Aalto University
School of Chemical Engineering
Master's Programme in Chemical Engineering

Toni Oleander

Machine learning framework for petrochemical process industry applications

Master's Thesis
Espoo, August 31, 2018

Supervisor: Professor Sirkka-Liisa Jämsä-Jounela
Instructors: Samuli Bergman, M.Sc. (Tech.)
Tomi Lahti, M.Sc. (Tech.)

Aalto University
School of Chemical Engineering
Master's Programme in Chemical Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Toni Oleander

Title:
Machine learning framework for petrochemical process industry applications

Date:	August 31, 2018	**Pages:**	ix + 118
Professorship:	Process control	**Code:**	Kem-90
Supervisor:	Professor Sirkka-Liisa Jämsä-Jounela		
Instructors:	Samuli Bergman, M.Sc. (Tech.) Tomi Lahti, M.Sc. (Tech.)		

Machine learning has many potentially useful applications in process industry, for example in process monitoring and control. Continuously accumulating process data and the recent development in software and hardware that enable more advanced machine learning, are fulfilling the prerequisites of developing and deploying process automation integrated machine learning applications which improve existing functionalities or even implement artificial intelligence.

In this master's thesis, a framework is designed and implemented on a proof-of-concept level, to enable easy acquisition of process data to be used with modern machine learning libraries, and to also enable scalable online deployment of the trained models. The literature part of the thesis concentrates on studying the current state and approaches for digital advisory systems for process operators, as a potential application to be developed on the machine learning framework.

The literature study shows that the approaches for process operators' decision support tools have shifted from rule-based and knowledge-based methods to machine learning. However, no standard methods can be concluded, and most of the use cases are quite application-specific.

In the developed machine learning framework, both commercial software and open source components with permissive licenses are used. Data is acquired over OPC UA and then processed in Python, which is currently almost the de facto standard language in data analytics. Microservice architecture with containerization is used in the online deployment, and in a qualitative evaluation, it proved to be a versatile and functional solution.

Keywords:	machine learning, framework, intelligent digital assistant, OPC UA, data analytics, process industry
Language:	English

Aalto-yliopisto
Kemian tekniikan korkeakoulu
Kemiantekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Toni Oleander		
Työn nimi:	Koneoppimiskehys petrokemianteollisuuden sovelluksille		
Päiväys:	31. elokuuta 2018	**Sivumäärä:**	ix + 118
Professuuri:	Prosessien ohjaus	**Koodi:**	Kem-90
Valvoja:	Professori Sirkka-Liisa Jämsä-Jounela		
Ohjaajat:	M.Sc. Samuli Bergman M.Sc. Tomi Lahti		

Koneoppimisella voidaan osoittaa olevan useita hyödyllisiä käyttökohteita prosessiteollisuudessa, esimerkiksi prosessinohjaukseen liittyvissä sovelluksissa. Jatkuvasti kerääntyvä prosessidata ja toisaalta koneoppimiseen soveltuvien ohjelmistojen sekä myös laitteistojen viimeaikainen kehitys johtavat tilanteeseen, jossa prosessiautomaatioon liitettyjen koneoppimissovellusten avulla on mahdollista parantaa nykyisiä toiminnallisuuksia tai jopa toteuttaa tekoälysovelluksia.

Tässä diplomityössä suunniteltiin ja toteutettiin prototyypin tasolla koneoppimiskehys, jonka avulla on helppo käyttää prosessidataa yhdessä nykyaikaisten koneoppimiskirjastojen kanssa. Kehys mahdollistaa myös koneopittujen mallien skaalautuvan käyttöönoton. Diplomityön kirjallisuusosa keskittyy prosessioperaattoreille tarkoitettujen digitaalisten avustajajärjestelmien nykytilaan ja toteutustapoihin, avustajajärjestelmän tai sen päätöstukijärjestelmän ollessa yksi mahdollinen koneoppimiskehyksen päälle rakennettava ohjelma.

Kirjallisuustutkimuksen mukaan prosessioperaattorin päätöstukijärjestelmien taustalla olevat menetelmät ovat yhä useammin koneoppimiseen perustuvia, aiempien sääntö- ja tietämyskantoihin perustuvien menetelmien sijasta. Selkeitä yhdenmukaisia lähestymistapoja ei kuitenkaan ole helposti pääteltävissä kirjallisuuden perusteella. Lisäksi useimmat tapausesimerkit ovat sovellettavissa vain kyseisissä erikoistapauksissa.

Kehitetyssä koneoppimiskehyksessä on käytetty sekä kaupallisia että avoimen lähdekoodin komponentteja. Prosessidata haetaan OPC UA -protokollan avulla, ja sitä on mahdollista käsitellä Python-kielellä, josta on muodostunut lähes de facto -standardi data-analytiikassa. Kehyksen käyttöönottokomponentit perustuvat mikropalveluarkkitehtuuriin ja konttiteknologiaan, jotka osoittautuivat laadullisessa testauksessa monipuoliseksi ja toimivaksi toteutustavaksi.

Asiasanat:	koneoppiminen, kehys, digitaalinen avustaja, OPC UA, data-analytiikka, prosessiteollisuus
Kieli:	Englanti

Acknowledgements

This master's thesis was done for NAPCON of Neste Engineering Solutions Oy between March and August in 2018.

I would like to thank Professor Sirkka-Liisa Jämsä-Jounela for supervising this thesis and also for the guidance during the project. I also want to thank my instructors M.Sc. Samuli Bergman and M.Sc. Tomi Lahti for the wise comments and the catching innovative thinking. I would like to thank all the previously mentioned and Lauri Haapanen for the help in defining the topic.

Special thanks go to the entire NAPCON Analytics team, and Ville Tähkävuori for the introduction to some machine learning topics and for many smaller things, including the university logos in a crisp vector format on the abstract page.

I am grateful for the opportunity to write the master's thesis for NAPCON, about an interesting topic. Last but not least, I wish to thank my parents for the support throughout the studies.

To balance the tiring factualness of this thesis, here is a Sudoku for you:

D		7	F				A		1				9	2	
C		B	6					E	A		7				
		A	0		2			9	6	C	5	7	F		E
	2		3		7				F		0	5	B		
	D		4	E	0		5				A		3		6
	6	E	8			4			D		F	B	A	1	
B		F					8				E		0	5	D
	A			B		F		5			C			E	
	4			0			B		C		3			8	
8	F	9		D				7					5		3
	B	2	5	7		3			9			E	4	D	
6		3		5				4		0	D	A		B	
		8	2	1		C				E		D		0	
4		5	B	2	9	D	7			3		1	E		
				4		A	E					9	C		5
	9	1				6		C				3	2		4

Espoo, August 31, 2018

Toni Oleander

Abbreviations and Acronyms

AE	Autoencoder
AI	Artificial intelligence
ANFIS	Adaptive neuro-fuzzy inference system
ANN	Artificial neural network
API	Application programming interface
ASIC	Application-specific integrated circuit
CEP	Complex event processing
DNN	Deep neural network
DR	Dimensionality reduction
ES	Expert system
FDA	Fisher discriminant analysis
FDI	Fault detection and isolation
GMDH	Group method of data handling
GPU	Graphics processing unit
GUI	Graphical user interface
HDFS	Hadoop Distributed File System
IDE	Integrated development environment
IE	Inference engine
IIoT	Industrial internet of things
KA	Knowledge acquisition
KB	Knowledge base
KBS	Knowledge-based systems
KDD	Knowledge discovery from database
KR	Knowledge representation
LDA	Linear discriminant analysis
MDP	Markov decision process
MES	Manufacturing execution system
ML	Machine learning
MOM	Manufacturing operations management
MOO	Multi-objective optimization

MPC	Model predictive control
MSE	Mean squared error
NMPC	Nonlinear model predictive control
OSS	Operator support system
PCA	Principal component analysis
PDF	Probability density function
PLS	Partial least square
RL	Reinforcement learning
SSE	Sum of squared errors
SVM	Support vector machine
UI	User interface

Contents

Chapter 1

Introduction

Machine learning has many practical and potential uses in process industry, for example in soft sensors, process monitoring, fault detection and predictive maintenance to name a few [1]. A common denominator of these applications is the data-driven approach: instead of manually programming all functionalities, systems are able to automatically learn from *big data*, which is a term used to describe the large amount of varied data collected continuously [2, p. 21].

Machine learning became more viable after the lack of data a few decades ago was fixed by the fast development of technology and the decreased cost of storing the data [2]. Another factor was the fundamental advance in training deep neural networks effectively [3]. Currently, the limiting factors are memory and processing power which usually translates into time: a self-driving car must be able to handle the data in real time and predict an oncoming accident before it happens. [4] One example of the significant development in computing power is one of the early autonomous vehicles in the 1970s, based on a lunar vehicle of NASA: the vehicle was able to move only about a meter at a time, pausing for half a minute to compute the next movement [5]. Nowadays, to speed up processing, the state-of-the-art machine learning systems commonly incorporate graphics processing units (GPU) [4] or application-specific integrated circuits (ASIC), such as the tensor processing unit (TPU) developed by Google [6].

Artificial intelligence (AI) is another growing topic, at least partly due to the overall development of machine learning which is an important part of many AI systems. Such systems include digital assistants that understand speech and can not only do what the user said, but also what the user actually intended to accomplish. Artificial intelligence systems may also help humans to operate machines or vehicles more optimally. However, machine learning only provides the learning part of AI, and additional decision mak-

ing is usually required. As a matter of fact, machine learning could be said to be mostly statistics and mathematical optimization with certain specific goals such as predictive analytics. [4, pp. 9–21] Artificial intelligence, on the other hand, tries to solve complex problems as humans do [7]. As the share of non-routine occupations and tasks keeps increasing due to the automation of routine and repetitive tasks, applications of artificial intelligence are predicted to become increasingly advantageous [8].

Machine learning and artificial intelligence are not new concepts: They have existed almost as long as digital computers, but especially artificial intelligence has followed the series of hype cycles well-known from many emerging technologies. [3] Even the term *AI winter* is used to describe the periods of disappointment caused by high expectations and the discovery of the limitations of AI systems [9]. During these periods, the term *artificial intelligence* was so unpopular that some scientist and engineers avoided using it [5], which may have contributed to the incoherent terminology of the present AI field. However, AI and machine learning are popular technologies again [3], and this time the background development in the subfields of AI, technological advances mentioned above and already useful-proven applications may result in a more visible success of artificial intelligence also in process systems engineering.

Expert systems are one well-known example of a system implementing artificial intelligence. Expert systems became somewhat unpopular after their limitations shattered the expectations in the 1990s [9, 10]. The challenges include acquiring, structuring and formalizing the usually heavily domain-specific knowledge that the system relies on [7, 11, 12]. Attempts to overcome this problem lead to a multitude of disciplines and domains such as cognitive studies of artificial intelligence, decision support systems, automated reasoning, knowledge automation, big data knowledge engineering, operations research and ontology engineering [7, 12–15]. Modern machine learning algorithms may provide solutions to some of these problems. For example, high-level decision making and planning is typically expressed as *Markov decision processes* for which stochastic dynamic programming (*e.g.* approximate dynamic programming in reinforcement learning) provides a solution [3]. Wagner [16] stated that the research related to expert systems has continued for over thirty years but the focus has moved from "classic" expert systems to a hybrid model of knowledge-based systems incorporating a variety of AI techniques.

1.1 Problem statement

The research questions for this master's thesis are:

- What is the current state of digital advisory systems for a process operator, to assist operation from the process control point of view?
- What approaches have been used to implement such advisory systems and how could modern machine learning be possibly used?
- How to design and implement a framework for developing and deploying machine learning applications, capable of satisfying process industry specific requirements?

1.2 Structure and objective of the thesis

The thesis is divided into two parts: The literature part (Chapters 2–5) reviews the current state of advisory and decision support systems for process operators. Approaches that are used in creating the systems that can be found in literature, are studied. Focus is given to modern machine learning methods as much as possible. Finally at the end of the literature part, case studies of advisory systems are reviewed.

In the experimental part (Chapters 6–9), a proof of concept for a machine learning framework is implemented. An architecture is first designed according to the requirements specification which is based on the requirements of the process control related applications in petrochemical industry. Finally, a couple of use cases are used to qualitatively evaluate the proof of concept.

1.3 Excluded contents

Both the literature part and the experimental part of the thesis have a common goal which is a set of process-specific applications using low-level machine learning libraries, developed on a framework which seamlessly integrates process control related applications and modern machine learning tools. However, reaching the final goal is clearly out of the scope of one master's thesis, and therefore, the literature part and the experimental part mostly have separate conclusions.

In the experimental part, the target is to develop the machine learning framework on a proof-of-concept level. Focus is primarily given to functionality, and therefore, unit tests or quantitative performance tests are not

implemented within the thesis work. Furthermore, individual machine learning algorithms will not be evaluated but the emphasis is more on creating a framework that supports multiple algorithms, and enables experimenting with, evaluating and comparing them.

Chapter 2

Theoretical background

In this chapter, an overview of advisory systems for a process operator is given, while also giving a better understanding of what actually is referred to with the term *advisory systems* in this thesis. In addition, the key concepts and terminology of machine learning are concisely summarized.

2.1 Process operator's advisory systems

Requirements for more optimized manufacturing in the petrochemical and other manufacturing industries are increasing along with strategies such as the Europe 2020, the Advanced Manufacturing Partnership in the USA, Industry 4.0 in Germany and China Manufacturing 2025 in China. Optimizations include for example increased energy efficiency, environmental protection, individualized customer needs, product cost reduction and diversified raw material sources. [14, 17, 18] In general, one way to realize these objectives is to make manufacturing processes more intelligent using modern methods of information technology such as the Internet of Things, cloud computing and big data technologies [1, 18]. A roof term *smart manufacturing* includes using data analytics to improve system performance and complement traditional methods of decision making [17]. The key technologies for making *smart factories* in petrochemical industry are summarized in Figure 2.1. In the figure, *virtual assistants* mentioned in *intelligent human-computer integration* aim to improve the overall human-machine collaborative decision-making. [18]

For petrochemical and other process industries, it is typical that fluctuations in concentrations and the diversity of chemical reactions and unit operations result in strongly nonlinear control problems with complex couplings and competing objectives. For a process operator, with limited knowledge of

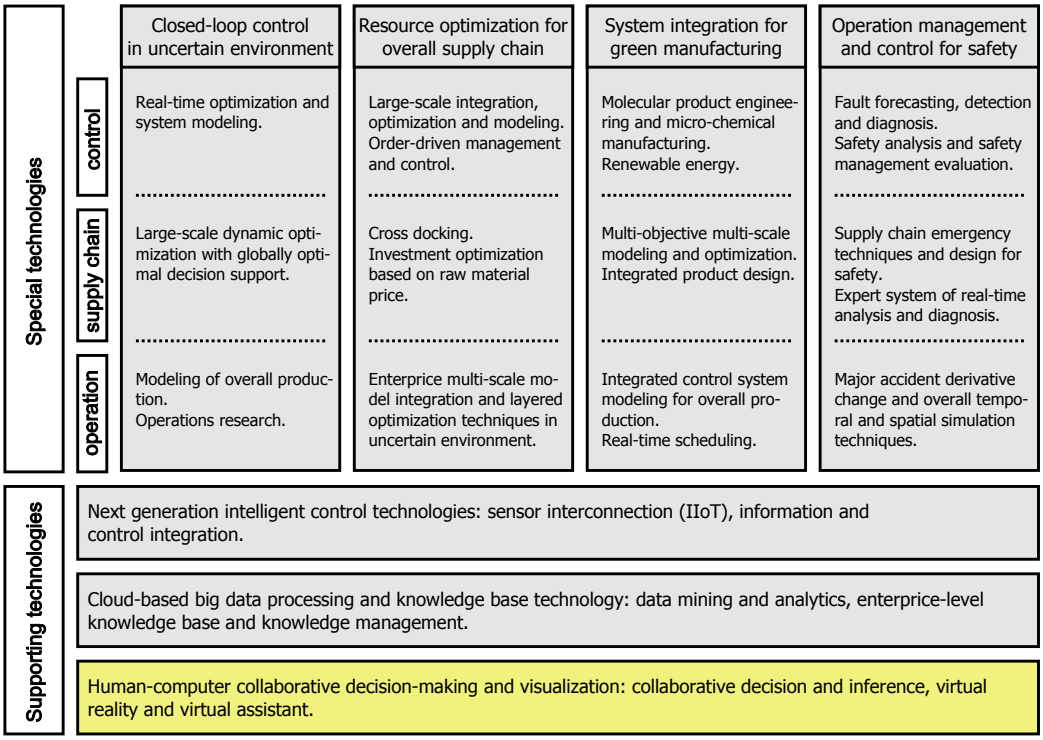


Figure 2.1: Key digital technologies of smart petrochemical manufacturing (see the original full figure in Appendix A) [18].

process dynamics it is difficult to make decisions that would reach the global optimum [14], which is usually done better by computers that are by comparison able to handle virtually an unlimited number of variables. On the other hand, human operators are able to use intuition and expert knowledge to make decisions in situations where the control system may not even have all the required input signals (*e.g.* sensors). [19]

Advisory systems (also *operator support systems* (OSS), *decision support systems* or *intelligent support systems*) try to combine the advantages of both human operators and computerized control systems [19] by giving advice to the operator and by augmenting the process operator's capacity to monitor and process information, and make decisions [20]. Advisory systems analyze process data and present it in a way that gives information about the events and trends in the process, and that way help in decision making. Systems based on both artificial intelligence and non-AI technologies have been used, and research has been done to combine them into a more robust system. [21]

Advisory systems can be built based on data-driven, analytical and knowl-

edge-based methods [21]. Data-driven and knowledge-based methods are interesting since large amounts of data are constantly accumulating into history databases in modern chemical plants. However, many of the best methods for extracting the knowledge from process data are *supervised* methods that require *labeling* the data, which requires knowledge of the process and a significant amount of work. On the other hand, *unsupervised* learning can be used to automatically discover groups of data: Extracting information and learning patterns from *dimensionally reduced* and *clustered* process data is called *knowledge discovery from database* (KDD), which includes *data mining*. [22] The steps of KDD are summarized in Figure 2.2.

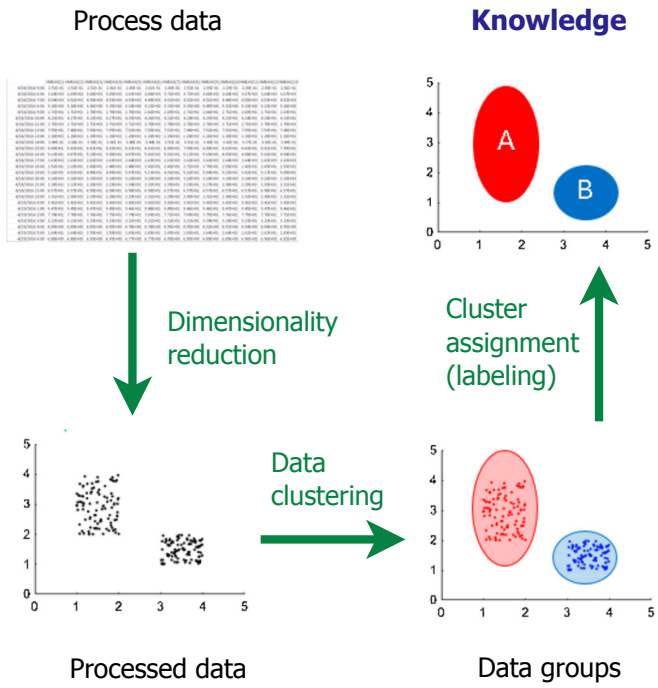


Figure 2.2: Outlined steps of knowledge discovery from database [22].

On a practical level, in order to combine supervised and unsupervised learning for an industrial application, such as an advisory system, a software framework with the following features is usually required: Separating or labeling the data, and training the model should be possible to be done also by non-experts in data science. The framework should also assist the user in identifying new knowledge and deploying the trained models. [22]

Compared with advisory systems, another analogous and closely related area of research is the development of autonomous vehicles, the goal of which

is to make driving more convenient and safer by reducing the probability of human error. While the environment is very different compared to chemical processes, similar challenges are faced with uncertainty due to unknown factors, noisy data and physical limitations of sensors *etc.* Many approaches have been proposed to implement decision making for autonomous driving, including rule-based systems (*e.g.* finite and hybrid state machines) in controlled environments, and knowledge base or ontology-based inference. Partially Observable Markov Decision Processes (POMDP) have also been suggested for determining the optimal control trajectory. [23]

2.2 Machine learning

Machine learning (ML) is a subfield of computer science that uses the theory of probability and statistics to recognize patterns and learn from data to make predictions and improve performance in a given task, without explicit programming [1, 24]. In process industry, machine learning has been used in data mining and analytics, *e.g.* for online soft sensing, quality prediction, anomaly detection, data clustering and dimensionality reduction. In general, by constructing models from process data, new useful information can be created, patterns in data can be identified and predictions can be made, and these in turn can be used *e.g.* in optimizing decision making processes. [1]

The machine learning algorithms most commonly used for data mining and analytics in the process industry, are presented in Figure 2.3.

2.2.1 ML terminology and algorithm classification

Machine learning methods can be categorized into *supervised* learning, *unsupervised* learning, *reinforcement* learning [1, 3] and *semi-supervised* learning. Supervised and unsupervised learning are the most commonly used methods in industrial applications. [1] A summary of the characteristics of selected machine learning algorithms is presented in Appendix B.

Learning from data generally means optimization where the (algorithm's internal) *cost function* (also known as *error function*, *objective function*, *loss function* when minimizing or *scoring function* when maximizing) guides the optimization by pointing out the changes in the internal parameters that are the most significant for outputting better predictions. During (supervised) learning, the input of the cost function can be *e.g.* all the variables that will define the learned model. The parameters of the cost function consist of all the labeled training data. Therefore, the output of the cost function gives a value for the goodness of the model, which then can be optimized

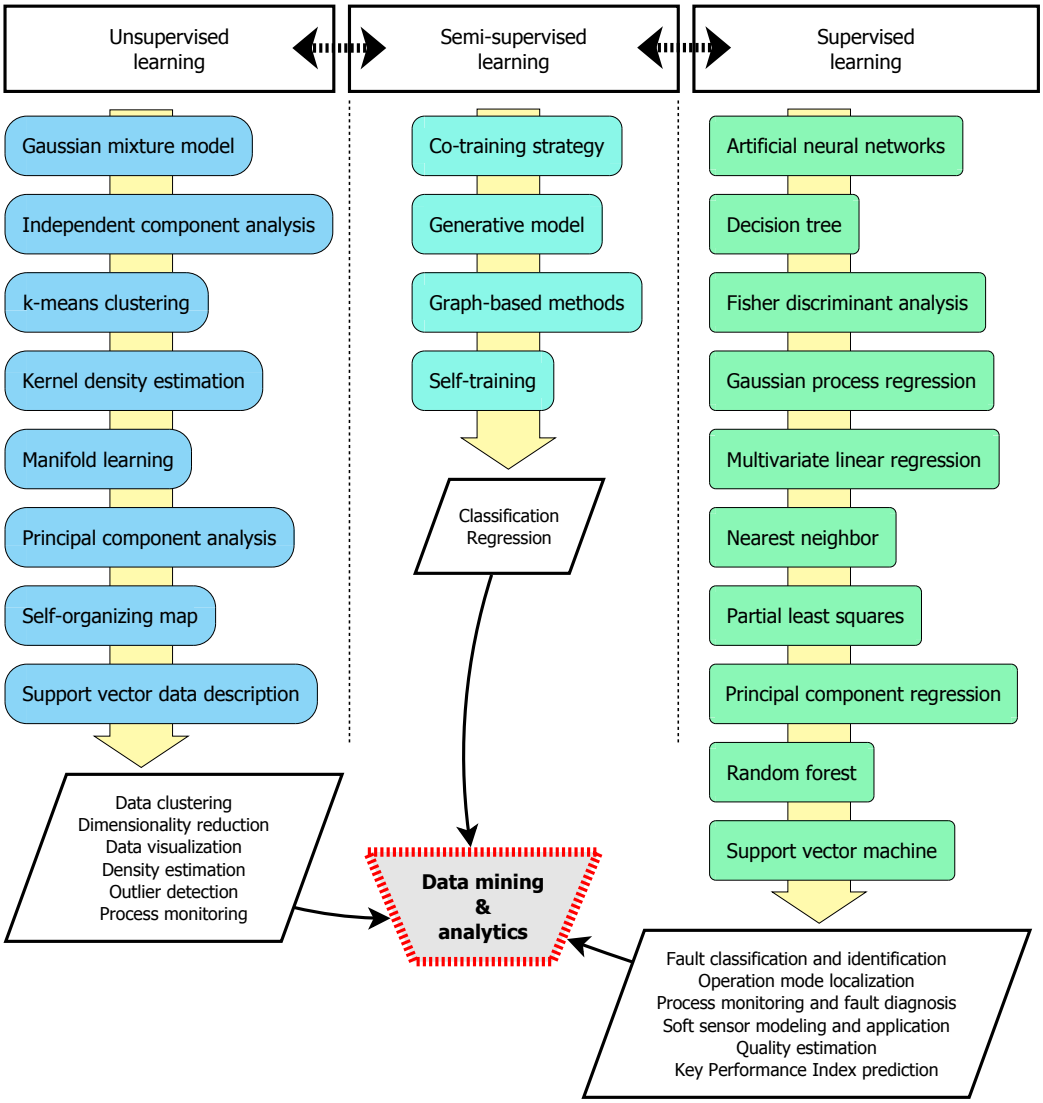


Figure 2.3: Machine learning algorithms most commonly used in process industry, and their applications [1].

using *e.g.* the gradient descent algorithm, resulting in the learned model, or rather the (at least locally) optimal internal parameters for the model. [4, pp. 167–177]

Most machine learning algorithms use their own cost functions, only the hyperparameters of which can be customized. A cost function that is used in optimization or validation, but not in the learning algorithm, is sometimes called an *external cost function*. [4, pp. 167–177]

The data X used for learning consists of a set of n *examples* (also known as *points*) x_i, so that $X = (x_0, ..., x_{n-1})$. By convention, X is either a (single-feature column) vector, or a matrix where the rows represent examples and the columns represent *features* (also known as *independent variables* or *predictors*). [4, p. 150][25] If the data is labeled, the training dataset is made of pairs (x_i, y_i) where $y_i \in Y$ are called *labels* (also known as *targets* or *responses*) of the examples x_i. Both the examples and the labels can be either quantitative or qualitative, *i.e.* continuous or discrete. [4, 26]

Supervised learning means learning from data X that includes associated target responses in the *response vector* (or in some cases *matrix*) Y which can be numeric values or qualitative *labels* [3, 4]. The learning involves mapping the labels y_i to the examples x_i with as accurate approximation as possible, and obtaining the conditional probability model $P(Y|X)$. The learned model can then be applied on new data, *i.e.* $f : X \to Y$ where f represents the learned model. Supervised learning can be categorized into *classification* and *regression*. In general, their difference is that in classification the output Y (*i.e.* labels) is discrete and in regression it is continuous. [27]

Unsupervised learning is used for discovering patterns and trends in multidimensional unlabeled data X. The learning involves obtaining information about the distribution of the data, $P(X)$, and it can be useful in clustering, dimensionality reduction and outlier detection [3].

Semi-supervised learning can be considered as a bridge between supervised and unsupervised learning. While being less often used in industrial applications, it has recently gained more attention in process industry. [1] However, the term 'semi-supervised' may be somewhat open to interpretations, since according to Ge *et al.* [1], any (un)supervised learning algorithm can be turned into semi-supervised with an appropriate modification or information integration. Either way, the idea of semi-supervised learning is to minimize the costly requirement of labeling the data, and thus a large amount of unlabeled data and only a small amount of labeled data is used for training the model [1]. The learning uses the unlabeled data to obtain the probability distribution of the input space $P(X)$ and jointly optimizes the prediction over the labeled and unlabeled data. In other words, $P(Y|X)$ and $P(X)$ are optimized together in a weighted combined objective. [3] In practice, semi-supervised learning can comprise for example first clustering all data, and then labeling the clusters based on the labeled data. Consequently, the unlabeled data helps in finding the boundary of the clusters more accurately, as illustrated in Figure 2.4. Semi-supervised learning usually requires assumptions, such as the cluster assumption: "If points are in the same cluster, they are likely to be of the same class" (where *class* is derived from the labels). [26]

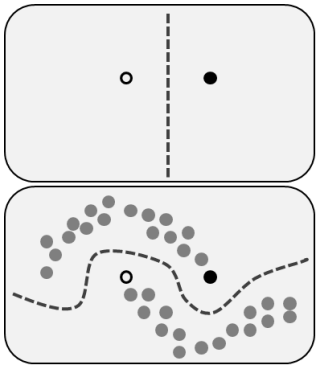


Figure 2.4: Semi-supervised learning: The influence of unlabeled data (grey circles) on the decision boundary (or learned model, dash line) [28].

Reinforcement learning is used in applications where the algorithm must make decisions (based on unlabeled data) that have consequences. Learning happens by trial and error, when positive or negative feedback is given to the decisions. [4] According to Ge *et al.* [1] reinforcement learning is rarely used in process industry applications. However, according to Silver [29] reinforcement learning is closely related to the theories of optimal control and operations research.

2.2.2 ML model development workflow

Developing machine learning models either manually or programmatically roughly follows the workflow presented in Figure 2.5. In general, the *extract, transform, load* (ETL) process is used to describe the process of combining multiple data sources, transforming the data into a proper format, and loading it to the final database [24]. In process automation, historical process data is often already available in a well-structured form in a database.

Dataset preparation includes *feature extraction* in which the relevant parts of the data are extracted, usually in a lossy manner. Then, the dataset is partitioned into at least two parts: the training and testing datasets, and possibly a validation dataset. The training dataset is used for adjusting the parameters (*i.e.* weights) of the model. The testing dataset is used for measuring the accuracy of the trained model. If alternative model types or architectures are compared, the third partition is a validation set that is used for the comparison. The proportions of the partitions (training, validation, test) are typically 70:20:10. [24]

Data preprocessing includes *feature engineering*, in which the selected

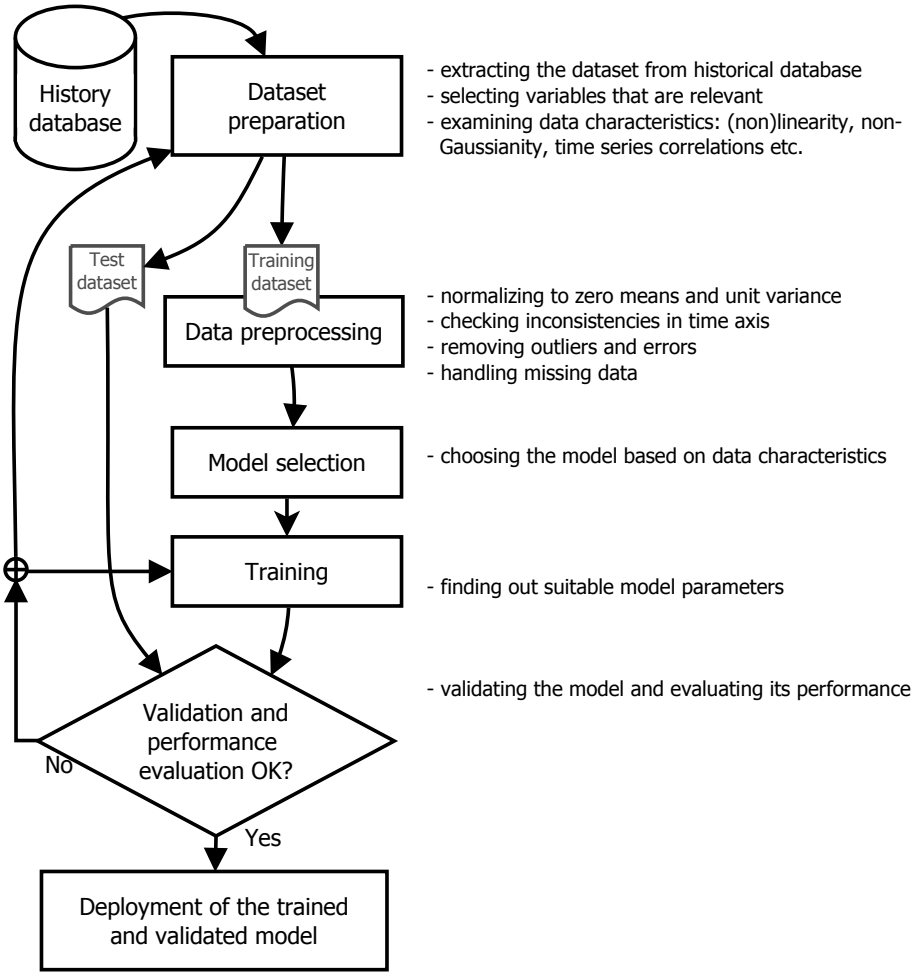


Figure 2.5: The process of creating a machine learning model [1].

data is transformed in a way that makes it easy for the model to generalize. The transformation can be *e.g.* a simple calculation, based on expert knowledge or even common sense. For instance, *one hot encoding* transforms discrete groups or integers into a binary array containing only ones and zeros. [24] For example for four integers between 0...3, one hot encoding would perform a transformation as follows:

$$[1,0,2,3] \rightarrow \begin{matrix}[[0,1,0,0] \\ [1,0,0,0] \\ [0,0,1,0] \\ [0,0,0,1]]\end{matrix}$$

Feature engineering is not usually a mandatory step, but in many cases it improves the quality of the model if done correctly. Other commonly used methods in preprocessing include PCA and autoencoders (see Section 3.1.1) which are used for dimensionality reduction. [24]

In the training step, an iterative optimization is performed to obtain the parameters of the model. One *iteration* defines one instance of calculating the error gradient and adjusting the model parameters. In traditional batching, the *batch* includes the entire training dataset, although in practice this is often impossible due to memory limits. By contrast, in *stochastic gradient descent* the number of samples per batch (*i.e. batch size*) is one. In practice, often the best compromise between these two methods is the *mini-batching*, in which a small subset of the training dataset is used at a time, until the whole dataset is used. Each pass of the entire training dataset is called an *epoch*. [24]

A common way to evaluate the fitness of the model in the current conditions, both before and after deploying, is to calculate error metrics. In regression, commonly used indicators include the mean absolute error, the median absolute error, and the mean squared error. In classification, general metrics include the accuracy, precision, recall and F-measure, and for visualization the confusion matrix. Clustering quality can be measured with the help of the silhouette coefficient, homogeneity, completeness and the V-measure. [24]

Chapter 3

Approaches for decision support in advisory systems

Process operations management (closely related to *manufacturing operations management* (MOM) and *manufacturing execution systems* (MES) [30–32]) consists of tasks from top-level scheduling and planning to low-level control and data acquisition. A lower level implements the decisions made on higher levels, but on the other hand, events on all levels affect the entire decision making chain. Integrated tools for decision support aim at improving decision making by automating it when possible, or by supporting it when human intervention is required, in order to reduce manufacturing cost and improve process safety and product quality. [33]

One particular sector of decision making is any manual control of a process, *i.e.* process operators' actions to control the process, especially during startups, shutdowns and abnormal conditions [33]. Advisory systems are often only one component of a wider framework that is designed for *e.g.* intelligent process monitoring, diagnosis, control and optimization, including process modeling and simulation. Especially *fault detection and isolation* (FDI) is a common motivator for such systems, probably due to the universality of FDI [34] in any manufacturing process.

An integrated decision support system does not usually try to replace human operators with computers because of the risks involved in supervisory decision making, liability and legal issues, and also limitations in intelligent systems. Therefore, in order to assist plant operators, the goal of an advisory system is to analyze data and present the information to the operator in an easily understandable manner. The information should contain insight to the near and distant future behavior of the process, with explanations and recommendations. [33]

In this chapter, approaches for implementing a decision support system

for a process operator are studied. The most common approaches can be categorized into data-driven, analytical and knowledge-based approaches [35], although one system may contain characteristics from multiple categories. In addition, an overview of the concept of agent-based decision support systems is given. Multi-objective optimization gives yet another approach which is sometimes used together with knowledge bases, resulting in knowledge-driven optimization.

3.1 Data-driven approach and methods

Data-driven methods have been successfully used in process monitoring applications based on models that are constructed almost entirely from process data. These methods include the principal component analysis (PCA), Fisher discriminant analysis, partial least squares (PLS) regression and canonical variate analysis. Especially PCA and PLS have been used to extract features from process data. [21]

3.1.1 Principal component analysis and autoencoder

Principal component analysis (PCA) is an unsupervised procedure used for dimensionality reduction and outlier detection. Dimensionality reduction helps for example visualization: High-dimensional process data can be projected into two or three dimensions which can be visualized while still conveying the characteristics of the state of the process. PCA is also used for preprocessing datasets for machine learning to improve results, compared to using the entire dimensionality of the observation space in training. Furthermore, PCA can be used to identify variables of the process data that most contribute to an event. [21] An example of using PCA for three-dimensional data is presented in Figure 3.1.

Autoencoder (AE) is an artificial neural network (ANN) that also can be used to learn an efficient representation (*i.e. encoding*) of a dataset [3]. Whereas PCA finds a linear orthogonal projection that maximally captures the variance [36], which consists of linear combinations of variables, autoencoder generalizes this to nonlinear combinations. Therefore, AE can be considered as a nonlinear generalization of PCA. [3]

Linear discriminant analysis (LDA) is yet another technique for dimensionality reduction, for labeled data. Compared to PCA, instead of maximizing the variance, LDA finds a linear orthogonal projection that maximizes the difference between two or more classes. Also, compared to clustering, the data used in LDA is labeled. [36]

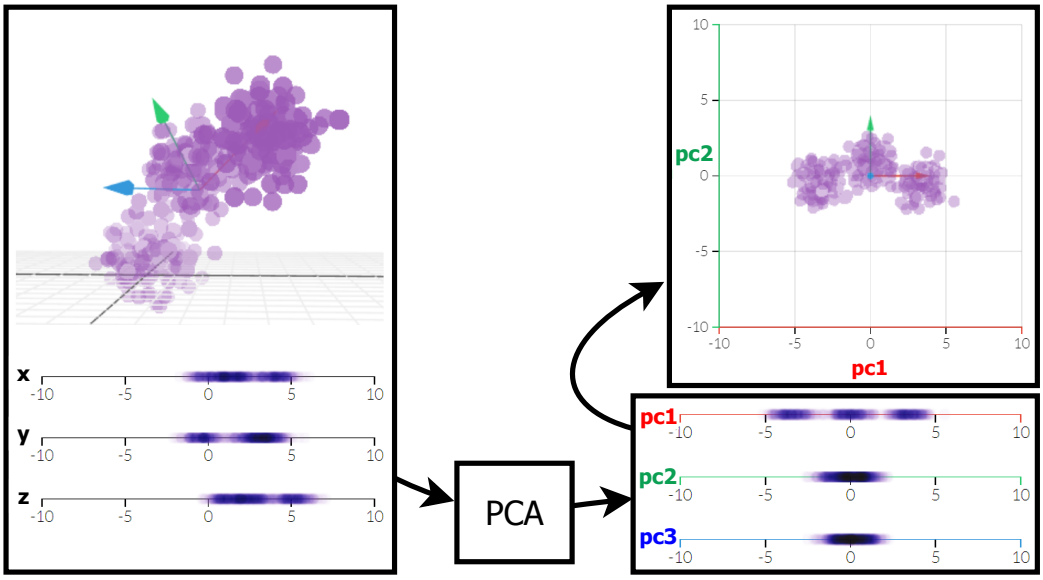


Figure 3.1: Dimensionality reduction from 3D to 2D with PCA. In case of 3D, PCA finds the optimal angle of view. [37]

3.1.2 Partial least squares regression

Partial least squares (also known as *projection on latent structures [38]*) (PLS) regression is a multivariate algorithm in supervised machine learning. PLS combines features from PCA (dimensionality reduction) and multiple linear regression. [38] After the PLS model has been created, it is possible to use only the *predictor matrix* X (for example process variables) to predict the Y matrix (for example product quality). Therefore, PLS has been used in soft sensors, other process monitoring and fault detection. [21]

During the learning, the objective is to find the *regression coefficient matrix* B to be able to build the model

$$Y = XB + F \tag{3.1}$$

where: $Y = Y_{n \times l}$ = response matrix; n samples by l (output) variables
$X = X_{n \times p}$ = predictor matrix; n samples by p (input) variables.
F = residual matrix of Y

This can be formed by first creating models

$$\begin{aligned} Y &= UQ^T + F \\ X &= TP^T + E \end{aligned} \tag{3.2}$$

where: U = score matrix of Y
Q = (orthogonal) loading matrix of Y
F = residual matrix of Y
T = score matrix of X
P = (orthogonal) loading matrix of X
E = residual matrix of X.

The score and loading matrices are developed so that the score $u_i \in U$ has the maximum covariance with $t_i \in T \; \forall i$. Thus the model parameter matrix R is formed so that $U = TR$. Consequently,

$$\begin{aligned} Y &= UQ^T + F \\ Y &= TRQ^T + F \\ Y &= (XP)RQ^T + F \\ Y &= XB + F \end{aligned} \tag{3.3}$$

where: $B = PRQ^T$. [1, 25, 38–40]

If the number of variables p (for example process variables used in X) is large compared to the number of samples n, the probability of X being singular is high, and the regression becomes unfeasible [38]. Therefore, it must often be assumed that the first few principal components capture most of the characteristics of the process. However, this assumption may not always be applicable. [21] In practice, the matrices are determined with iterative algorithms, such as NIPALS or SIMPLS [41].

3.1.3 Fisher discriminant analysis

Fisher discriminant analysis (FDA) is a supervised technique widely used in process industry for dimensionality reduction, data classification and process monitoring. For example, it can be directly used for classification of different operating modes of the process. Other examples include using FDA as a dimensionality reduction step before further classification, and differentiating various abnormal events and faults. The FDA algorithm performs classification by finding a transformation matrix that maximizes scatter between classes and minimizes it within classes. [1]

3.2 Analytical versus empirical approaches

In the analytical approach, rigorous first-principles process models are required to calculate model parameters p, residuals r and state estimations $\hat{x}$,

by means of measured input u and output y. For example, process monitoring and fault detection can be based on comparing estimated parameters $\hat{p}$ or states to the values associated with normal operating conditions (p, *e.g.* historical average value). Predefined thresholds ϵ are used to determine if a parameter difference Δp indicates an abnormal condition $|\Delta p| = |p - \hat{p}| > \epsilon$, as presented in Figure 3.2. [21, 42]

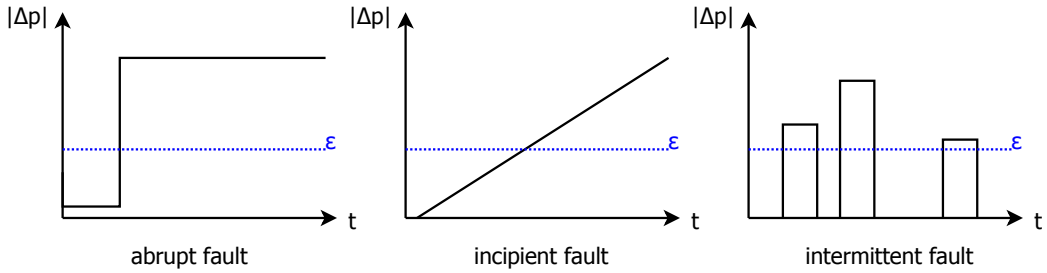


Figure 3.2: Time dependency and threshold of faults [43].

With state estimator (also known as observer) based methods, it is possible to reconstruct unmeasured states and calculate their residuals by estimating observable states of the system. Common methods include the Luenberger observer and the Kalman filter. [44, p. 1063][21]

In contrast to analytical approach, *system identification* provides tools to use empirical mathematical models and parameter estimation to model a process as a black box without the need of complete process knowledge [44]. In process industry, system identification is used for model-based engineering, design, control and optimization, such as model predictive control (MPC) [45]. System identification is closely related to machine learning, since in both, examples or observations are used to infer a function. For example, *applications-oriented optimal experiment design* (AOED) in system identification is related to reinforcement learning (RL) as an *optimal control* problem. [45]

In general, the link between system identification and machine learning is not yet strong [46], and according to Rajeswaran [47], due to its background in statistics and signal processing, system identification is not always utilizing modern computational power. On the other hand, neural networks have been used especially in modeling complex nonlinear processes (*e.g.* in *nonlinear MPC* (NMPC) [48]), and according to Ogunmolu *et al.* [49], deep neural network (DNN) is a suitable replacement for system identification of nonlinear regressive models, although DNN requires selecting hyperparameters, choosing the model structure and tuning weights. However according

to Tsai *et al.* [48], ANN based models of highly nonlinear processes are usually too inaccurate for MPC if the training data is not good and sufficient enough. New input patterns and subsequent unreliable prediction by an ANN can be recognized from notable decrease in the probability density function calculated from the training data [48].

3.3 Knowledge base and AI-based approaches

Knowledge-based approaches are usually built on a knowledge base and an inference engine, and they are designed to perform process monitoring, control and diagnosis. *Knowledge-based systems* (KBS) also aim at solving complex problems, such as prediction, detection, recommendation and automated reasoning, by using uncertain, conflicting and non-quantifiable information. Machine learning and human expertise are often used for creating knowledge-based systems. [21, 50] Knowledge-based solutions can be based on expert systems, fuzzy logic, machine learning and pattern recognition [21].

3.3.1 Expert systems

Expert systems (ES) are knowledge-based software systems that capture domain specific knowledge into a knowledge base, and use that in conjunction with an inferencing (reasoning) procedure to solve decision making problems that usually require human experts [51]. In the control field, expert systems are useful for online operations due to their ability to explain a sequence of reasoning by incorporating symbolic rule-based knowledge that relates a situation to actions [35].

The first expert systems were created already in the 1960s, and they were the first step in the evolution of *knowledge engineering* [15]. Since then, numerous case studies have been published containing information about the challenges and advantages of expert systems [16]. The major challenges include acquiring, structuring and formalizing the usually heavily domain-specific knowledge that the system relies on [7, 11, 12]. Since the beginning of the 21st century, knowledge engineering has focused on big data techniques [15], and expert systems have evolved from a classic model the knowledge base of which is generated by one or more humans, to a hybrid model (see Section 3.3.3: Hybrid expert systems) of a knowledge-based system that utilizes various AI tools and techniques [16], such as fuzzy logic, machine learning and pattern recognition techniques [21].

Nowadays expert systems are often considered obsolete [10], but on the other hand, they are also said to be among the most mature, widespread and

successful branches of AI applications [52] and according to Weathington [10], no clear successor with the same capabilities can be identified for an expert system, at least in its modern shape. *Drools* is one example of software that can be used as a rule engine for an expert system, and its open source project is still alive. Drools is a business rules management system (BRMS) solution developed and productized by Red Hat. Drools comprises a set of components including an inference engine and a knowledge base manager. [53, 54]

3.3.1.1 Architecture

Expert systems usually consist of a *knowledge-base* (KB), an *inference engine* (IE) and a user interface (UI). The knowledge base may contain *shallow* knowledge based on heuristics, and *deep* knowledge based on factual models, *i.e.* mathematical, structural or behavioral models. [35] *Knowledge representation* (KR) schemas are used to organize the knowledge in the knowledge base, and they include rule-based, frame-based, case-based and cognitive map based schemas [16, 51]. Rule-based are generally most common, while cognitive maps have gained increased attention recently. [16] Frames specify attributes and relationships for objects, whereas rules are simply if-then statements [51].

The working memory (or workspace) contains the information about the current status of the case or problem. The inference engine evaluates the rules of the knowledge base, and orchestrates the applying of reasoning methods on the working memory. The reasoning methods include forward and backward chaining, hypothesis testing, heuristic search methods, and meta-rules. In forward chaining, inferring advances from the facts to the goal, whereas in backward chaining the inference engine attempts to match an hypothesized goal with the then-part of a rule. [21, 51, 55, 56] Forward chaining is usually used with open-ended problems, such as in planning, and the backward chaining is used in classification problems where the states of the conclusion are discrete and limited [51].

The *explanation facility* is used to explain, either in natural language or as a sequence of reasoning, how the expert system achieved the inference [51]. A typical system architecture of a rule-based expert system is presented in Figure 3.3.

3.3.1.2 Knowledge acquisition in expert systems

The knowledge base is an important component of an expert system, but acquiring the knowledge from domain experts and building the knowledge base has been the bottleneck of building knowledge-based systems, such as expert

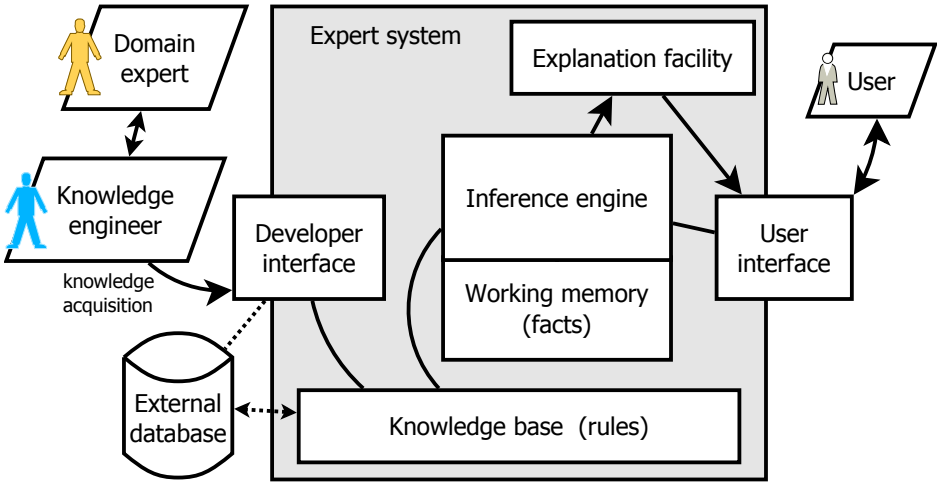


Figure 3.3: A general architecture used in expert systems [21, 51, 55].

systems [12]. Early methods of *knowledge acquisition* (KA) included manual or computerized interview processes. Later on, automated KA methods have been studied, involving ANNs, Bayesian networks and case-based reasoning. [16] In ANNs, symbolic information can be included into the network learning algorithm [21].

3.3.2 Fuzzy logic

Fuzzy logic provides a mechanism to represent uncertain knowledge using graded statements instead of strictly boolean statements. For example, if given data points $x_i \in X$ (*e.g.* temperature) (X is usually called *a universe of discourse* [57]) can be classified into *(sub)sets* $A, B, C, ... \in X$ (*e.g.* hot, warm, cold), a membership function μ for each set maps the points to a value between $[0, 1]$ with respect to the degree of the corresponding membership:

$$\mu_A(x) : X \rightarrow [0, 1] \tag{3.4}$$

where: μ_A = membership function of set A.

Thus, a degree of membership, the value of which is 1, indicates a full membership to the set. Fuzzy logic can also be used to represent qualitative properties with a degree of certainty. [21] A membership function is illustrated in Figure 3.4.

Membership functions and a knowledge base consisting of fuzzy rules, can be heuristically created with expert knowledge [43]. Fuzzy rules are

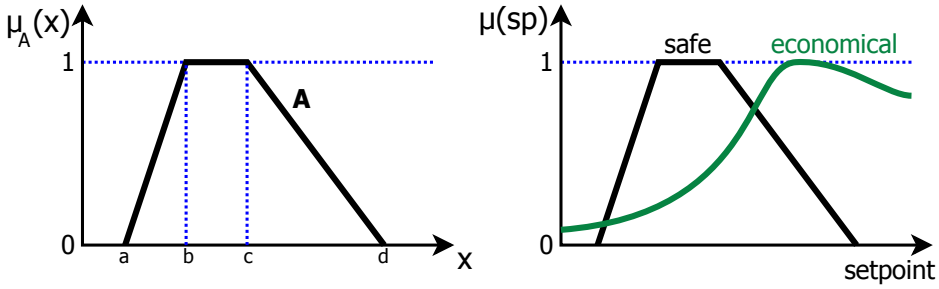


Figure 3.4: Trapezoidal membership function, and an example with fuzzified controller setpoint regions.

expressed as a function of the sets (or linguistic variables, *e.g.* "if x is A and y is B then z is C" [57]). However, determining the correct set of rules and their weighting for a complex system is usually difficult and time consuming, and thus, neural networks are often used for learning the best membership functions. This combination of ANN and fuzzy logic is used *e.g.* in control applications, such as controlling a continuous stirred tank reactor involving a non-linear open-loop unstable process, or controlling a pulp digester with a fuzzy inference system consisting of a connectionist network. [21] *Fuzzy clustering* such as fuzzy c-means (FCM), where data points can belong to multiple clusters, can be used *e.g.* for process monitoring without detailed process knowledge [43].

3.3.3 Hybrid expert systems

Hybrid expert systems are *hybrid intelligent systems* that combine expert systems with multiple intelligent technologies, such as ANNs and fuzzy logic, in order to overcome the challenges of creating expert systems while taking advantage of both the explanation capabilities of an expert system and the data-driven construction of ANNs. Most hybrid expert systems can be categorized into neural network based *connectionist* expert systems (also known as *neural* expert systems), including *neuro-fuzzy* and *rough neural* expert systems. [58]

3.3.3.1 Fuzzy expert systems

Fuzzy expert systems use fuzzy logic, which improves the handling of uncertainty [59]. A general architecture is presented in Figure 3.5. Fuzzification transforms crisp input values to fuzzy values using the membership functions. Clustering (*e.g.* fuzzy c-means) can be used to define the sets and their

membership functions. Fuzzy rule base is the knowledge base, as discussed in section 3.3.2, and the fuzzy inference engine usually follows the *Mamdani fuzzy inferencing* [60], which is focused on interpretability, or the *Takagi-Sugeno-Kang inferencing*, which aims at precise fuzzy modeling focused on accuracy [61]. Finally, in defuzzification the fuzzy value is transformed to a definite value, for example using a weighted average model [60].

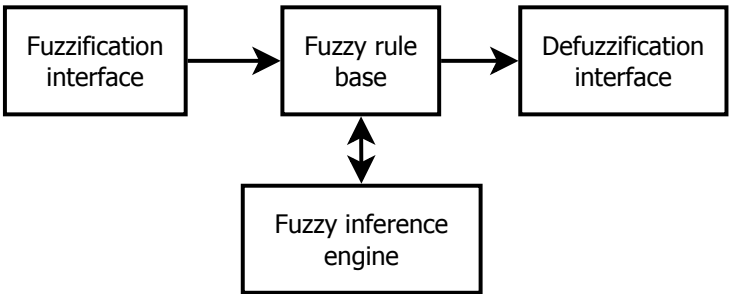


Figure 3.5: General core architecture of a fuzzy expert system [60].

Clustering can be used for automatic rule generation, if an approximate relationship from input to output can be represented by clustered input and output. Since the cluster centroids are used to represent the entire cluster, derived rules become automatically inexact or fuzzy by nature. [62] For example, Determan *et al.* [62] describe a method to form an ith rule as an if-then statement denoted as implication

$$\begin{aligned} &\overbrace{(x_{k1} \in A_{i1}) \wedge (x_{k2} \in A_{i2}) \wedge ... \wedge (x_{kn} \in A_{in})}^{n} \\ &\Rightarrow \underbrace{(y_{i1} \in B_{i1}) \wedge (y_{i2} \in B_{i2}) \wedge ... \wedge (y_{im} \in B_{im})}_{m} \end{aligned} \tag{3.5}$$

where: x_{kj} = n normalized rule input values for the kth datapoint x_k, when $j \in [1, n]$
$A_{i1}, ..., A_{in}$ = fuzzy sets defined by input clusters
$y_{i1}, ..., y_{im}$ = rule output values
$B_{i1}, ..., B_{im}$ = fuzzy sets defined by output clusters

The truth value of the membership $x_{kn} \in A_{in}$ can be defined using exponential (Gaussian) membership functions μ_{ij} derived from cluster parameters:

$$\mu_{ij}(x_{kj}) = \exp\left(-\frac{1}{2}\left(\frac{x_{kj} - x^*_{ij}}{\sigma_{ij}}\right)^2\right), j \in [1, n] \tag{3.6}$$

where: x^*_{ij} = estimated cluster mean
σ_{ij} = estimated standard deviation of the cluster

If a crisp truth value is required, alpha-cut could be used:

$$A := A_\alpha = \{x \in X | \mu(x) \geq \alpha\} \tag{3.7}$$

Otherwise, since the output of the membership function represents the *match strength* of the input x_{kj} with the associated set A_{ij}, the match strengths can be combined into *rule strength* ξ:

$$\xi_i(x_k) = \exp\left(-\frac{1}{2}\sum_{j=1}^{n}\left(\frac{x_{kj} - x^*_{ij}}{\sigma_{ij}}\right)^2\right) \tag{3.8}$$

In fuzzy inference, the defuzzification algorithms cause greater rule match strength to give greater weight to the consequence of the rule. One well-known defuzzification method is the *center of gravity* algorithm that gives the system output y as follows:

$$y = \frac{\sum_{i=1}^{n}\xi_i y^*_i}{\sum_{i=1}^{n}\xi_i} = \sum_{i=1}^{n}\frac{\xi_i}{\sum_{i=1}^{n}\xi_i}y^*_i \tag{3.9}$$

where: n = number of rules
y^*_i = vector of output cluster centroids
$\frac{\xi_i}{\sum_{i=1}^{n}\xi_i}$ = normalized weight of each rule

Thus, the rule set output is the weighted sum of the individual rule outputs, where the weight is the rule strength that is normalized by the sum of all rule strengths. [62]

Genetic algorithms can be used for further improving the process, for example for determining the number of clusters, which affects the number of rules, and for determining cluster parameters such as the amount of cluster overlapping [60, 62].

3.3.3.2 Neural expert systems

Neural expert systems provide automatic knowledge acquisition from a set of examples by employing feedforward trained ANNs, and the inference is improved by an ability to handle more generalized and incomplete cases [58]. System architectures with regard to the relation between ANN and ES, vary from fully-integrated models to loosely-coupled, to stand-alone architectures.

In fully-integrated models, components share the data structures, whereas in loosely-coupled models ANN and ES components are separate, and output of one component is passed to the other. In stand-alone models the components are independent, which makes the comparison of the components possible. [63]

In classic rule-based expert systems, the knowledge is represented as well-specified if-then statements, and consequently, the inference engine cannot handle noisy or incomplete data. On the other hand, since the reasoning is serial algorithmic by nature, results are easily explainable. In neural networks, the knowledge is unstructured and it is stored as synaptic weights between neurons. Thus, the input data does not have to completely match the data used for training, which enables so called *approximate reasoning*. On the other hand, knowledge is embedded into the entire network, acting as a black box model, and any change in the synaptic weights may cause unpredictable results. [59, 63]

Combining the ES and ANN results in a neural expert system that uses a trained neural network as its knowledge base. The network consists of neurons connected by weighted links. The weights w define rules by determining the strength of the associated neuron inputs, as presented in Figure 3.6. The inference engine draws an inference if the sum of weighted known inputs to a neuron is greater than the sum of the absolute value of weights of the unknown inputs:

$$\underbrace{\sum_{i=1}^{n} x_i w_i}_{known} > \underbrace{\sum_{j=1}^{n} |w_j|}_{unknown} \tag{3.10}$$

where: i = known neuron inputs
j = unknown neuron inputs
n = number of neuron inputs [63]

Neuro-fuzzy (expert) systems aim at combining the learning abilities and parallel computation of ANNs with the linguistic knowledge representation and explanation abilities of fuzzy systems. Neuro-fuzzy systems can be trained as an ANN to develop fuzzy if-then rules and define the membership functions for the input and output variables of the system. A neuro-fuzzy system can be divided into five layers (Figure 3.7), the three of which are hidden representing membership functions and fuzzy rules:

- Layer 1, *input*: Neurons transmit external crisp signals to the next layer, *i.e.* $y_i^{(1)} = x_i^{(1)}$. For a neuron n, it is common practice to denote

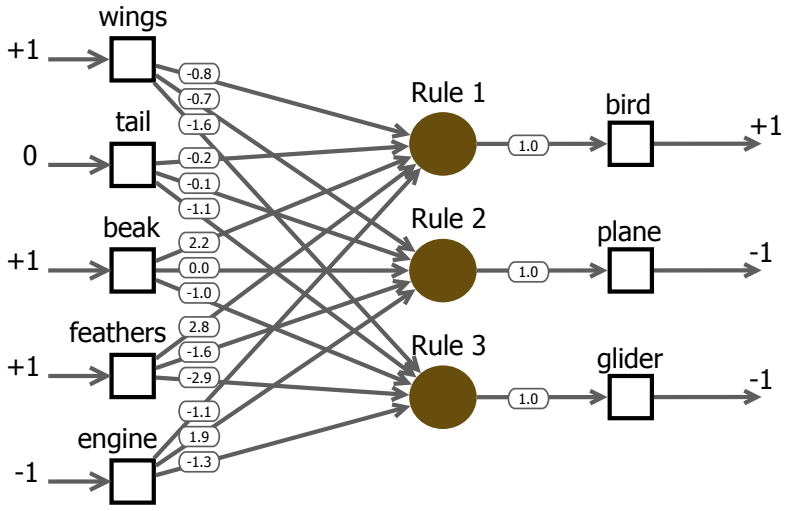


Figure 3.6: An example of a neural knowledge base for classifying flying objects. Input $\in \{-1, 0, 1\}$, where -1=false, 0=unknown and 1=true. [63]

the layer number l as superscript, and the neuron number h on that layer (or input feature number on the input layer) as subscript as follows: $n_h^{[l]}$. Here the input of a neuron is denoted as $x_h^{(l)}$ and output as $y_h^{(l)}$.

- Layer 2, *fuzzification*: Neurons, *e.g.* neuron A1, represent a fuzzy subset that is one of the inputs for the fuzzy rules. The input of the neuron A1 is a crisp value, based on which the neuron determines the degree of membership of the input to the set A1. The activation functions of the neurons are derived from the membership functions.
- Layer 3, *fuzzy rules*: Each neuron represents a single fuzzy rule. For example in Figure 3.7, neuron R1 represents the Rule 1, which is a function of subsets A1 and B1. Rule output is calculated as a product of the inputs.
- Layer 4, *output membership*: Neurons represent fuzzy sets that are outputs of the fuzzy rules. A neuron combines its inputs using probabilistic OR, or fuzzy union.
- Layer 5, *defuzzification*: The layer combines the rule output memberships into a single output membership function and determines the defuzzified crisp or numerical output of the entire system. For example the sum-product method calculates the output as a weighted average of the centroids of all output membership functions. [63] If the third

layer outputs crisp values, layers 4 and 5 are replaced with one output layer [61].

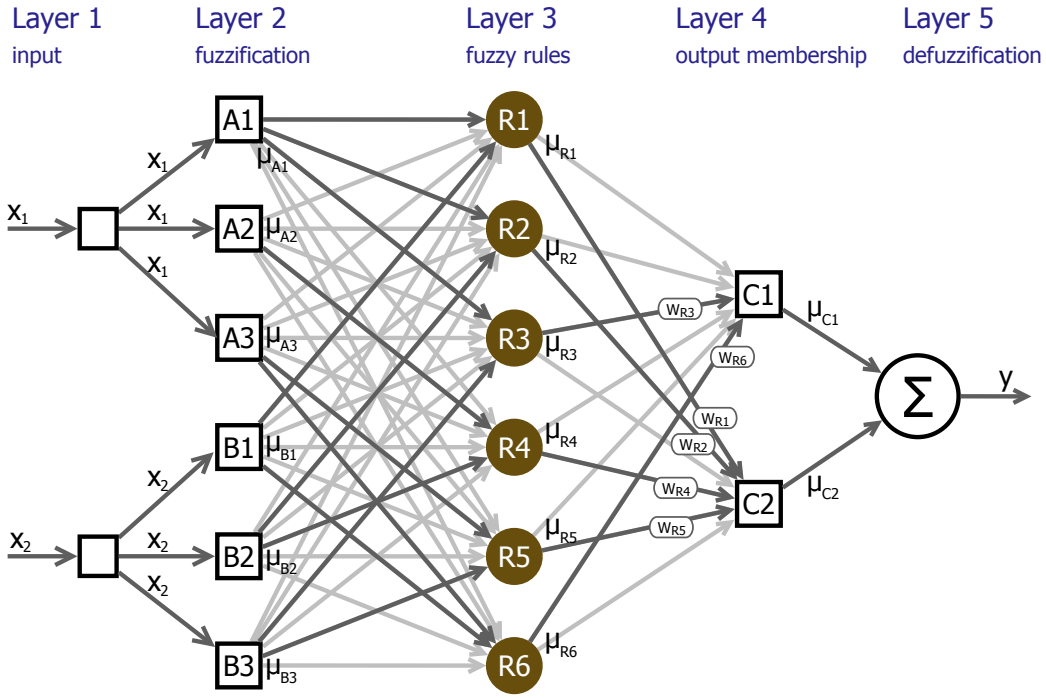


Figure 3.7: Neuro-fuzzy system. Light arrows represent unused subsets in rules. [63]

Back-propagation and other standard ANN learning methods can be used for training neuro-fuzzy systems. Given a good set of examples, fuzzy if-then rules can be created automatically. On the other hand, also human experts can create rules. [63]

Further improvements of neuro-fuzzy systems include the *adaptive neuro-fuzzy inference system* (ANFIS), where the added fourth layer normalizes the firing strengths of the rules, and the fifth layer receives the initial inputs of the first layer (assuming that the ANFIS system is represented as six layers to correspond the diagram in Figure 3.7, instead of a five-layer system, where the numbering starts from the second layer [64]) [65, 66]. ANFIS combines the least-squares and the back-propagation gradient descend methods to find the effective parameters of a Sugeno-type fuzzy inference system [67]. The general form of a Sugeno fuzzy rule is

$$\begin{aligned} &\text{IF } (x_1 \text{ is } A_1) \text{ AND } (x_2 \text{ is } A_2) \text{ AND } ... \text{ AND } (x_m \text{ is } A_m) \\ &\qquad\qquad \text{THEN } y = f(x_1, x_2, ..., x_m) \end{aligned} \tag{3.11}$$

where: x_i = input variables, $i \in [1, m]$
A_j = fuzzy sets, $j \in [1, m]$ [63]

Wavelet transform can be used to improve time series data prediction, and fuzzy c-means clustering can be used to decrease the number of fuzzy rules reducing the problem of curse of dimensionality [68].

ANFIS has been used for process prediction and modeling. For example, Kassem *et al.* [69] successfully predicted viscosity and density of biodiesel-petroleum diesel blends as a function of temperature and volume fraction of biodiesel. ANFIS proved to output more accurate prediction than an ANN model.

Kurnaz *et al.* [64] used ANFIS as three parallel fuzzy controllers to implement an autonomous control of an unmanned aerial vehicle (UAV). The inputs for the controllers were the altitude error, the air speed error and the bank angle error with their derivatives, and the outputs were the throttle position, the elevator position and the bank angle, respectively. The ANFIS controllers were evaluated against requested flight trajectories. Stable control and fast reaction time was achieved, although some external disturbances caused unstable performance.

For online system modeling problems, Ouyang *et al.* [70] describe a method for extracting fuzzy if-then rules by dynamically merging clusters in input and output data. Thus, existing clusters are refined and new ones created as new training data is inputted. The workflow is presented in Figure 3.8.

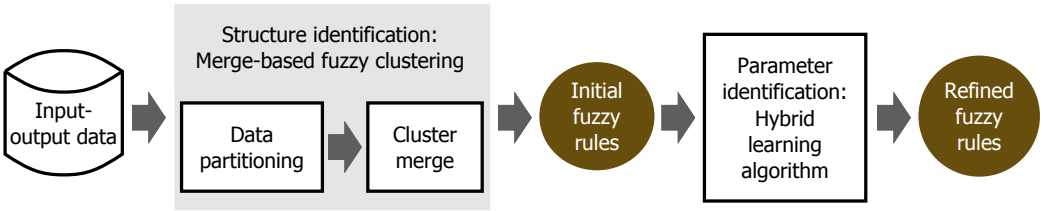


Figure 3.8: Dynamic fuzzy rule extraction from online process data [70].

3.4 Agent based decision support

In artificial intelligence and especially in computer science, *agents* are generally defined as systems or software capable of autonomous reasoning in order to achieve one or multiple goals [71]. Agents usually get input from their environment, and based on that modify that environment, without having a

complete control over it, however. Learning is not a requirement, and further definition greatly depends on the application. [72]

Compared to classic expert systems, agents are more autonomous and act on their environment directly, whereas expert systems give advise to a third party, who then modifies the environment. On the other hand, some real-time expert systems in process control are agents by this definition. [72]

Intelligent agents are able to interact with other agents and humans, perceive changes in their environment and respond quickly enough to achieve the goal, and also proactively recognize opportunities and take initiative actions. *Multi-agent systems* consist of multiple agents, and decentralized data and computation without a global control system. [72] Also a knowledge base common to all agents, may exist [73].

Koumboulis *et al.* [74] proposed a multi-agent based solution for assisting process operator's decision making. The decision making system consists of two kind of agents: supervisory control agents and an operator agent. The supervisory control agent executes *e.g.* controller selection and tuning, whereas the operator agent takes actions usually done by human operators, such as setpoint selection and fault detection and management, leaving only the higher-level decisions to the human operator. The operator agent is implemented according to the DAI-DEPUR expert system architecture, which consist of four levels: data, knowledge, reasoning and supervisory levels. The knowledge level comprises multiple agents, each of which describes the behavior of one subsystem of the process, by using the data level, clustering, simulation and knowledge acquisition. These agents send their information to the supervisory level, which infers the state of the process. The reasoning level is based on a case library that contains information about previously experienced situations and their consequent control actions (or solutions). Heuristic functions are used to retrieve the best matching case, and the proposed solution is evaluated with simulation. For example the optimal setpoints are stored in the case library, and they are selected to be the optimal solution after the following steps: the human operator chooses the high-level goal, the supervisory level infers the process state, and, the reasoning level finds the optimal case and evaluates its actions in the current situation by simulating.

Calderwood *et al.* [73] presented a multi-agent system that concentrates on modeling and handling uncertainty that is caused by *e.g.* noise and error in sensor data, and conflicting and incomplete data from multiple sources or agents. Uncertainty modeling is based on the Dempster-Shafter theory, and the AgentSpeak language was used to model the plans and goals of the system.

Gofuku *et al.* [75] developed a multi-agent based operator support system

prototype for an oil refinery. The goal of the system is to support and guide human operators during an abnormal condition of the plant, and also to serve as a dynamic operation permission system, the idea of which is to check that the operator's actions comply with sequential operation rules and manuals. The prototype was evaluated with two scenarios: the decrease of a crude input flow, and the shutdown of the plant. In addition, the influence of an action, on the future plant behavior, is also predicted using model-based reasoning. The model was created with multi-level flow modeling, which is able to estimate the plant behavior qualitatively only, although quantitative simulation was also considered. The agent-based architecture of the system is presented in Figure 3.9.

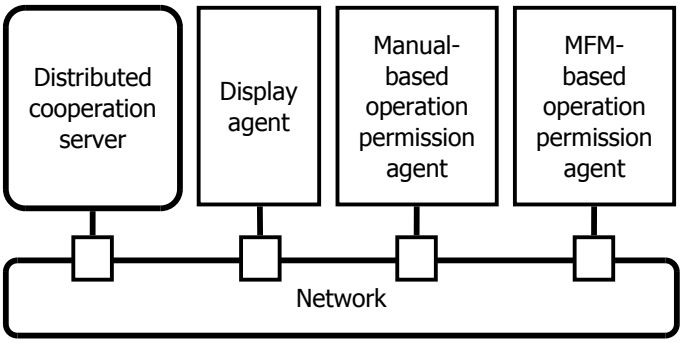


Figure 3.9: The architecture of the prototype of dynamic operation permission system for oil refineries [75].

3.5 Knowledge-driven multi-objective optimization

The theory of multi-objective optimization concentrates on methods to simultaneously optimize multiple conflicting objective functions. These methods can be applied quite universally almost in any decision making, therefore being interesting from the perspective of advisory systems. Knowledge-driven multi-objective optimization combines the multi-objective optimization with knowledge bases.

3.5.1 Multi-objective optimization

In *multi-objective optimization* (MOO), decision making means selecting one solution out of all Pareto optimal solutions [76]. A MOO problem is generally

expressed as

$$\begin{aligned} &\text{minimize} && \mathbf{F}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})\} \\ &\text{subject to} && \mathbf{x} \in S \end{aligned} \tag{3.12}$$

where: $f_i = m\ (\geq 2)$ conflicting objectives that are simultaneously minimized, $f_i : \mathbb{R}^n \to \mathbb{R}$

$\mathbf{x} = [x_1, x_2, ..., x_n]$ = variable vector that belongs to the non-empty feasible region $S \subset \mathbb{R}^n$

$S =$ the feasible region formed by the constraints of the problem and the bounds of the variables. [36]

A variable vector $\mathbf{x}_1$ *dominates* $\mathbf{x}_2$ if

$$\begin{aligned} &1. \quad f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2) \ \ \forall i \in \{1, 2, ..., m\} \\ \text{and} \quad &2. \quad \exists j \in \{1, 2, ..., m\} \text{ such that } f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2) \end{aligned} \tag{3.13}$$

This domination is denoted as $\mathbf{x}_1 \prec \mathbf{x}_2$. A vector $\mathbf{x}^*$ is called *Pareto optimal* if $\nexists \mathbf{x}$ such that $\mathbf{x} \prec \mathbf{x}^*$. The set of all $\mathbf{x}^*$ forms the Pareto optimal set, the projection of which in the objective space, $\mathbf{F}(\mathbf{x}^*)\ \forall \mathbf{x}^*$, is called the *Pareto optimal front*. [36]

The n-dimensional *decision space* formed by the variables, and the m-dimensional *objective space* formed by the objectives, are illustrated in Figure 3.10.

Population-based evolutionary algorithms are the most common methods to solve multi-objective optimization problems. These algorithms can be classified into Pareto-dominance based (*e.g.* MOGA, NSGA, NPGA, PAES, NSGA-II and SPEA2), decomposition-based (*e.g.* MOGLS, MOEA/D and NSGA-III) and indicator-based methods (*e.g.* HypE, IBEA and SMS-EMOA). High-dimensional objective space ($m > 10$) causes the curse of dimensionality problem with Pareto-dominance based algorithms, and thus, the latter two are more suitable in that case. During the optimization, a *MOO dataset* is formed. Each row in the MOO dataset consists of the decision vector components, corresponding objective values and possibly some additional information. Thus, the dataset can be divided into infeasible and feasible sets. [36]

According to Bandaru *et al.* [36], the MOO dataset can be used for extracting knowledge, by using data mining methods such as descriptive statistics (*e.g.* mean, skewness, contingency), visual data mining (*e.g.* clustering, dimensionality reduction) and machine learning (*e.g.* decision trees). With these methods, it is possible to for example reveal the overall structure of the objective space and associate clusters in it to corresponding clusters in the

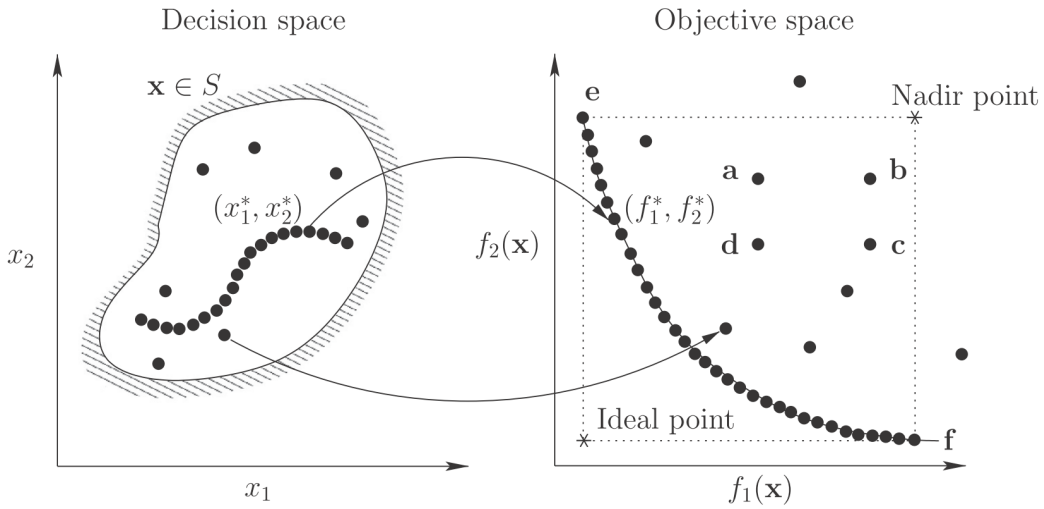


Figure 3.10: The decision space and the objective space (both two-dimensional) in multi-objective optimization [36].

decision space. Some methods, especially data visualization, may require expert knowledge to make understanding the knowledge possible, which usually results in subjective interpretation of the extracted knowledge.

Machine learning can be used for non-visual data mining and knowledge extraction from MOO datasets. In supervised learning, *decision tree* (*classification tree* if the labels are discrete, or *regression tree* if continuous) algorithms, such as ID3 and C4.5, are used to create decision rules, *e.g.* in the form of a statement

$$\text{if } (x_1 < v_1) \wedge (x_2 > v_2) \wedge ... \text{ then (Class Y)} \tag{3.14}$$

These algorithms recursively divide the training dataset into two subgroups, by a feature that maximizes the dissimilarity between the subgroups. Thus, explicit knowledge representation is acquired. [36] For example, an objective function in MOO can be discretized using the decision variables x_i as the division variables [77].

For implicit representation, *e.g.* self-organizing maps are used for visualization, and support vector machines (SVM) and ANNs can be used for black-box modeling [36]. For example, SVMs are used for predicting the ranks of new MOO solutions [78].

3.5.2 Knowledge-driven optimization

According to Bandaru *et al.* [36], interest in the application of multi-objective optimization to real-world problems is increasing, and consequently, knowledge-driven optimization may become an important topic in the near future. In *knowledge-driven optimization* (KDO) the user or decision maker modifies the optimization problem or algorithm by using the knowledge gained in a post-optimal analysis. KDO can be performed online or offline, as illustrated in Figure 3.11. In offline KDO, data mining and the optimization algorithm are separate, and thus, any MOO algorithm can be used unchanged.

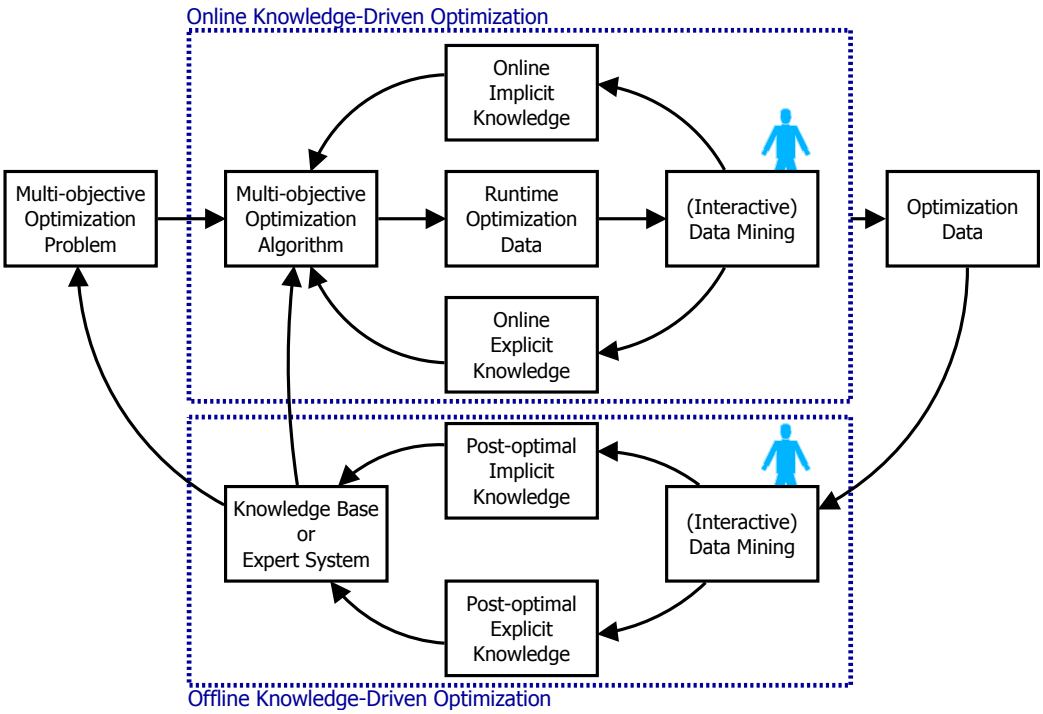


Figure 3.11: A framework for knowledge-driven optimization [36].

For example in offline KDO, regression analysis can be carried out on a set of Pareto optimal solutions, to obtain analytical relationships between the variables. When these relationships are used as constraints on the original MOO problem and a local search is performed on the previously obtained solutions, a better approximation of the Pareto front can be generated. As another example, post-optimal knowledge can be used for parameter tuning of the optimization algorithm, and expert systems can be used for automatic MOO problem and algorithm modification if a knowledge base is built, as presented in Figure 3.12. [36]

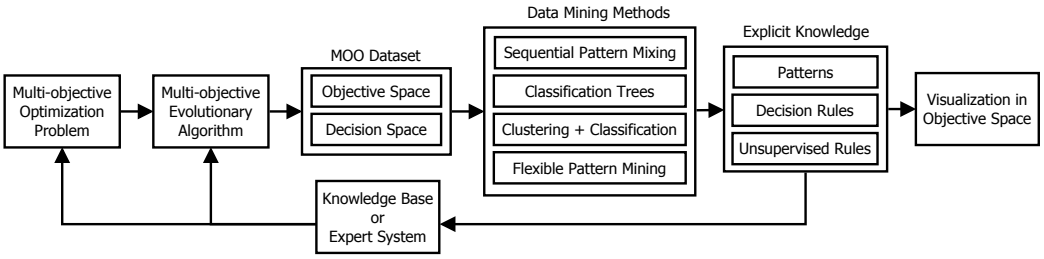


Figure 3.12: A method to control the optimization algorithm and to modify the original MOO problem, by using extracted knowledge in an expert system [79].

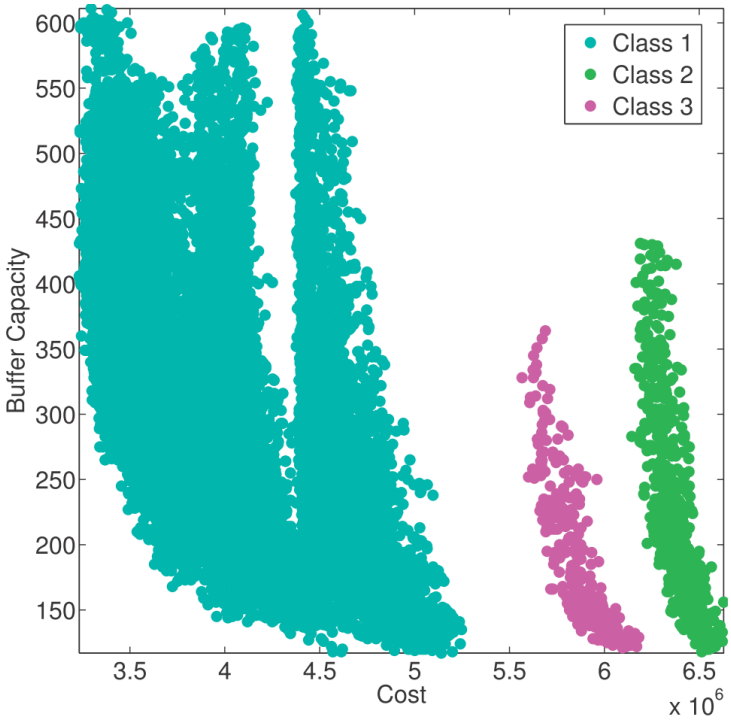
Online KDO aims at extracting knowledge during optimization, which could be used to obtain faster convergence of the Pareto optimal front. For example, dimensionality reduction can be used for identifying redundant objectives in the objective space. On the other hand, online KDO can also be *data stream mining*: For example, if a reference point is provided by the decision maker, the input data stream can be filtered to obtain only solutions close to the reference point. [36]

In a case study by Bandaru *et al.* [79], a MOO problem with three objective functions (including cost and buffer capacity) was solved using NSGA-II. The results for the Pareto optimal solutions for cost C and a buffer capacity B is presented in Figure 3.13. Then DBSCAN clustering was performed in the objective space, which resulted in three clusters, Classes 1–3. Based on this, it was possible to create rules in the decision space for these clusters, as shown in Figure 3.13.

3.6 Conclusions

The idea of creating digital advisory systems for process operators, especially for control room use, has been under research for decades. However, most of the high-level methods used lead to either the research of artificial intelligence or generic methods used for example in fault detection. The research of artificial intelligence has been quite periodical, and as a consequence, currently the majority of the information that can be found in literature applicable to process control related advisors, is at least fifteen years old. More recent articles about the topic can be found, but in a closer examination, they do not offer much improvement or other new relevant information.

Bonnin [24] presented the development of artificial intelligence in four



Class	SEL	UNSEL	Rule
1	78.84	0.00	$B_2 \geq 1.5 \ \wedge \ B_8 \geq 1.5 \ \wedge \ B_{16} < 23.5 \ \wedge \ldots$
2	95.93	0.00	$B_2 < 1.5 \ \wedge \ B_{15} < 18.5 \ \wedge \ B_{16} < 16.0 \ \wedge \ldots$
3	90.06	0.00	$B_2 \geq 1.5 \ \wedge \ B_8 < 1.5 \ \wedge \ B_{27} < 31.5 \ \wedge \ldots$

Figure 3.13: Clustered objective space and the obtained rules for the decision space variables (*e.g.* a buffer capacity B_j) [79].

steps, shown in Figure 3.14. According to this, the development has shifted from rule-based algorithms to machine learning, but on the other hand, in artificial intelligence, all the other three algorithm types will be used to enable solving tasks that had never been considered during the development of the system.

While machine learning and especially neural networks show significant potential in inference and decision making, in many contexts the inability to explain the sequence of reasoning and the need of a substantially large dataset for training, are mentioned as the biggest challenges. Different solutions to these shortcomings have been proposed to this day, such as in a recent rule-embedded neural network (ReNN) approach where long-term dependencies are modeled with the help of rules from domain experts. [80] However, no standard solutions can be found in literature, but most of the

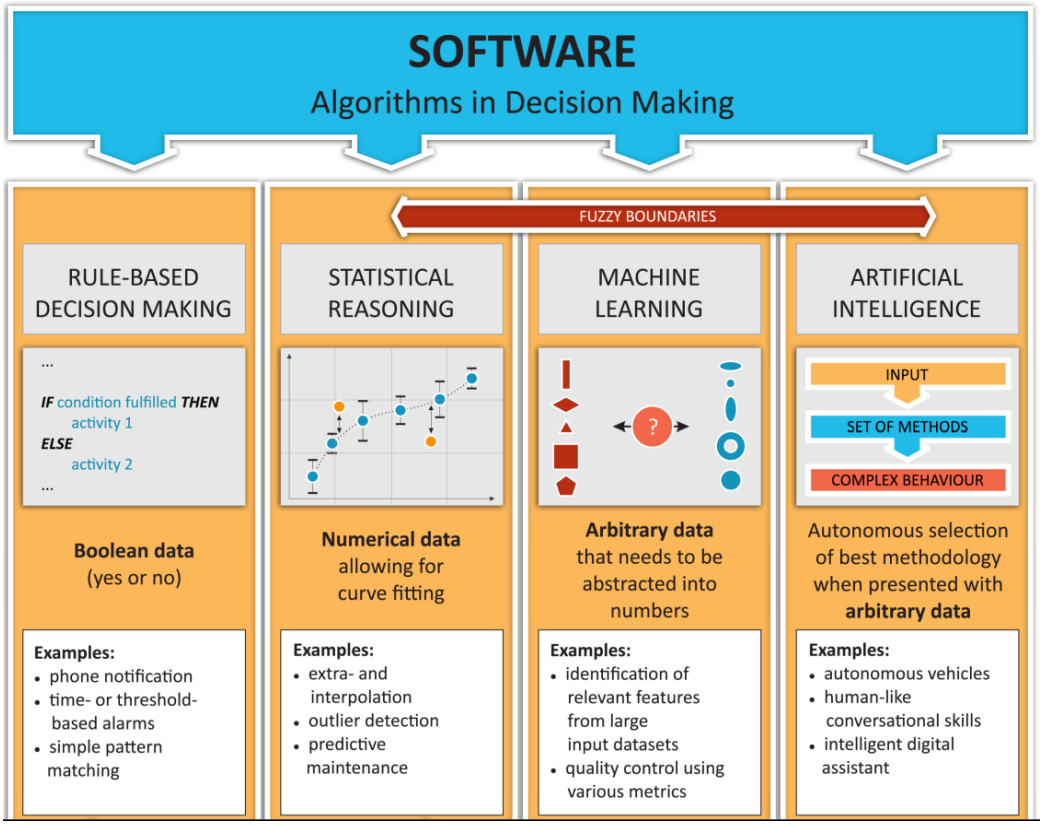


Figure 3.14: The four stages of development towards true or strong artificial intelligence [24, p. 9].

proposed methods are very experimental, and often only tested in one specific case.

Artificial intelligence products in the consumer-oriented market are often called *intelligent digital assistants*, including *e.g.* Google Assistant and Apple Siri. In most cases these intelligent assistants are based on natural language processing, image recognition and recommender systems, all of which mostly rely on machine learning algorithms. Similar systems have been developed also for industrial use, such as the Sophos-MS framework, proposed by Longo *et al.* [81]. The idea of Sophos-MS is to provide the operator with augmented reality (AR) contents such as interactive operating manuals and tutoring systems. Speech recognition and 3D models of equipment are used with a dedicated headset, as shown on the right side of Figure 3.15. [81] However the concept of augmented reality for process operators is not new, and in the end, it only provides another user interface. Any software that provides the

artificial intelligence functionality is application-specific, and not covered by the Sophos-MS framework.

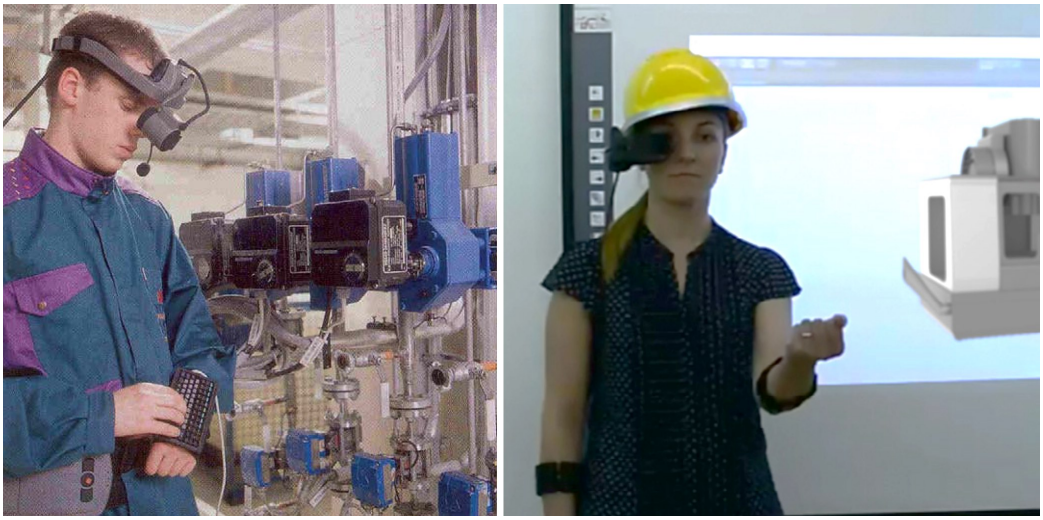


Figure 3.15: Implementations of augmented reality for a process operator, by Valmet Automation Oy in the 1990s [82] (left), and by the Sophos-MS framework [81] in 2017 (right).

Chapter 4

Artificial neural networks in process modeling and control

Since machine learning, and especially neural networks, have been concluded to be important tools in artificial intelligence, and on the other hand, the current best methods of advanced process control (APC) get quite close to the definition of artificial intelligence, it is worth looking into the state of neural networks from the perspective of advanced process control.

In this chapter, the use of artificial neural networks in process modeling and model predictive control (MPC) is briefly reviewed. The scope is not exhaustive, but mainly topics that have figured in relevant case studies and articles, are covered.

4.1 Process modeling

In theory, artificial neural networks are able to model any continuous non-linear relation between variables with arbitrary accuracy. In practice, while neural modeling does not need any analytical knowledge of the process, at least its dynamics must be known to be able to select a correct ANN model, and also a representative dataset is required for training. If one model cannot approximate a dynamic system in all operating points, a Takagi-Sugeno-Kang fuzzy model (equation 3.11 on page 27) can be used to connect multiple partial models to different operating ranges. However, defining the borders of these ranges is not an easy task, and usually expert knowledge, clustering, group method of data handling (GMDH) or some automatic or genetic optimization is required. [83]

Compared to polynomial models, neural models can be more precise and usually have better interpolation and extrapolation properties. Compared to

fuzzy models, neural models tend to be more precise with less parameters. [84]

When modeling a process with a neural network, time series data may require using a dynamic network that contains memory. These ANNs can be classified into short-term and long-term memory networks, depending on the retention time. Short-term memory usually refers to memory that still represents the near-current state of the process, whereas long-term memory may contain even permanently stored data or slowly changing weights. [83]

4.1.1 Multilayer perceptron and deep neural network

Multilayer perceptron (MLP) network is one of the most common artificial neural network used for static process modeling [83, 85]. MLP is a feedforward network, meaning that calculations flow from input to output without cyclic connections (or feedback loops) in the neurons. Backpropagation is used in training to adjust the weights, and usually a nonlinear activation function (or transfer function), such as a sigmoid function, is used. [24]

A classic multilayer perceptron network usually comprises one hidden layer, whereas a single-layer perceptron contains only the input and output layers. Thus, MLP can be seen as a special case of a deep neural network (DNN). While there is no universally accepted definition for the *multilayer perceptron*, in this thesis, *deep neural network* and *deep learning* refer to neural networks with more than one hidden layer. In general, MLP and DNN are suitable for both regression and classification tasks. In regression, a linear output function is used to get a real output value. In classification, *e.g.* sigmoid output function is used. [24]

For MLP networks, usually multiple parallel MISO (multi-input single-output) models are used, and they are trained independently. Compared to one MIMO (multi-output) model, MISO models may perform better if the dynamics are different in different parts of the process. [84, p. 32]

Deep neural networks started becoming more popular after finding network structures and training algorithms with satisfactory performance-complexity ratio. Some milestones and developed models include the *deep belief network* in 2006, *deep Boltzmann machine* in 2009, *denoising auto encoder* in 2010, *deep convolutional neural network* in 2012 and *attention-based LSTM* in 2016. Nowadays, new model architectures are developed and proposed on a weekly basis. [17]

Training the ANN usually means minimizing a nonlinear multimodal cost function using a gradient-based algorithm, which easily causes the training to get stuck in local minima resulting in unsatisfactory results. To mitigate this issue, some global optimization methods have been developed, such as *adaptive random search* (ARS) and *simultaneous perturbation stochastic*

approximation (SPSA). [83]

In addition, there are algorithms for finding the optimal structure of ANN to optimize the complexity and reduce the overfitting phenomenon. These include the pruning algorithms *optimal brain damage* (OBD) and *optimal brain surgeon* (OBS). [83, 86]

4.1.2 Tapped delay lines and recurrent networks

There are two main methods to add dynamic capabilities into an ANN: tapped delay lines and feedback loops. *Tapped delay lines* (TDL) use external memory to include previous time steps to the input of the neural network. If known, the order of the process can be used as the number of previous time steps:

$$y_m(k+1) = f(y(k), ..., y(k-m), u(k), ..., u(k-m)) \tag{4.1}$$

where: f = nonlinear mapping or ANN
$u(k)$ = input of the model at time step k
$y_m(k)$ = output of the model
$y(k)$ = output of the process
m = order of the process

In practice, there are limitations including possible problems with some dynamic phenomena, *e.g.* hysteresis, and also using a high value for the m results in an unfeasible number of inputs to the ANN. [83] Consequently, a suitable value for m may have to be selected by trial and error. Also model stability issues may occur, especially if the model output instead of process output, is used for the delayed inputs [83], as follows:

$$y_m(k+1) = f(y_m(k), ..., y_m(k-m), u(k), ..., u(k-m)) \tag{4.2}$$

The other method, *feedback loops*, are used in *recurrent neural networks* which can be classified into *globally*, *partially* and *locally* recurrent networks. In the globally recurrent, feedback is allowed between neurons of the same or different layers, whereas in the locally recurrent, feedback takes place only inside the neuron model. Partially recurrent networks, such as Elman neural network, may provide the optimal compromise performancewise. In the Elman network, one of the two hidden layers (context layer) is used to memorize the previous activations of the other hidden layer, as shown in Figure 4.1. [83]

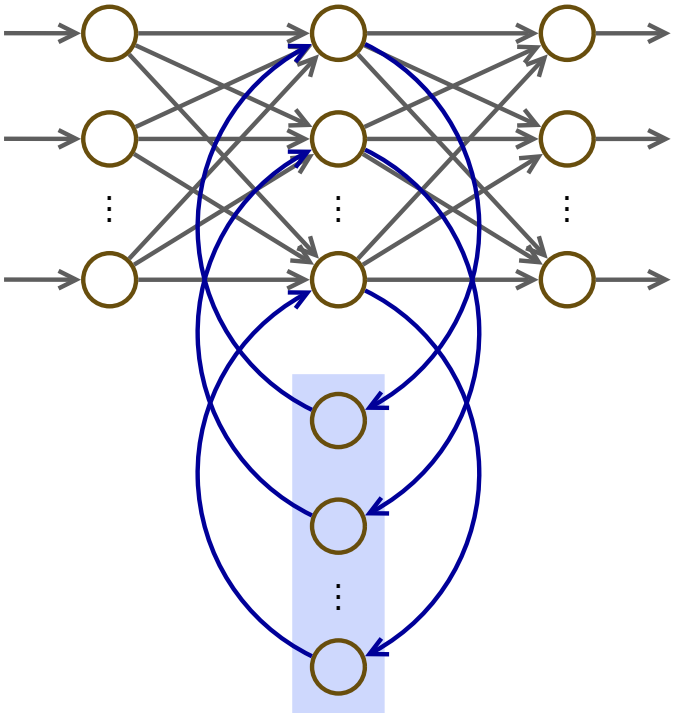


Figure 4.1: The topology of the Elman network.

4.2 ANN-based model predictive control

Model predictive control (MPC) is one of the advanced process control methods. MPC can be categorized into linear and nonlinear MPC according to the linearity of the model that is used for process prediction, and their combination:

- linear MPC algorithms with quadratic optimization
- nonlinear MPC algorithms with nonlinear optimization
- suboptimal MPC algorithms with successive online linearization and quadratic optimization [84]

Linear models include polynomial models *e.g.* ARX and ARMAX, whereas nonlinear models can be created with analytical models, artificial neural networks and fuzzy models [86]. Nonlinear models enable more precise predicting since the majority of technological processes are nonlinear by nature. However, nonlinear models with nonlinear constraints result in a nonlinear cost function in optimization, requiring nonlinear MPC algorithms. In addition, there is no method to guarantee that an optimal solution from the

optimization is the global optimum, which poses a risk in online model predictive control. [84].

Since the process model gives the process output as a function of the manipulated variables, and the opposite is pursued to solve the control problem, either optimization or the inverse model is required [84]. The general idea of the MPC is shown in Figure 4.2, where the lower graph is the result of the optimization, which is recalculated for the entire prediction horizon at each time step k.

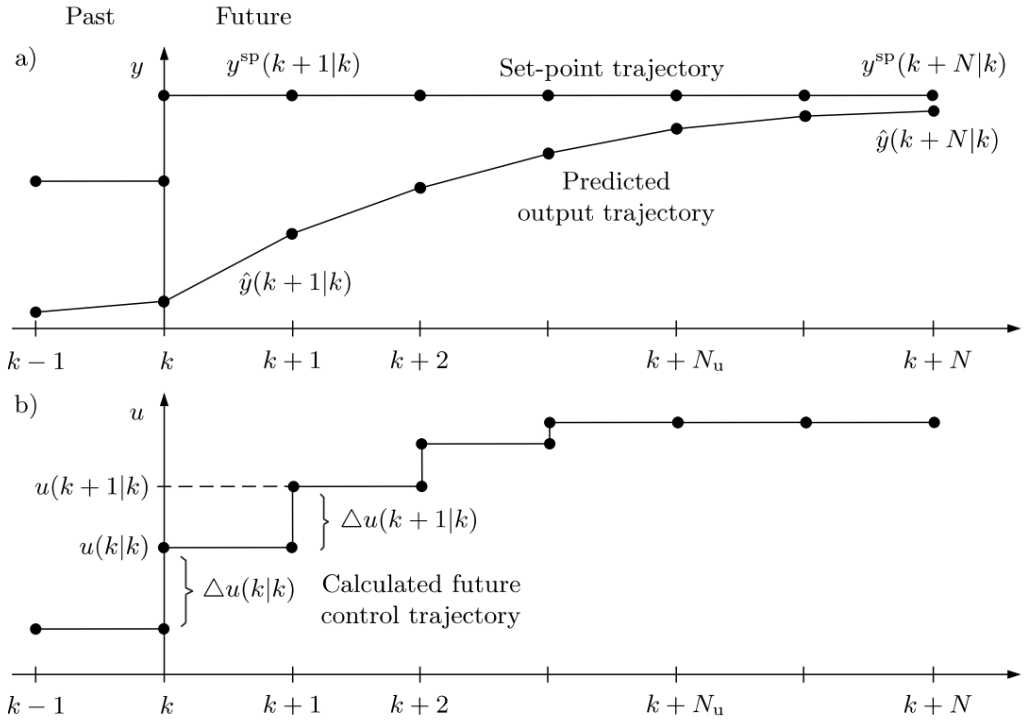


Figure 4.2: The basic idea of model predictive control with *a)* the setpoint and the predicted output trajectories and *b)* the calculated future control trajectory [84].

4.2.1 Predicted output trajectory

Since the model used for prediction in MPC is never perfect due to simplifications and unknown disturbances, a disturbance term is usually added to the prediction:

$$\hat{y}_m(k+p|k) = y_m(k+p|k) + d_m(k) \tag{4.3}$$

where: $\hat{y}_m(k+p|k)$ = predicted value (for the time step $k+p$) of a process output, calculated at the time step k
$y_m(k)$ = output of the model
$d_m(k)$ = estimation of the unmeasured disturbance, which is assumed to be constant for the entire prediction horizon.

One way to calculate the $d_m(k)$ is to use the difference between the real measured process output and the model output:

$$d_m(k) = y(k) - y_m(k|k-1) \tag{4.4}$$

where: $y_m(k|k-1)$ = model output for the time step k, calculated at the time step $k-1$. [84]

4.2.2 ANN based NMPC

A neural model can be used directly for process prediction if nonlinear optimization such as sequential quadratic programming (SQP) or interior point optimizer (IPOPT) is used, as shown in Figure 4.3. From the model accuracy point of view, this is the ideal way of using the ANN. However, the nonlinear optimization problem may be non-convex and computationally too demanding compared to the sampling time in fast processes. [84]

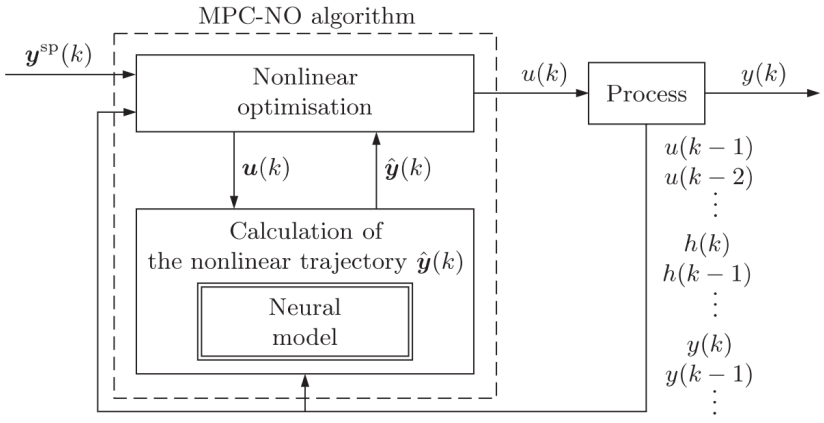


Figure 4.3: MPC with nonlinear optimization [84].

Another *suboptimal* but more computationally efficient option is to use online linearization so that the prediction trajectory becomes a linear function of the calculated future control sequence. These methods include the

MPC-NPL (nonlinear prediction and linearization for the current operating point) shown in Figure 4.4. [84] According to Lawryńczuk [84], many more

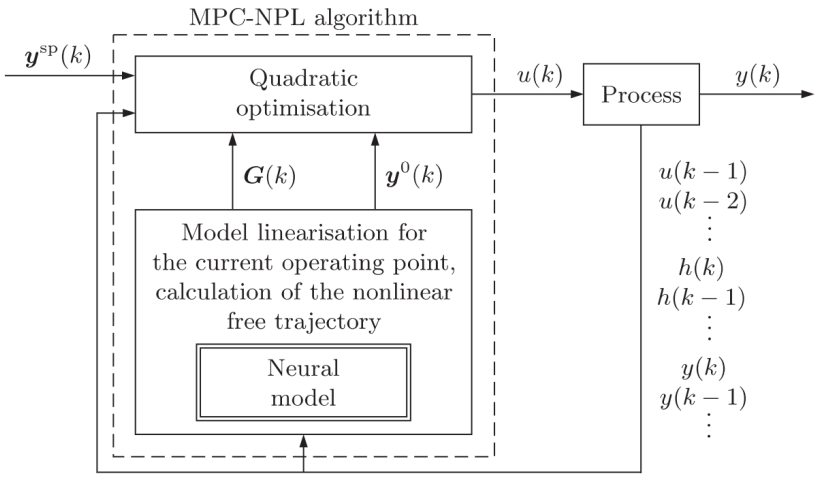


Figure 4.4: MPC with nonlinear prediction and linearization for the current operating point [84].

advanced algorithms have been proposed, for example MPC-SL, MPC-NPLT, MPC-NPLPT and MPC-NNPAPT, which should be used if required for an acceptable control performance, usually in strongly nonlinear processes such as distillation.

For setpoint optimization, MPC can be used in multi-layer control on top of PID controllers, as shown in Figure 4.5. The fourth layer in the system, local steady-state optimization (LSSO) calculates the optimal setpoints for the MPC layer, usually based on *e.g.* economics constrained by safety and product quality. Instead of a (nonlinear) steady-state model, a dynamic model is rarely used. The topmost management layer determines the operating conditions for the LSSO layer, *i.e.* parameters and constraints for the cost function of LSSO. [84]

4.2.3 Inverse ANN based NMPC

In *direct inverse control*, inversion of the process model is used as the controller. If the forward model of a process f is unknown, but enough is known about its properties, such as nonlinearity and time delay, a suitable structure for an artificial neural network can be deduced. After that an ANN can be trained to approximate the inverse function f^{-1}. [86, p. 47]

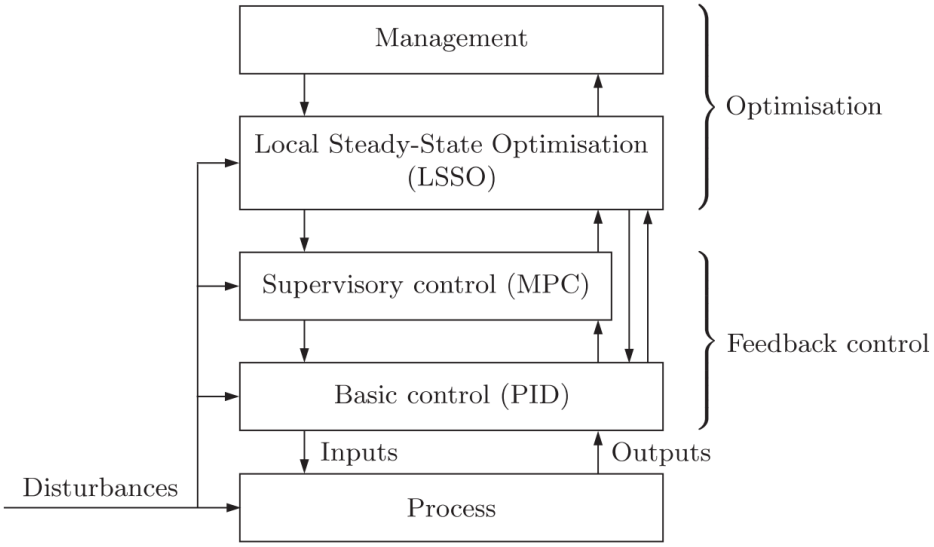


Figure 4.5: The structure of the classical multi-layer control system [84].

There are at least two strategies for obtaining the the inverse model: In *generalized* training, the inverse model is obtained offline even without any process model. The inputs of the neural network include the (delayed) process output and the setpoints. In *specialized* training, an online algorithm is used, related to *model-reference adaptive control*. A forward model is trained before training the inverse model, and therefore, specialized training is more complex to implement in practice, requiring more design parameters. On the other hand, specialized training makes it possible to create a controller that is able to make the process output closely follow the reference. [86]

A drawback of the inverse ANN based NMPC comes from the general non-invertibility of neural networks. When followed backwards, neural network mappings derived from time series data may lead to multiple possible trajectories.

Chapter 5

Case studies for machine learning and operator's advisory systems

In this chapter, five case studies of systems that enable or aim at implementing advisory functionalities for a process operator, are reviewed. Because of the lack of example cases in the field of petrochemical industry, other industries and processes are accepted as well. Overall conclusions are given in Chapter 10.

5.1 DiaSter system

DiaSter is a software package for process modeling, diagnostics, fault detection, supervisory control and decision support for industrial processes [87], such as chemical industry. The system is able to detect improper states of a process, and give advice and operating instructions to the operator. [83]

DiaSter is developed as a university project by a research team from the Warsaw University of Technology, Silesian University of Technology, Rzeszów University of Technology and University of Zielona Gora, and it is the successor of an older system AMandD developed in Warsaw University of Technology [87]. The position of DiaSter in the automation pyramid is presented in Figure 5.1.

According to Syfert *et al.* [87] (in 2011), DiaSter is a unique solution, and at that time ready to be prepared also for commercial use. Most of the information available of the system is from years 2011–2014, and from universities that took part in developing the system.

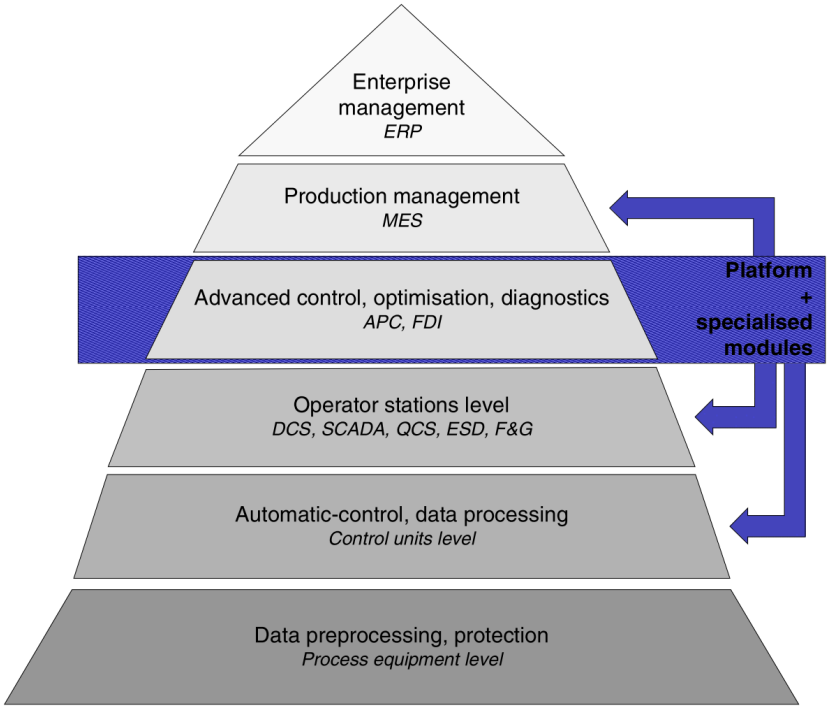


Figure 5.1: DiaSter in the automation pyramid [87].

5.1.1 System architecture

DiaSter is a software platform that enables a set of programs, specialization packages and plug-ins (.dll) together to implement *e.g.* simulation, modeling, soft sensors and fault diagnosis. The main components of the DiaSter system are presented in Figure 5.2. A common information model defines the data exchanged between the components, a central configuration environment handles configuration data for all components, and a central archival database stores values of built-in or custom data types. Distributed calculation over a computer cluster is possible. [87]

5.1.2 Process modeling and control

For identification, a modeling module (MITforRD) allows creating static and dynamic models, *e.g.* linear models, neural networks or fuzzy logic based models. For neural models, two model types are available: locally recurrent networks, and ANN of the GMDH type. [83] The workflow of the identification can be seen in the GUI presented in Figure 5.3.

Syfert *et al.* [83] present an example for neural network modeling for the

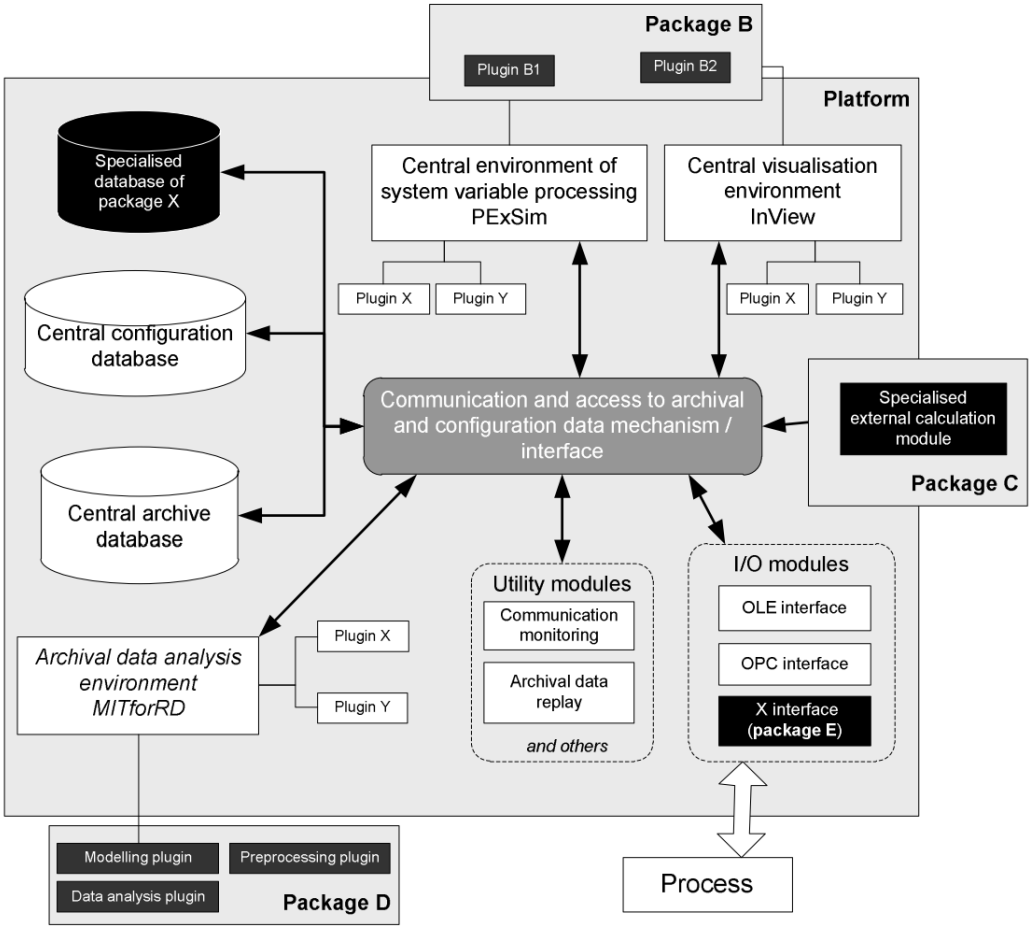


Figure 5.2: The architecture of DiaSter. Black blocks are specialization packages. [87]

simple three-tank benchmark process, using a locally recurrent network (and also the GMDH ANN). The *three-tank benchmark process*, or *the three-tank system*, is a laboratory-scale system that consists of three tanks in series, the bottom parts of which are connected with a pipe. From Tank 3, water pours out freely due to gravity, to a storage reservoir. From the storage, water is pumped through a control valve to Tank 1. From there, water flows freely via Tank 2 to Tank 3, approximately according to Torricelli's law. The structure of the locally recurrent network was selected by trial and error, and a model with one hidden layer with seven neurons was chosen. A first-order infinite impulse response filter, and a hyperbolic tangent activation function was used. The three inputs for the ANN were the control valve position x_v, pressure before the control valve p, and the flow on the inlet of Tank 1, f_1,

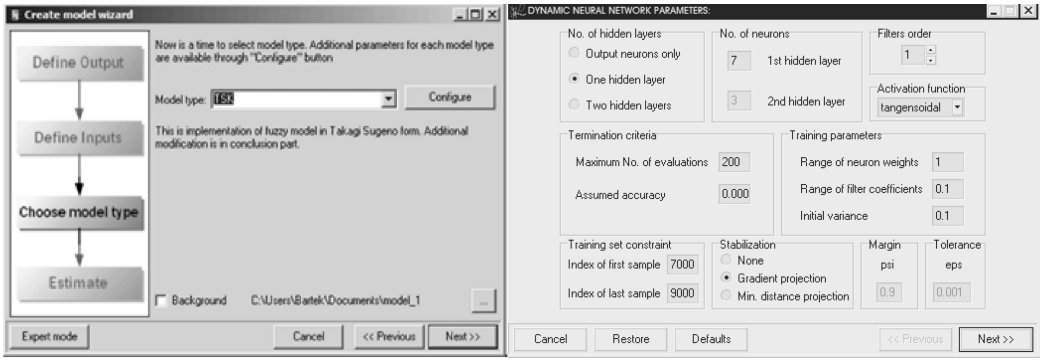


Figure 5.3: GUI for system identification in DiaSter [83].

and the input data was scaled to the range between $[0, 1]$. The output was the liquid level in Tank 3, l_3, thus resulting in a model

$$l_3 = f_{NN}(x_v, f_1, p) \tag{5.1}$$

where: f_{NN} = nonlinear mapping by the locally recurrent neural network

The ANN was trained using the adaptive random search (ARS) algorithm with initial variance $\sigma_0 = 0.1$. The maximum number of iterations or epochs was 200, and the training set included 2000 samples. The stability of the model was guaranteed by the gradient projection method. The testing data comprised 13000 samples, and the results were mostly good with SSE = 11.67 and MSE = $8.9 \cdot 10^{-4}$. In addition, the model was tested with data from some fault cases, such as decreased water pump efficiency. As expected, it was possible to detect a fault from the model output residual, if the source of the fault had some relation with the ANN input variables.

For nonlinear model predictive control, at least MPC-SL (suboptimal MPC algorithm with successive linearization) and MPC-NPL (suboptimal MPC algorithm with nonlinear prediction and linearization) algorithms are available [84, p. 97].

5.1.3 Visualization

The data generated by DiaSter can be sent to external systems, visualized there, or presented directly to the operator with DiaSter. A visualization module is used for graphical user interfaces (GUI), the displays of which are created as plug-ins. [83] An example of a GUI for an operator is presented in Figure 5.4.

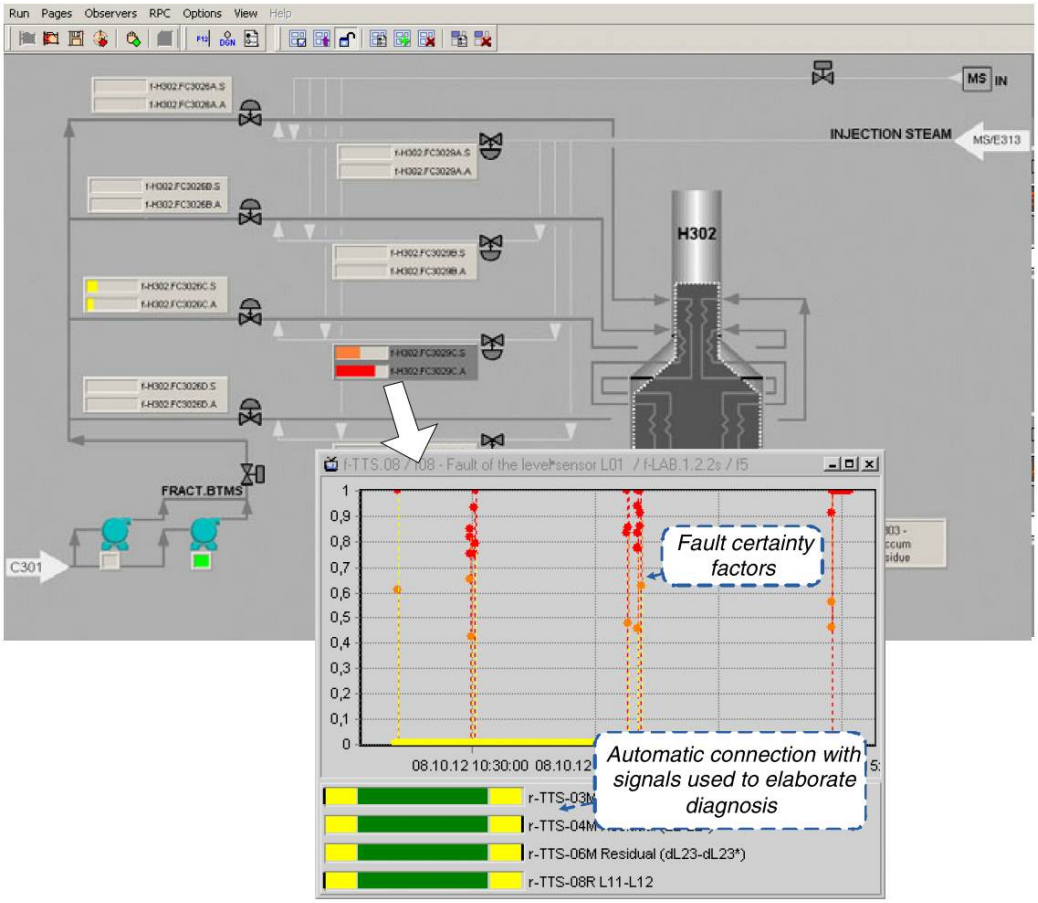


Figure 5.4: An example of a process operator's interface in DiaSter, displaying diagnostics of the process [87].

5.1.4 Conclusion

In conclusion, DiaSter is a versatile and easy to extend platform, somewhat similar to Matlab with Simulink, however with focus on supervisory process control and modeling. According to Syfert *et al.* [87], the system is in educational use, and commercial applications are planned, but no further information is available.

5.2 Probabilistic advisory system based on Bayesian probability

Puchr and Herout [19, 88] describe the implementation of a probabilistic advisory system based on multiple earlier projects, such as the European Union funded ProDaCTool (Decision support tool for complex industrial processes based on probabilistic data clustering) and ProDisMon (Probabilistic distributed industrial system monitor). The goal of the system is to help the operator to optimally set a set of global parameters of a complex industrial process.

The process is considered as stochastic, and the input and output of the system controlling the process are considered as random variables. It is assumed that these variables vary around their mean value during a given state of the process, forming clusters of points in a region of the multi-dimensional data space. A probability density function (PDF) is used to describe the location of a point. A short history of a mixture of these probability density functions is used to describe the state or behavior of the process. Uncertainty and updating of the PDFs is handled with Bayesian statistics. Bayesian probability is used to form the mathematical model for the PDF:

$$f(\Theta|y(T)) = \frac{f(y(T)|\Theta)f(\Theta)}{f(y(T))} \tag{5.2}$$

where:
- f = probability density function (PDF)
- Θ = parameter describing the state or behavior of the process, $\Theta \in \langle 0, 1\rangle$
- $y(T)$ = process data vector as a function of index T
- $f(\Theta) = f(\Theta|y(0))$ = prior probability density function of the parameter Θ at the start of estimation
- $f(\Theta|y(T))$ = posterior probability density function expressing the current knowledge of the value for Θ, at the moment of T
- $f(y(T))$ = normalization constant that can be omitted if the posterior probability function is normalized otherwise. [19, 88]

In a simplified workflow of the advisory system, first a criterion Θ that characterizes the requested state of the process, is chosen. From the history data, a time period corresponding to that state, is selected, and the data from that period is presented as probability density functions. Next, corresponding probability density functions are acquired in the current state of the

process, and they are compared with the historical ones. Recommendations are generated to the operator, how to change the state of the process to the requested state. When the probability density functions match, the process is in the requested state. [19, 88] The overall structure of the advisory system is presented in Figure 5.5.

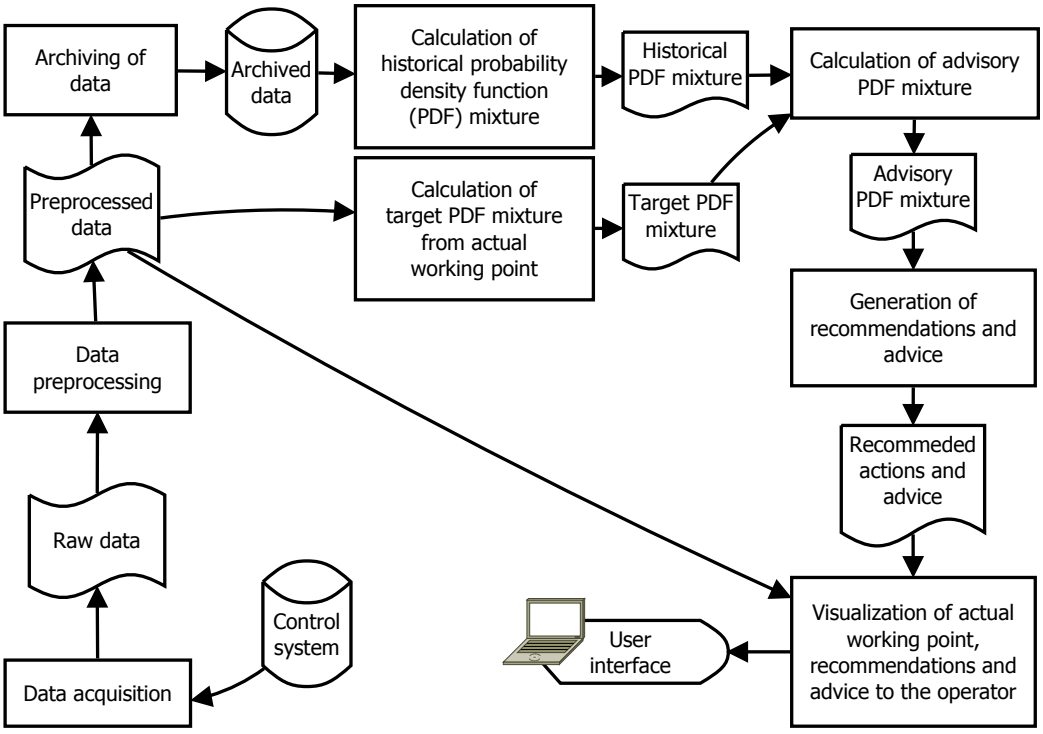


Figure 5.5: Logical structure of the probabilistic advisory system. The probability density function is abbreviated to *PDF*. [19]

Puchr [19] states that the intended form of the advice to the operator is sentences, *e.g.* "increase parameter X to value Y", but also mentions that the shortest path from a process state to the requested state, is not always feasible. In addition, the order of actions taken may affect the transition. No ready solution was proposed yet.

A pilot of the system was tested for a small rolling mill. The approach proved to be well applicable, although data processing (Bayesian probability calculation) caused some performance issues. Need for better generation of advices and visualization was identified, since the test showed that these were the key parts of the advisory system. Any obscure information easily confuses the operator who then loses the confidence in the advisory system. [19]

5.3 Operator support systems in nuclear power plants

Human error has been identified as one of the major causes of accidents in safety critical processes, especially in nuclear power plants. Therefore, many different kind of *operator support systems* (OSS) have been developed to support the process operators' cognitive processes, such as perception and situation assessment in nuclear power plants. These systems can be classified to passive and active: Passive systems aim to represent all the available information to the operator in an optimal way, improving operator's abilities. Active systems include intelligent advisors and diagnostic systems that are developed using *e.g.* knowledge bases, neural networks and genetic algorithms. [89]

5.3.1 An integrated operator support system

Lee *et al.* [89] proposed an advisory system that integrates multiple separate support systems into one system, as shown in Figure 5.6. Implementations for a fault diagnosis system and an operation validation system were presented. The operation validation system works in a similar manner than the system by Gofuku *et al.* [75] described on page 29, except that also quantitative predictions for actions are available.

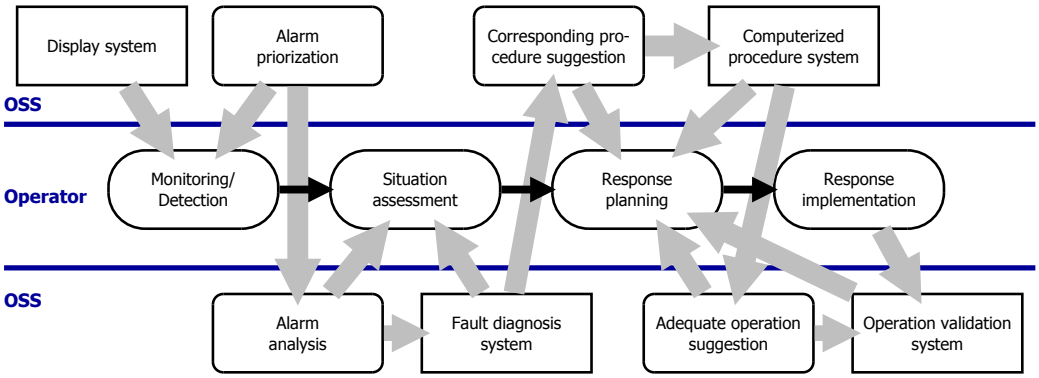


Figure 5.6: An integrated operator support system to support operator's cognitive process [89].

The fault diagnosis system consists of a modified *dynamic neural network* (MDNN) and a *dynamic neuro-fuzzy network* (DNFN), which partially act as one double-redundant system. MDNN handles digital inputs such as

alarms and on-off information, whereas DNFN handles analog inputs such as measurements. The proposed MDNN is based on multi-layer perceptron network, and the modifications make it handle time deviations in recognized patterns (*e.g.* alarms). [89]

The system was tested with a group of students as operators, by measuring the operator performance based on workload required to accomplish a task in a fault case, and accuracy. Workload was measured by a subjective self-assessment of an operator (NASA-TLX), in which the required mental effort was rated. The accuracy was measured as the failure probability (of the given task), which was based on diagnosis error and operation error. The overall result was positive, but in some cases the advisory system also had adverse effects. [90]

5.3.2 A pattern-based operator support system for fault diagnosis

Ayodeji *et al.* [91] present an operator support system for decision making during abnormal transients or disturbances of the process, based on artificial neural networks and a "large knowledge base, which is developed by collecting the plant's time-dependent transient data from the system". The authors state that this kind of data-driven approach has replaced the probabilistic methods of diagnostics, such as fault trees, Bayesian network and Markov chain, since these are too dependent on expert knowledge, too prone to error, and cannot be easily updated. However, even though the authors do not elaborate, judging by the earlier research by Vinod *et al.* [92], the knowledge base in this case means a trained neural network, the training of which requires data (fault cases) generated with an external simulator.

In the proposed system, the input vectors of ANNs are normalized to zero mean unit standard deviation, and also PCA is applied to reduce the dimensionality from about forty variables into about ten features. The ANNs have two outputs, indicating the location and size of the fault. [91]

Two different neural networks were tested: recurrent *Elman neural network* (ENN) and *radial basis function network* (RBFN). RBFN has one hidden layer, the size of which is automatically increased until the mean square error goal for the training dataset is met. RBFN is faster and computationally less demanding compared to other type of multilayer perceptron networks. ENN is able to predict sequential data due to its temporary memory available in the form of recursive loops in the hidden layer. This enables recognizing events even with slightly different time scales. The ENN was trained in Matlab using gradient descend with momentum (traingdx) and

adaptive learning rate back propagation (ALBP) with tangent-sigmoid activation function in the hidden layer and a linear activation function in the output layer. The overall performance of RBFN was better. [91]

5.4 Use of fuzzy neural network for rule generation in activated sludge process

Du *et al.* [93] implemented a fuzzy-neural network system to extract fuzzy rules from a set of numerical process data, in order to enable heuristic reasoning in process control. The goal was to assist ordinary operators to work at the level of an experienced operator who is able to use heuristic control rules to control the sludge process. With the learning capabilities of neural networks and the reasoning capabilities of fuzzy rules, the system was able to learn the complex relations in the process while generating logic linguistic rules for heuristic reasoning.

The neural network was a three-layer feedforward network having one hidden layer of four nodes. The weights of the network were real numbers and the inputs were fuzzified. The input variables were the feed flow rate q, feed concentration u and recycle ratio α, and all of these were fuzzified to three levels (L=low, M=medium, H=high), resulting in nine inputs total for the ANN. The output variable was sludge age θ, fuzzified the same way, giving three output neurons. θ was considered as the manipulated variable for the process. The structure of the ANN is shown in Figure 5.7.

A simulator was used to create the process data for training the ANN. Data was normalized based on the normal operating range of the variables. Conventional membership functions were used for the fuzzification of the input variables, as shown in Figure 5.8, and a customized function was used for the recycle ratio α since its fuzzy value is relative to the feed condition. The rules generated are presented in Appendix C. [93]

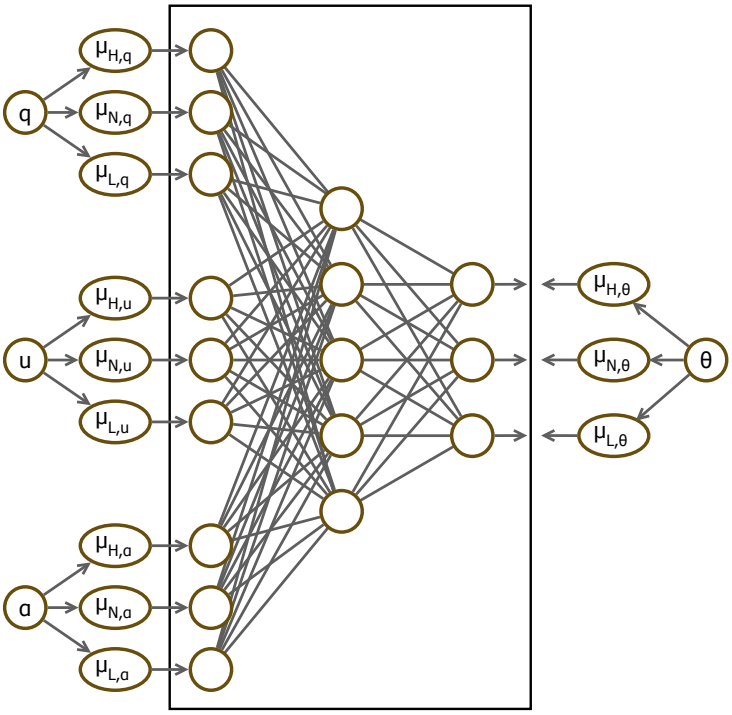


Figure 5.7: Neuro-fuzzy network for creating heuristic control rules [93].

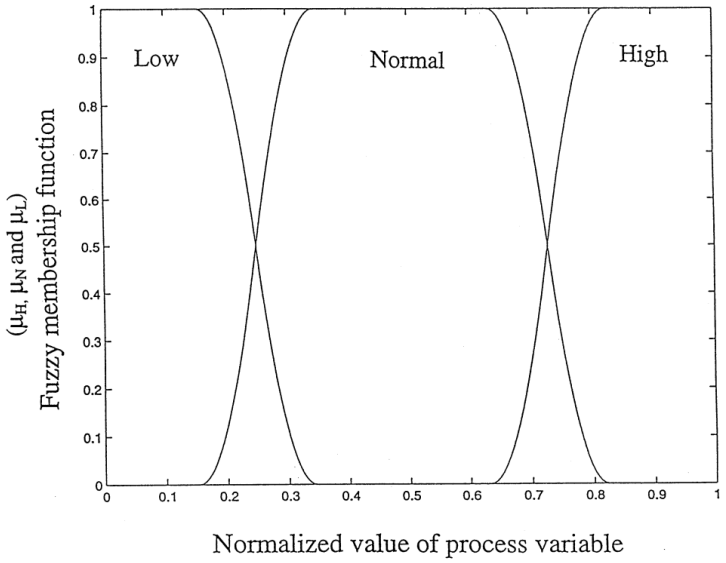


Figure 5.8: Fuzzification of the input variables [93].

Chapter 6

Framework objective and requirements specification

In the literature part of this thesis, the current status of digital advisory systems for process operators was reviewed, these systems being one potential application that could make use of machine learning in the field of process industry. The literature study shows that both supervised and unsupervised methods have been used to implement process operator's advisory systems, but otherwise more specifically, the methods have been quite application-specific. This creates special requirements for the software that is used to implement the machine learning part of the advisory system, especially in terms of modularity and extensibility. On the other hand, the characteristics of production software in process industry also create their own special requirements, *e.g.* in terms of communication protocols, application platforms, scalability and information security. Therefore, a platform design and a framework is required, to enable controlled development and deployment of machine learning applications.

The goal of the experimental part of this thesis is to design and implement a proof-of-concept for a machine learning framework for petrochemical process industry applications, with respect to the given requirements specification. In this chapter, the requirements are presented. However, as always in the process of defining new software architecture, defining the requirements, the architecture and the implementation is an iterative process [94], as shown in Figure 6.1.

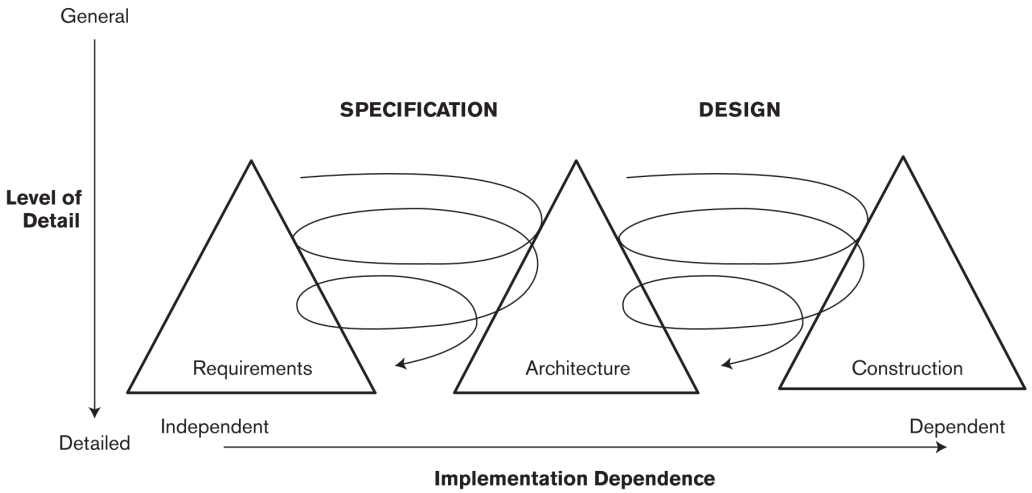


Figure 6.1: The process of defining software architecture. The curly arrows represent the iterative process. [94, p. 76]

6.1 Software context, data acquisition and the user

The framework will be implemented to be a part of or compatible with NAPCON Analytics which is an advanced data analytics and visualization solution for process industry. Utilizing its existing components is not a requirement but probably makes implementing faster and the end result more coherent.

The primary data source will be NAPCON Informer and the communication protocol for the process data will be OPC UA. History data will be used for machine learning model development, and it must be possible to easily select correct variables and the time interval with a graphical user interface. Predictions should be generated in real time with real-time data. The user will be the user of NAPCON Analytics, however the framework should double as an internal tool for data analytics and machine learning research.

6.2 Functionality and supported algorithms

The first two use cases for the framework are data mining and soft sensor development. Data mining includes for example data preprocessing and clustering. For data mining, modern software libraries must be supported, such

as Scikit-learn. The development of soft sensors will be done with modern deep learning libraries, and the framework should easily support the addition of new machine learning libraries and tools.

Soft sensors and similar online deployments require a model server that supports updating the model while keeping the application (*e.g.* a soft sensor) interface unchanged. Multitenancy should be supported, and the applications developed with the framework should easily be compatible with multiple client applications, especially within NAPCON Suite.

Chapter 7

Comparison and choice of compatible components

Analyzing the requirements in Chapter 6 reveals that they can be categorized into the following tasks which are also roughly the steps of the development of a machine learning model, as shown in Figure 2.5 (on page 12):

- data acquisition
- data preprocessing and basic machine learning
- deep neural network training
- model deployment
- model management
- user interface

Several open source components with permissive licenses are available for most of these tasks. However, choosing the optimal and compatible components without locking too much into one ecosystem in a restrictive way is not a trivial task. Furthermore, there is a high risk that using multiple open source components results in a high technical dept design pattern with too much glue code (*i.e.* code that makes inherently incompatible codebases or software components compatible with each other) and experimental configuration, since the actual machine learning code tends to form only a small fraction of real-world machine learning systems, as shown in Figure 7.1 [95]. According to Sculley *et al.* [95], common architecture smells of machine learning systems include the *multiple-language smell* which is usually a consequence of using components that are easily available but implemented in different languages, usually making testing and maintenance more difficult. The chosen components should also be compatible as they are, since in forking, the

support from the open source community (including compatible updates) is more or less lost. In *forking*, a copy of the source code is made, to start new independent development, usually by another group of developers. In this chapter, available and the most suitable options are briefly reviewed and an initial choice for a component for each task is presented. The licenses and API languages of the mentioned software products are listed in Appendix E.

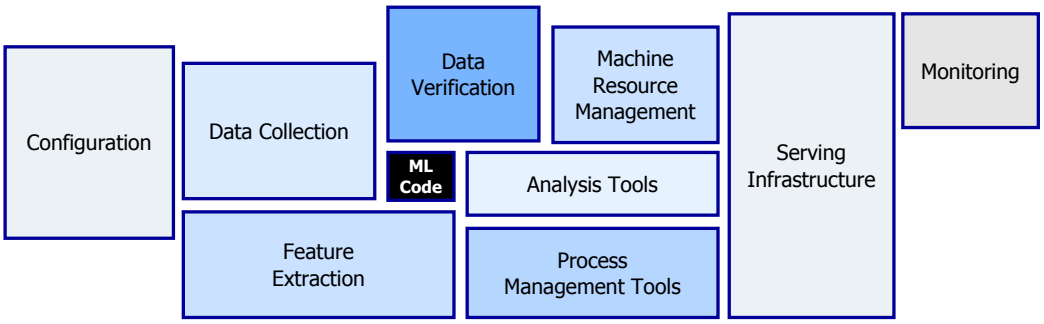


Figure 7.1: Real-world machine learning systems often consist of mostly supportive components for the actual machine learning component (black box in the figure) [95].

7.1 Data acquisition

In the requirements, the primary source of data is an OPC UA server. NAPCON Analytics natively supports OPC UA, therefore being the primary option, and also adequate when the amount of data and the nature of the processing are such that the resources of one computer are enough. However, using large datasets with machine learning algorithms may require specialized file systems, databases and parallel processing capabilities. Apache Hadoop and its ecosystem is probably the most popular open source solution in that area.

The Hadoop Distributed File System (*HDFS*) makes it possible to store large files across multiple computers. HDFS uses a master–slave (NameNode–DataNodes) architecture, the slave nodes of which serve the read and write requests coming from the client of the file system. The master node (NameNode) stores the metadata and directs the traffic from clients to the DataNodes. HDFS is fault tolerant in case of disk failure. [96]

Introduced in 2007, Apache *Hadoop* is an open source implementation of the combination of the MapReduce processing engine and HDFS [96].

MapReduce is a programming model that enables processing big datasets in a parallelized and distributed way. The current Apache MapReduce uses the Apache YARN (Yet-Another-Resource-Negotiator) which is a framework for implementing applications for distributed processing. [97] Hadoop and MapReduce are best at linear processing of big datasets when the speed of processing is not critical [98]. While Apache Mahout library can be used for machine learning, Hadoop with MapReduce is currently not very viable and versatile platform for machine learning, especially if iterative computation is required [96].

Apache Spark, which was initially developed at the University of California, Berkeley, is based on MapReduce and is often considered as an improvement over Hadoop's MapReduce. The main concept of Spark is the Resilient Distributed Dataset (RDD) which can be seen as an immutable (*i.e.* read-only) distributed shared data storage. Spark supports iterative computation, has stream processing capabilities and utilizes in-memory computation making it generally faster than MapReduce on Hadoop. Spark has also been considered as easier to program, although the learning curve can be quite steep. [96, 99][4, p. 144] MLlib is a machine learning library for Spark, and it includes algorithms for classification, regression, clustering and collaborative filtering [99]. Neither MLlib nor Mahout offer deep learning capabilities, at least for deeper neural networks than a multilayer perceptron with one hidden layer [96].

Apache Flink has more improved streaming capabilities compared to Spark. FlinkML is a machine learning library for Flink, and its supported algorithms include SVMs, multiple linear regression, k-nearest neighbor and multiple data preprocessing algorithms. [99, 100] However, while Flink may become a replacement for Spark, it is still quite a new project, and thus if the streaming features are not required, Spark is more mature and therefore probably a safer choice [101].

While setting up the Hadoop environment is out of the scope of this thesis, and doing more advanced machine learning in a parallelized manner is probably quite difficult in practice, the machine learning framework should be easily compatible with the Hadoop ecosystem for future extensibility. In this thesis, NAPCON Analytics and its OPC UA client is used for data acquisition.

7.2 Data preprocessing and basic machine learning

In addition to the preprocessing capabilities of the Hadoop ecosystem, there are two well-known options for data preprocessing and feature engineering in machine learning: Caret package with R language, and Scikit-learn with Python.

Another framework worth mentioning is H_2O, which is written in Java and consists of both Apache-2.0 licensed open source projects and commercial products developed by the company H_2O.ai [102]. The open source components provide a distributed in-memory machine learning platform, somewhat comparable to Scikit-learn, but more scalable [103]. One of the components, Sparkling Water, integrates Apache Spark into the platform, making H_2O compatible with the Hadoop ecosystem. H_2O can be used with Java, Python, R and Scala, and in addition, a notebook-style web user interface is provided, the language of which is CoffeeScript. Trained models can be exported and deployed in POJO (Plain Old Java Object) or MOJO (Model Object Optimized) formats.

R programming language and environment was created for statistical computing and graphics, and it is mostly written in the R dialect of S programming language. It is possible to link and run C, C++ and Fortran code at run time. R and its Caret package provide a versatile collection of statistical tools, for example for (non)linear modeling, time series analysis, classification and clustering [104].

Scikit-learn is a Python library which implements a wide collection of machine learning and data preprocessing algorithms for medium-scale unsupervised and supervised problems. Due to its BSD license and Python language, it is easy to use Scikit-learn also in commercial applications. Scikit-learn has a moderate number of dependencies, and some of them are compiled or written in C++ for efficiency. In terms of speed, Scikit-learn is generally faster than many other machine learning libraries for Python, such as *mlpy*. [105] A flow diagram for choosing the correct machine learning algorithm is presented in Appendix D, and the workflow is presented in Figure 7.2.

In terms of functionality, Scikit-learn is often compared to R language, and they are often found quite comparable. While R with Caret is sometimes found more user-friendly, Scikit-learn is often found faster. However, the most significant advantages of Scikit-learn over R (and GPL-3 licensed Caret) is its BSD license and the universality of Python language. [106–108] In this thesis, Python and Scikit-learn along with the NumPy, SciPy and Pandas libraries are the primary choices for data preprocessing.

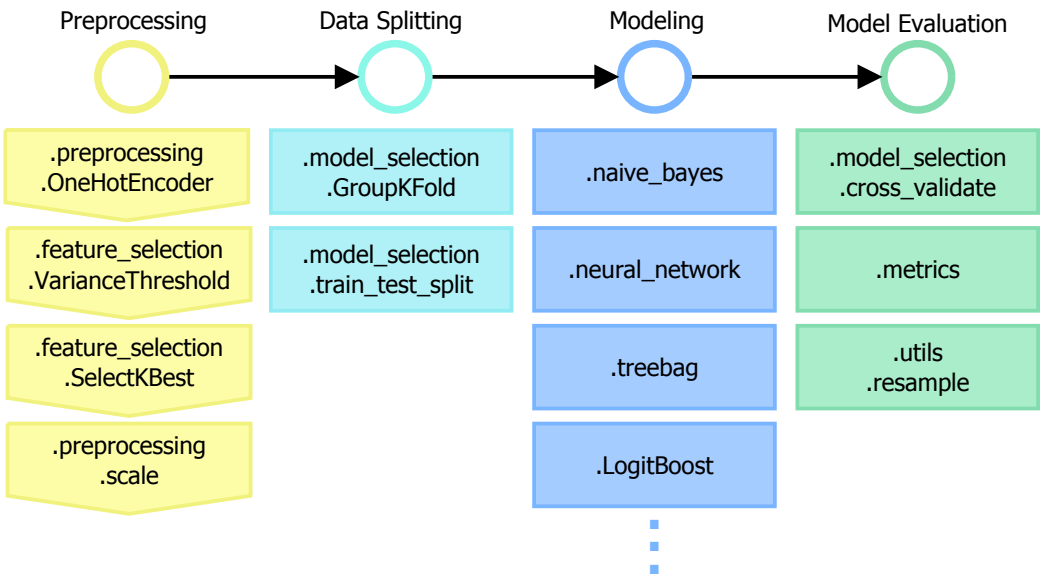


Figure 7.2: Data mining workflow in Scikit-learn [106].

7.3 Deep neural network training

There are many open source deep learning frameworks with permissive licenses, in many cases initially developed or later backed up by some large company, such as Google, Microsoft, Facebook or Amazon [109]. A summarizing comparison of the currently most popular deep learning frameworks is presented in Table 7.1. Recurrent neural network (RNN) modeling capability is included as an criterion because of its importance when dealing with time-series data. The licenses and API languages are listed in Appendix E.

According to the Table 7.1, three frameworks stand out: TensorFlow, PyTorch and Microsoft Cognitive Toolkit (also known as CNTK). Since the frameworks are at their best in slightly different use cases, and the framework designed in this thesis should be adaptable to versatile use cases, it should be easy to add a support for a new deep learning framework. As a starting point, TensorFlow and PyTorch will be chosen as the supported frameworks.

7.3.1 TensorFlow

TensorFlow is a low level deep learning framework and numerical library developed by Google. TensorFlow uses static data flow graphs to define how a series of deep learning algorithms process the batches of data (*i.e. tensors*). [4, p. 140] Input nodes of the graph are called *placeholders*, and weights, biases

Table 7.1: A comparison of deep learning frameworks, on a scale of 0–3 (-/•/••/•••), summarizing multiple sources. Since it is virtually impossible to make a universally applicable and summarizing comparison, this table should not be used for comparing the frameworks in one given use case. Moreover, this table can be considered as a representation of the general average opinion about the current state of the frameworks. [110, 111]

Framework	**Community support and tutorials**	**Distributed execution**	**RNN modeling capability**	**Usability and portability**	**Speed**	**Multi-GPU support**	**Keras compatible**
Caffe	•	-	-	•	•	•	-
Caffe2	•	••	•	••	••	•	-
CNTK	•	••	•••	•	••	••	•
MXNet	•	•	•	••	••	•••	-
PyTorch	••	••	••	••	••	••	-
TensorFlow	•••	••	••	••	••	••	•
Theano	•	-	••	•	••	•	•
Torch	•	-	••	•	••	••	-

and other mutable values of the graph are called *variables*. In addition to static graphs (*define-and-run*), there is a new *define-by-run* interface (called *Eager Execution*) introduced but still in its early stages. A define-by-run paradigm usually makes debugging and prototyping easier. Due to the low-level nature of TensorFlow, higher level abstractions have been built on top of it. The best known option is to use *Keras* with TensorFlow as its backend. [111]

TensorFlow supports distributed execution in a cluster that consists of TensorFlow servers and clients created with TensorFlow itself. A client program is usually written in Python or C++, and it builds the computational graph and creates a TensorFlow *session* which uses the servers in the cluster. Cluster management programmatically with Kubernetes is planned but not yet supported. [112]

TensorFlow has separate install packages for CPU and GPU enabled versions. In addition, a web-based visualization tool is provided, called *TensorBoard*, which visualizes the summary data that is written to disk during training. Third-party libraries make it possible to use TensorBoard with PyTorch and CNTK as well. TensorFlow does not natively support the *ONNX* format (Open Neural Network Exchange) which is used for saving and deploying the models. [111]

7.3.2 PyTorch

PyTorch is a define-by-run deep learning framework developed by Facebook. PyTorch resembles and is partly based on Torch, while being also well integrated with Python. The imperative nature and the Python API of PyTorch have been considered as an advantage in prototyping and research. Tensors in PyTorch are similar to NumPy arrays, and *variables* are the nodes in the computational graph including the mutable values. A *module* is used to store the weights of one layer of the neural network. [111]

PyTorch supports distributed execution with four different backends: TCP, MPI, Gloo and NCCL, each of which has a different set of supported functionalities for CPU and GPU (CUDA) tensors [113]. Most of the deep learning libraries are able use Nvidia GPUs, and thus CUDA and cudNN libraries, for faster calculation, such as matrix multiplication. Possible future alternatives for GPU programming frameworks include OpenCL, HIP by AMD and MKL-DNN by Intel. [111] The CPU and GPU enabled versions are in the same PyTorch package. A visualization tool *Visdom* can be used with PyTorch, and ONNX is supported. [111] Caffe2 is being merged with PyTorch [114].

7.4 Model deployment and management

The two main issues in the model deployment are the *model persistence* (saving the trained model for later use) and the *inference* system (*i.e.* prediction serving) that is independent of the machine learning framework used for training. The inference system should also be easily usable from different applications and environments, with high availability.

For Scikit-learn and other Python models in general, the recommended way to implement model persistence is to serialize the model with Pickle [115]. *Pickle* can be used for serializing Python objects into a byte stream (or a binary file) and for de-serializing them back into Python objects elsewhere. However, the most significant limitation of pickling is that the versions of Scikit-learn when pickling and unpickling, must match, or otherwise the unpickling is not supported or at least guaranteed to succeed. [116] In addition to picking, a small software project *Sklearn-porter* can be used to transpile some Scikit-learn models into Java or C [115].

The prediction serving for pickled Scikit-learn models can be done *e.g.* by wrapping the model into an independently deployable Python-based web application, for example by using *Flask*. This idea of turning learned models into *microservices* is often used in (open source) inference systems. In addition, if the running environment is included with the model by using containers for example, the microservice architecture also partially solves the problem with different software versions when unpickling, although it does not help updating the version compatibility of a model.

Another approach for model persistence is to use an interoperable intermediate format, such as ONNX (Open Neural Network Exchange). *ONNX* is an open source format for neural networks, initiated by Microsoft and Facebook, with IBM and Amazon joined later. [115] Google and the TensorFlow community have not shown as much interest in ONNX, but some third-party converters exist. [117] While the ONNX format and its compatibility with various machine learning frameworks is in its early stages, it is an advantage if the inference system supports the ONNX format.

Currently, the best known open source inference servers with permissive licenses include TensorFlow Serving, Clipper, Model Server for Apache MXNet (MMS) and Apache PredictionIO. Other open source projects with less permissive licenses include DeepDetect (LGPLv3 [118]) and PlaidML (AGPLv3). Apache PredictionIO is compatible with Spark MLLib, and therefore worth further examination if the Spark ecosystem is used. In this thesis, Clipper is the chosen tool for model management and deployment, mostly because of its Apache-2.0 license and versatility compared to Tensor-

Flow Serving.

7.4.1 TensorFlow Serving

TensorFlow Serving is an inference product developed by Google, for serving TensorFlow models in a production environment. TensorFlow Serving aims to provide a high-performance API for querying predictions and deploying new models without modifying their front-end applications. TensorFlow Serving supports TensorFlow models only, although it should be possible to add support for other ML frameworks with a C++ API, by implementing a custom model-specific wrapper in C++, called *servable* (see Figure 7.3). TensorFlow Serving uses the gRPC protocol between the server and the client, and a RESTful API is also available for classification, regression and prediction on TensorFlow models. Version management is supported to update or roll back to an old version of a model. [115, 119, 120]

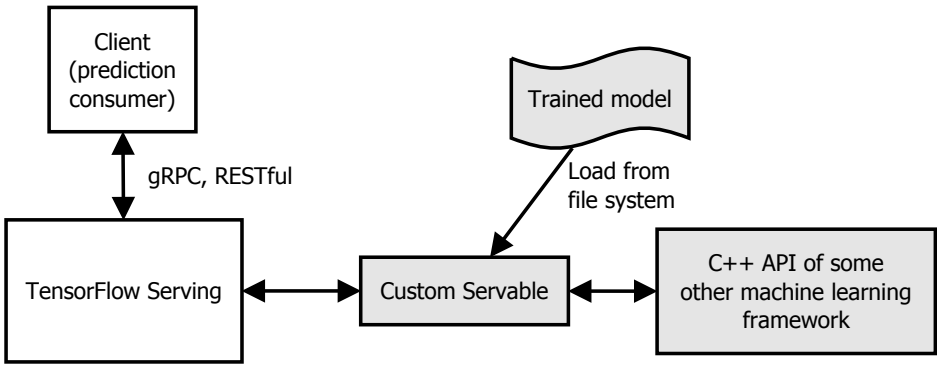


Figure 7.3: Custom Servable in TensorFlow Serving [115].

7.4.2 Clipper

Clipper is a general-purpose open source prediction serving system, developed by the RISELab of the University of California, Berkeley. Clipper is written in C++ and Python, and it deploys the models as microservices which expose the inference interface by using a RESTful API. Similar to TensorFlow Serving, Clipper supports model versioning, *i.e.* updating or rolling back to an old model without changing the front-end application. However, Clipper supports not only TensorFlow models, but also pure Python, Scikit-learn, PySpark, PyTorch and MXNet models. In addition, implementing a support for a new machine learning framework should be quite easy. [115, 121]

The runtime components of Clipper form a cluster, including four main components, as shown in Figure 7.4: a query processor, a management tool, a metrics server and a configuration database. The models are deployed as Docker containers by using a Python API. [122]

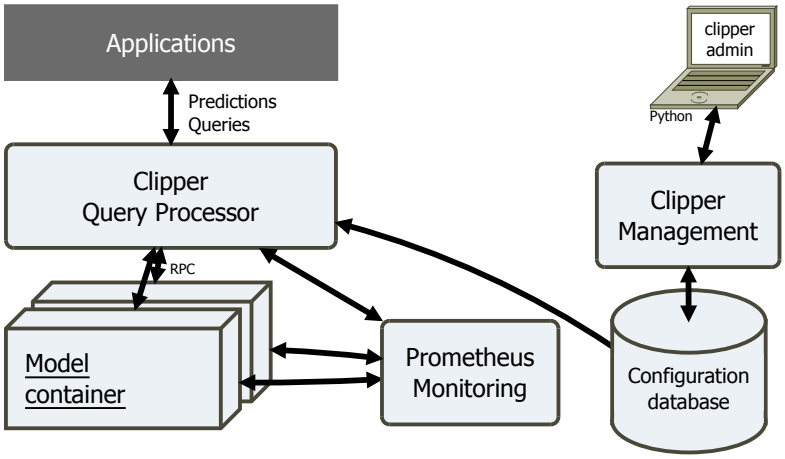


Figure 7.4: The architecture of the Clipper runtime cluster [122].

The architecture of the Clipper's query engine can be divided into two layers: the model abstraction and the model selection layers, as presented in Figure 7.5 [121].

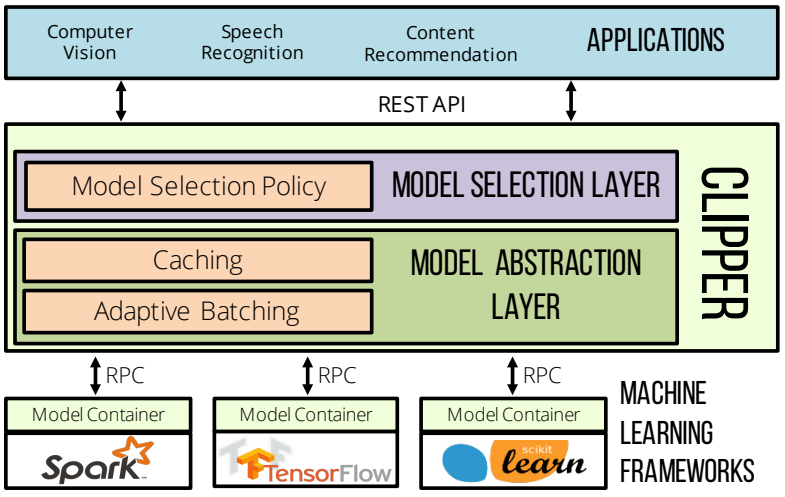


Figure 7.5: The data flow and prediction stack of Clipper [121].

Prediction requests from end-user applications are first processed in the *model selection* layer which dispatches the queries to one or more models. The *model abstraction* layer provides a common prediction interface, resource isolation and caching, and optimizes the query workload for batch oriented machine learning frameworks. Therefore, it uses the prediction cache or assigns the query to an adaptive batching query associated with the correct model. A cross-language RPC protocol is used in the communication between the model containers and Clipper. The model abstraction layer returns the results to the model selection layer which combines the results and confidence estimates into the final prediction which in turn is returned to the end-user application. [121]

In terms of prediction throughput and latency, according to Crankshaw (one of the developers of Clipper) *et al.* [121], Clipper is comparable to TensorFlow Serving. Figure 7.6 shows the results of a comparison where three TensorFlow models were deployed and tested with both Clipper and TensorFlow Serving. The models were a 4-layer convolutional neural network trained on the MNIST dataset, a 8-layer AlexNet trained on the CIFAR-10 dataset, and Google's 22-layer Inception-v3 network trained on the ImageNet database.

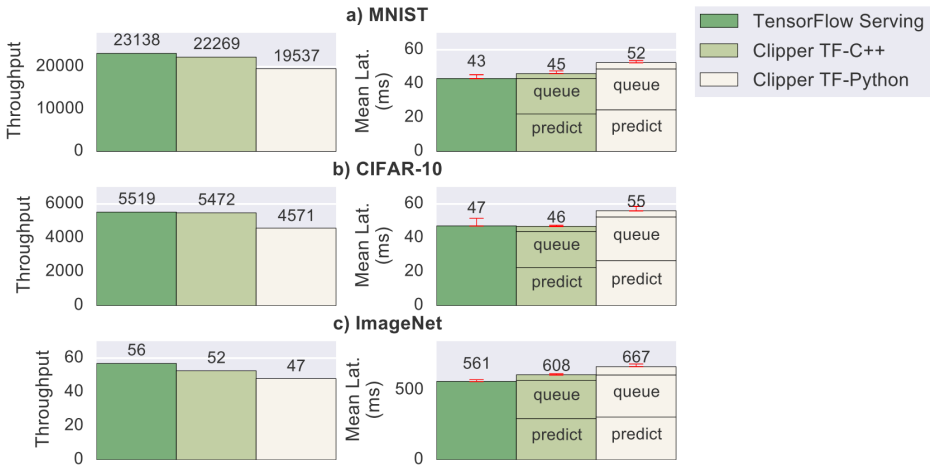


Figure 7.6: Comparison of the relative peak throughput and mean latency, between Clipper and TensorFlow Serving. Both Python and C++ APIs of TensorFlow were tested. The *predict* measurement includes the time spent in the inference (*i.e.* in TensorFlow code) and the *queue* measurement represents the time when the model container is waiting for the GPU to become available. [121]

The deployed models are stateless, which means that resource intensive models can be replicated across multiple machines. Clipper also supports linear ensemble methods, to improve prediction accuracy, although implementations for ensemble methods are not included in the current version (0.3). [121]

7.5 User interface

Most of the open source machine learning frameworks do not include a graphical user interface but only an API, for example for Python. However, graphical user interfaces exist in commercial products. The available user interfaces can be categorized into the following groups: API only, notebook-style UI and (flow-style) graphical UI.

Fully graphical user interfaces often visualize the data flow, where all the operations are visualized as blocks. A good example of such a GUI is the Microsoft Azure Machine Learning Studio, presented in Figure 7.7.

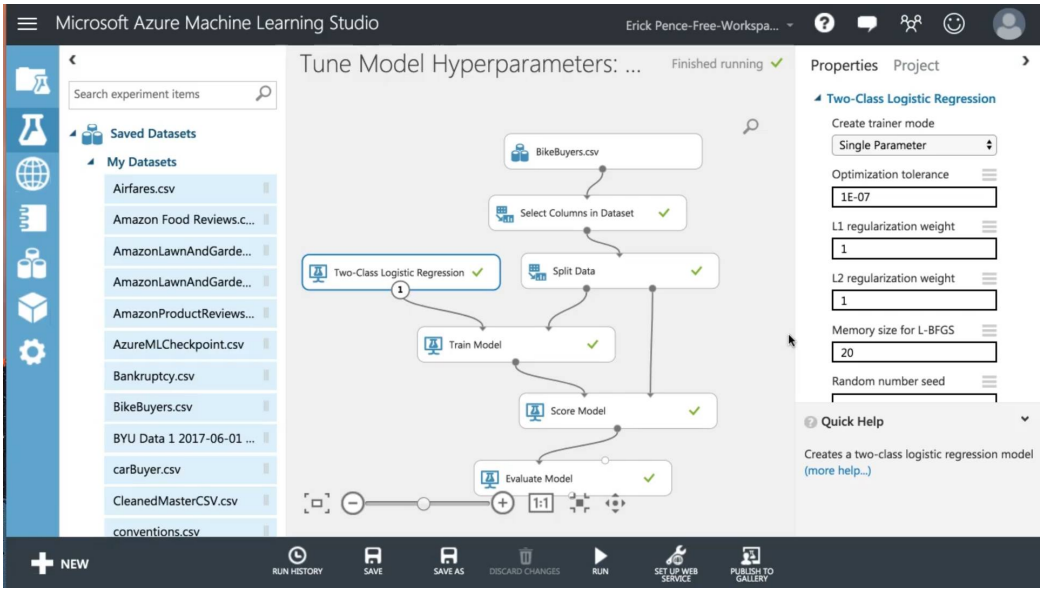


Figure 7.7: The GUI of Microsoft Azure Machine Learning Studio [123].

TensorBoard for TensorFlow looks somewhat similar, although it is used only for visualizing the computational graph (Figure 7.8). This kind of flow-type user interface could probably be built for example on top of Apache NiFi (see Figure 7.9), Node-RED, Pothos framework [124] or Darwin [125]. In NiFi and Node-RED, custom processing blocks can be written in Java

and JavaScript respectively. However, they are more than graphical user interfaces, and especially using NiFi for GUI purposes should probably be considered only if the Hadoop ecosystem is used.

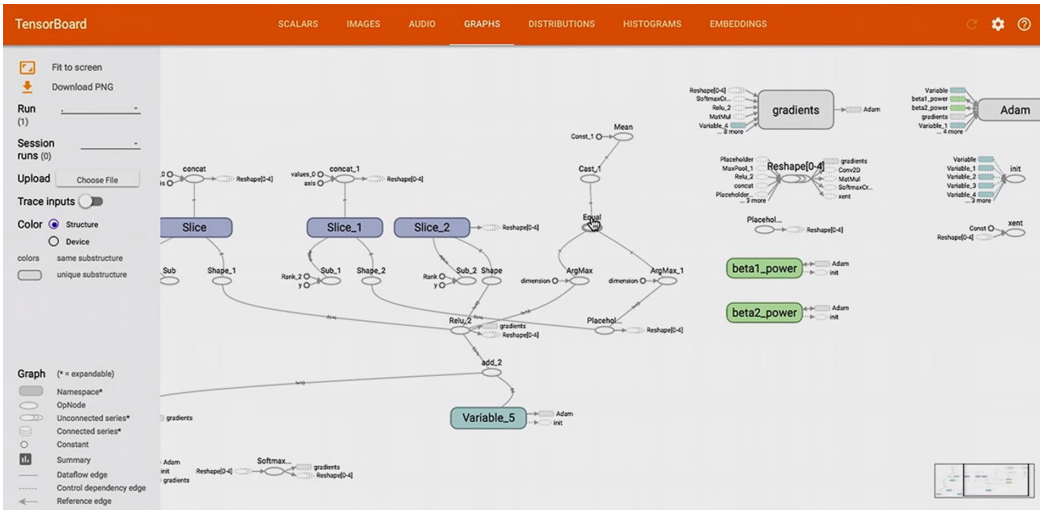


Figure 7.8: The user interface of TensorBoard [126].

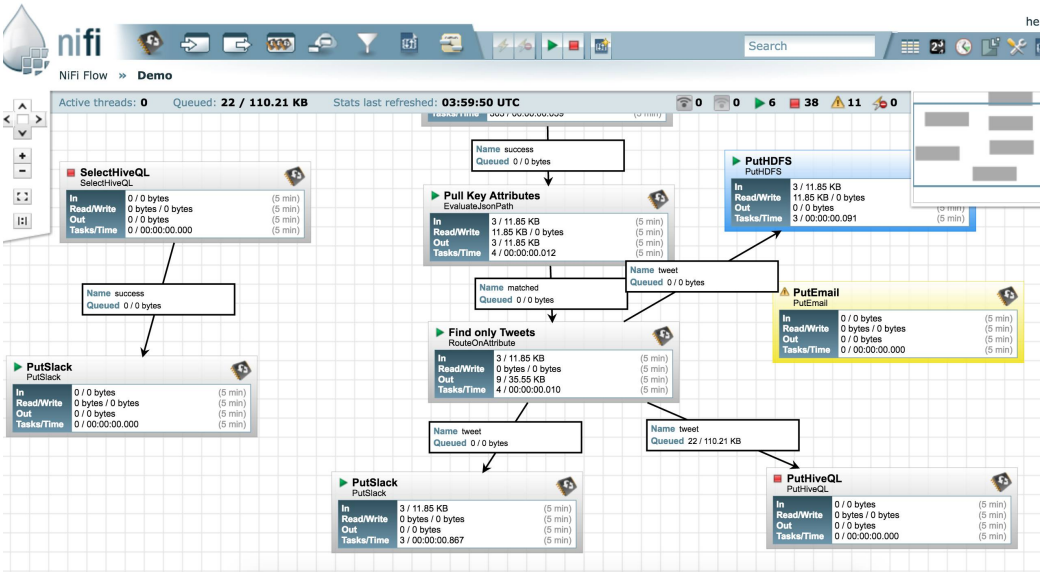


Figure 7.9: The user interface of Apache NiFi [127].

Notebooks are often used as semi-graphical user interfaces for machine learning. However, notebooks are more web-based code editors or light IDEs

than fully graphical user interfaces. One well-known example is the H_2O's (graphical) user interface Flow, shown in Figure 7.10. Other well-known notebook projects include Jupyter, R Markdown and Apache Zeppelin.

The common component in notebooks is a cell of code that can be run individually, after which the output is printed below the cell. By default, the output is text (such as standard output), but it can also be graphical UI elements, images or even embedded web pages. Currently one of the most popular notebook projects is Jupyter, which was also selected (along with its default Python 3 kernel) as the initial user interface in this thesis.

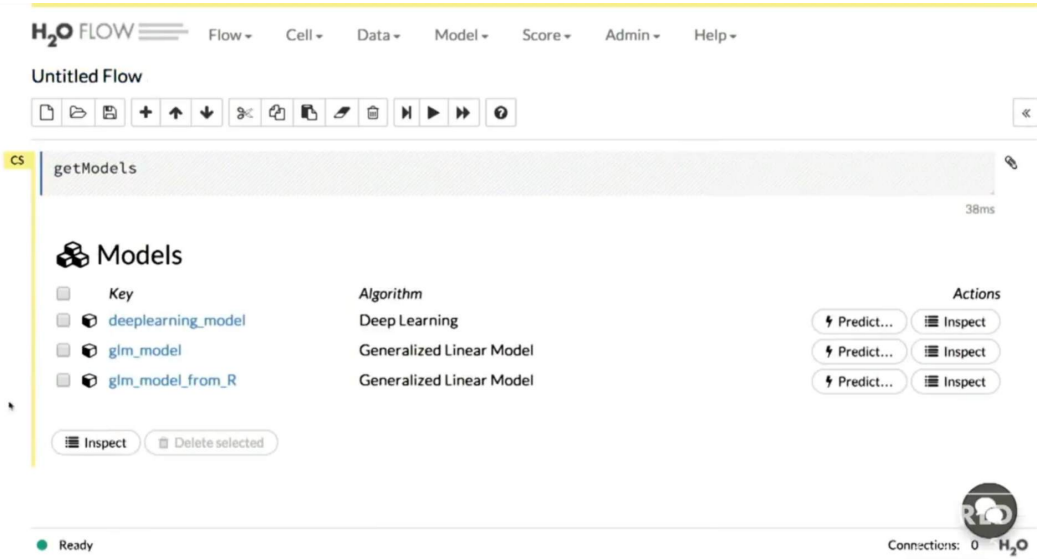


Figure 7.10: H_2O Flow is the notebook-style user interface of H_2O [128].

7.5.1 Jupyter Notebook

Jupyter Notebook is an open source web application based on IPython, which is an interactive Python shell, also used as the default execution environment in Jupyter. Jupyter itself is language-neutral, and the language used in the notebook can be changed by switching the kernel which is responsible for the code execution, tab completion, *etc.* [129] The Jupyter project is being constantly developed, and compared to many alternatives, it has a large community because of its universality. The user interface of Jupyter is presented in Figure 7.11.

As an user interface for a machine learning framework, a notebook is clearly not as easy to use than a dedicated GUI. However, since Python

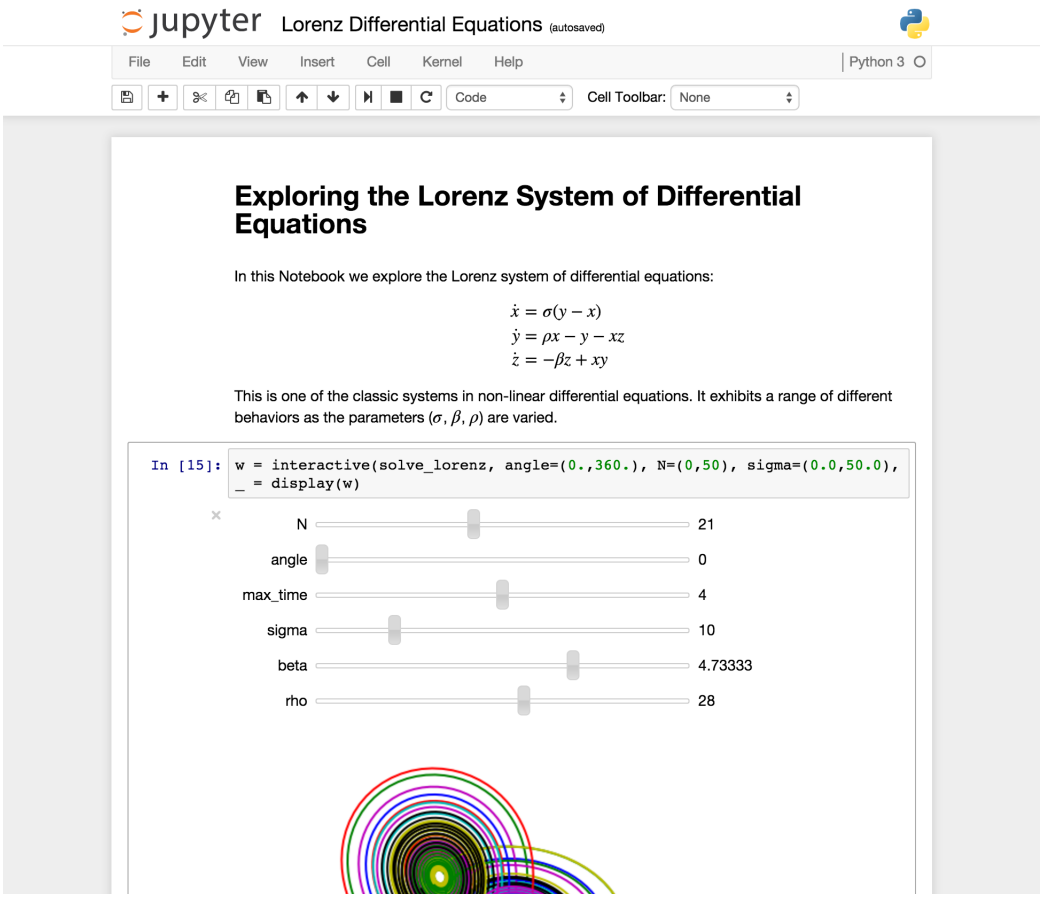


Figure 7.11: The user interface of Jupyter [130].

has become one of the de facto standard languages in machine learning, a notebook-style UI has multiple advantages: Most machine learning tutorials online are available in the IPython notebook format. In addition, it is easy to share own experiments or development, since the notebook can easily be downloaded in JSON format as an .ipynb file. Due to the plain text format, version control tools can also be used with the notebooks, although the version control of the source code itself is somewhat difficult due to the JSON encapsulation. Furthermore, compared to a dedicated GUI, it is easy to add support for new machine learning tools or frameworks since they can be easily installed and used immediately without GUI development.

Widgets can be used to add GUI elements, such as dropdown menus, buttons, sliders, progress bars *etc.* and also more advanced widgets are already available, such as 3D plots and interactive tables. [131, 132] Writing

documentation as a part of a notebook is also possible since the type of a cell can be changed to Markdown, and for example LaTeX equations are supported. [133]

The notebook front end communicates with the IPython kernel using JSON messages sent with ZeroMQ (or ØMQ) which implements the ZeroMQ Message Transfer Protocol (ZMTP). The kernel is not connected to the notebook file, as shown in Figure 7.12, but a new kernel process is started for each notebook. [134]

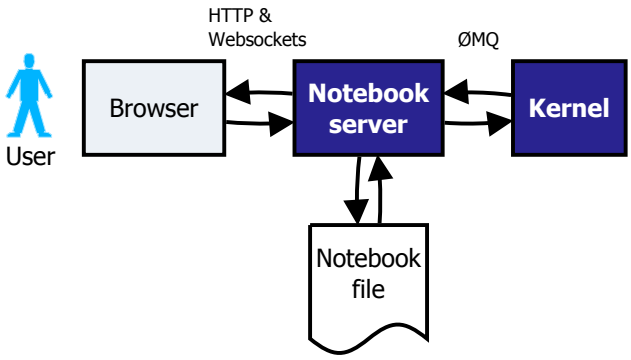


Figure 7.12: The components of Jupyter [134].

Chapter 8

Framework architecture

Based on the requirements specification in Chapter 6 and the selected components in Chapter 7, a practical high-level architecture was designed for the machine learning framework. In this chapter, the architecture is presented and an overview of the implemented parts and their additional required components is given.

8.1 Architecture overview

The designed architecture is presented in Figure 8.1. As mentioned in Section 6.1, the framework is implemented to be a part of NAPCON Analytics, and the primary data source is NAPCON Informer. Training with big data (the lower part of the diagram) and the background deployment are out of the scope of this thesis, but it is important to keep the machine learning framework compatible with them, and therefore, they are included in the architecture.

NAPCON Informer is an advanced OPC UA database server for industrial large scale applications with low latency requirements, certified by the OPC Foundation. NAPCON Informer also implements the serving of historical data via OPC UA, and it is also extensible due to plugin-based user-definable OPC UA information models. The extensibility, along with the other products of the NAPCON Suite, gives multiple options to implement methods for utilizing the predictions and other real-time results from the machine learning framework.

The machine learning environment, code named *ML factory* in the architecture, is a single-user environment for experimenting with process data and machine learning, and also for developing and deploying machine learning applications in Python. Process data is transferred to the environment from

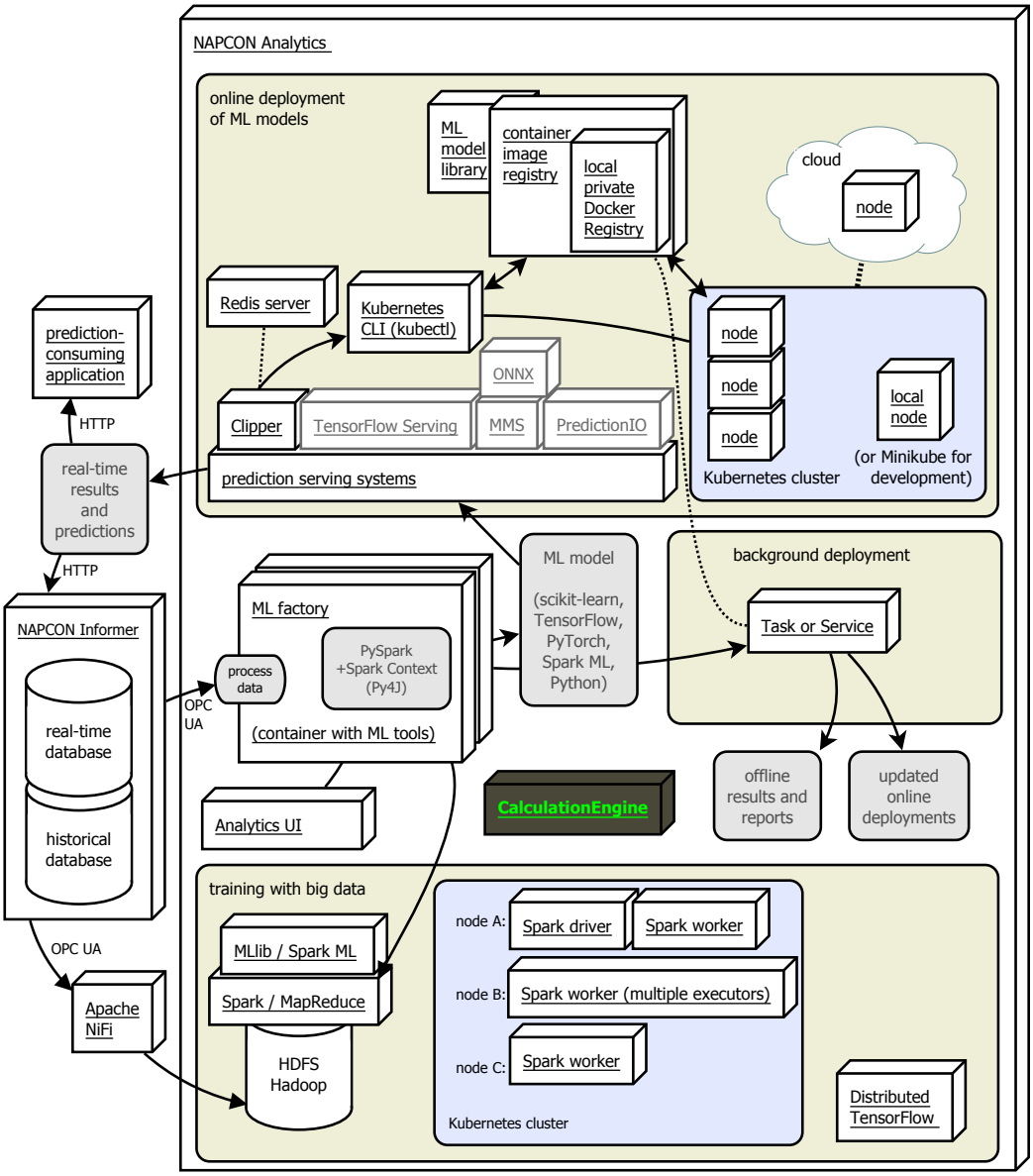


Figure 8.1: The high-level architecture of the designed machine learning framework. The implementations for the *background deployment* and the *training with big data* are excluded from this thesis. The background deployment primarily comprises any machine learning tasks that run in the background modifying the state of either a database or a deployed ML model.

NAPCON Informer via OPC UA. The ML factory also serves the notebook user interface, which is used as a part of NAPCON Analytics UI.

The ML factory is deployed as a container, and the container management and the data transfer are orchestrated by the calculation engine of NAPCON Analytics. The support for this orchestration was implemented into the calculation engine with a multi-user support. Upon request by the user, the data transfer is started for the requested variables and time span, and in case the ML factory does not exist yet, it is created for the user. In the current implementation, while ML factories can be created for multiple users, the data source and the online model deployment are shared resources for all users. Therefore, in this proof of concept, all the users and clients within one NAPCON Analytics deployment are also in the same information security context.

The ML factory is equipped with the tools selected in Chapter 7. For example, it is possible to visualize the process data, preprocess the data, train machine learning models, and develop Python applications that use the model. In the current implementation in this thesis, Clipper's Python client is used to deploy the models, but other prediction serving systems could be used as well, such as TensorFlow Serving.

The Redis database instance is used by Clipper, for persistently storing its internal configuration state. Since the deployed models are built into Docker containers (as explained in Section 7.4.2), Clipper uses Kubernetes and its command line interface (kubectl) for container management. The model containers are pushed into a private container registry which doubles as a library or storage for the trained models.

8.2 Docker

The core idea of *Docker* is to be able to pack an application with all the needed dependencies into a single standardized unit for deployment. Docker was initially created as an internal project in a platform-as-a-service company dotCloud, and in 2013 Docker was released as open source. Currently, Docker is developed by Docker Inc. and the development is supported by many big companies, *e.g.* Google and Microsoft. [135]

Essentially, a Docker *image* is a group of filesystem layers sequentially stacked to form the final union filesystem of the image, which is then run in an isolated environment by the kernel of the host machine [136]. Layers contain the information of the filesystem changes relative to the parent layer. When building the image *e.g.* with a *Dockerfile*, each step becomes a layer and consequently, an intermediate image. [135] The layered structure of Docker images makes it possible to quickly deploy a machine learning model as a self-contained microservice because most of the contents already exist in the

base image, and only the application itself is added as a layer. Updating the model only requires updating the application layer.

The running instance of an image is called a *container*. When a container is started, a writable layer is added on top of the (read-only) layers defined by the image. Any changes in the writable layer can be committed, resulting in a new image. [135]

Compared to traditional virtualization with a virtual machine that is completely isolated with its own BIOS and operating system running on top of the host machine, Docker containers run within the same kernel running on the host machine. This makes running containers lightweight since there is no overhead from running a guest operating system or a virtualization hypervisor. [135] However, the downside is that *e.g.* a Linux container does not have all the capabilities of a fully-featured virtual machine. Some capabilities can be added to a container, but usually by compromising the advantages of virtualization, especially the isolation between the container and the host system or between containers.

Docker images can be pulled from and pushed to a *Docker registry*, which is an application for storing the images. There are publicly available online registries, such as Docker Hub, serving a large number of repositories. However, deploying an own registry server is also possible. [135]

8.3 Kubernetes

Kubernetes (also abbreviated as *K8s*) is an open source tool for deploying containers across a cluster of computers, and it can be used for the orchestration and automated deployment and management of Docker containers. Kubernetes is developed by Google, and it is based on the best practices developed with Google's internal container system Borg, which can be seen as the predecessor to Kubernetes. The alternatives to Kubernetes include Docker Swarm and Apache Mesos. [135]

The role of Kubernetes is important especially with microservices where automated management is required, for example to enable communication between containers without manually opening network ports, or to enable automated model deployment balancing across a cluster. [135]

The key concepts of a Kubernetes cluster include the nodes, pods and services, as shown in Figure 8.2. *Nodes*, more specifically a master node and a set of worker nodes form the cluster, and they are either physical servers or virtual machines with Kubernetes installed. The master node is dedicated to handling and managing the cluster. The state of the cluster is stored in etcd, which is a fast and reliable key-value store that contains information

about what pods should be running and on which nodes *etc.* The API server accepts HTTP requests using JSON, and it is used as a front end to the state of the cluster. All the other components interact through the front end, and it is also used for the centralized management of the cluster since only the API server is connected to etcd. The scheduler and controller manager is continuously monitoring the instances of the deployed applications, and ensures their availability in case a node goes down or is deleted. In the worker nodes, *kubelet* is the process listening to commands coming from the master node. [135]

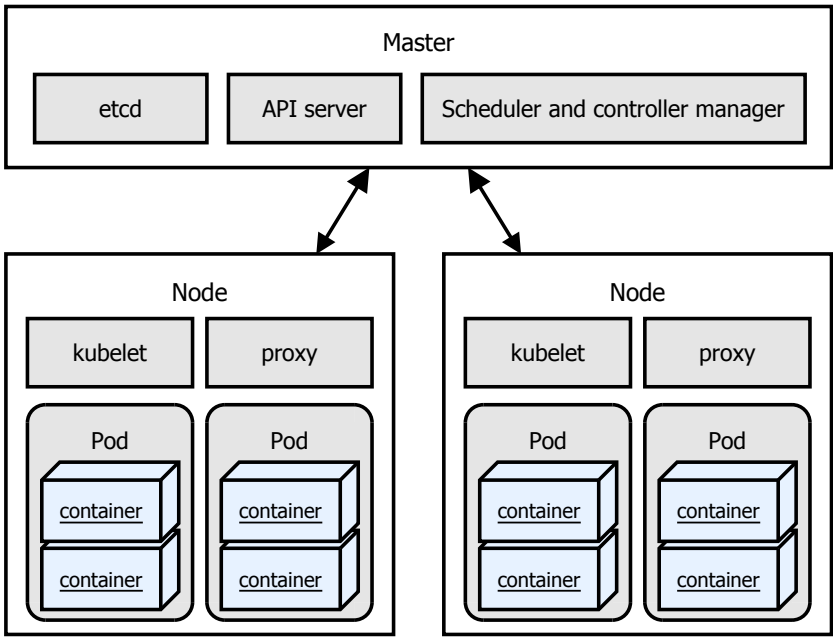


Figure 8.2: The architecture of Kubernetes [135].

A *pod* consists of one or more Docker containers, and it is the basic unit of execution in the Kubernetes platform. The containers running in the same pod share the disk, network namespace and security context. Therefore, the communication between the containers is usually done over localhost, and volumes (*e.g.* a local hard disk or Amazon Web Services storage) attached to the pod can be mounted in one or multiple containers. An example of a pod could consist of two containers: one including the deployed application and the other providing necessary tools to run the application, such as a database. Pods are stateless by design and therefore may be killed and disposed by the master node. [135]

A *service* is an abstraction that can be used to persistently provide access

to an application deployed over a group of pods, as defined by the *deployment* and its *ReplicaSet*. When using a service, Kubernetes takes care of load balancing by distributing the traffic to different instances of the application. [135]

While the API server of a Kubernetes cluster can be used with standard HTTP requests, *kubectl* is a command-line interface that makes interacting with the cluster faster and more convenient. In addition, the web-based *Kubernetes Dashboard* can be used as an GUI for managing and troubleshooting the cluster and the deployed applications. *Minikube* can be used for creating a local Kubernetes cluster with a single node, for development purposes. [135]

Chapter 9

Qualitative evaluation with use cases

A proof-of-concept system was constructed according to the architecture in Chapter 8, and a couple of use cases were used to qualitatively evaluate the framework against the requirements in Chapter 6. In this chapter, these use cases are presented and also illustrated, although the development of a graphical user interface was not included in the proof of concept, and therefore, the GUI is lacking in most cases. Furthermore, the solutions for the problems of the use cases are not necessarily accurate from the data science point of view, but they are enough to represent the workflow.

9.1 Process data acquisition and visualization

By using the graphical user interface of NAPCON Analytics, process data for three pressure measurements and one level measurement, for the time span of fifteen hours, was selected and transferred to the machine learning environment, by using OPC UA. Figure 9.1 illustrates the user interface which is already implemented in NAPCON Analytics. When typing the variable names, automatic completion and suggestions for matching database tags help finding the correct tag. Tab completion is also used in the machine learning environment where the selected dataset can be accessed with the `OPCUA_Dataset` object, as shown in Figure 9.2. In the figure, the user interface of the machine learning environment is embedded into the web-based GUI of NAPCON Analytics, and Google Chrome was used to display the webpage in all of the use cases.

The dataset was plotted using three different visualization libraries. In Figure 9.3, Matplotlib and an interactive plot capable of panning and zoom-

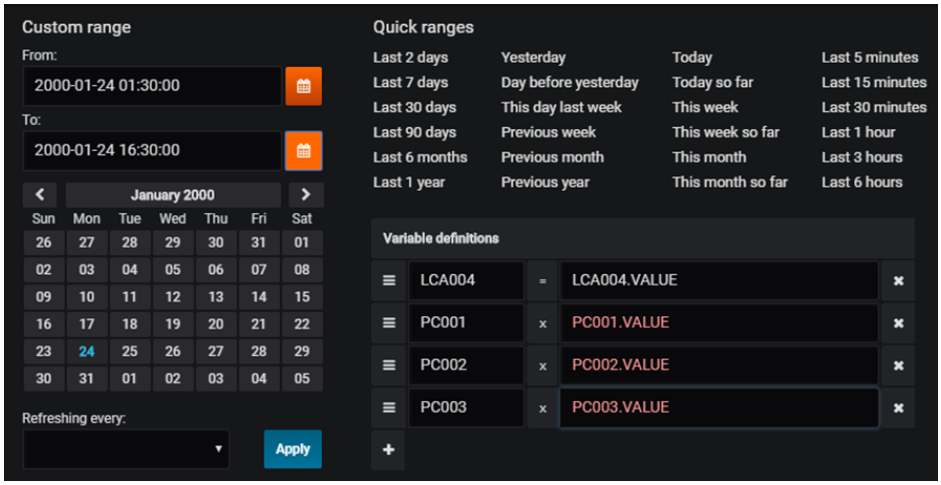


Figure 9.1: Choosing the variables and the time interval.

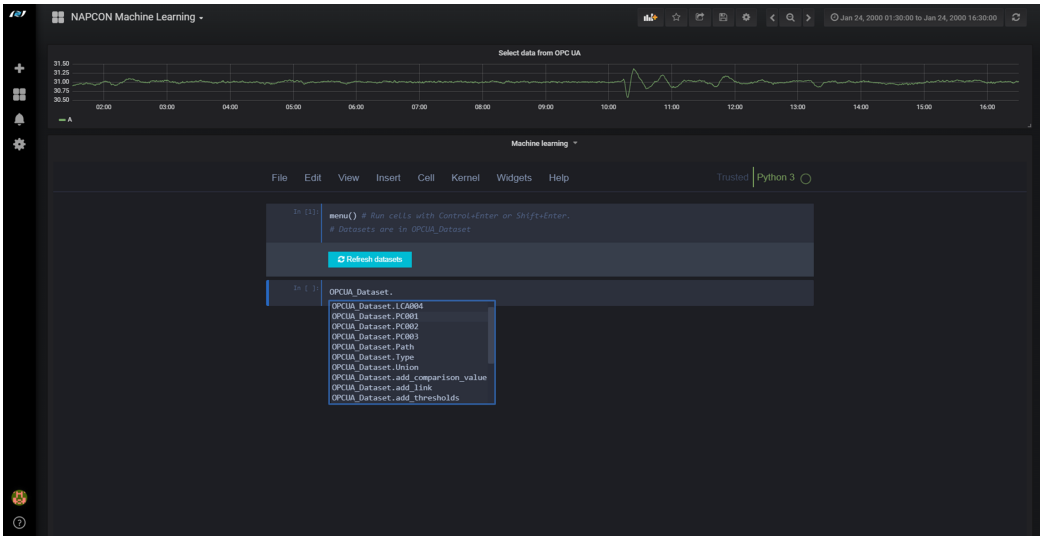


Figure 9.2: Accessing the dataset in the machine learning environment.

ing, are used for visualizing the dataset. Notebook widgets could be used to eliminate the need of code, by displaying *e.g.* a dropdown menu and a 'draw' button instead.

Figure 9.3: Visualizing the dataset by using two different plotting libraries.

9.2 Process data clustering for detecting operating points

As a continuation of the previous use case, the same dataset was used for generating and evaluating a use case where an operating state of a process is detected. The generated data should be considered as toy data, and the evaluation methods are accurate enough only for representing the workflow.

Clustering is probably the most common general unsupervised method to generate subgroups of similar types of observations [137], which is one way to detect operating points of a process. For the three pressure measurements, a new operating point was artificially created by shifting the values so that the new operating point could not be easily detected for example as global minima or maxima (see Figure 9.4).

The modified dataset was clustered by using the agglomerative clustering

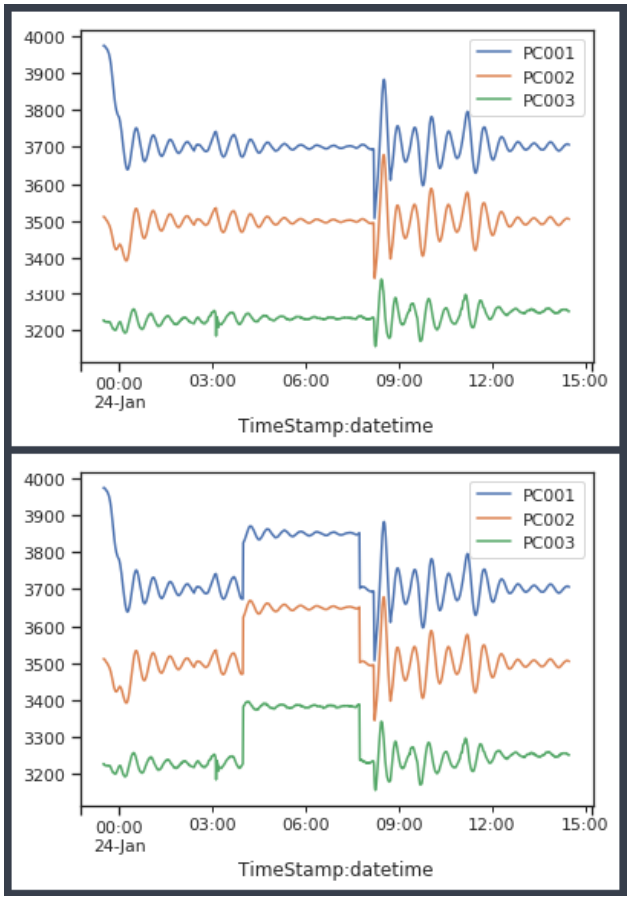


Figure 9.4: The original dataset (upper plot), and the generated dataset (lower plot) where the artificially added operating point can be seen in the middle of the time series.

algorithm of Scikit-learn, and the number of clusters was set to three. The results were visualized with a scatter matrix, as shown in Figure 9.5, where the clusters are marked with different colors. The algorithm was able to separate the operating points reasonably well. While being out of the scope of this thesis, experimenting with different algorithms and tuning their parameters could be easily done in the same environment by utilizing Python and the straightforward API of Scikit-learn.

```
In [29]:
from sklearn.cluster import AgglomerativeClustering
from sklearn.neighbors import kneighbors_graph

model = AgglomerativeClustering(linkage='average',
                                connectivity=None,
                                n_clusters= 3)

array_for_clustering = dataframe_reduced_mod.values[:,1:3]
model.fit(array_for_clustering)

In [30]:
# Add clustering results to the original dataset for visualization
dataframe_reduced_mod = dataframe_reduced_mod.assign(cluster = model.labels_)

In [31]:
sns.pairplot(dataframe_reduced_mod, vars=['PC001','PC002','PC003'], hue='cluster')

<seaborn.axisgrid.PairGrid at 0x7f87d6f3cbe0>
```

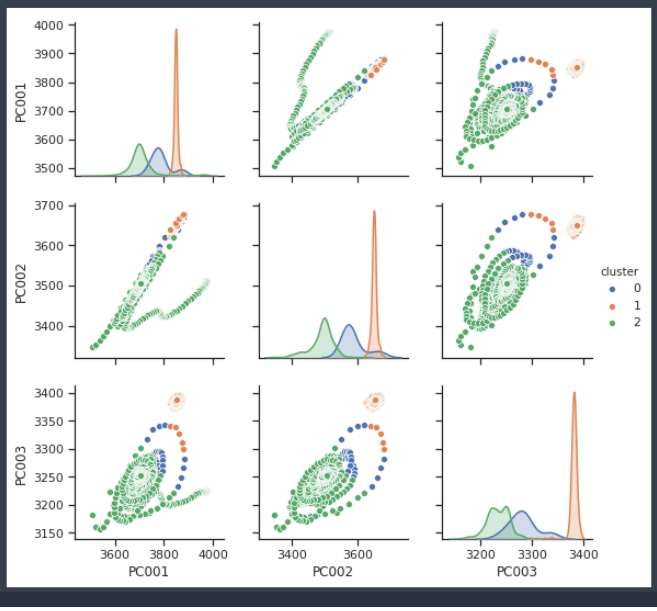


Figure 9.5: The operating points of the process were detected with the agglomerative clustering of Scikit-learn. The results can be visualized as a scatterplot matrix that is colored according to the clusters.

9.3 Training, deploying and querying a neural network model

In this use case, a simple neural network was trained using PyTorch. The architecture of the neural network was defined as one linear layer with $28 \cdot 28$ inputs and two outputs, as shown in Figure 9.6. An external dataset was used and uploaded to the machine learning environment by using the file

manager included in Jupyter. The dataset was a small part of the MNIST dataset which is a large database of labeled handwritten digits. The input of the neural network takes in one image with the resolution of $28 \cdot 28$ pixels, and predicts whether the image represents either the number one or two.

```
# Define a simple NN model, for recognizing MNIST handwritten numbers, in this case only whether
# the number is 1 or 2
class BasicNN(nn.Module):
    def __init__(self):
        super(BasicNN, self).__init__()
        self.net = nn.Linear(28 * 28, 2) # input is an image of 28x28 pixels, output either '1' or '2'

    def forward(self, x):
        if type(x) == np.ndarray:
            x = torch.from_numpy(x)
        x = x.float()
        x = Variable(x)
        x = x.view(1, 1, 28, 28)
        x = x / 255.0
        batch_size = x.size(0)
        x = x.view(batch_size, -1)
        output = self.net(x.float())
        return F.softmax(output)
```

Figure 9.6: Defining the neural network in PyTorch.

After training the model, it was deployed to the online deployment platform (the upper part of Figure 8.1). The deployment can be done with only one line of code, which again could be replaced with a couple of graphical UI elements that could be easily implemented as notebook widgets. The API used for the deployment hides the complexity of the online deployment architecture shown in Figure 8.1.

After the model was deployed, predictions could be queried with a standard HTTP request. In this case, the request was made in the same machine learning environment with the Requests library in Python because of its convenience, but the request could as well be sent from any other application that could benefit from the use of machine learning models. A randomly selected image of a handwritten digit was parsed into JSON format and sent in the HTTP request. The queried image and the response from the model is presented in the Figure 9.7 below. The latency limit for the deployed application was set to 100 ms, and the model was able to reliably generate predictions within the latency requirement, even with the modest hardware used to evaluate the proof of concept.

Figure 9.7: The input image sent in the HTTP query, and the resulting response from the deployed application using the neural network model.

9.4 Discussion

The use cases in this chapter show that the designed framework is able to qualitatively fulfill the requirements described in Chapter 6. The framework is well integrated with the NAPCON Analytics utilizing some of its technologies, while adding the capabilities to develop and deploy machine learning models trained with process data. As a proof of concept, the system is mostly ready to be deployed on dedicated hardware to enable quantitative evaluation which includes finding the possible bottlenecks in terms of usability and throughput of data and predictions.

Being only an initial proof of concept, there are naturally many areas for development. Next, real-world use cases should be well defined and then executed on the created system, to find out the most important targets of development, and to be able to further refine the requirements specification and also the architecture. The user interface should also be improved.

Out of the components used in the system, the model deployment in general, probably requires development the most: While the API of Clipper is simple and easy to use, a considerable amount of knowledge is required to actually be able to use the deep learning frameworks and create models that can be serialized or otherwise deployed. Furthermore in terms of production use, Clipper has some critical bugs that have not been fixed yet in the current version (0.3), although bug reports and development can already be found in GitHub. In addition, Clipper does not implement a way to manage the

Docker registry, and therefore, it is not possible to fully delete and clean up deployed models in the current implementation.

Chapter 10

Conclusions and future work

The success of modern artificial intelligence applications in everyday life has inspired the development of AI based applications also in the field of process industry. One use case for such an application is an advisory system for process operators, which augments the process operator's capacity to process and monitor information, resulting in better decision making in terms of safety and profitability.

The concept of advisory systems for process operators is not new, and the development is somewhat connected to the research of artificial intelligence. The minor recent theoretical advancement, as concluded in Section 3.6, does not necessarily make the older methods of digital advisory systems obsolete, but it raises the question if successful real-world use cases exist. As a conclusion from the numerous applicable case studies, including the ones in Chapter 5, in practice there are no standard generally accepted approaches for developing such systems, and even the oldest methods are used in some recent works. Furthermore, while giving also good results, most of the use cases are designed for and tested in quite a narrow and specific cases, which makes it difficult to evaluate the applicability and scalability of the methods used. Therefore, the most realistic approach for an advisory system seems to be a combination of multiple smaller applications, each of which are implemented with the best methods already available.

Machine learning has proven to be a very useful tool in developing intelligent or otherwise complex applications, partly due to the increased ability to utilize deep neural networks which can be used for precise and nonlinear black-box modeling. In process industry and especially advanced process control, system identification is used for similar purposes. However, the popularity of machine learning has resulted in fast development in the software (and also compatible hardware) that can be used to make more complex machine learning tasks feasible. Utilizing these resources in advanced process

control might result in noticeable improvements in control systems in the future.

One step towards the integration of modern machine learning and process industry applications was taken in the experimental part of this thesis, where a framework for developing and deploying machine learning applications was designed and implemented on a proof-of-concept level. The microservice architecture that was used, proved to be functional and advantageous, especially by ensuring the easy extensibility in terms of both the machine learning libraries used and the client applications using the deployed models.

The use of open source software as components of a machine learning framework gives a good head start in implementing new features. However, the drawbacks include the risk of accumulating technical dept: Especially small open source projects with permissive licenses tend to promise a lot, and even the documentation may look extensive, but eventually, if the software is not functional enough as it is, or the problems cannot be solved with the help of the documentation (or community), the lack of knowledge of the codebase causes significant overhead in resolving the issues.

Future work related to the framework architecture presented in this thesis, includes quantitative performance testing in real-world use cases. It is to be expected that the versatility of the framework causes some negative impact on the performance, compared to an application-specific tightly-coupled solution.

While most machine learning tasks can be done on a single computer given decent hardware, future work also includes making the framework compatible with big data, which is the term used when the resources of one computer are not enough to store or process all the data. Big data handling can be done with dedicated tools, the best known of which are probably in the Apache Hadoop ecosystem. When a cluster of resources is set up, the current implementation of the machine learning framework should be easily adaptable to using it. In some cases, also workarounds such as random or stratified subsampling of a large dataset, can be used to process big data with given memory limits. On the other hand, big data tools usually already implement similar procedures, along with streaming, mini-batch learning and online learning. The planning of big data compatibility should be started with a literature review, since studies related to big data architectures can be found quite easily, such as the article by Sarnovsky *et al.* [99], about a big data processing platform for process industry factories.

Bibliography

[1] Ge, Z., Song, Z., Ding, S.X. and Huang, B., Data Mining and Analytics in the Process Industry: The Role of Machine Learning, *IEEE Access* **5** (2017) 20590–20616, doi:10.1109/ACCESS.2017.2756872, URL `http://ieeexplore.ieee.org/document/8051033/`.

[2] Talonen, J., *Advances in Methods of Anomaly Detection and Visualization of Multivariate Data*, Doctoral dissertation, Aalto University, Espoo 2015, 124 p.

[3] Lee, J.H., Shin, J. and Realff, M.J., Machine learning: Overview of the recent progresses and implications for the process systems engineering field, *Computers & Chemical Engineering* **114** (2017) 111–121, doi:10.1016/j.compchemeng.2017.10.008.

[4] Mueller, J. and Massaron, L., *Machine learning for dummies*, For dummies, John Wiley & Sons, Inc, Hoboken, New Jersey 2016, ISBN 978-1-119-24551-3, 410 p.

[5] Markoff, J., Behind Artificial Intelligence, a Squadron of Bright Real People (2005), URL `https://www.nytimes.com/2005/10/14/technology/behind-artificial-intelligence-a-squadron-of-bright-real-people.html`, accessed 2018-03-25.

[6] Sato, K., Young, C. and Patterson, D., An in-depth look at Google's first Tensor Processing Unit (TPU) | Google Cloud Big Data and Machine Learning Blog, URL `https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu`, accessed 2018-03-14.

[7] Kornienko, A.A., Kornienko, A.V., Fofanov, O.B. and Chubik, M.P., Knowledge in Artificial Intelligence Systems: Searching the Strategies

for Application, *Procedia - Social and Behavioral Sciences* **166** (2015) 589–594, doi:10.1016/j.sbspro.2014.12.578.

[8] Thibodeau, P., Meet the virtual woman who may take your job (2015), URL https://www.computerworld.com/article/2990849/robotics/meet-the-virtual-woman-who-may-take-your-job.html, accessed 2018-03-14.

[9] Paine, J., AI Newsletter (2005), URL https://web.archive.org/web/20131109201636/http://www.ainewsletter.com/newsletters/aix_0501.htm#w, accessed 2018-03-25.

[10] Weathington, J., 40-year-old AI innovation may solve your big data problems, URL https://www.techrepublic.com/article/40-year-old-ai-innovation-may-solve-your-big-data-problems/, accessed 2018-03-16.

[11] Mizoguchi, R., Gofuku, A., Matsuura, Y., Sakashita, Y. and Tokunaga, M., Human media interface system for the next generation plant operation, in *IEEE SMC '99 Conference Proceedings*, IEEE, Tokyo, Japan 1999, pp. 630–635, URL http://ieeexplore.ieee.org/document/815625/.

[12] Mizoguchi, R. and Bourdeau, J., Using Ontological Engineering to Overcome Common AI-ED Problems, *International Journal of Artificial Intelligence in Education* **11** (2000) 107–121.

[13] Tommila, T., Hirvonen, J. and Pakonen, A., Fuzzy ontologies for retrieval of industrial knowledge, *VTT Working Papers* **153** (2010) 54.

[14] Sun, B., Jämsä-Jounela, S.L., Todorov, Y., Olivier, L.E. and Craig, I.K., Perspective for equipment automation in process industries, *IFAC-PapersOnLine* **50** (2) (2017) 65–70, doi:10.1016/j.ifacol.2017.12.012, URL http://linkinghub.elsevier.com/retrieve/pii/S2405896317335474.

[15] Wu, X., Chen, H., Wu, G., Liu, J., Zheng, Q., He, X., Zhou, A., Zhao, Z.Q., Wei, B., Li, Y., Zhang, Q. and Zhang, S., Knowledge Engineering with Big Data, *IEEE Intelligent Systems* **30** (5) (2015) 46–55, doi:10.1109/MIS.2015.56, URL http://ieeexplore.ieee.org/document/7155445/.

[16] Wagner, W.P., Trends in expert system development: A longitudinal content analysis of over thirty years of expert system case studies, *Expert Systems with Applications* **76** (2017) 85–96, doi:10.1016/j.eswa.2017.01.028, URL http://linkinghub.elsevier.com/retrieve/pii/S0957417417300386.

[17] Wang, J., Ma, Y., Zhang, L., Gao, R.X. and Wu, D., Deep learning for smart manufacturing: Methods and applications, *Journal of Manufacturing Systems* **48** (2018) 144–156, doi:10.1016/j.jmsy.2018.01.003, URL http://linkinghub.elsevier.com/retrieve/pii/S0278612518300037.

[18] Li, D., Perspective for smart factory in petrochemical industry, *Computers & Chemical Engineering* **91** (2016) 136–148, doi:10.1016/j.compchemeng.2016.03.006, URL http://linkinghub.elsevier.com/retrieve/pii/S009813541630059X.

[19] Puchr, I., Probabilistic advisory subsystem as a part of distributed control system of complex industrial process, in *Technical report no. DCSE/TR-2015-01*, University of West Bohemia, Pilsen 2015, 80 p.

[20] International Atomic Energy Agency, Operator support systems in nuclear power plants, in *Proceedings of a Specialists meeting*, Moscow, Russian Federation 1993, pp. 1–8.

[21] Uraikul, V., Chan, C.W. and Tontiwachwuthikul, P., Artificial intelligence for monitoring and supervisory control of process systems, *Engineering Applications of Artificial Intelligence* **20** (2) (2007) 115–131, doi:10.1016/j.engappai.2006.07.002.

[22] Thomas, M.C., Zhu, W. and Romagnoli, J.A., Data mining and clustering in chemical process databases for monitoring and knowledge discovery, *Journal of Process Control* **67** (2017) 160–175, doi:10.1016/j.jprocont.2017.02.006, URL http://linkinghub.elsevier.com/retrieve/pii/S095915241730032X.

[23] Hubmann, C., Becker, M., Althoff, D., Lenz, D. and Stiller, C., Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles, in *Intelligent Vehicles Symposium*, IEEE, Redondo Beach, CA, USA 2017, ISBN 978-1-5090-4804-5, pp. 1671–1678, doi:10.1109/IVS.2017.7995949, URL http://ieeexplore.ieee.org/document/7995949/.

[24] Bonnin, R., *Machine learning for developers*, Packt Publishing Ltd., Birmingham 2017, ISBN 978-1-78646-696-9, 247 p.

[25] Anonymous, Statsoft, Partial Least Squares (PLS), URL `http://www.statsoft.com/Textbook/Partial-Least-Squares`, accessed 2018-03-26.

[26] Chapelle, O., Schölkopf, B. and Zien, A., *Semi-supervised learning*, The MIT Press, Cambridge, Massachusetts, London, England 2010, ISBN 978-0-262-51412-5, 518 p.

[27] Thorström, M., *Applying machine learning to key performance indicators*, Master's thesis, University of Gothenburg, Gothenburg, Sweden 2017, 64 p., URL `http://publications.lib.chalmers.se/records/fulltext/250254/250254.pdf`.

[28] Techerin, Semi-supervised learning (2018), URL `https://en.wikipedia.org/w/index.php?title=Semi-supervised_learning&oldid=822065999#/media/File:Example_of_unlabeled_data_in_semisupervised_learning.png`, accessed 2018-03-21.

[29] Silver, D., RL Course by David Silver - Lecture 1: Introduction to Reinforcement Learning. University College London, URL `https://www.youtube.com/watch?v=2pWv7GOvuf0`, accessed 2018-03-21.

[30] Wubbolt, C. and Patterson, J.T., Considerations for Validation of Manufacturing Execution Systems, *Journal of Validation Technology* **1** (2012) 80–84.

[31] Piiparinen, E. and Tilja, H., *Raaka-ainetoimittajien toimitusvarmuus ja -kyky*, Bachelor's thesis, Lapland University of Applied Sciences, Tornio 2015, 46 p.

[32] Littlefield, M., What Is Manufacturing Operations Management?, URL `http://blog.lnsresearch.com/bid/115598/What-Is-Manufacturing-Operations-Management`, accessed 2018-03-23.

[33] Rengasamy, R., *A framework for integrating process monitoring, diagnosis and supervisory control*, Doctoral dissertation, Purdue University 1995, 142 p.

[34] Thirumarimurugan, M., Bagyalakshmi, N. and Paarkavi, P., Comparison of fault detection and isolation methods: A review, in *Intelligent Systems and Control (ISCO), 2016 10th International Conference*, IEEE, Coimbatore, India 2016, ISBN 978-1-4673-7807-9, pp.

1–6, doi:10.1109/ISCO.2016.7726957, URL http://ieeexplore.ieee.org/document/7726957/.

[35] Chiang, L.H., Braatz, R.D. and Russell, E., *Fault detection and diagnosis in industrial systems*, Advanced textbooks in control and signal processing, Springer, London 2001, ISBN 978-1-85233-327-0, 279 p.

[36] Bandaru, S., Ng, A.H. and Deb, K., Data mining methods for knowledge discovery in multi-objective optimization: Part A - Survey, *Expert Systems with Applications* **70** (2017) 139–159, doi:10.1016/j.eswa.2016.10.015, URL http://linkinghub.elsevier.com/retrieve/pii/S0957417416305449.

[37] Powell, V., Principal Component Analysis explained visually, URL http://setosa.io/ev/principal-component-analysis/, accessed 2018-03-26.

[38] Abdi, H., Partial least squares regression and projection on latent structure regression (PLS Regression), *Wiley Interdisciplinary Reviews: Computational Statistics* **2** (1) (2010) 97–106, doi:10.1002/wics.51, URL http://doi.wiley.com/10.1002/wics.51.

[39] Wang, Z.X., He, Q.P. and Wang, J., Comparison of variable selection methods for PLS-based soft sensor modeling, *Journal of Process Control* **26** (2015) 56–72, doi:10.1016/j.jprocont.2015.01.003, URL http://linkinghub.elsevier.com/retrieve/pii/S0959152415000050.

[40] Bro, R., Partial Least Squares Regression, URL https://www.youtube.com/watch?v=AxmqUKYeD-U, accessed 2018-03-26.

[41] Jämsä-Jounela, S.L., Zakharov, A. and Jain, T., Partial least squares, in *Lecture presentation, KE-90.5120 Process Monitoring*, Aalto University, Espoo 2015, 46 p.

[42] Ding, S.X., *Model-based fault diagnosis techniques: design schemes, algorithms and tools*, 2nd edn., Advances in industrial control, Springer, New York 2013, ISBN 978-1-4471-4798-5, 501 p.

[43] Ruusunen, M. and Paavola, M., Quality Monitoring and Fault Detection in an Automated Manufacturing System - a Soft Computing Approach, in *Report A No 19, Control Engineering Laboratory, University of Oulu*, Oulu 2002, 33 p.

[44] Ogunnaike, B.A. and Ray, W.H., *Process dynamics, modeling, and control*, Topics in chemical engineering, Oxford University Press, New York 1994, ISBN 978-0-19-509119-9, 1260 p.

[45] Ljung, L., Hjalmarsson, H. and Ohlsson, H., Four Encounters with System Identification, *European Journal of Control* **17** (5-6) (2011) 449–471, doi:10.3166/ejc.17.449-471, URL `http://linkinghub.elsevier.com/retrieve/pii/S0947358011709712`.

[46] Pillonetto, G., The interplay between system identification and machine learning, *ArXiv e-prints* **1612.09158** (2016) 16, URL `https://arxiv.org/abs/1612.09158`.

[47] Rajeswaran, A., Machine Learning vs System Identification? (2015), URL `https://cs.stackexchange.com/q/48947`, accessed 2018-03-27.

[48] Tsai, P.F., Chu, J.Z., Jang, S.S. and Shieh, S.S., Developing a robust model predictive control architecture through regional knowledge analysis of artificial neural networks, *Journal of Process Control* **13** (5) (2003) 423–435, doi:10.1016/S0959-1524(02)00067-7.

[49] Ogunmolu, O., Gu, X., Jiang, S. and Gans, N., Nonlinear Systems Identification Using Deep Dynamic Neural Networks, *ArXiv e-prints* **1610.01439** (2016) 8, URL `http://arxiv.org/abs/1610.01439`.

[50] Kim, D., Lin, Y., Han, S.C., Kang, B.H. and Lee, S., RDR-based Knowledge Based System to the Failure Detection in Industrial Cyber Physical Systems, *Knowledge-Based Systems* **150** (2018) 1–13, doi:10.1016/j.knosys.2018.02.009, URL `http://linkinghub.elsevier.com/retrieve/pii/S0950705118300601`.

[51] Joshi, K., Management Information Systems, Course material, University of Missouri–St. Louis, 1998, URL `http://www.umsl.edu/~joshik/msis480/chapt11.htm,http://www.umsl.edu/~joshik/msis480/chapt16.htm`, accessed 2018-03-21.

[52] Boaye Belle, A., Lethbridge, T.C., Garzón, M. and Adesina, O.O., Design and implementation of distributed expert systems: On a control strategy to manage the execution flow of rule activation, *Expert Systems with Applications* **96** (2018) 129–148, doi:10.1016/j.eswa.2017.11.033, URL `http://linkinghub.elsevier.com/retrieve/pii/S0957417417307911`.

[53] Anonymous, Drools, Business Rules Management System (Java, Open Source), URL http://www.drools.org/, accessed 2018-08-28.

[54] Sadik, A.R., *Design and Implementation of an Expert System for Monitoring and Management of Web-Based Industrial Applications*, Master's thesis, Tampere University of Technology, Tampere 2013, 91 p.

[55] Goja, A., Man, Marriage and Machine – Adventures in Artificial Advice, URL https://www.codeproject.com/Articles/179375/Man-Marriage-and-Machine-Adventures-in-Artificia, accessed 2018-03-28.

[56] Gavrilov, A.V., Hybrid Intelligent Systems, Lecture series, URL http://insycom.ru/html/teaching/2007/his.htm, accessed 2018-03-16.

[57] Bose, B., Expert system, fuzzy logic, and neural network applications in power electronics and motion control, *Proceedings of the IEEE* **82** (8) (1994) 1303–1323, doi:10.1109/5.301690, URL http://ieeexplore.ieee.org/document/301690/.

[58] Sahin, S., Tolun, M. and Hassanpour, R., Hybrid expert systems: A survey of current approaches and applications, *Expert Systems with Applications* **39** (4) (2012) 4609–4617, doi:10.1016/j.eswa.2011.08.130, URL http://linkinghub.elsevier.com/retrieve/pii/S0957417411012619.

[59] Kurzyn, M., Expert systems and neural networks: a comparison, in *IEEE Comput. Soc. Press*, IEEE, Dunedin, New Zealand 1993, pp. 222–223, doi:10.1109/ANNES.1993.323038, URL http://ieeexplore.ieee.org/document/323038/.

[60] Yunwu, W., Using Fuzzy Expert System Based on Genetic Algorithms for Intrusion Detection System, in *International Forum on Information Technology and Applications*, IEEE, Chengdu, China 2009, ISBN 978-0-7695-3600-2, pp. 221–224, doi:10.1109/IFITA.2009.107, URL http://ieeexplore.ieee.org/document/5231228/.

[61] Sreedevi, E. and Padmavathamma, M., A rule based neuro-fuzzy expert system model for diagnosis of diabetes, *International Journal of Advanced Research in Computer Science* **5** (8) (2014) 236–239.

[62] Determan, J.C. and Foster, J.A., A Genetic Algorithm for Expert System Rule Generation, in *Genetic and Evolutionary Computation Conference*, San Francisco, California 2001, 39 p.

[63] Fung, C. C., L., Intelligent Systems, Lecture series, Murdoch University (2003), URL http://ftp.it.murdoch.edu.au/units/ICT219/Lectures/.

[64] Kurnaz, S., Cetin, O. and Kaynak, O., Adaptive neuro-fuzzy inference system based autonomous flight control of unmanned air vehicles, *Expert Systems with Applications* **37** (2) (2010) 1229–1234, doi:10.1016/j.eswa.2009.06.009.

[65] Khoshnevisan, B., Rafiee, S., Omid, M. and Mousazadeh, H., Development of an intelligent system based on ANFIS for predicting wheat grain yield on the basis of energy inputs, *Information Processing in Agriculture* **1** (1) (2014) 14–22, doi:10.1016/j.inpa.2014.04.001, URL http://linkinghub.elsevier.com/retrieve/pii/S2214317314000031.

[66] Yusof, N., Ahmad, N.B., Othman, M.S. and Nyen, Y.C., *A Concise Fuzzy Rule Base to Reason Student Performance Based on Rough-Fuzzy Approach, Fuzzy Inference System - Theory and Applications*, InTech 2012, ISBN 978-953-51-0525-1, 504 p.

[67] Mostafaei, M., ANFIS models for prediction of biodiesel fuels cetane number using desirability function, *Fuel* **216** (2018) 665–672, doi:10.1016/j.fuel.2017.12.025, URL http://linkinghub.elsevier.com/retrieve/pii/S0016236117315958.

[68] Zare, M. and Koch, M., Groundwater level fluctuations simulation and prediction by ANFIS- and hybrid Wavelet-ANFIS/Fuzzy C-Means (FCM) clustering models: Application to the Miandarband plain, *Journal of Hydro-environment Research* **18** (2018) 63–76, doi:10.1016/j.jher.2017.11.004, URL http://linkinghub.elsevier.com/retrieve/pii/S1570644316303926.

[69] Kassem, Y., Çamur, H. and Bennur, K.E., Adaptive Neuro-Fuzzy Inference System (ANFIS) and Artificial Neural Network (ANN) for Predicting the Kinematic Viscosity and Density of Biodiesel-Petroleum Diesel Blends, *American Journal of Computer Science and Technology* **1** (1) (2018) 8–18, doi:10.11648/j.ajcst.20180101.12.

[70] Ouyang, C.S., Lee, W.J. and Lee, S.J., A TSK-Type Neurofuzzy Network Approach to System Modeling Problems, *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)* **35** (4) (2005) 751–767, doi:10.1109/TSMCB.2005.846000, URL http://ieeexplore.ieee.org/document/1468248/.

[71] Anonymous, Tutorialspoint, Artificial Intelligence Quick Guide, URL https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_quick_guide.htm, accessed 2018-04-11.

[72] Rudowsky, I., Intelligent agents, in *Proceedings of the Americas Conference on Information Systems*, New York 2004, 8 p.

[73] Calderwood, S., Liu, W., Hong, J. and Loughlin, M., An Architecture of a Multi-Agent System for SCADA - Dealing With Uncertainty, Plans and Actions:, in *Proceedings of the 10th International Conference on Informatics in Control, Automation and Robotics*, vol. 1, SCITEPRESS Digital Library 2013, pp. 300–306, doi:10.5220/0004594603000306.

[74] Koumboulis, F.N. and Tzamtzi, M.P., Automation agents embedded in industrial decision support systems, in *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, vol. 1, IEEE, Vienna, Austria 2005, pp. 51–57.

[75] Gofuku, A. and Sato, T., Dynamic operation permission system for oil refinery plants, *International Journal of Intelligent Control and Systems* **14** (2) (2009) 149–157, doi:10.1109/ICNSC.2009.4919371, URL http://ieeexplore.ieee.org/document/4919371/.

[76] Miettinen, K., *Nonlinear multiobjective optimization*, no. 12 in International series in operations research & management science, Kluwer Academic Publishers, Boston 1999, ISBN 978-0-7923-8278-2, 298 p.

[77] Sugimura, K., Obayashi, S. and Jeong, S., Multi-objective optimization and design rule mining for an aerodynamically efficient and stable centrifugal impeller with a vaned diffuser, *Engineering Optimization* **42** (3) (2010) 271–293, doi:10.1080/03052150903171084, URL http://www.tandfonline.com/doi/abs/10.1080/03052150903171084.

[78] Chun-Wei Seah, Ong, Y.S., Tsang, I.W. and Siwei Jiang, Pareto Rank Learning in Multi-objective Evolutionary Algorithms, in *World Congress on Computational Intelligence*, IEEE, Brisbane, QLD, Australia 2012, pp. 1–8, doi:10.1109/CEC.2012.6252865, URL http://ieeexplore.ieee.org/document/6252865/.

[79] Bandaru, S., Ng, A.H. and Deb, K., Data mining methods for knowledge discovery in multi-objective optimization: Part B - New developments and applications, *Expert Systems with Applications* **70** (2017)

119–138, doi:10.1016/j.eswa.2016.10.016, URL http://linkinghub.elsevier.com/retrieve/pii/S0957417416305474.

[80] Wang, H., ReNN: Rule-embedded Neural Networks, *ArXiv e-prints* **1801.09856** (2018) 6, URL https://arxiv.org/abs/1801.09856.

[81] Longo, F., Nicoletti, L. and Padovano, A., Smart operators in industry 4.0: A human-centered approach to enhance operators' capabilities and competencies within the new smart factory context, *Computers & Industrial Engineering* **113** (2017) 144–159, doi:10.1016/j.cie.2017.09.016, URL http://linkinghub.elsevier.com/retrieve/pii/S0360835217304291.

[82] Anonymous, Aalto-University, Johdanto automaatioon, Automaatiojärjestelmien perusteet, Automaatio 1, Image source: Valmet Automation Oy. Course material (2015).

[83] Korbicz, J. and Kościelny, J.M. (eds.) *Modeling, Diagnostics and Process Control*, Springer, Berlin, Heidelberg 2011, ISBN 978-3-642-16653-2, 384 p., doi:10.1007/978-3-642-16653-2.

[84] Lawryńczuk, M., *Computationally Efficient Model Predictive Control Algorithms*, *Studies in Systems, Decision and Control*, vol. 3, Springer International Publishing, Cham 2014, ISBN 978-3-319-04228-2, 316 p., doi:10.1007/978-3-319-04229-9.

[85] Amirkhani, S., Nasirivatan, S., Kasaeian, A. and Hajinezhad, A., ANN and ANFIS models to predict the performance of solar chimney power plants, *Renewable Energy* **83** (2015) 597–607, doi:10.1016/j.renene.2015.04.072, URL http://linkinghub.elsevier.com/retrieve/pii/S0960148115003614.

[86] Agachi, P.S. (ed.) *Model based control: case studies in process engineering*, Wiley-VCH, Weinheim 2006, ISBN 978-3-527-31545-1, 277 p.

[87] Syfert, M., Wnuk, P. and Kościelny, J.M., Introduction to DiaSter–Intelligent System for Diagnostics and Automatic Control Support, in *Mechatronics*, Springer 2011, pp. 385–393.

[88] Puchr, I. and Herout, P., Probabilistic advisory system for operators can help with diagnostics of rolling mills, in *21st International Conference on Process Control*, IEEE, Štrbské Pleso, Slovakia 2017, pp. 132–136.

[89] Lee, S.J., Mo, K. and Seong, P.H., Development of an Integrated Decision Support System to Aid the Cognitive Activities of Operators in Main Control Rooms of Nuclear Power Plants, in *Symposium on Computational Intelligence in Multicriteria Decision Making*, IEEE, Honolulu, USA 2007, pp. 146–152, doi:10.1109/MCDM.2007.369429, URL `http://ieeexplore.ieee.org/document/4222995/`.

[90] Lee', S.J. and Seong, P.H., Theoretical and Experimental Impact Analysis of Decision Support Systems for Advanced MGR Operators, in *Proceedings of the International Youth Nuclear Congress*, Interlaken, Switzerland 2008, 12 p.

[91] Ayodeji, A., Liu, Y.k. and Xia, H., Knowledge base operator support system for nuclear power plant fault diagnosis, *Progress in Nuclear Energy* **105** (2018) 42–50, doi:10.1016/j.pnucene.2017.12.013, URL `http://linkinghub.elsevier.com/retrieve/pii/S0149197017303190`.

[92] Vinod, S.G., Babar, A., Kushwaha, H. and Venkat Raj, V., Symptom based diagnostic system for nuclear power plant operations using artificial neural networks, *Reliability Engineering & System Safety* **82** (1) (2003) 33–40, doi:10.1016/S0951-8320(03)00120-0, URL `http://linkinghub.elsevier.com/retrieve/pii/S0951832003001200`.

[93] Du, Y., Tyagi, R. and Bhamidimarri, R., Use of fuzzy neural-net model for rule generation of activated sludge process, *Process Biochemistry* **35** (1-2) (1999) 77–83, doi:10.1016/S0032-9592(99)00035-7, URL `http://linkinghub.elsevier.com/retrieve/pii/S0032959299000357`.

[94] Rozanski, N. and Woods, E., *Software systems architecture: working with stakeholders using viewpoints and perspectives*, Addison-Wesley, Upper Saddle River 2005, ISBN 978-0-321-11229-3, 546 p.

[95] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.F. and Dennison, D., Hidden Technical Debt in Machine Learning Systems, in *NIPS 2015 Workshop*, Montreal, Canada 2015, 9 p., URL `https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf`.

[96] Landset, S., Khoshgoftaar, T.M., Richter, A.N. and Hasanin, T., A survey of open source tools for machine learning with big data in the Hadoop ecosystem, *Journal of Big Data* **2** (1) (2015) 36, doi:10.1186/s40537-015-0032-1, URL `http://www.journalofbigdata.com/content/2/1/24`.

[97] Roman, J., The Hadoop Ecosystem Table, URL https://hadoopecosystemtable.github.io/, accessed 2018-08-01.

[98] Bekker, A., Spark vs. Hadoop MapReduce: Which big data framework to choose, URL https://www.scnsoft.com/blog/spark-vs-hadoop-mapreduce, accessed 2018-08-02.

[99] Sarnovsky, M., Bednar, P. and Smatana, M., Big Data Processing and Analytics Platform Architecture for Process Industry Factories, *Big Data and Cognitive Computing* **2** (1) (2018) 3, doi:10.3390/bdcc2010003, URL http://www.mdpi.com/2504-2289/2/1/3.

[100] Anonymous, Flink, Apache Flink 1.5 Documentation: FlinkML - Machine Learning for Flink, URL https://ci.apache.org/projects/flink/flink-docs-release-1.5/dev/libs/ml/, accessed 2018-08-03.

[101] Mushketyk, I., Apache Flink vs. Apache Spark (2017), URL https://brewing.codes/2017/09/25/flink-vs-spark/, accessed 2018-08-02.

[102] Robinson, J., Introduction to H2O.ai (2018), URL https://medium.com/@jamal.robinson/introduction-to-h2o-ai-1ba51a884f02, accessed 2018-08-07.

[103] Bartczuk, K., How does H2O compare to TensorFlow? - Quora (2017), URL https://www.quora.com/How-does-H2O-compare-to-TensorFlow, accessed 2018-08-07.

[104] Anonymous, R Foundation, T., R: What is R?, URL https://www.r-project.org/about.html, accessed 2018-08-03.

[105] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A. and Cournapeau, D., Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research* **12** (2011) 6.

[106] Xie, Z., Ye, Z., Jones, S., Roggenburg, M., Root, C., Prasutchai, T. and Lanham, M.A., Caret Versus Scikit-learn A Comparison of Data Science Tools (2018), URL http://matthewalanham.com/Students/2018_PURC_caretvsscikit.pdf, accessed 2018-08-06.

[107] Berhane, F., Machine Learning with Python and R-Scikit-learn vs Caret-part1, URL http://datascience-enthusiast.com/R/ML_python_R_part1.html, accessed 2018-08-06.

[108] Hefele, C., Pros and Cons of R vs Python Scikit learn | Kaggle, URL https://www.kaggle.com/getting-started/5243, accessed 2018-08-06.

[109] Saunders, A.A., A Comparison of Deep Learning Frameworks (2018), URL https://www.exastax.com/deep-learning/a-comparison-of-deep-learning-frameworks/, accessed 2018-08-06.

[110] Rubashkin, M., Getting Started with Deep Learning (2017), URL https://www.svds.com/getting-started-deep-learning/, accessed 2018-08-06.

[111] Ahmed, A., Choosing a Machine Learning Framework in 2018 (2018), URL https://agi.io/2018/02/09/survey-machine-learning-frameworks/, accessed 2018-08-06.

[112] Anonymous, TensorFlow, Distributed TensorFlow, URL https://www.tensorflow.org/deploy/distributed, accessed 2018-08-08.

[113] Anonymous, PyTorch, Distributed communication package - torch.distributed. PyTorch documentation, URL https://pytorch.org/docs/master/distributed.html#module-torch.distributed, accessed 2018-08-08.

[114] Anonymous, Blog, C., Caffe2 and PyTorch join forces to create PyTorch 1.0 (2018), URL http://caffe2.ai/blog/2018/05/02/Caffe2_PyTorch_1_0.html, accessed 2018-08-08.

[115] Melentyev, A., Machine Learning Engineering - Model interoperability (2017), URL https://www.andrey-melentyev.com/model-interoperability.html, accessed 2018-08-08.

[116] Anonymous, Scikit-learn, 3.4. Model persistence — scikit-learn 0.19.2 documentation, URL http://scikit-learn.org/stable/modules/model_persistence.html, accessed 2018-08-09.

[117] Anonymous, GitHub, ONNX Support · Issue #12888 · tensorflow/tensorflow, URL https://github.com/tensorflow/tensorflow/issues/12888, accessed 2018-08-10.

[118] Anonymous, DeepDetect, Deep Learning API and Server in C++11 with Python bindings and support for Caffe, Tensorflow, XGBoost and TSNE (2018), URL https://github.com/jolibrain/deepdetect, accessed 2018-08-26.

[119] Vikati, A., The Rise of the Model Servers (2018), URL https://medium.com/@vikati/the-rise-of-the-model-servers-9395522b6c58, accessed 2018-08-09.

[120] Olston, C., Fiedel, N. and Gorovoy, K., TensorFlow-Serving: Flexible, High-Performance ML Serving, in *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA 2017, 8 p.

[121] Crankshaw, D., Wang, X., Zhou, G., Franklin, M.J., Gonzalez, J.E. and Stoica, I., Clipper: A Low-Latency Online Prediction Serving System, in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*, Boston, MA, USA 2017, 17 p.

[122] Anonymous, Clipper.ai, Basic Concepts :: Clipper, URL http://clipper.ai/tutorials/basic_concepts/, accessed 2018-08-12.

[123] Keith, M., Azure Machine Learning Studio: Tune Model Hyperparameters, URL https://www.youtube.com/watch?v=1D2AB_kRmWU, accessed 2018-08-13.

[124] Anonymous, Pothosware, Welcome to Pothosware, URL http://www.pothosware.com/, accessed 2018-08-13.

[125] Gould, S., drwn: Darwin: A Framework for Machine Learning Research and Development (2018), URL https://github.com/sgould/drwn, accessed 2018-08-13.

[126] Mané, D., Hands-on TensorBoard (TensorFlow Dev Summit 2017), URL https://www.youtube.com/watch?v=eBbEDRsCmv4, accessed 2018-08-13.

[127] Spann, T., Using HiveQL Processors in Apache NiFi 1.2 - Hortonworks, URL https://community.hortonworks.com/articles/45706/using-the-new-hiveql-processors-in-apache-nifi-070.html, accessed 2018-08-13.

[128] Wang, A., Intro to R, Python, Flow at H2O World 2015, URL https://www.youtube.com/watch?v=LtZiK6iCJvE, accessed 2018-08-13.

[129] Perez, F., Project Jupyter, URL https://speakerdeck.com/fperez/project-jupyter, accessed 2018-08-13.

[130] Parente, P., Notebooks — From Doodles to Dashboards: Notebooks as a Cloud Platform, URL https://ibm-et.github.io/defrag2015/?full#notebook, accessed 2018-08-14.

[131] Anonymous, Jupyter, Project homepage - Widgets, URL http://www.jupyter.org, accessed 2018-08-14.

[132] Anonymous, Jupyter, Widget List - 7.4.0 documentation, URL https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html, accessed 2018-08-14.

[133] Anonymous, Jupyter, Markdown Cells — Jupyter Notebook 5.6.0 documentation, URL http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html, accessed 2018-08-14.

[134] Anonymous, Jupyter, How IPython and Jupyter Notebook work — Jupyter Documentation 4.1.1 alpha documentation, URL https://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html, accessed 2018-08-14.

[135] Krochmalski, J., *Docker and Kubernetes for Java Developers*, Packt Publishing, Birmingham 2017, ISBN 978-1-78646-839-0, 302 p.

[136] Grubor, S., *Deployment with Docker*, Packt Publishing Ltd., Birmingham 2017, ISBN 978-1-78646-900-7, 272 p.

[137] Miller, C., Nagy, Z. and Schlueter, A., A review of unsupervised statistical learning and visual analytics techniques applied to performance analysis of non-residential buildings, *Renewable and Sustainable Energy Reviews* **81** (2018) 1365–1377, doi:10.1016/j.rser.2017.05.124, URL http://linkinghub.elsevier.com/retrieve/pii/S1364032117307608.

[138] Mueller, J. and Massaron, L., Choosing the right algorithm for machine learning, URL http://www.dummies.com/programming/big-data/data-science/machine-learning-dummies-cheat-sheet/, accessed 2018-03-20.

[139] Anonymous, Scikit-learn, Scikit-learn 0.19.2 documentation, URL http://scikit-learn.org/stable/, accessed 2018-08-03.

Appendix A

Key technologies of smart petrochemical manufacturing

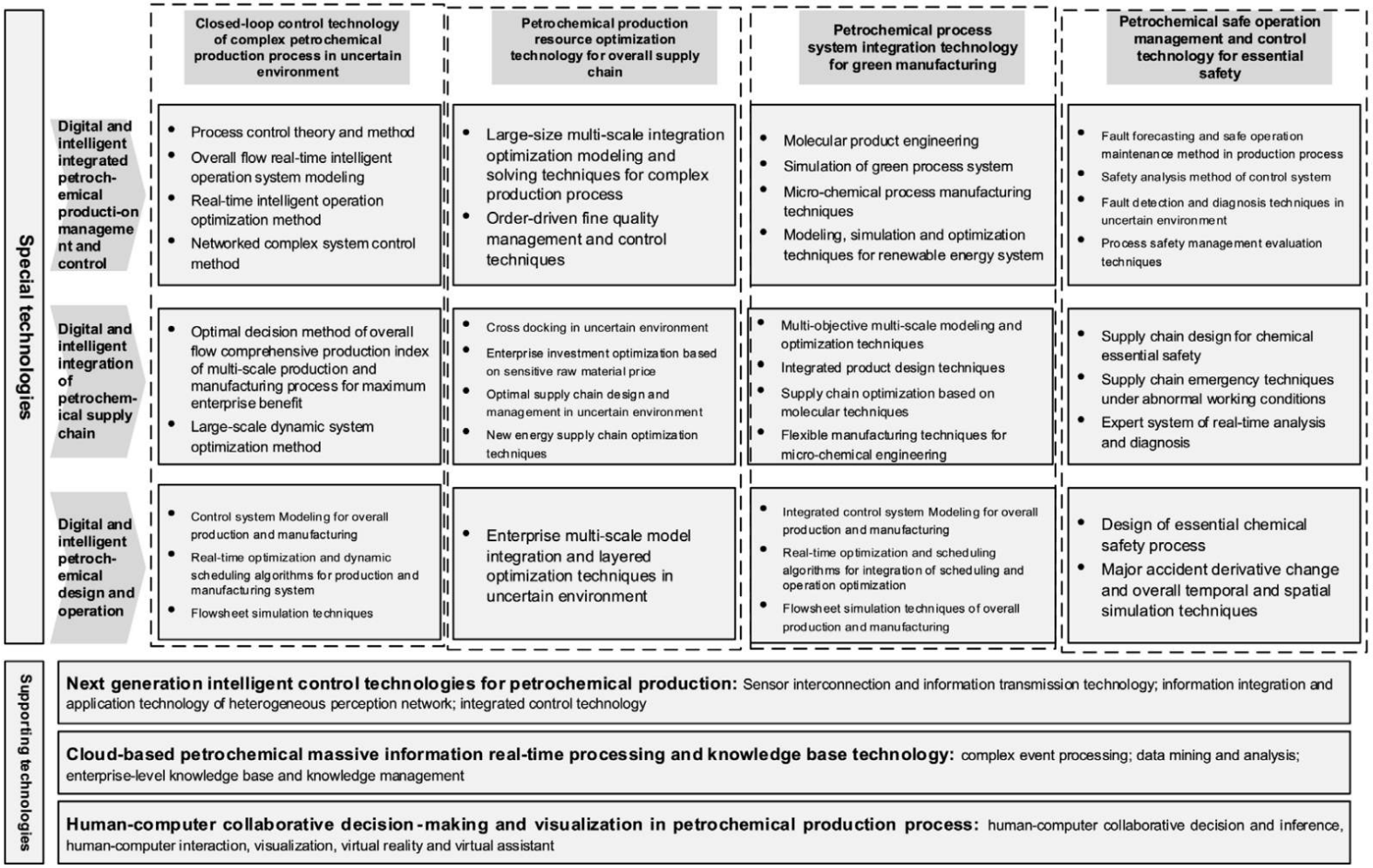


Figure A.1: Key technologies of smart petrochemical manufacturing [18].

Appendix B

Comparison of machine learning algorithms

Table B.1: A summary of the characteristics of selected machine learning algorithms [138].

Algorithm	**Used in**	**Pros**	**Cons**
Adaboost	Face detection	Automatically handles missing values. No need to transform any variable. Does not overfit easily. Only few parameters to tweak. Can leverage many different weak-learners.	Sensitive to noisy data and outliers. Never the best in class predictions.
Gradient Boosting	At almost any machine learning problem. Search engines (solving the problem of learning to rank).	Can approximate most nonlinear functions. Best in class predictor. Automatically handles missing values. No need to transform any variable.	May overfit if run for too many iterations. Sensitive to noisy data and outliers. Does not work well without parameter tuning.

K-means	Segmentation	Fast in finding clusters. Can detect outliers in multiple dimensions.	Suffers from multicollinearity. Clusters are spherical, cannot detect groups of other shape. Unstable solutions, depends on initialization.
K-nearest Neighbors	Computer vision. Multilabel tagging. Recommender systems. Spell checking tasks	Fast, lazy training. Can naturally handle extreme multiclass problems (like tagging text).	Slow and cumbersome in the predicting phase. Can fail to predict correctly due to the curse of dimensionality.
Linear regression	Baseline predictions. Econometric predictions. Modelling marketing responses.	Simple to understand and explain. Seldom overfits. Using L1 and L2 regularization is effective in feature selection. Fast to train. Easy to train on big data thanks to its stochastic version.	Hard to make it fit nonlinear functions. Can suffer from outliers.
Logistic regression	Ordering results by probability. Modelling marketing responses.	Simple to understand and explain. It seldom overfits. Using L1 and L2 regularization is effective in feature selection. The best algorithm for predicting probabilities of an event. Fast to train. Easy to train on big data thanks to its stochastic version.	Hard to make it fit nonlinear functions. Can suffer from outliers.

Naive Bayes	Face recognition. Sentiment analysis. Spam detection. Text classification.	Easy and fast to implement, doesn't require too much memory and can be used for online learning. Easy to understand. Takes into account prior knowledge.	Strong and unrealistic feature independence assumptions. Fails estimating rare occurrences. Suffers from irrelevant features.
Neural Networks	Image recognition. Language recognition and translation. Speech recognition. Vision recognition.	Can approximate any nonlinear function. Robust to outliers. Works only with a portion of the examples (the support vectors).	Very difficult to set up. Difficult to tune because of too many parameters, one of which is the topology of the network. Difficult to interpret. Easy to overfit.
PCA	Removing collinearity. Reducing dimensions of the dataset.	Can reduce data dimensionality.	Implies strong linear assumptions (components are a weighted summations of features).
Support Vector Machines	Character recognition. Image recognition. Text classification.	Automatic nonlinear feature creation. Can approximate complex nonlinear functions.	Difficult to interpret when applying nonlinear kernels. Suffers from too many examples, after 10 000 examples it starts taking too long to train.
SVD	Recommender systems	Can restructure data in a meaningful way.	Difficult to understand why data has been restructured in a certain way.
Random Forest	At almost any machine learning problem. Bioinformatics.	Can work in parallel. Seldom overfits. Automatically handles missing values. No need to transform any variable. No need to tweak parameters. Can be used by almost anyone with excellent results.	Difficult to interpret. Weaker on regression when estimating values at the extremities of the distribution of response values. Biased in multiclass problems toward more frequent classes.

Appendix C

Rules generated in a neuro-fuzzy system (case study)

Case study in section 5.4.

Table C.1: The IF-THEN rules generated by the neuro-fuzzy system. H, N, or L means that the respective fuzzy membership function is > 0.9. Otherwise, for example in Case 20, u can be interpreted as *a bit low* since $\mu_{L,u} = 0.75$. [93]

Case	**IF** q **and**	**IF** u **and**	**IF** α **and**	**THEN** θ
1	L	L	L	L (0.81 0.17 0.00)
5	L	L	L (0.60 0.32 0.00)	N
9	L	L (0.75 0.21 0.00)	L	L (0.89 0.12 0.00)
14	L	L (0.75 0.21 0.00)	N (0.32 0.60 0.00)	N
20	L	L (0.75 0.21 0.00)	H (0.00 0.13 0.88)	H
21	L	N	L (0.88 0.13 0.00)	L (0.68 0.26 0.00)
22	L	N	N (0.41 0.50 0.00)	N (0.14 0.85 0.00)
26	L	N	H	H (0.00 0.14 0.85)
27	L	N (0.00 0.61 0.32)	L	L (0.84 0.15 0.00)
29	L	N (0.00 0.61 0.32)	L (0.72 0.23 0.00)	N
37	L	N (0.00 0.61 0.32)	N (0.00 0.48 0.43)	H
40	L	H	L	L
56	L	H	N (0.40 0.87 0.00)	H (0.00 0.30 0.63)
59	N (0.23 0.73 0.00)	L	L	L
63	N (0.23 0.73 0.00)	L	L (0.66 0.27 0.00)	N (0.15 0.85 0.00)
65	N (0.23 0.73 0.00)	L	L (0.51 0.4 0.00)	N
77	N (0.23 0.73 0.00)	L	H	H (0.00 0.18 0.80)
78	N (0.23 0.73 0.00)	L (0.67 0.27 0.00)	L	L (0.87 0.13 0.00)
83	N (0.23 0.73 0.00)	L (0.67 0.27 0.00)	N (0.38 0.53 0.00)	N
95	N (0.23 0.73 0.00)	N	H	H (0.00 0.41 0.50)
97	N (0.23 0.73 0.00)	N	L	L
107	N (0.23 0.73 0.00)	N	L (0.62 0.31 0.00)	N
118	N (0.23 0.73 0.00)	N (0.00 0.61 0.32)	L	L (0.91 0.11 0.00)
132	N (0.23 0.73 0.00)	N (0.00 0.61 0.32)	N (0.00 0.50 0.40)	N (0.12 0.90 0.00)
135	N (0.23 0.73 0.00)	H	L	L (0.87 0.14 0.00)
145	N (0.23 0.73 0.00)	H	H (0.00 0.18 0.80)	N (0.36 0.55 0.00)
148	N (0.00 0.66 0.27)	L (0.75 0.27 0.00)	L (0.85 0.14 0.00)	L (0.60 0.32 0.00)
155	N (0.00 0.66 0.27)	L (0.75 0.21 0.00)	N (0.00 0.72 0.23)	N
157	N (0.00 0.66 0.27)	L (0.75 0.21 0.00)	H	H (0.00 0.19 0.78)
171	N (0.00 0.66 0.27)	N (0.00 0.61 0.32)	H	L
190	H	L	L	L
208	H	L	N	N (0.17 0.81 0.00)
220	H	L	H	H (0.00 0.22 0.74)
222	H	L (0.75 0.21 0.00)	L	L
236	H	L (0.75 0.21 0.00)	N (0.00 0.49 0.42)	N
243	H	L (0.75 0.21 0.00)	H	N (0.00 0.67 0.27)
259	H	N	H	H (0.00 0.13 0.88)

Appendix D

Scikit-learn: choosing a machine learning algorithm

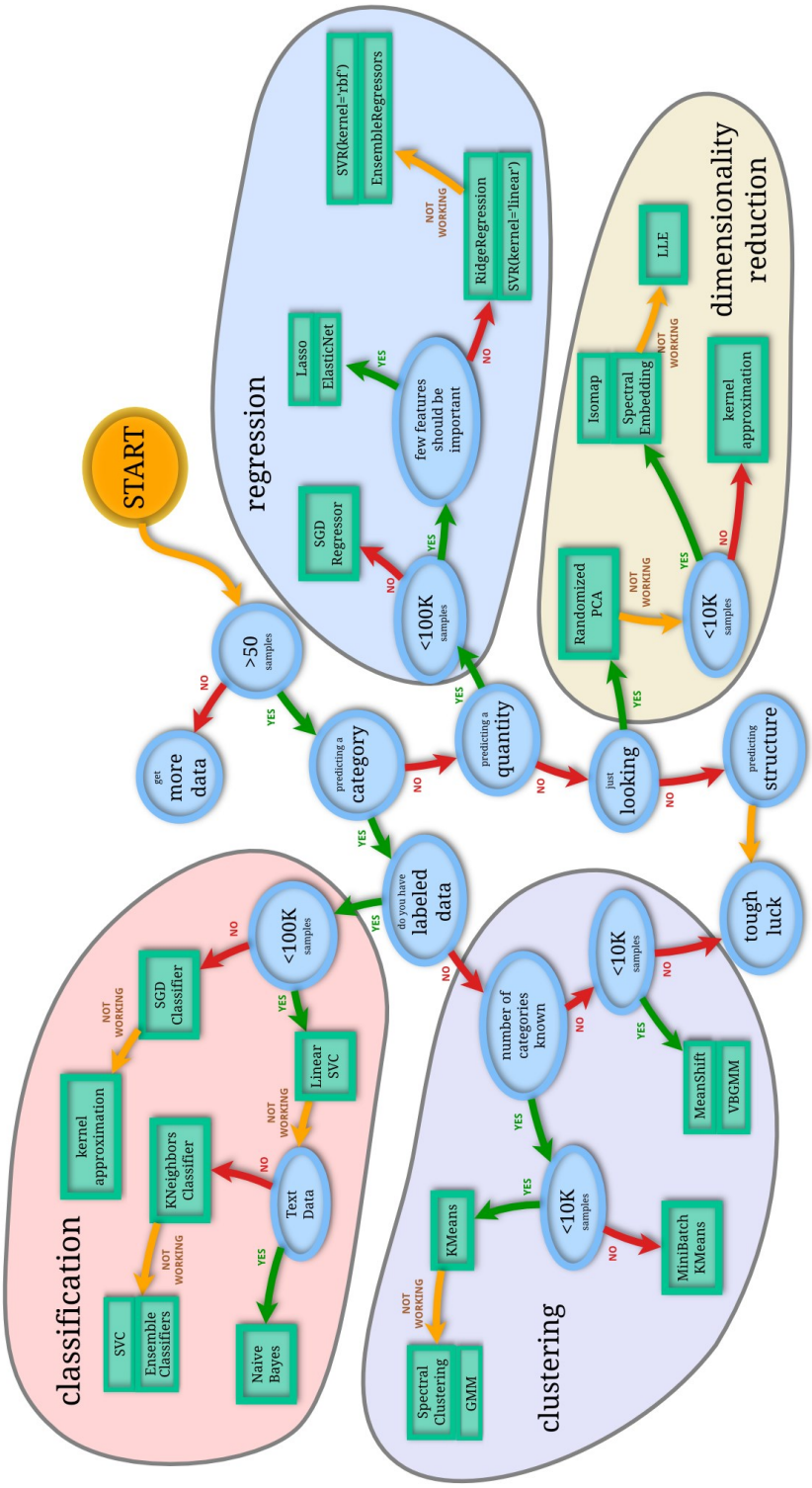


Figure D.1: A flow diagram for choosing the correct machine learning algorithm in Scikit-learn [139].

Appendix E

List of licenses and languages of the software discussed

Table E.1: A summary of the licenses, implementation languages and API languages of all the machine learning tools, frameworks and other software discussed in this thesis. The BSD licenses listed are of either the 2-clause or the 3-clause version, unless otherwise stated.

Software	**License**	**API languages (or UI)**	**Written in**
Caffe	BSD	Python, C++	C++
Caffe2 (merged to PyTorch)	Apache-2.0	Python, C++	C++
Caret (for R)	GPLv2, GPLv3	R	R
Clipper	Apache-2.0	Python, R, (RESTful)	C++, Python
Darwin	BSD	C++	C++
DeepDetect	LGPLv3	C++	C++
Deeplearning4j	Apache-2.0	Java, Scala	Java, C, C++, CUDA
Docker	Apache-2.0	(CLI)	Go
Drools	Apache-2.0	Java	Java
Flink	Apache-2.0	Scala, Java, Python, SQL	Scala (Java)

H2O	Apache-2.0	Java, Python, Scala, R	Java, Python, R
Hadoop	Apache-2.0	MapReduce: Java	Java
Jupyter	BSD 3-Clause	(web), Kernel: *e.g.* Python	Python, JavaScript, CSS, HTML
Keras	MIT	Python, R	Python
Kubernetes	Apache-2.0	(CLI)	Go
Mahout	Apache-2.0	Java, Scala	Scala (Java)
Matplotlib	Matplotlib (BSD)	Python	Python
Microsoft CNTK	MIT	Python (Keras), C++, (CLI)	C++
MLlib	Apache-2.0	Java, Python, Scala, R	Scala (Java)
Model Server for Apache MXNet (MMS)	Apache-2.0	(CLI)	Python
MXNet	Apache-2.0	C++, Python, R, Scala, and others	C++, Python, R, Scala, and others
NiFi	Apache-2.0	(web)	Java
Node-RED	Apache-2.0	(web), JavaScript	JavaScript
NumPy	BSD 3-Clause	Python	Python, C
ONNX	MIT	-	Python, C++
Pandas	BSD 3-Clause	Python	Python
PlaidML	AGPLv3	Python (Keras), C++, C	C++, Python
Plotly.js	MIT	Python, R, Node	JavaScript
Pothos	BSL-1.0	C++	C++
PredictionIO	Apache-2.0	Java, Python	Java
PyTorch	BSD	Python	Python, C, CUDA
Scikit-learn	BSD	Python	Python, Cython, C, C++

SciPy	BSD 3-Clause	Python	Python, Fortran, C, C++
Seaborn	BSD 3-Clause	Python	Python
Sklearn-porter	MIT	Python	Python
Spark	Apache-2.0	Scala, Java, Python, R, SQL	Scala (Java)
TensorBoard	Apache-2.0	(web)	Python, HTML, others
TensorFlow	Apache-2.0	Python (Keras), C, C++, Java, R, Go, Julia	C++, Python, CUDA
TensorFlow Serving	Apache-2.0	Python, (gRPC), (RESTful)	C++
Theano	BSD	Python (Keras)	Python
Torch	BSD	Lua, C	C, Lua
Vespa	Apache-2.0	Java, HTTP	Java, C++
Visdom	Attribution-NonCommercial 4.0	Python, Lua, (web)	Python, JavaScript