A.3 Generating large datasets

Table A.2: Dataset sizes for each type of dataset. A separate training set was used for the MPCA since it is trained without fault events, while the SVM models was trained on a training set with a reduced size due to its computational complexity.

Dataset	Total batches	Normal batches	Fault batches
Training	1000	500	100 of each type
Training (MPCA)	1000	1000	0
Training (SVM)	250	125	25 of each type
Validation	100	50	10 of each type
Test	1000	500	100 of each type
OOD test	1000	500	100 of each type

A.4 Model development

A.4.1 MPCA model

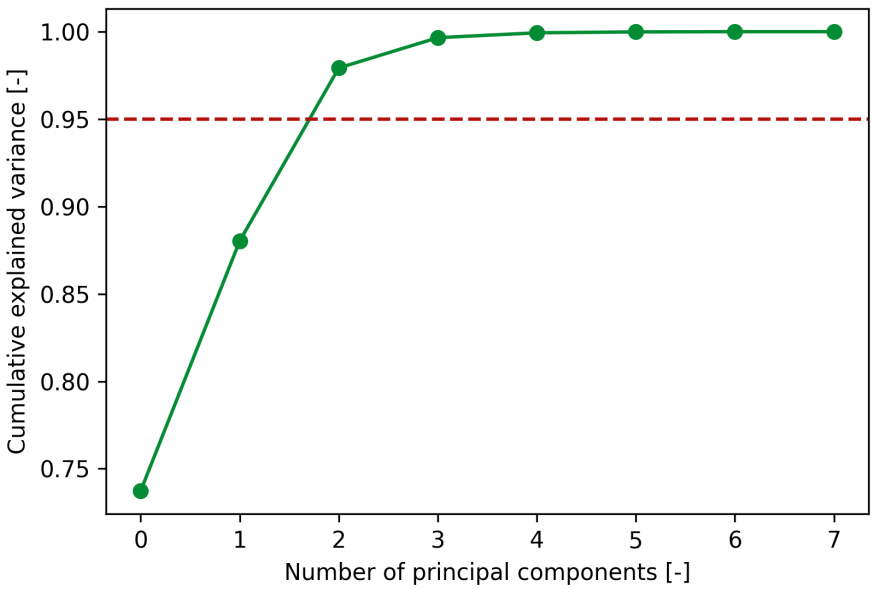


Figure A.5: Cumulative explained variance of the PCA model for each number of principal components used. The dataset consists of 1000 batches without fault events. The red dotted line shows the 95% threshold used for choosing the number of principal components used for the MPCA model.

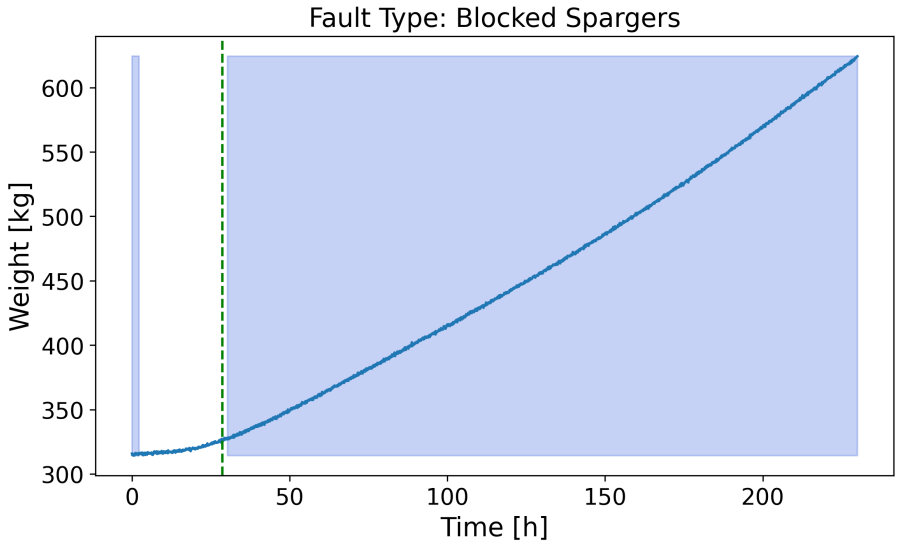


Figure A.6: Fault detection of MPCA model on an illustrative batch with blocked spargers. MPCA model configuration: components = 3, $\alpha = 0.01$, moving time window = 1 h (6 time units). Green dotted line indicates beginning of fault. Blue marked area shows time intervalls classified as having a fault by the MPCA model.

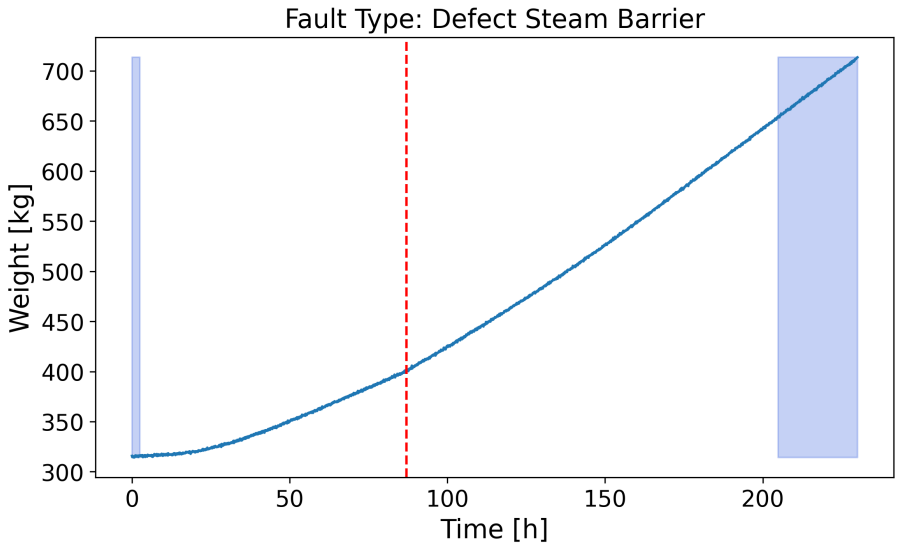


Figure A.7: Fault detection of MPCA model on an illustrative batch with defect steam barriers. MPCA model configuration: components = 3, $\alpha = 0.01$, moving time window = 1 h (6 time units). Red dotted line indicates beginning of fault. Blue marked area shows time intervalls classified as having a fault by the MPCA model.

A.4.2 PLS-DA

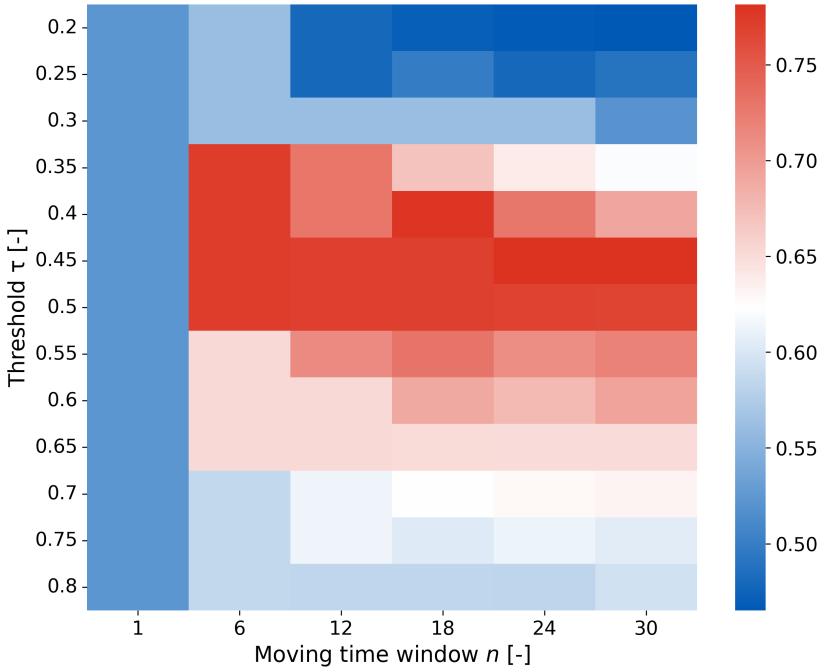


Figure A.8: Accuracy of the PLS-DA model with 6 components depending on the threshold α and the moving time window size n. The model is trained on the 1000 batch training set. The accuracy is evaluated on the 100 batch test set. The optimal set of hyperparameters is at $\tau = 0.45$ and $n = 24$.

A.4.3 SVM

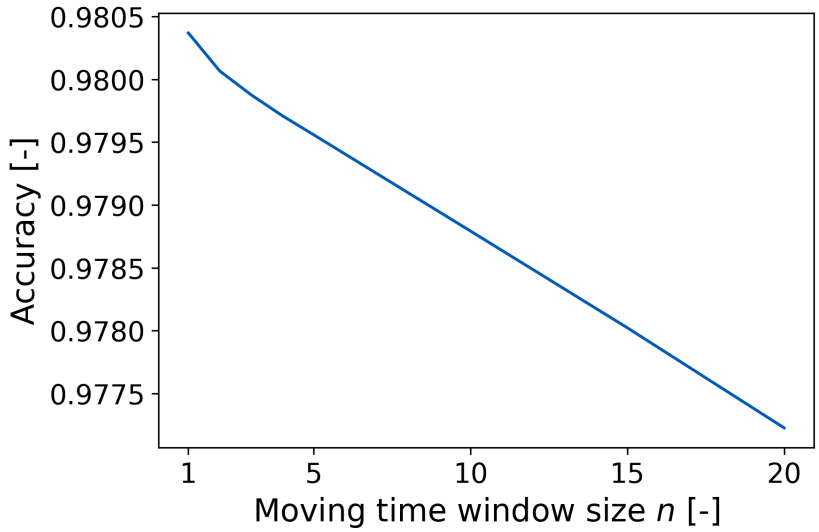


Figure A.9: Influence of the moving time window on the accuracy of the SVM model. The hyperparameters of the model are chosen according to the grid search (see Figure 4.4), meaning $C = 1$ and $\gamma = 1$.

A.4.4 ANN

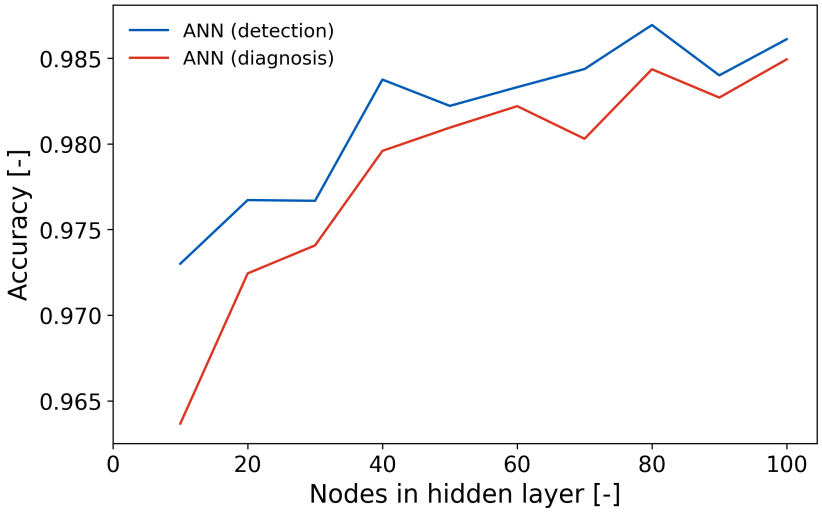


Figure A.10: Influence of the number of nodes on the hidden layer on the ANN models for fault detection and fault diagnosis. The curve of the ANN model for detection shows the accuracy of Figure 4.5 trained at a weight decay of $1 \cdot 10^{-8}$. The curve of the ANN model for diagnosis stems from Figure A.11 at a weight decay of $1 \cdot 10^{-6}$.

A.4.5 ANN for fault diagnosis

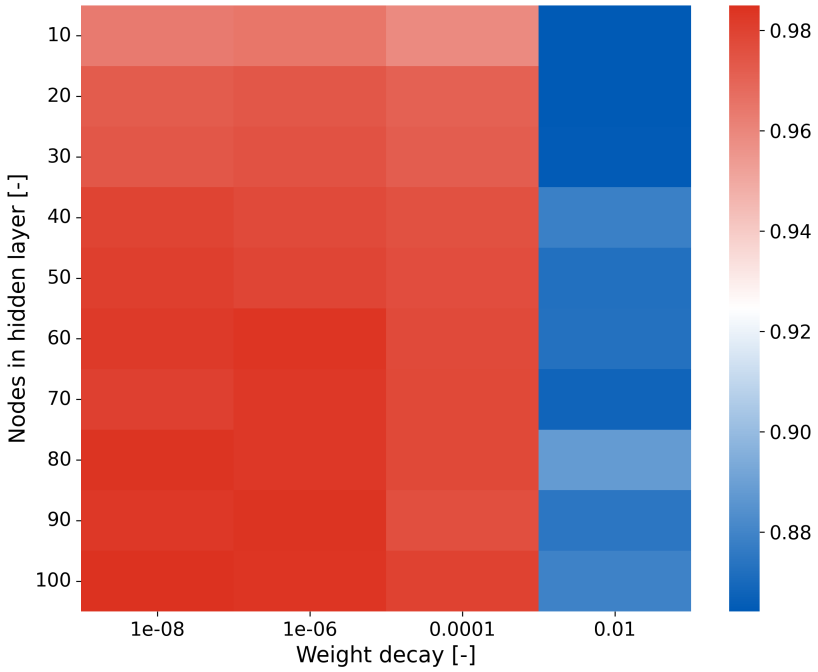


Figure A.11: Accuracy of the ANN model for diagnosis depending on the number of nodes in the hidden layer and the weight decay during training. The models were trained for 100 epochs with callback to the model state with the highest accuracy. The learning rate was $1 \cdot 10^{-3}$.

A.4.6 Model evaluation

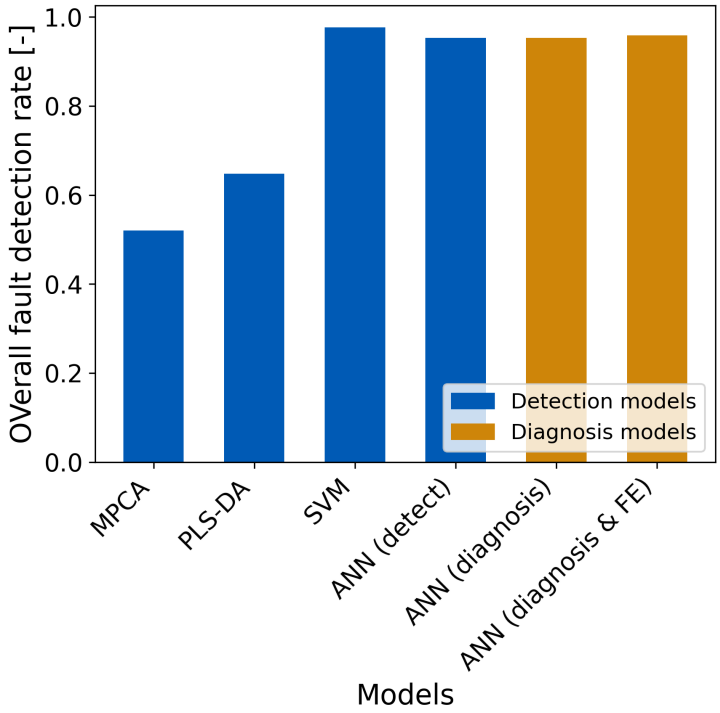


Figure A.12: Overall fault detection rate of all optimized models for fault detection and fault diagnosis.

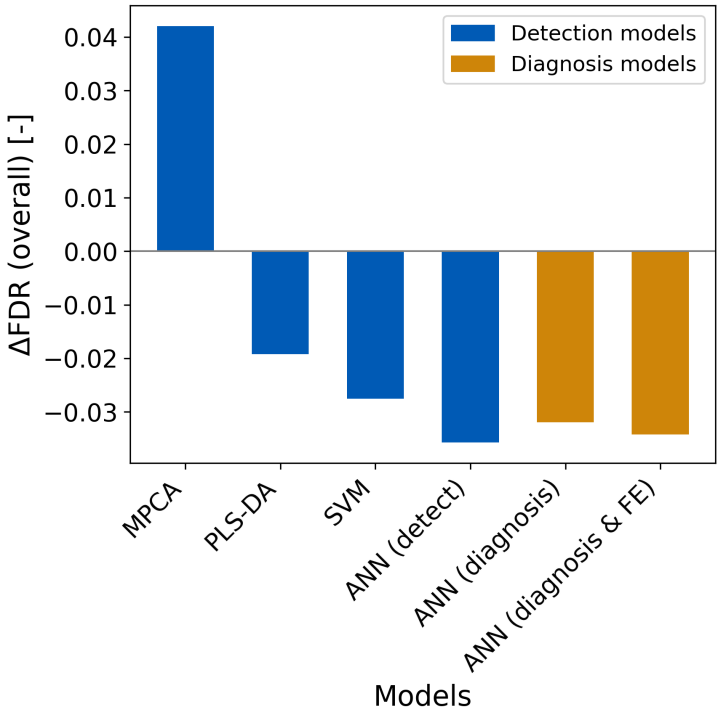


Figure A.13: Difference in the overall FDR between the normal test set and the OOD test set for each fault detection model. Each model configuration is according to their final state as seen in Chapter 4.

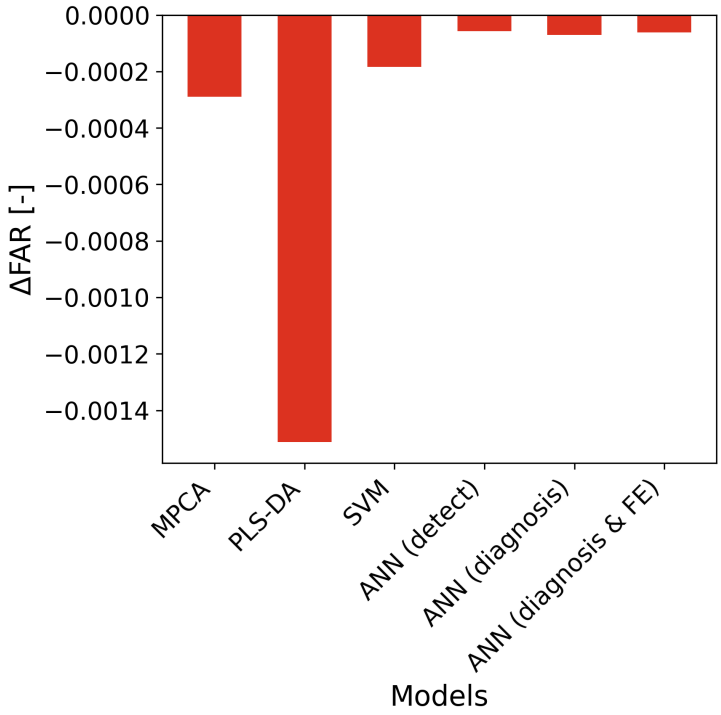


Figure A.14: Difference in the overall FAR between the normal test set and the OOD test set for each fault detection model. Each model configuration is according to their final state as seen in Chapter 4.

Technical

University of

Denmark

Søltofts Plads, Building 228A

2800 Kgs. Lyngby

Tlf. 4525 2822

https://www.kt.dtu.dk/research/prosys