

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

Ville Tähkävuori

Machine learning framework for OPC UA data (Industry 4.0)

Master's Thesis
Espoo, April 1, 2019

Supervisor: Professor Sirkka-Liisa Jämsä-Jounela
Instructors: Alexandre Boriouchkine D.Sc. (Tech.)
Lauri Saurus M.Sc. (Tech.)

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

ABSTRACT OF
 MASTER'S THESIS

Author:	Ville Tähkävuori		
Title:	Machine learning framework for OPC UA data (Industry 4.0)		
Date:	April 1, 2019	Pages:	iix + 71
Professorship:	Process control	Code:	Kem-90
Supervisor:	Professor Sirkka-Liisa Jämsä-Jounela		
Instructors:	Alexandre Boriouchkine D.Sc. (Tech.) Lauri Saurus M.Sc. (Tech.)		
<p>Machine learning has rapidly gained popularity in all industries with the increase of computational power and data gathering capabilities. Process industry is a good candidate for machine learning based modeling due to the large amounts of data gathered and need for accurate process state predictions.</p> <p>In this work the viability of combining the OPC UA protocol with existing open source machine learning libraries to create data driven models and generate real time predictions was studied.</p> <p>Scikit-learn was used to generate soft sensor style models for the butane content of a debutanizer column output. The data for offline model training was dynamically fetched from an OCP UA server and with a trained model predictions could be generated in real time.</p> <p>The accuracy of the generated models needs to be further researched with better methodology and larger datasets.</p>			
Keywords:	machine learning, OPC UA, framework, process industry		
Language:	English		

Aalto-yliopisto
 Kemian tekniikan korkeakoulu
 Kemiantekniikan koulutusohjelma

DIPLOMITYÖN
 TIIVISTELMÄ

Tekijä:	Ville Tähkävuori		
Työn nimi:	Koneoppimiskehys OPC UA datalle (Industry 4.0)		
Päiväys:	1. toukokuuta 2019	Sivumäärä:	iix + 71
Professuuri:	Prosessien ohjaus	Koodi:	Kem-90
Valvoja:	Professori Sirkka-Liisa Jämsä-Jounela		
Ohjaajat:	Tekniikan Tohtori Alexandre Boriouchkine Diplomi-insinööri Lauri Saurus		
<p>Koneoppiminen on kasvattanut suosiotaan nopeasti kaikilla toimialoilla laskenta-tehon ja datankeruun kasvaessa. Prosessiteollisuus on hyvä kandidaatti koneoppimispohjaiselle mallinnukselle suurien datamäärien sekä vaadittujen tarkkojen prosessimallien takia.</p> <p>Tässä työssä tutkittiin mahdollisuutta OPC UA protokollan yhdistämistä olemassaolevien avoimen lähdekoodin koneoppimiskirjastojen kanssa mittausdataan perustuvien mallien opettamiseksi ja reaaliaikaisten ennusteiden luomiseksi.</p> <p>Scikit-learn kirjastoa käytettiin luomaan malleja butaaninpoistokolonnin ulostulon butaanipitoisuuden ennustamiseen. Data mallien offline opetukseen ladattiin dynaamisesti OPC UA palvelimelta ja valmiiksi opetetulla mallilla ennusteita voitiin generoida reaaliaikaisesti.</p> <p>Luotujen mallien tarkkuutta täytyy tutkia tarkemmin paremmalla metodologialla ja suuremmilla datamäärillä.</p>			
Asiasanat:	koneoppiminen, OPC UA, kehys, prosessiteollisuus		
Kieli:	Englanti		

Acknowledgments

This master's thesis was done for NAPCON of Neste Engineering Solutions Oy between June 2017 and April 2019.

I wish to thank Professor Sirkka-Liisa Jämsä-Jounela for supervising this thesis and my instructors D.Sc. Alexandre Boriouchkine and M.Sc. Lauri Saurus for their detailed comments and help throughout this thesis. Additional thanks to Antti Räisänen and Markus Sintonen for helping in defining the topic of the thesis in its early stages

Special thanks to my manager Lauri Haapanen for helping with literally everything, Toni for always sitting next to me and Johanna for making sure this thesis got finished. I would also like to thank the whole NAPCON team for always being helpful and supportive during my thesis.

Espoo, April 1, 2019

Ville Tähkävuori

Abbreviations and Acronyms

AI	Artificial Intelligence
OPC UA	OPC Unified Architecture
ANN	Artificial Neural Network
DNN	Deep Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
MLaaS	Machine Learning as a Service
PCA	Principal Component Analysis
API	Application Programming Interface
HDS	Hydrodesulphurization
DMS	Dimethyl sulfide
TPU	Tensor Processing Unit
ASIC	Application Specific Integrated Circuit
GPU	Graphics processing unit
CPU	Central processing unit
SVM	Support Vector Machine
SOM	Self-Organizing Map
KDE	Kernel Density Estimation
PLS	Partial Least Squares

Contents

Abbreviations and Acronyms

1	Introduction	1
2	Description of mathematical methods in machine learning	3
2.1	Machine learning model development process	3
2.1.1	Data preparation	4
2.1.2	Data Pre-processing	4
2.1.3	Model selection and training	4
2.1.4	Deployment	5
2.2	Machine learning methods	5
2.2.1	Supervised learning	5
2.2.1.1	Multiple regression linear models	6
2.2.1.2	Partial least squares regression	7
2.2.1.3	k -nearest neighbors	7
2.2.1.4	Artificial neural networks	7
2.2.1.5	Deep neural networks	9
2.2.1.6	Tree based methods	10
2.2.1.7	Kernel based methods	11
2.2.2	Unsupervised learning	13
2.2.2.1	Principal Component Analysis	14
2.2.2.2	K -Means clustering	15
2.2.2.3	Self-organizing map	16
2.2.2.4	Gaussian mixture model	16
2.2.3	Reinforcement learning	17
2.2.4	Ensemble learning	18
3	Description of machine learning software libraries and solutions	19
3.1	Open source software	19
3.1.1	scikit-learn	19

3.1.2	TensorFlow	20
3.1.3	R Programming Language	21
3.1.4	Apache Spark / MLlib	21
3.1.5	H2O	22
3.2	Proprietary software	23
3.2.1	MATLAB	23
3.2.2	Wolfram Mathematica	24
3.3	Machine Learning as a Service	24
3.3.1	Azure Machine Learning	25
3.3.2	Amazon Machine Learning	26
3.3.3	Google Cloud Machine Learning	26
3.3.4	BigML	26
3.3.5	IBM Watson Machine Learning	27
4	State of the art in machine learning in process industries	28
4.1	Soft sensors	28
4.2	Process monitoring and fault detection	30
4.2.1	Predictive maintenance	31
4.3	Reinforcement learning for process control	32
5	Machine learning framework requirements for process industries	33
5.1	Process data acquisition and storage	33
5.1.1	OPC Unified Architecture	33
5.1.1.1	Historical data access	34
5.1.1.2	Real-time access	34
5.2	Data preprocessing	34
5.3	Machine learning methods for process industry applications . .	35
5.4	Model training	35
5.5	Model deployment	36
5.6	User interface	37
5.6.1	Model Training UI	37
6	Implementation of an OPC UA enabled machine learning framework	41
6.1	Choosing a machine learning library	41
6.2	Architecture of machine learning framework	42
6.2.1	OPC UA connection	43
6.2.2	Model training	45
6.2.3	Model deployment	46
6.2.4	User Interface	47

6.3	Process description	49
6.4	Gathering training and test data	50
6.4.1	ProsDS process model	50
6.4.2	Test run	51
7	Results and Discussion	54
7.1	Framework requirements results	54
7.1.1	Process data acquisition and storage	54
7.1.2	Data preprocessing	54
7.1.3	Algorithms	55
7.1.4	Model Deployment	55
7.1.5	User Interface	55
7.2	Model performance evaluation	56
7.3	Model performance	56
8	Conclusions and future work	60
A	Debutanizer column model in ProsDS	70
B	Data visualization tools in Azure Machine Learning Studio	71

Chapter 1

Introduction

As the global community has become more environmentally conscious due to the increased knowledge on how human influence is contributing to global warming and other environmental problems, a major global trend in process industry during the past decade has been increasing effectiveness and reducing the pollution. Other driving factor in increasing process efficiency has been an increase in global competition. These global trends have culminated in European Unions (EU) case in the 2020 package, which is a set of binding legislations requiring requiring 20% cuts in greenhouse gases and 20% increase in energy efficiency by the year 2020. Other large economies have published similiar plans. (Ge et al., 2017)

To achieve these goals of increased efficiency and reduced pollution, processes have become much more complex, and the amount of monitoring and instrumentation in all process industries has increased a lot. This, in combination with the continuously decreasing price of sensors, storage space, etc. has lead to more data being collected. (Ge et al., 2017; Harding et al., 2006)

Most of this process data goes to a history database without any detailed analysis ever being done with it. The data is mostly used to make real time decisions about process operation. With current methods, the value gained from these huge amounts of raw data has diminishing returns and lots of potential is being lost. To gain more value from high volume datasets, new data mining methods must be adopted. (Ge et al., 2017; Harding et al., 2006)

In the past few years, machine learning has seen increase as a popular data processing technique. It is seen as a potentially very powerful tool to unlock hidden potential from the large amounts of being gathered. Process industry has noticed potential for applying machine learning in process monitoring, soft sensors, fault diagnosis and other important applications. Machine learning is a wide set of methods that can automatically detect patterns in datasets, and produce models which can generate predictions about

future data, without being explicitly programmed to understand the data. (Murphy, 2012; Ge et al., 2017)

The objective of this thesis is to define a systematic framework for utilizing machine learning in process industries. The structure of the thesis is as follows: The literature part gives an overview of machine learning, including history and description of mathematical methods. This survey of state of the art of machine learning in process industry applications is used to define requirements for a machine learning framework for process industries. The experimental part of the thesis will focus on implementing the requirements defined in the literature part in a demo framework of machine learning for OPC UA process data using an existing machine learning software implementation. The framework was tested using a industrial dynamic process simulator.

Chapter 2

Description of mathematical methods in machine learning

Machine learning enables computers to model systems independently by utilizing large volumes of data. Machine learning algorithms can spot patterns from complex datasets and in some cases create models that are more effective than ones programmed by humans.

Machine learning is a subset of Artificial Intelligence (AI) research. At first expert systems with knowledge-based approaches came to dominate AI research. The current form of data-driven machine learning began to gain shape in the 1980s and flourished in 1990s after it started to get applied to smaller, more manageable problems instead of general artificial intelligence. Machine learning is a vast area of study that covers linear regression models with least squares to complicated multilayer artificial neural networks. Machine learning contains many branches of science, including mathematics, statistical analysis and computer science. (Hastie et al., 2009)

First, this chapter will describe the standard procedure for creating a machine learning model, then different types of machine learning methods are presented.

2.1 Machine learning model development process

Machine learning model development consists of three steps: Data preparation, pre-processing and model training.

2.1.1 Data preparation

Data preparation is the initial step in machine learning model development. The aim of the data preparation step is to get an overview of the data. It contains the extraction of the dataset from historical database, decisions of sample and variable selection. (Ge et al., 2017)

2.1.2 Data Pre-processing

After preparation, the data often requires some pre-processing before it is ready for model training.

Removing outliers and obvious errors from the training dataset improves the model performance. Missing values should then be handled by missing value estimation or by completely removing the sample. (Ge et al., 2017)

Differences in variable scales need to be addressed. The absolute scale of variables can vary a lot, even depending on the unit of measurement (for example, kPa vs Pa). For Principal Component Analysis (PCA), the variables must all transformed to zero mean and unit variance which makes the different variables equal in scale. Data transformations step must be carefully decided, as it depends significantly on the model being trained. (Ge et al., 2017)

2.1.3 Model selection and training

The next step is to select the appropriate model for training. This is not an easy task, and there are no standard methods for selecting the model. (Ge et al., 2017)

The first factor to be considered is the model complexity. A linear model should be used if possible, but if the data is highly nonlinear, more complex models such as Artificial Neural Networks or Support Vector Machines must be used. (Ge et al., 2017)

After the model has been selected, it must be trained with a learning algorithm that is depended on the mode being used. Often models also contain so called hyperparameters that can not be learned from training data, but must be manually tuned.

Before the model is ready for deployment, it's performance must be validated. Many methods for model validation are available. The simplest way is to remove a part of the training dataset, and use it to check how the trained models performs on data it has not been learning from. In K-Fold Cross Validation, the whole dataset is separated into k subsets, and each is used as validation set while the rest is used as training data. This gives a better

indicator on how well the model will perform, because it removes the effect of what separation was chosen. (Ge et al., 2017)

2.1.4 Deployment

After training and validation, the model is ready for offline or online deployment. In case of any changes to the environment being modeled, the model might require maintenance. This can be performed offline by repeating the steps defined earlier, or online by some algorithms. (Ge et al., 2017)

2.2 Machine learning methods

This section looks at common machine learning methods and explains how they work and what they can be used for. The methods have been roughly divided under supervised learning, unsupervised learning, reinforcement learning and ensemble learning.

2.2.1 Supervised learning

In supervised machine learning, during the training process, the model is supplied with labeled training examples, pairs of input objects (most often vectors) and expected output values. The machine learning algorithm then infers a function from the training data that can map new inputs to output values. Mathematically this can be expressed as giving a set of training examples of the form $\{(x_i, y_i), \dots, (x_n, y_n)\}$ where x_i is a vector of inputs and y_i is the expected value, then training a model so that it can create function that can map any $x \rightarrow y$. (Murphy, 2012; Hastie et al., 2009)

Figure 2.1 shows the basic workflow for supervised machine learning workflow. Training set is split into training- and test data. Training data is used to tune the model parameters, and test data is used after training to validate the performance of the trained model.

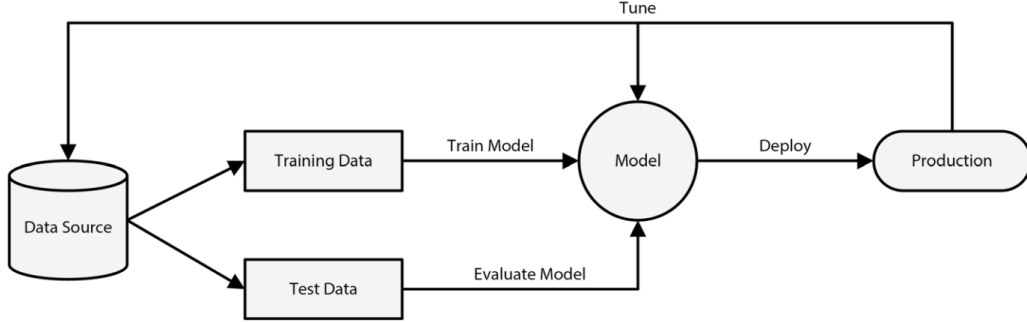


Figure 2.1: Supervised machine learning workflow (Landset et al., 2015)

In supervised learning it is especially important that the training and test dataset is large and representative of the phenomenon we wish to model. If the training set is too narrow, the resulting model will only be accurate for the training set, and will probably produce bad predictions for inputs outside the area where the model was trained. (Murphy, 2012)

Broadly speaking, supervised machine learning problems can be divided into regression and classification. Regression means the predicted value will be a continuous numeric value. For time series data, this usually means forecasting into the future or predicting a value that is difficult to measure in real time. In classification, the output is in two or more discrete categories.

2.2.1.1 Multiple regression linear models

Linear regression models are one of the most widely used models for regression. In linear regression, output response is assumed to be a linear combination of the inputs. (Murphy, 2012)

Linear regression models are simple, but often provide good fits for the given data. They were mostly developed in the precomputer era. (Hastie et al., 2009)

The general form of an linear regression model is show in 2.1, input vector $X_{1...p}$ and coefficients $\beta_{0...p}$ (Hastie et al., 2009):

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j \quad (2.1)$$

The coefficients $\beta_{0...p}$ are estimated from training data. The most common way to estimate them is the least squares method, where the coefficients are selected to minimize the residual sum of the squares (2.2). (Hastie et al., 2009):

$$RSS(\beta) = \sum_{i=1}^N (y_i - f(x_i))^2 \quad (2.2)$$

Other methods for estimating the coefficients for the linear model are for example Ridge regression and Lasso. They are called regularization methods, and their aim is to prevent overfitting to the data. Generally they yield much better models than regular least squares, especially for high dimensional data or low amount of samples. (Hastie et al., 2009)

2.2.1.2 Partial least squares regression

Partial Least Squares (PLS) is a method for creating linear regression models. PLS is very common in many machine learning tasks, especially in soft sensors and process monitoring. (Ge et al., 2017; Bishop, 2006)

In regular multiple linear regression, the variables are assumed not to be collinear. In many cases, this assumption doesn't hold very well, especially when the number of input variables grows. PLS is best used when there are many highly collinear input variables. Even though the number of process variables can be very large, the amount of underlying factors that determine the output might be small. PLS aims to extract these factors, and use them in creating the model. (Ge et al., 2017; Bishop, 2006)

2.2.1.3 k -nearest neighbors

k -nearest neighbors is one of the simplest machine learning algorithms. It can be used in both classification and regression. In the former, objects are classified by the majority of its k -nearest neighbors, where k is a positive integer. In the latter, the objects value is an average of its nearest neighbors. Adding weighing to the neighbors based on how far they are is a simple way to improve the algorithm. (Bishop, 2006; Murphy, 2012; Hastie et al., 2009)

Computing the k -nearest neighbors algorithm requires the whole training data set to be stored in memory, which can be challenging with sufficiently large data sets (Bishop, 2006).

Using Dynamic Time Warping (DTW) as distance measure with k -nearest neighbors classification with time series data can produce very good results (Wang et al., 2013).

2.2.1.4 Artificial neural networks

Artificial neural network (ANN) is a computational system inspired by the human brain. ANNs comprise of artificial neurons and weighted links be-

tween them. ANNs have been used for many tasks that are difficult to express via traditional rule-based programming, such as computer vision and speech recognition. (Murphy, 2012; Hastie et al., 2009)

ANNs have their beginnings in 1943 when McCulloch and Pitts (1943) released their paper "A logical calculus of the ideas immanent in nervous activity". It presented a model of an artificial neuron whose output depends on the weighted sum of its inputs, plus a threshold value which had to be exceeded in order to activate the output. Research for modern ANNs with hidden layers started to gain traction in the late 1980's when backpropagation algorithm (Rumelhart et al., 1986) for fitting neuron parameters for ANNs with hidden layers became popular. (Murphy, 2012; Hastie et al., 2009)

ANNs can perform very complicated tasks, but can be challenging to set up. The network architecture must first be determined (number of layers and neurons, and the connections between them). After this the network is specialized (trained) to a given problem by varying the connection weights. Sometimes the network architecture simply does not work for the specific problem and no amount of learning will fix it. In addition, ANN training suffers from the existence of local minima. (Nielsen, 2015)

One of the simplest forms of ANNs is the feedforward network, where data flows from the input layer through hidden layer(s) towards the output layer. Figure 2.2 shows a simple feed forward ANN.

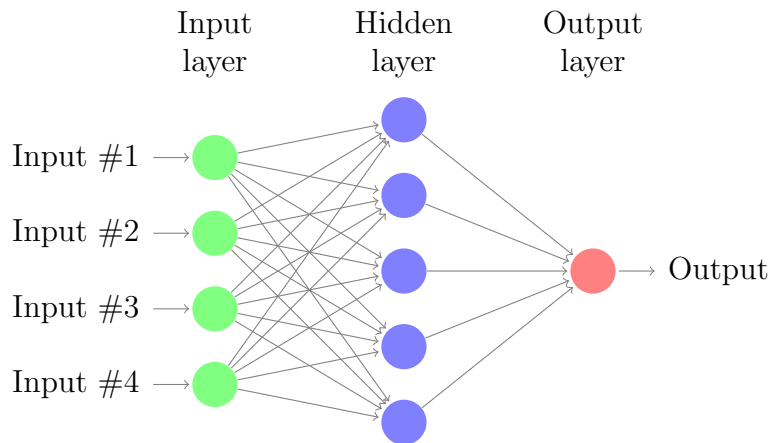


Figure 2.2: A simple feed forward neural network with a single hidden layer and a single output. (Fauske, 2006)

ANNs are *universal*, that is they can approximate any function to arbitrary precision. That is, for any function $f(x)$ there exists a neural network

whose output given the input x is a close approximation of $f(x)$. This statement is true for functions with many inputs, and works even with single hidden layer networks. The approximation can be made arbitrarily close to the original function just by adding neurons to the hidden layer. This property of universality combined with training algorithms make ANNs very attractive. (Nielsen, 2015)

Perhaps more interesting for time series machine learning purposes is a type of network where connections between neurons form loops. This gives the network a form of state, which enables them to process dynamic temporal behavior. The network presented in Figure 2.3 is almost the same as the feed forward network in Figure 2.2 except for the loop-backs in the hidden layer, shown here in bold.

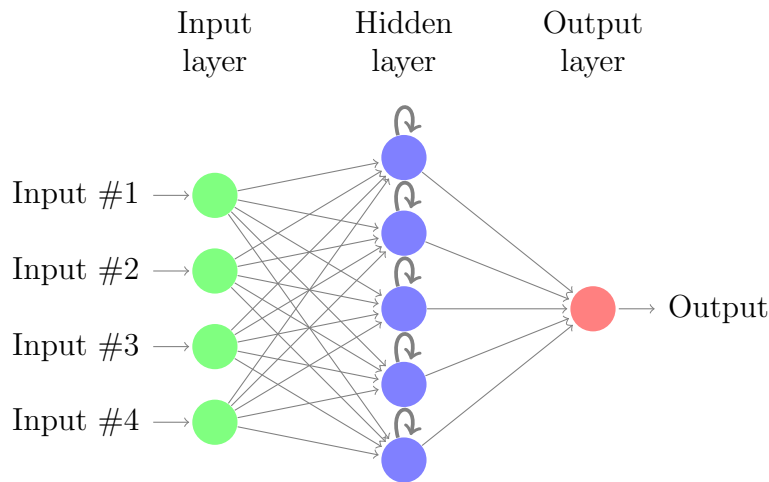


Figure 2.3: A recurrent neural network.

Long Short-Term Memory (LSTM) is a special Recurrent Neural Network (RNN) architecture presented by Hochreiter and Schmidhuber (1997). It has been shown to produce state-of-the-art results in applications with sequential data. LSTM networks are good at handling long term temporal relations in data. (Che et al., 2016)

2.2.1.5 Deep neural networks

A Deep Neural Network (DNN) contains multiple hidden layers between the input and output layers. In theory, DNNs have existed for as long as ANNs, but they suffered from lack of good training algorithms and slow computer hardware. In 2006, computers had become faster and Hinton et al. (2006)

published a breakthrough paper describing how DNNs could be trained effectively. Since 2006, Deep Learning and DNNs have become extremely popular in machine learning. (Schmidhuber, 2015)

Figure 2.4 shows a fully connected feedforward DNN with four hidden layers and five neurons each, but the number of layers and neurons can be substantially higher (Schmidhuber, 2015).

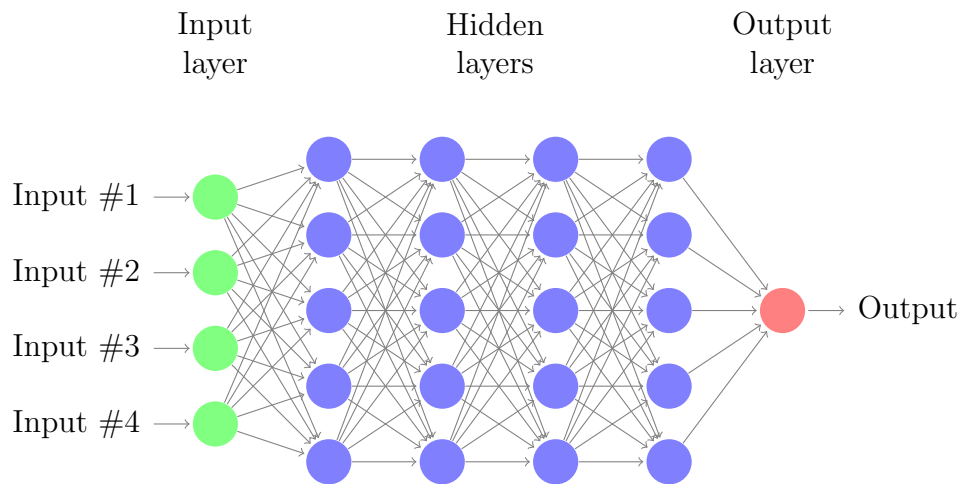


Figure 2.4: A feedforward Deep Neural Network.

DNNs are good at learning representation of high dimensional data. DNNs have showed extremely good performance in many important applications, easily outperforming other machine learning methods. DNNs have even outperformed humans in pattern recognition tasks on a limited set. (Schmidhuber, 2015)

2.2.1.6 Tree based methods

Decision trees are a class of machine learning methods that recursively partition the input data space to purer output. Decision trees can be thought of as a sequence of if-then-else statements. Decision trees power is that they can fit almost any data distribution, but this can result in overfitting of the data, but this means that they can be prone to overfitting. (Aldrich and Auret, 2013)

Figure 2.5 shows a simple decision tree which can classify two dimensional data into five categories.

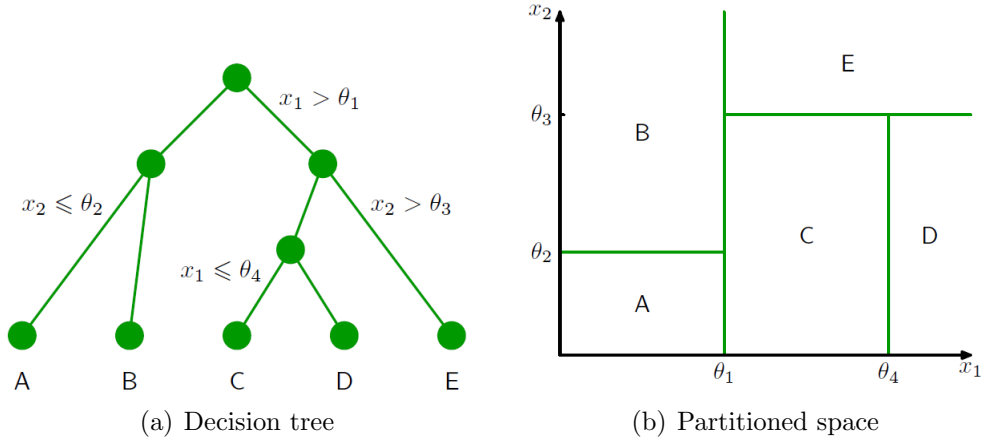


Figure 2.5: A simple classification decision tree (Bishop, 2006).

Random forest or random decision forest is an ensemble learning method using multiple decision trees. Random forests are created by training different decision trees from randomly chosen subset of *input* variables and data cases. (Murphy, 2012)

In a technique called bootstrap aggregating (bagging), when random forest method is used to predict something from input, the individual decision trees in the random forest vote on the output value, as shown in equation 2.3, where $f_m(x)$ is m 'th decision trees output (Murphy, 2012):

$$f(x) = \sum_{m=1}^M \frac{1}{M} f_m(x) \quad (2.3)$$

Random forests are good at preventing overfitting to the training data, which can be a problem using individual decision trees.

2.2.1.7 Kernel based methods

Kernel methods are a class of machine learning methods. Kernel methods get their name from the kernel functions they use. A kernel function is a similarity function, it takes two inputs and returns how similar they are. There are many kinds of kernels that can be used (linear, polynomial, many specialized kernels), the choice of the kernel can be very specific to the problem being studied. (Murphy, 2012)

Support Vector Machine (SVM) is one of the most important kernel based methods in use today. Original SVM algorithm dates back to a paper by Vladimir Vapnik in 1963, but the current standard method is based on a

paper by Cortes and Vapnik (1995). SVMs can be used for supervised classification, regression and unsupervised clustering.

In clustering, SVMs generate maximum margin hyperplanes for the data. Figure 2.6 shows a maximum margin hyperplane generated from a trained two class SVM with linearly separable data. Samples that lie exactly on the margin, are called support vectors.

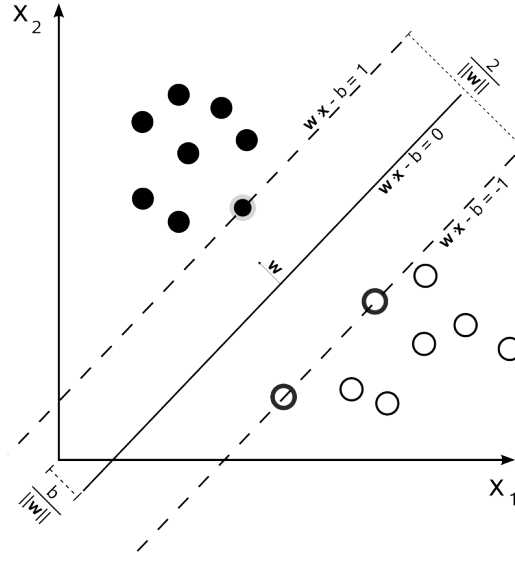


Figure 2.6: Maximum margin hyperplane between two classes generated by an SVM used for classification (Anonymous, 2008)

If the data is not linearly separable, which is often the case, SVMs aim to map the data to higher dimension to gain linear separability. Figure 2.7 shows an example with non linearly separable data.

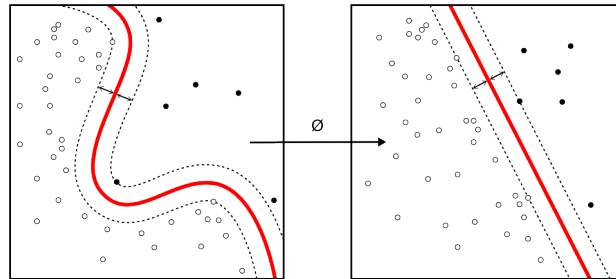


Figure 2.7: Mapping (Φ) of non linearly separable data to a higher dimension to make the it linearly separable (Anonymous, 2016).

The 'kernel trick' allows for usage of very high dimensional kernels without requiring to do the computationally expensive transformation. These kernels must allow for expressing the similarity score directly using the features in the original mapping. (Bishop, 2006)

The advantages of SVMs are the high versatility achievable by choosing specific kernels, it is a convex optimization problem (the training can't get stuck on a local minimum). The disadvantages are also related to the versatility of the kernel functions. Achieving good generalization performance requires lots of knowledge of the data being modeled and specifics on how the different kernels perform and their hyperparameters. (Bishop, 2006; Murphy, 2012)

SVMs can also be extended for use in regression tasks by choosing the optimization restrictions differently. This is often called Support Vector Regression (SVR). SVR retains most of the advantages and disadvantages of SVMs. (Bishop, 2006)

SVMs can be applied for machine learning with time series data by choosing correct kernel methods. Gudmundsson et al. (2008) used dynamic time warping based kernel for speech recognition with promising results.

2.2.2 Unsupervised learning

Unsupervised learning is a machine learning task used for drawing inferences from the unlabeled data containing only inputs without any desired output values. The need for unsupervised learning methods is caused by the massive amounts of unlabeled data compared to labeled data. (Murphy, 2012; Hastie et al., 2009)

The most common unsupervised learning method is clustering, where the unlabeled data is assigned into separate "natural" clusters, hoping that these clusters will tell us something about the data contained in them. Principal Component Analysis (PCA) is another common unsupervised learning method, often used in conjunction with supervised learning methods for feature extraction. (Murphy, 2012; Hastie et al., 2009)

Unsupervised learning applications in process industries are dimensionality reduction, outlier detection, process monitoring and data visualization (dimensionality reduction).

Unlike in supervised learning, in unsupervised learning the accuracy of the models cannot be automatically evaluated against labeled data.

2.2.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure used for dimensionality reduction.

The goal of PCA is to reduce the amount of dimensions in the data. It does this by finding the directions in the data which cause the largest and smallest variations. Figure 2.8 shows a graphical example in two dimensions, but PCA can be applied to data with any amount of dimensions. In higher dimensions, multiple principal components can be found, under the constraint that the n th principal component is orthogonal to all principal components before it. (Aldrich and Auret, 2013; Ge et al., 2017)

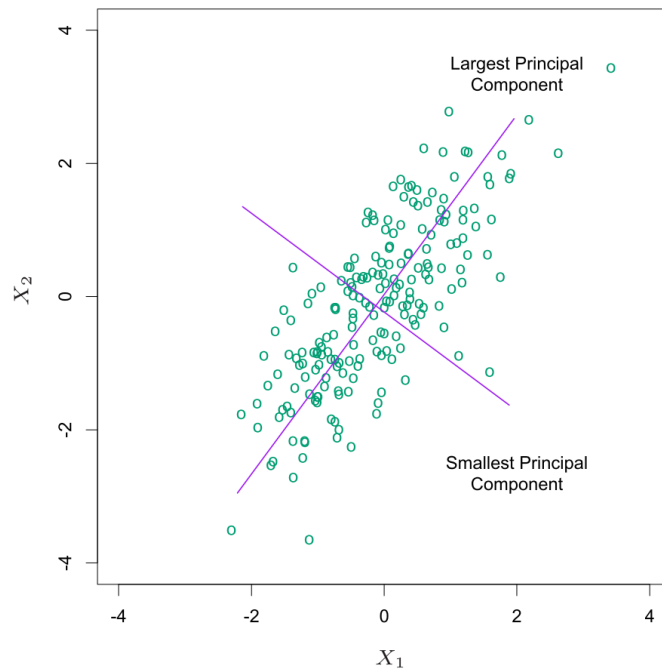


Figure 2.8: Largest and smallest principal components of some data points. (Hastie et al., 2009)

Due to the nature of how PCA works, the principal components generated will be completely uncorrelated, which greatly aids in data analysis. (Ge et al., 2017)

PCA is useful with data that contains high-dimensional correlated data. High-dimensional data is difficult to work with, but PCA can compress higher dimensional data to fewer dimensions by finding linear combinations between

variables while preserving large fraction of the original data content. (Aldrich and Auret, 2013; Ge et al., 2017)

Many variations have been made to PCA to improve performance in complex applications, including nonlinear PCA, dynamic PCA, adaptive PCA and mixture PCA. (Ge et al., 2017)

2.2.2.2 K -Means clustering

k -means clustering is a simple clustering algorithm where the goal is to partition observations into k different clusters. Algorithm 1 shows the standard k -means algorithm. (Bishop, 2006)

Algorithm 1 K -means algorithm (Hastie et al., 2009)

- 1: give initial (random) values for cluster centers
 - 2: **repeat**
 - 3: **Assign** each observations to cluster whose center is closest (euclidean distance)
 - 4: **Update** each cluster center location
 - 5: **until** converged (assignments no longer change)
-

Figure 2.9 shows a visualization of how k -means clustering works iteratively.

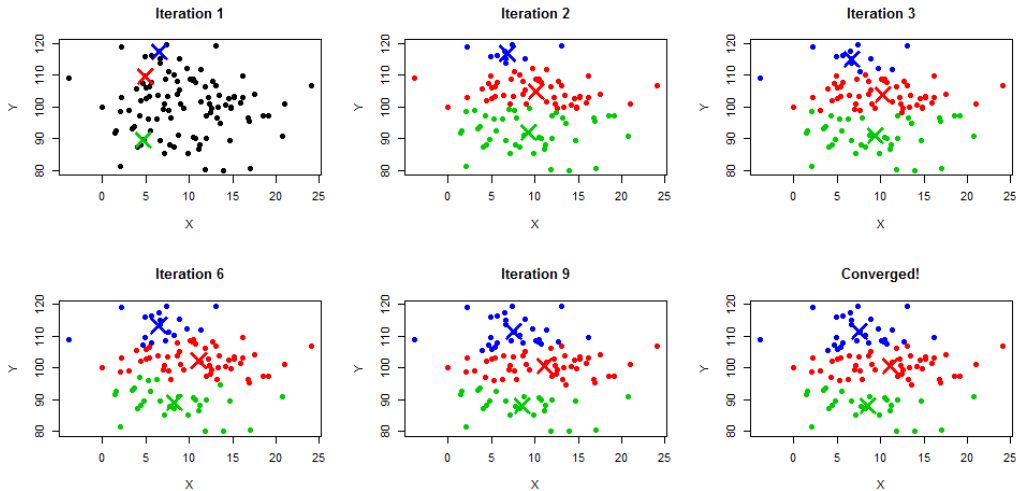


Figure 2.9: k -means iterations with 50 data points and 3 random initial cluster centers. (Johnson, 2017)

This direct implementation can be rather slow, as the number of euclidean distance calculations done in each iteration of the loop is rather large. Many proposals to speed up the algorithm have been made, such as precomputed data structures which map nearby point to same subtree, and mathematical tricks like triangle inequality which eliminate unnecessary distance calculations. (Bishop, 2006)

Unlike k -nearest neighbors algorithm, we do not have to keep the whole training dataset in memory after the model has been trained.

While k -means is easy to use and relatively simple to understand and implement, it has some caveats and limitations that need to be taken into account:

The amount of clusters hyperparameter, k , needs to be set manually, and as the initial cluster centers are set randomly, sometimes the clustering is not optimal. Due to these two factors k -means often requires more than one try to get a useful clustering result.

Euclidean distance is not a very useful measure of similarity with really high dimensional data, referred often to as the "curse of dimensionality". The standard k -means clustering algorithm only uses straight lines to divide the data, which can be insufficient for some datasets.

2.2.2.3 Self-organizing map

A self-organizing map (SOM) is a type of ANN that can be used for unsupervised learning. SOMs (sometimes called Kohonen Maps) learn data mapping from high-dimensional data to low-dimensional (typically two) representation. (Kohonen, 1997; Kadlec et al., 2009).

SOMs are trained with competitive training algorithms, which differ from the error correcting algorithms used in supervised learning process (backpropagation).

2.2.2.4 Gaussian mixture model

Gaussian mixture model is a probabilistic model which gives a probability of a sample belonging in subpopulation. It does not require labeling of subpopulations, and can learn them automatically. Mixture models are especially useful for characterization of processes with multiple operating points and product grades. (Bishop, 2006; Ge et al., 2017)

2.2.3 Reinforcement learning

In reinforcement learning, the machine learning algorithms output influences the environment it is acting in, and the algorithm receives feedback on how it is performing. The goal in reinforcement learning is to find a policy that maximizes the cumulative rewards (positive feedback). (Spielberg et al., 2017)

Figure 2.10 shows the essence of reinforcement learning, how the agents actions interact with the state of the environment, which produces new state and reward feedback. (Sutton and Barto, 1998)

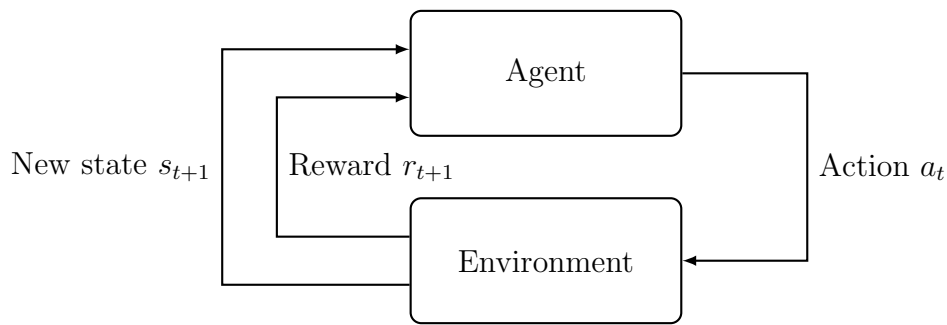


Figure 2.10: Basic reinforcement learning scheme. (Bacon, 2013)

Storing the raw policy might be impractical, especially with high dimensional state-action data. Instead, ANNs and other function approximators are used with reinforcement learning. (Spielberg et al., 2017)

One important challenge that affects reinforcement learning but does not arise at all in supervised learning problems is exploitation versus exploration. To gain positive feedback (succeed in the task it has been given), the reinforcement learning algorithm must *exploit* what it knows already, but to make better decisions in the future, it must also *explore* its environment to learn how to make better decisions in the future to gain higher rewards. Neither aspect can be ignored, or the algorithm will fail at the task. (Sutton and Barto, 1998)

Reinforcement learning combined with Deep Learning has proven to be extremely promising approach for complicated tasks such as playing strategy games such as go, chess and shogi. DeepMind, a company owned by Google, developed a general reinforcement learning algorithm that could teach itself to play these strategy games at a superhuman level, beating other state of the art game engines after just a couple of hours of playing against itself, with no external input other than the rules of these games. (Silver et al.,

2017)

2.2.4 Ensemble learning

In ensemble learning, multiple base models are used to improve the performance (prediction, classification, etc.). The main goal of ensemble learning is to reduce the possibility of an inaccurate model being used, that happens to accurately model the test data, because of model misconfiguration. (Murphy, 2012; Polikar, 2006)

There are multiple reasons to use ensemble learning systems. Sometimes good performance in training data does not imply good generalization performance, and different machine learning algorithms generalize differently from the same training data. Large amounts of data might be infeasible to process for a single algorithm, and combining multiple algorithms is usually more efficient. Ensemble systems have also been used for the exact opposite problem, too little data. (Polikar, 2006)

The chapter on tree based methods (2.2.1.6) will take a look at Random Forest method which is an ensemble learning system based on multiple decision trees.

Chapter 3

Description of machine learning software libraries and solutions

Having seen the demand for machine learning, many solutions have appeared on the market. This chapter presents available open-source, proprietary and service based machine learning software. Machine learning method offering and maturity, and ease of integration with OPC UA data are evaluated.

3.1 Open source software

In open source software the source code is fully available. It is generally free and can be used within commercial products without any fees or obligations, though this may depend on the license. Some licenses may oblige you to share any modifications you made, example of such a license is the GNU General Public License (GPL) (Anonymous, 2007a).

There are three different kinds of open source machine learning software solutions, general machine learning, neural network focused, and distributed focused. This means there is no single best library to choose from.

3.1.1 scikit-learn

Scikit-learn is a machine learning library for the Python programming language. It is designed to be as simple as possible to install and use, and features a large library of regression, classification and clustering algorithms. It is not designed with massive datasets and cluster computing in mind. scikit-learn depends on NumPy (numerical operations library) and SciPy (scientific library). scikit-learn has a very healthy community with lots of users, and new versions are being released periodically with new features and bug fixes.

(Pedregosa et al., 2011)

scikit-learn has one of the largest set of machine learning tools for a single library. Most important supervised learning algorithms include Neural Networks, Support Vector Machines, Nearest Neighbors and Decision Trees.

scikit-learn also has tools for data preprocessing, feature selection and model evaluation built in. Data preprocessing tools include transformations like standardization and normalization, binarization and imputating missing values.(Buitinck et al., 2013)

scikit-learn is licensed under the BSD license, which gives the user almost unlimited freedom as long as the BSD copyright and license notice are included. (Pedregosa et al., 2011)

3.1.2 TensorFlow

TensorFlow is a machine learning library primarily designed for deep learning, developed by Google Brain. It is based on Google Brains research project in very-large-scale deep learning which started in 2011. TensorFlow itself was released in 2015. Currently, TensorFlow has a large community of developers and users behind it and it is being used in production environments at Google and other companies. (Abadi et al., 2015; Anonymous, 2017k)

TensorFlow is written in Python, C++ and CUDA. Official Application programming interfaces (API) are provided for Python, C++ and Java. The community has developed bindings for many other programming languages, including C#, Rust and Scala. TensorFlow can run on multiple CPUs and GPUs to vastly increase the speed at which model training happens. (Abadi et al., 2015; Anonymous, 2017k)

TensorFlow gets its name from tensors, which in this case mean n-dimensional matrices. Models in TensorFlow are based on computational graphs, which consists of nodes of operations, which can be simple as adding or multiplying two constants or tensors, or more complicated functions. These computational graphs are then run in parallel. TensorFlow's main purpose is to provide an interface for expressing complicated machine learning algorithms and then running them on any level of system, ranging from mobile devices to very-large-scale distributed server farms with hundreds of Central Processing Units (CPU) and Graphics processing units (GPU). (Abadi et al., 2015)

While TensorFlow is primarily used for deep learning with neural networks, TensorFlow's API allows for implementation of general machine learning algorithms. TensorFlow ships with a module that implements some popular algorithms:

- K-means clustering

- Random Forests
- Support Vector Machines
- Gaussian Mixture Models
- Linear/Logistic regression

TensorFlow is licensed under Apache 2.0 which permits commercial use. (Abadi et al., 2015)

3.1.3 R Programming Language

R is a language and environment for statistical computation and graphics. On top of containing the most common elements of any programming language (conditionals, loops, functions, I/O facilities) it contains data preprocessing tools, and large library of statistical data analysis tools. (Anonymous, 2017j)

ANNs can be implemented using R. Single-hidden-layer neural networks are implemented in a package called `nnet`, which is shipped with base R. Recurrent neural network and some deep learning flavors of neural network packages also exist for R. (Hothorn, 2017)

The base R language is licensed under GNU General Public License (GPL). GPL permits commercial use of the software. Most packages are licensed under GPL or similiar license, but some have a noncommercial clause. (Anonymous, 2017j, 2007a)

3.1.4 Apache Spark / MLlib

Apache MLlib is a machine learning library for Apache Spark. Spark is a popular open source general-purpose cluster computing system for large amounts of data, which provides an interface for programming clusters of computers with parallelism and fault-tolerance. MLlib provides a complete machine learning pipeline with data pre-processing tools, multiple machine learning models and deployment tools. Spark has APIs for Java, Python, Scala and R languages. (Meng et al., 2016)

MLlib has a large library of algorithms for basic statistics, classification, regression and feature extraction and selection. Supervised and unsupervised algorithms are presented in Table 3.1.

Table 3.1: List of machine learning algorithms supported by Apache MLlib.

Supervised	Deep Neural Networks, Random Forests, Generalized Linear Model, Gradient Boosting Machine, Naïve Bayes Classifier, Stacked Ensembles
Unsupervised	Generalized Low Rank Models, K-Means clustering, Principal Component Analysis

Landset et al. (2015) performed a survey of open source hadoop based machine learning tools, according to which MLlib provided the most well rounded algorithm library. Distributed machine learning generally have a smaller algorithm library compared to non-distributed system due to how many machine learning algorithms cannot be distributed. (Landset et al., 2015)

3.1.5 H2O

H2O is an open source machine learning platform/library written in Java. It aims to provide a complete system for distributed machine learning, including data handling, distributing model training tasks and deploying the models. (Anonymous, 2017e)

H2O has a decently large list of supported algorithms. A list of algorithms is presented in Table 3.2.

Table 3.2: List of machine learning algorithms supported by H2O.

Supervised	Deep Neural Networks, Random Forests, Generalized Linear Model, Gradient Boosting Machine, Naïve Bayes Classifier, Stacked Ensembles
Unsupervised	Generalized Low Rank Models, K-Means clustering, Principal Component Analysis

The native deep learning in H2O is based on multi-layer feedforward ANN. Other types of deep neural networks such as Convolutional Neural Networks and Recurrent Neural Networks are supported by H2O Deep Water project, which integrates other deep learning libraries (TensorFlow, Caffe). (Anonymous, 2017e)

H2O provides multiple ways to control it. It has a built-in web-based interface called Flow which allows H2O to be controlled without any programming experience. Python, R and Java have native APIs to programmatically control H2O. A REST API is also provided, which can be used from almost any programming language. (Anonymous, 2017e)

Getting data into H2O can be done in many ways. Table 3.3 shows supported data types and data sources. (Anonymous, 2017e)

Table 3.3: Data types and sources accepted by H2O system.

Data types	CSV, ORC, SVMLight, ARFF, XLS, XLSX, Avro, Parquet
Data sources	Local, Remote File, Amazon S3, HDFS, JDBC

H2O is licensed under Apache-2.0, which is very permissive. It allows for H2O to be used commercially. (Anonymous, 2017e, 2004)

According to Landset et al. (2015), H2O and Apaches' MLlib are quite similar. Where H2O trumps the other frameworks are deep learning capabilities.

3.2 Proprietary software

Proprietary software costs money and the source code is usually not available to the customer. Vendors who sell proprietary software often provide incentives such as customer support on top of the software to help sell them.

3.2.1 MATLAB

MATLAB (from the words **matrix laboratory**) is a proprietary numerical computing programming language and environment continuously developed by MathWorks since 1984.

MATLAB offers large amount of ready made tools for supervised and unsupervised machine learning purposes. For regression, regular linear models including PLS, decision trees, k-nearest neighbors and neural networks are supported. For classification, SVM, logistic regression, discriminant analysis and neural networks are supported. For clustering, k-means, GMMs, SOMs and hidden Markov models are supported. This list is not exhaustive. As MATLAB is also an complete programming environment with a large toolset,

implementing ones own models is possible and made easier by the available tools. (Anonymous, 2017g)

Applications / Algorithms written in MATLAB can be compiled as stand-alone executables, C/C++ libraries, .NET components, Java classes and Excel Add-ins. This helps with model deployment, as buying MATLAB to run the developed model is not necessary.

OPC UA support can be added to MATLAB using OPC Toolbox. It supports namespace browsing, reading and writing, and historical data reading.

A single user commercial base MATLAB license costs 2000€ with toolboxes ranging from 1000€ to 2000€ (Statistics and Machine Learning Toolbox is 1000€, OPC Toolbox is 1150€). MathWorks does not make clear how much multiple licenses would cost and how freely applications developed in MATLAB can be distributed.

3.2.2 Wolfram Mathematica

Wolfram Mathematica is a technical computing environment developed by Wolfram Research. Originally it was developed for symbolic mathematical calculation, but now it is used in many areas of science and engineering computation.

Often the most difficult part in solving a machine learning problem is choosing a right algorithm. Mathematica tries to mitigate this by automating the process.

Mathematica has experimental but advanced support for ANNs. Arbitrary networks can be created and trained through supervised and unsupervised learning. Recurrent neural networks, including LSTM, are supported with built-in functions. The neural network package has some built-in functionality for time series analysis. Both CPU and GPU can be used for training and using the models. (Anonymous, 2017l)

Programs written in Wolfram Language can be deployed in many ways, including but not limited to: cloud hosted web API, embedded programs, linkable library.

Pricing for Mathematica depends on what license level you buy. For a standard desktop individual professional license, Mathematica costs 3345€ (excluding VAT).

3.3 Machine Learning as a Service

Machine Learning as a Service (MLaaS) is a new system where everything from training the models and deploying them in end application is managed

by the provider. Usually they run proprietary algorithms, but they can also be just hosting providers for running open source implementations.

3.3.1 Azure Machine Learning

Azure Machine Learning is a Machine Learning as a Service (MLaaS) provided by Microsoft. It provides a framework for creating and running predictive machine learning models in the cloud.

Azure ML is a fully managed service, where Microsoft provides all the servers infrastructure required for training and running the models. Models are created in Azure Machine Learning Studio, which can be accessed using a web browser. Some of the distinguishing features of Azure Machine Learning are collaboration capabilities, automatic versioning and visual workflow. (Anonymous, 2017c)

Figure 3.1 shows the basic workflow in Azure ML.

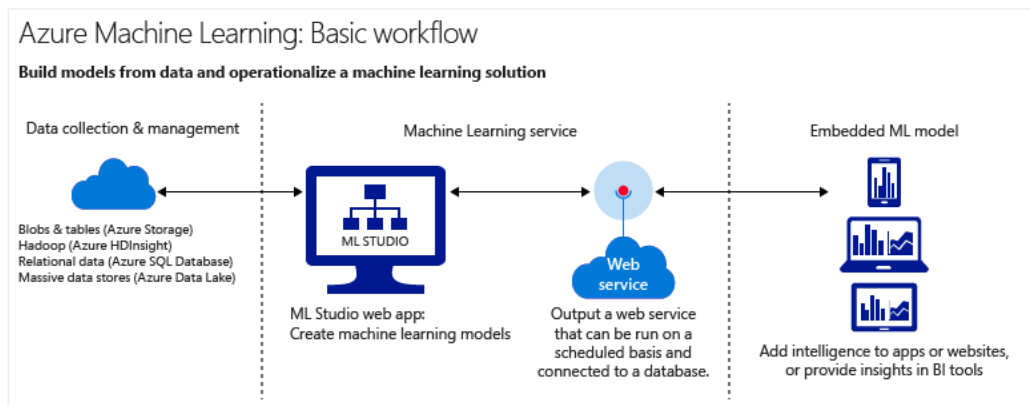


Figure 3.1: Azure machine learning workflow. (Anonymous, 2017c)

Azure ML studio supports data input from the following sources (Anonymous, 2017c):

- A Web URL using HTTP
- Hadoop using HiveQL
- Azure blob storage
- Azure table
- Azure SQL database or SQL Server on Azure VM

- On-premises SQL Server database
- A data feed provider, OData currently

Azure ML has a large set of algorithms to choose from. In addition to the built in data manipulation and machine learning blocks, Azure ML Studio also supports using Python and R programming languages for processing the data.

After the mode has been created and trained using Azure ML Studio, it can be deployed as a web service. These web services can be queued in real-time or batches of 1000 predictions.

Azure ML doesn't currently have straightforward integration with OPC UA, but Microsoft's other product, Azure IoT Suite, recently received OPC UA support (George, 2016).

3.3.2 Amazon Machine Learning

Amazon Machine Learning is a fully managed MLaaS provided by Amazon. It provides a framework for creating and running predictive machine learning models online.(Anonymous, 2017b)

As of writing this, the Amazon ML only supports input as comma separated values (csv) files from Amazon S3 service or from Amazon Redshift, a managed data storage service.

Amazon Machine Learning uses the pay for use model, with no minimum fees or upfront costs. Cost is determined by how much computing time is used to train the models and how many predictions are generated.(Anonymous, 2017b)

3.3.3 Google Cloud Machine Learning

3.3.4 BigML

BigML is a fully managed MLaaS. It provides end-to-end solutions for most machine learning tasks.

BigML offers some generic machine learning algorithms, such as those discussed in Chapter 2, but they are mostly focused on their proprietary models and algorithms. For regression and classification tasks, BigML offers their proprietary tree based algorithms. For clustering they offer their proprietary implementation of K -Means. No research was found on the performance and maturity of their proprietary machine learning models. (Anonymous, 2017d)

BigML supports many types of remote data sources, but data formats supported are limited to CSV, JSON, and ARFF.

What sets BigML apart from other MLaaS providers like Microsoft's AzureML, is that they provide totally private deployments for customers. This enables the companies to have more confidence in their data security, as the data will never have to leave their own server rooms, unlike when using cloud based machine learning services where your data is sent to the cloud. BigML can also be hosted on the customers choice of cloud hosting, such as Azure or Amazon AWS. (Anonymous, 2017d)

Hosted BigML pricing is tiered, starting from free and going up to \$10,000 per month. The highest tier provides unlimited storage, 64 GB max dataset size and 64 parallel tasks. Private cloud or on-premises deployment starts from \$45,000 per year for 1 server / 8 cores and goes up to \$2,250,000 per year for unlimited servers. (Anonymous, 2017d)

3.3.5 IBM Watson Machine Learning

IBM offers MLaaS solutions under the Watson Machine Learning brand. It is focused on providing a set of REST APIs for managing machine learning model deployments. It does not offer advanced tools for data analysis or model training, and training models requires writing your own programs in Python or Scala. (Anonymous, 2017f)

Watson Machine Learning supports deploying of models created in Apache Spark MLlib, scikit-learn and IBMs proprietary SPSS Modeler. (Anonymous, 2017f)

The service is priced in three tiers: Free, Standard and Professional. Free plan includes 5000 predictions and 5 compute hours, Standard plan costs \$0.50 / 1000 predictions and \$0.45 / hour of computer hours. (Anonymous, 2017f)

Chapter 4

State of the art in machine learning in process industries

This chapter describes the state of the art in machine learning in process industries, including soft sensors, process monitoring, fault detection and predictive maintenance.

4.1 Soft sensors

In industrial processes, some variables will always be difficult to measure directly. Some of these difficult to measure variables are also closely related to the final product quality. Examples of difficult to measure variables in oil refining are distillation end points and fuel cold properties (freeze and cloud points).

Soft sensors (sometimes smart or virtual sensors) are software sensors that instead of measuring a variable directly, use predictive models, either white-box models based on the underlying physical and chemical principles, or data-driven black-box models, to calculate desired variables from a process. Models based on first principles are difficult to develop for massive scale oil refining operations, which gives motivation for data-driven approaches. In this section we will take a look at machine learning methods for soft sensors. (Kadlec et al., 2009)

Soft sensor development process closely follows the procedure described in section 2.1. Figure 4.1 shows a general picture of how data-driven soft sensors are developed. Data collection is important, and the data used to train the machine learning model should cover a large amount of probable operating conditions. In variable selection, we wish to reduce the amount of variables used in the soft sensor to avoid the "curse of dimensionality". The

variable selection usually involves both preliminary expert knowledge of the process with some trial and error.

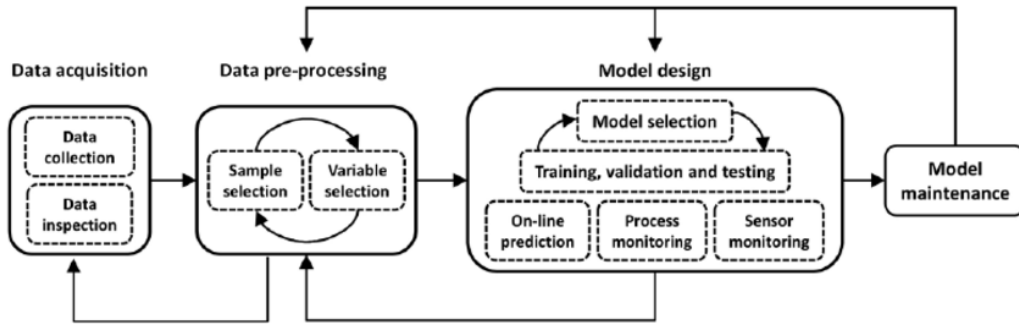


Figure 4.1: Soft sensor development and maintenance workflow (Haimi et al., 2011)

The original and still most important use of soft sensors in process industries is to predict process variables that can only be determined at low sampling rates or through off-line analysis only. As these variables are often related to the product quality, having them available is very beneficial for process control. (Kadlec et al., 2009)

First generation data-driven soft sensors were built with the traditional offline machine learning approach, where historical data is used to train a model for the soft sensor and then they were only used online. Complex processes where operating conditions are changing constantly require online adaptive soft sensor modeling for optimal performance. (Kadlec et al., 2011; Ge and Song, 2010)

Changes in processes can include process input (raw material) change, equipment wear and catalyst deactivation. The earliest adaptive mechanisms were based on moving window with a maintenance mechanism that depended on laboratory results. (Kadlec et al., 2011)

Many machine learning methods have been applied to smart sensors. According to Kadlec et al. (2009), PCA, partial least squares, various forms of ANNs, and SVMs are popular.

Gonzaga et al. (2009) used a simple feed forward ANN to provide on-line estimates of polyethylene terephthalate viscosity.

Shang et al. (2014) introduce a novel semi-supervised deep learning technique for data-driven soft sensors. Their method is applied to estimate the heavy diesel 95% cut point of a crude distillation unit. Their method was much more accurate when compared to traditional one hidden layer ANN,

SVM or PLS. Training the deep learning model takes longer than other models, but was still completely reasonable at less than 5 minutes on a modern consumer computer. Prediction computation time with the neural network stays constant after the architecture has been decided, whereas the complexity of SVM model increases as more training data is used and eventually becomes too slow for real time prediction.

Yan et al. (2017) present a deep learning denoising autoencoder neural network (DAE-NN) approach for soft sensors. Denoising autoencoders goal is to be able to reconstruct the original input data from corrupted (noisy) input data. The neural network was applied in a case study to predict oxygen content in flue gases of a coal-fired power plant. Twenty two variables were used as inputs to the neural network, 7420 process datasets were used in training and 3710 in testing. Their approach proved to be more accurate than currently much used SVM regression and simpler neural network architectures, even with PCA pre-treatment for the data. The downside for their DAE-NN was the highest computational time used for training.

Pani et al. (2016) used a PCA and an ANN to develop a soft sensor for measuring butane concentration from a debutanizer column bottom product, which is the same process and estimated variable that will be explored in the experimental part of this thesis.

4.2 Process monitoring and fault detection

Process monitoring is important for successful operation of a process. Process upsets, equipment failures and events that might affect product quality must be detected early. (Aldrich and Auret, 2013)

Both supervised and unsupervised machine learning methods are used in process monitoring and fault detection problems. (Ge et al., 2017)

The most used machine learning methods in data driven process monitoring applications have historically been PCA and other nonlinear dimensionality reduction algorithms, such as nonlinear PCA and kernel PCA (Aldrich and Auret, 2013). The linear features extracted by PCA are then used with statistical methods, usually Hotelling's T^2 and Q -statistic to detect abnormal behavior. (Qin, 2012)

Soft sensors are also employed in process monitoring, as they can be used to predict the product quality. If the predicted product quality falls below certain limits, the process is considered to be abnormal. (Qin, 2012)

Figure 4.2 shows a diagram of how PCA and soft sensors (PLS in this case) are utilized in process monitoring.

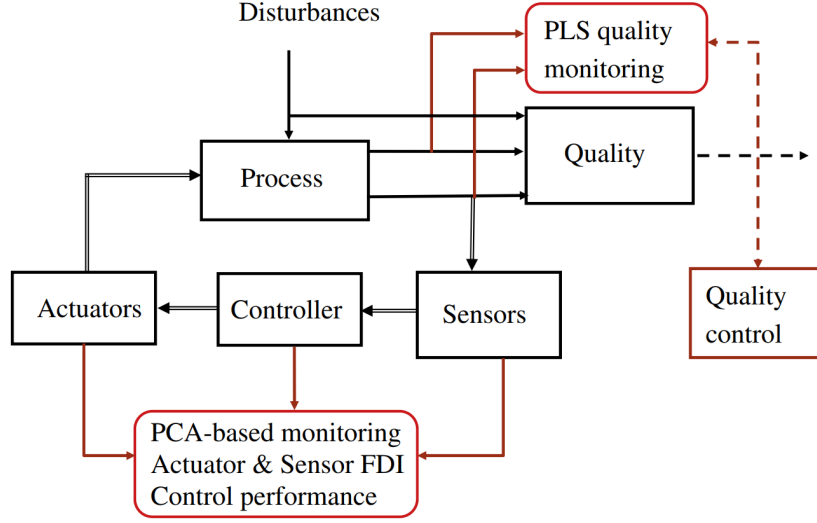


Figure 4.2: Process and product quality are monitored using PCA and PLS soft sensor (Qin, 2012).

If abnormal conditions in a process are detected, it is desirable to know what caused the fault. Fault identification is the process in which after abnormal process conditions are detected, the cause of the fault (faulty sensor, actuator, or other) is identified. (Qin, 2012)

4.2.1 Predictive maintenance

Unscheduled shutdowns of manufacturing processes are very expensive and potentially dangerous. The most common method of maintenance for important systems is conventional preventive maintenance based on a schedule determined on technicians experience or statistics on how often the system failed in the past. Most systems that must be depended on are serviced on a schedule that is usually more frequent than would strictly be necessary, to make sure that the system does not fail. This over eager maintenance results in wasted resources. (Aldrich and Auret, 2013)

Predictive maintenance refers to a maintenance scheme where maintenance is performed based on the actual level of wear in the equipment, instead of statistically determined schedule. Machine learning enables the identification of failure signatures from historical data, providing a way to determine level of wear. Predictive maintenance aims to give insight on what is the probability of equipment failure in the near future, and what is the remaining lifetime of the equipment before failure. For predictive maintenance to

work, sufficient amount of historical data which shows previous failure modes is required to support prediction. (Aldrich and Auret, 2013)

4.3 Reinforcement learning for process control

Designing good process control systems is a time consuming and difficult task. It requires careful analysis of the process and detailed mathematical modeling. (Spielberg et al., 2017)

Reinforcement learning in general was discussed in section 2.2.3. Spielberg et al. (2017) presents how reinforcement learning could be applied in process control applications. Their method doesn't require time consuming modeling and automation design process, instead the model would learn the behavior of the process directly by interacting with it. This means the model can be applied to many different kinds of processes with minimal work, and the model could automatically learn new control policies after changes to the process.

Chen and Wang (2014) showed how reinforcement learning can be used to optimize operation conditions for cost on a CO₂ capture process using combined information from simulation and real process data. Simulations are easier and cheaper to perform than experiments with real equipment, and give access to process variables that would be inaccessible in the real process.

Chapter 5

Machine learning framework requirements for process industries

To fully utilize machine learning in production, many subsystems (data gathering, preprocessing, model training, and deployment) are required to work together. A framework combines all of these subsystems in a larger single system that makes developing machine learning solutions easier. Defining requirements for such a system gives software developers a clear goal to work towards. (Hastie et al., 2009)

This chapter will discuss the requirements for an (OPC UA) based machine learning framework for process industries. OPC UA and its useful features for the framework are presented. Algorithms and software discussed in the earlier chapters are evaluated with process industry use in mind.

5.1 Process data acquisition and storage

The first requirement for any machine learning is acquiring training data. This section will take a look at how OPC UA can be used for gathering measurement data, accessing historical data and reading and writing measurements in real time.

5.1.1 OPC Unified Architecture

OPC UA is a platform-independent communication standard for all industrial domains. It was created to replace the original OPC specification, which was locked to Microsoft Windows platform. The main goal of OPC UA was to provide interoperability between systems of different manufacturers,

rich information modeling capabilities, reliable communication and Internet access, which necessitates strong security.

5.1.1.1 Historical data access

To accurately train machine learning models to predict most possible cases, lots of data from the process is needed. In some cases over very long time period, even years. The OPC UA standard defines historical data access for gathering and accessing OPC UA data. (Anonymous, 2015)

The OPC UA standard defines methods for easy access to historical data over any time range that data has been historized in the OPC UA server. For a standard following client and server, the user need not know any implementation details but only call history read functions, which should be implemented by the client. (Anonymous, 2015)

5.1.1.2 Real-time access

In some applications such as soft-sensors, near real-time access to process measurements and operator display systems are needed. With OPC UA, implementing a soft sensor which reads process measurements from the OPC UA server as soon as they are updated, calculates the predicted value, then sends the value to OPC UA server is fairly simple using subscriptions and publishing. (Anonymous, 2017h)

5.2 Data preprocessing

Raw process data contains noise, outliers, missing values and wildly varying scales between different measurements. Data quality is important factor to success of machine learning projects. It can have a significant result on the performance of the machine learning algorithm, and poor performance of a machine learning model can often be improved with better data preprocessing. (Kotsiantis et al., 2006).

Data transformations tools such as standardization and normalization are a major part of preprocessing, as a common requirement for many algorithms is for the data to look like standard normally distributed data with zero mean and unit variance. Other sometimes useful data transformations are discretization and binarization of data, which help reduce the possible values of continuous features which speeds up machine learning with massive datasets. (Pedregosa et al., 2011; Kotsiantis et al., 2006).

5.3 Machine learning methods for process industry applications

Process industry is a very diverse field with many possible applications for machine learning. The methods range from the simple least squares data fitting for linear models (Opfell and Sage, 1958) to deep learning for soft sensors measuring product quality (Yan et al., 2017).

Linear models with LS or PLS are still useful, and should be considered before nonlinear models (Souza et al., 2016; Opfell and Sage, 1958).

PCA is a popular method for dimensionality reduction to avoid the curse of dimensionality. Many processes easily contain tens of measurements, and using PCA for preprocessing the data has been shown to increase the accuracy of the resulting machine learning model. (Aldrich and Auret, 2013; Bishop, 2006; Kadlec et al., 2009)

ANNs have become very popular for all machine learning applications, and the same holds for process industry applications. Simple ANNs with PCA have been used for soft sensors (Kadlec et al., 2009), though in the past few years, deep learning methods have been at the forefront of data-driven soft-sensor research. (Shang et al., 2014; Yan et al., 2017). Studies have also shown potential for usage with fault detection & diagnosis (Aldrich and Auret, 2013) and process control with reinforcement learning (Hoskins and Himmelblau, 1992).

Scikit-learn implements most of the methods discussed in this section, but lacks in deep learning capabilities (Pedregosa et al., 2011). TensorFlow and Pytorch are more specialized towards deep neural networks, but don't implement any other machine learning methods. (Abadi et al., 2015)

In conclusion, almost all the popular machine learning methods have uses in process industries, and a framework should implement as many of them as possible to make testing different methods for a particular problem easier.

5.4 Model training

Depending on the machine learning model and amount of data used for training, model training might be fast and straightforward and require very little computing resources, or it might take days on a very powerful supercomputer.

Some companies have developed dedicated machine learning hardware. Google developed an ASIC (Application-specific integrated-circuit) called Tensor Processing Unit (TPU) (Sato et al., 2017). ASICs are integrated circuits designed for very specific use, which usually makes them much more

efficient than a generic CPU at that specific task. The second generation of TPUs, called Cloud TPUs, are also capable of ANN training and prediction. (Smith, 1997; Sato et al., 2017)

Machine learning with massive datasets which cant fit into the main memory of a single computer (Typically 8 GB for lower end machines, up to 64 GB for powerful desktop PCs and the largest amount of memory offered by AWS for a single computer instance is 2 TB (Anonymous, 2017a)) require specialized distributed algorithms (Leskovec et al., 2014). In Chapter 3 Apache Spark and H₂O were the only software packages that provided distributed machine learning algorithms. Dataset sizes in process industries can vary from fairly small to extremely large.

MLaaS providers remove the need for worrying about the hardware requirements of machine learning. (Anonymous, 2017c,b).

5.5 Model deployment

Generally machine learning model deployment is much easier than model training. Generating predictions with a trained machine learning model usually only requires a couple of simple calculations which almost any modern device can perform in real time with no problems.

Most programming languages offer serialization interfaces that allows the developer to export objects in some easily storable / transportable format (byte stream, text) and later recreate the original object from the serialized data. Scikit-learn and H₂O advocate usage of object serialization for model persistence and deployment. (Pedregosa et al., 2011; Anonymous, 2017e)

In the case of ANNs, model deployment is much easier than training. A trained ANN model is just a map of neurons, their connections and weights, which can be easily implemented in any programming language. Graphical processing units (GPU) are much better at parallel operations than Central processing units (CPU), which allows them to quickly execute operations required in ANNs, but programming for GPUs is more difficult, and not all neural network implementations support GPU acceleration. For most simple ANN models, CPU based implementations are fast enough for real time deployments, but for sufficiently complicated Deep Neural Networks, even deployment might require specialized hardware for real-time applications. (Jang et al., 2008)

In MLaaS solution, deployment is often done via a web service that is hosted by the service provider. This allows for very easy deployment to many products that have Internet connectivity, but rules out offline applications. OPC UA can be made to work with web based machine learning deployments

with software that communicates with both the service and OPC UA server, if some latency is acceptable. (Anonymous, 2017c,b).

Especially for soft sensors in process industries, automatic comparison between laboratory data and soft sensor output could help detect problems in the models.

5.6 User interface

A simple to use User Interface (UI) greatly benefits the usability of a framework. There are many kinds of possible UI designs, ease of use, configuration options and development time have to be considered.

Model training and model deployment are two separate tasks and should be done in two different applications. Training requires a lot of functionality that is unnecessary during deployment. Trained models are most easily transferred as serialized objects, (eg. Python's pickle module), but human readable model representations would be advantageous in some situations.

Data visualization is critical for any machine learning task. It is the easiest and fastest way to spot problems in data, decide what variables to use for prediction, and other data shape related tasks. During this thesis, I used NAPCON Information Manager for visualizing historized time-series data directly from the OPC UA server.

5.6.1 Model Training UI

A very powerful but difficult to use UI is to programmatically use the APIs of the machine learning and OPC UA libraries. This allows for the greatest control of what is happening with the data, but is very time consuming and error prone. It also has the greatest barrier for use, as programming knowledge is required. No UI development is necessary.

A minimal UI would be a configuration file that lets the user customize some parts of the process, such as input and output variables, the model used and its parameters. A configuration file might be sufficient for some specialized tasks, but not for a more complete software solution, as having a configuration file that allows for all the possible settings that should be controllable would be large and complicated to navigate. Configuration files are easy to implement and most programming languages have built in support for some format. Figure 5.1 shows what a configuration file supported by the Python programming language for a machine learning soft-sensor model training task might look like.

```

[Server]
address = opc.tcp://localhost:51312/
[Inputs]
nodeids = i=1234, i=1235, i=1236
[Predicted value]
nodeid = i=1237
[Training data]
begin = 2000-01-05 00:00:00.0000
end = 2000-01-05 10:36:00.000
[Test data]
begin = 2000-01-05 10:36:00.000
end = 2000-01-06 15:00:00.000
[Model]
#Supported models linear.ridge , linear.ls , MLPRegressor
model = linear.ridge
[Model params]
alpha = 0.5

```

Figure 5.1: A configuration file in the Python format

Having a graphical user interface (GUI) for the framework would greatly benefit usability and decrease machine learning task design time. Most people are used to GUIs, and in general, they provide the user with the most information and allow for fast development. GUI development is very costly, but will probably pay itself back with usability improvements over the alternatives.

Out of the MLaaS providers discussed previously in this thesis, Microsoft Azure Machine Learning Studio offers the most graphically advanced UI. Other MLaaS providers like Amazon, BigML and IBM focus more on providing APIs for programmatically controlling the machine learning services, and maybe simple form based tools for creating models and data visualization. Azure ML Studio offers a powerful visualization that shows all the steps in the model training process in a small footprint and easy to understand way. Figure 5.2 shows a sample case where data is preprocessed and split into training and test set, and two different clustering algorithms are trained.

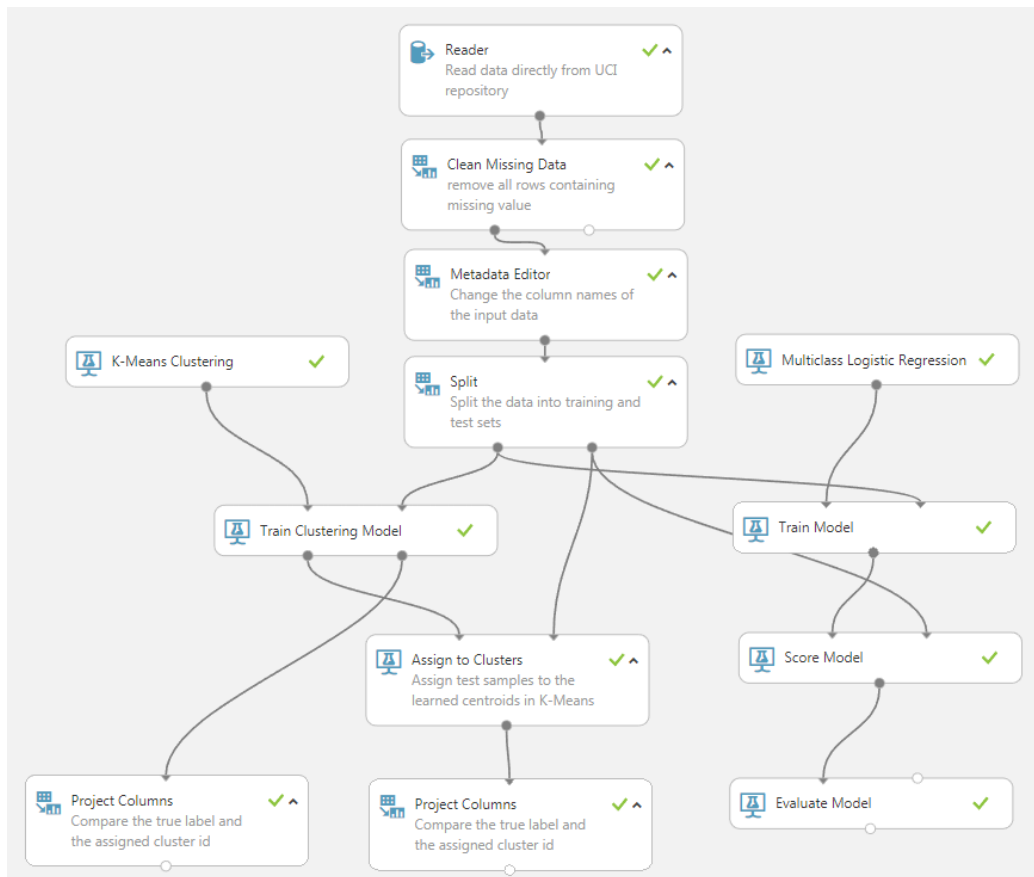




Figure 5.2: Azure Machine Learning Studio UI shows how the data flows through preprocessing, model training and model validation

Machine learning model blocks, like the K-Means Clustering block, have easy to use settings, shown in Figure 5.3. Appendix B shows data visualization tools.


▲ K-Means Clustering

Number of Centroids 


3

Metric 

Cosine ▼

Initialization 

K-Means++ ▼

Iterations 

200

START TIME 7/14/2015 11:07:41 AM

END TIME 7/14/2015 11:07:42 AM

ELAPSED TIME 0:00:01.170

STATUS CODE Finished

STATUS DETAILS None

[View output log](#)

Figure 5.3: Changing model parameters in Azure Machine Learning studio

Chapter 6

Implementation of an OPC UA enabled machine learning framework

The main focus of the experimental part is building a general machine learning framework for OPC UA data based on the requirements discussed in the earlier chapter. The most important topics related to the experimental part of the thesis from the literature review are Soft Sensors in Chapter 4.1 and process data acquisition and storage in Chapter 5.1.

The framework will be tested with Neste Engineering Solutions' NAPCON Informer OPC UA server and the data for machine learning is gathered using NAPCON Simulator. The evaluation will be based on how well the framework meets the requirements discussed in Chapter 5 and how well the models perform.

6.1 Choosing a machine learning library

The choice of what machine learning library to use depends a lot on the intended application. As this was to be a small demo rather than a large scale deployment, libraries intended for large scale distributed computing systems such as Apache Spark and H2O were left out.

Deep Learning focused libraries are usually quite complicated to use, especially without prior knowledge of neural networks. Providing an easy to use UI that also allows for the finesse of control allowed and even required by these deep learning libraries would be complicated.

Proprietary software (MATLAB, Wolfram Mathematica) will not be used, due to cost and academic reasons.

Scikit-learn was chosen for this demo because of many reasons. It is a mature project that has been in development for over 10 years, and still receives regular updates. It has a large library of machine learning algorithms with a well designed and simple API, which allows for simple UI that can be used without a lot of prior machine learning and programming. Almost all models in scikit-learn implement fit, predict and score methods which makes it easy to quickly prototype with different models.

Scikit-learn has been used by many large IT companies and institutions in production environments, including but not limited to Spotify, Booking.com and Inria. User testimonials of software libraries for industry use are much harder to find in general, for any. The author has prior knowledge in the Python programming language, which is used with scikit-learn.

6.2 Architecture of machine learning framework

After scikit-learn was chosen as the machine learning library, the architecture around it had to be designed to allow for interfacing with OPC UA server to read process data and write predictions and giving the user some kind of interface to choose process variables and machine learning model and parameters.

Early on the decision was made to develop two separate programs, the model training program and model deployment program. The model training program is used to rapidly prototype different machine learning models with data input from OPC UA server, while the model deployment program is used to generate predictions in real time with live OPC UA data. This separation allows for easier development of the system, and errors in the model training program are less likely to influence the more critical deployment program. Figure 6.1 shows an overview of the whole architecture.

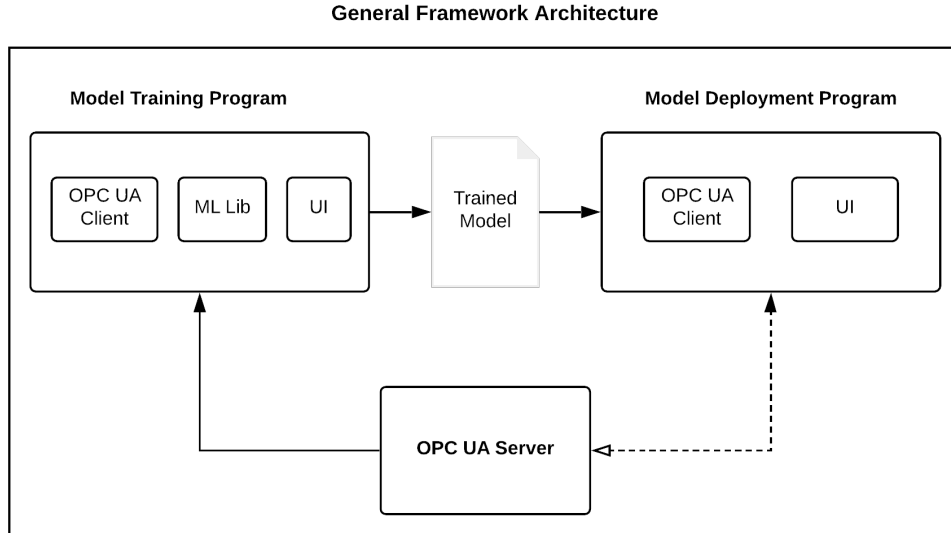


Figure 6.1: General architecture of the machine learning framework showing the separation of training and deployment.

Transferring the trained models between the two programs is handled via object serialization. Object serialization (named *pickling* in Python) is the process of transforming objects in program memory to a format that can be stored, for example in a file. Deserialization (*unpickling* in Python) is the opposite process, where objects can be created from a file, as they were at the time of serialization. (Anonymous, 2017i)

When the model training program is successfully executed, it serializes the trained model and saves it to a file which can then be deserialized in the deployment program and generate predictions. Information about the trained model, such as creation date, type of machine learning model and model parameters, is saved in a separate text file.

6.2.1 OPC UA connection

The OPC UA connection must be handled in both model training and deployment programs. Implementing the OPC UA protocol from scratch is a large task, and not necessarily required, as many open source implementations exist. In contrast to the quantity and quality of open source machine learning libraries, not quite as many open source OPC UA libraries exist,

and their quality in general is not as high. This can possibly be explained by the generally more in-house approach of process industry companies, and how new the OPC UA standard is.

For the purposes of this thesis, a Python based open source OPC UA implementation FreeOpcUa was chosen. The simplest reason was because scikit-learn, the machine learning library used for this thesis, only supports Python API officially. Working with only one language simplifies the system considerably. FreeOpcUa is also licensed under LGPL (GNU Lesser General Public License), which permits commercial usage of the library and doesn't require publishing your own source code (Anonymous, 2007b).

The important OPC UA features used in the framework are reading the historical process data in model training, and subscribing to nodes and publishing generated predictions in the model deployment.

6.2.2 Model training

The model training program contains a few key parts that will be separately discussed in this section. Figure 6.2 shows a general overview of the program.

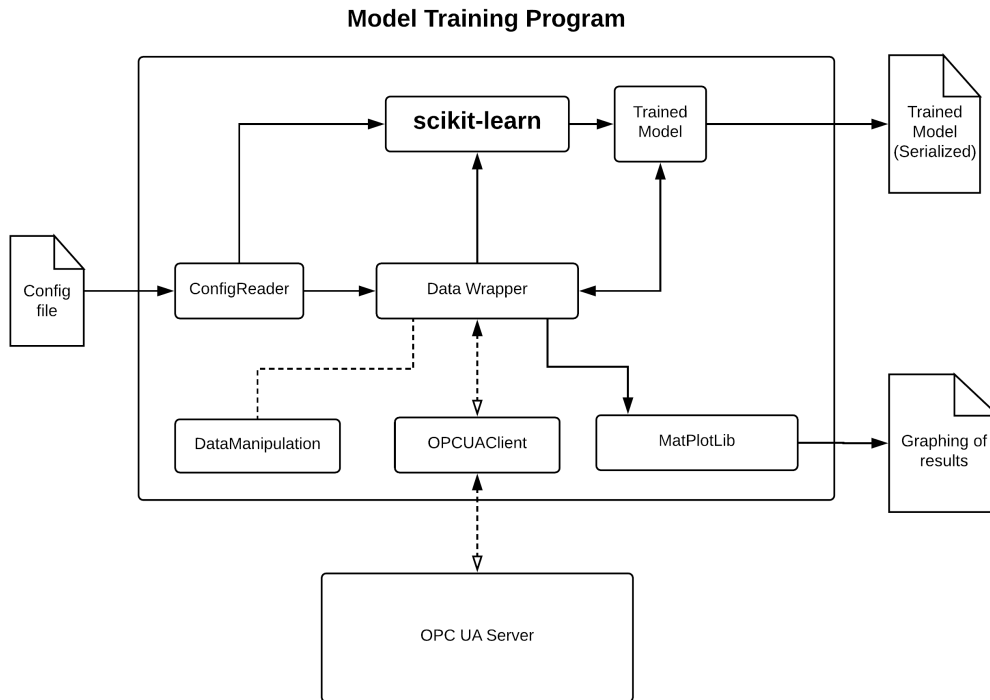


Figure 6.2: General architecture of the model training program.

ConfigReader handles reading the configuration file, which contains all the user configurable parameters. It is simple and internally uses Python's built-in configparser which can easily read human editable configuration files. Section 6.2.4 explains what the configuration contains.

DataWrapper class is the most important part of the system, as it handles moving the data between OPC UA client and machine learning library. Raw OPC UA data contains a lot of variables (timestamp, measurement unit, etc.) which are not compatible with scikit-learn, but must still be tracked. It provides methods (using **DataManipulation** class) for getting the training and testing datasets as scikit-learn-compatible NumPy arrays. The timestamps units are later used for plotting.

DataManipulation class is a helper for **DataWrapper**, which provides

DataWrapper the data preprocessing tools for manipulating the data to fit scikit-learn and also performs data normalization and inverse normalization if so specified by the user.

The task of the OPC UA Client class in model training is to retrieve historical node value data. For machine learning purposes, we want the same number of samples from all measurements. Some tricks have to be performed to get same number of measurements at specific timestamps.

The OPC UA standard specifies ReadAtTimeDetails functionality for reading a node values at specific time stamps, interpolating between two measurements if necessary (Anonymous, 2015). ReadAtTime is provided because OPC UA performs compression of data by not storing new values constantly if the difference between samples wasn't large enough. However, FreeOPCUA and Informer do not seem to support this feature together. Instead, ReadRaw functionality will be used. It returns all historical values of a node between specified startTime and endTime. ReadAtTime equivalent functionality was implemented manually by first using ReadRaw and then writing the code for linear interpolation.

After the specified model training and verification data has been retrieved from the OPC UA Server, it transferred to scikit-learn for model training. Parameters for model training are read from configuration file. The trained model is saved to a file, and Matplotlib is used for plotting the results.

6.2.3 Model deployment

As discussed in the general architecture, the model deployment program was implemented as a separate entity from the model training program. After a suitable model has been developed with the model training program, the serialized model is transferred to a server that should continuously run the soft sensor. Figure 6.3 shows the general architecture of the deployment program.

The model training program reads the serialized soft sensor model containing the actual fitted machine learning model, plus possible scalers which are used to transform the data prior to using the models (Data transformations were discussed in Chapter 2.1). Config file contains information about which OPC UA nodes to use for inputs and output. The order of process variables must match with model training step, but node id's do not have to match, the reason is that the trained model can be used in a different environment from where it was trained.

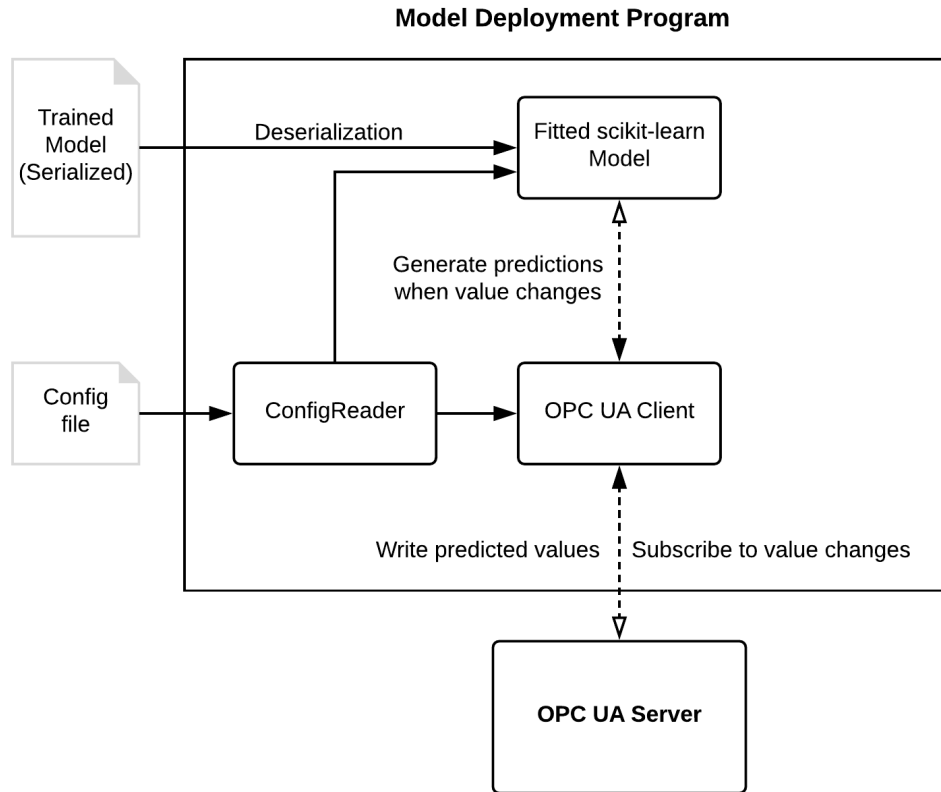


Figure 6.3: General architecture of the model deployment program.

To acquire real time data from the OPC UA server, the implementation uses the subscription service which is defined in the OPC UA Services specification. This way the programmer doesn't have to poll the values continuously. When a new value is received, a new thread is launched for calculating the soft sensor output and writing it to the OPC UA server.

6.2.4 User Interface

The user interface is text based with no graphical user interface, other than plotting provided by Matplotlib. All user input is read from a configuration file, and the program is executed without any user interaction.

```

[Server]
address = opc.tcp://localhost:51312/UA/OPCUA_Server

[Inputs]
nodeids = ns=3;i=568780, ns=3;i=270288,
ns=3;i=483112, ns=3;i=331607,
ns=3;i=571684, ns=3;i=542644,
ns=3;i=490372, ns=3;i=331019

[Predicted value]
nodeid = ns=3;i=705642

[Training data]
begin = 2000-01-05 00:30:00.000
end = 2000-01-06 00:30:00.000

[Test data]
begin = 2000-01-06 13:30:00.000
end = 2000-01-07 03:00:00.000

[Model]
#example is from
#http://scikit-learn.org/stable/modules/generated/
#sklearn.neural_network.MLPRegressor.html
model_module = neural_network
model_name = MLPRegressor
model_params = hidden_layer_sizes = 16,
activation = 'tanh'

```

Figure 6.4: A configuration file in the Python format

The configuration file is very self explanatory. In the Server block, **address** points to the OPC UA server which is to be used. In the Inputs block, **nodeids** is a list of nodes (measurements) that are to be used as the inputs for training the machine learning model.

The implementation supports all regression supervised learning algorithms in scikit-learn. For example, the **linear_model** module has many regression algorithms available, and they can be found in the API Reference (Anonymous, 2017m).

6.3 Process description

The debutanizer is part of a hydrodesulfurization (HDS) process for straight-run gasoline, known as BERP3 at the Porvoo refinery. In the process, the straight-run gasoline goes through the debutanizer, through preheating to the actual HDS reactor. After the HDS reactor the almost sulphur free gasoline flows to the fractionation unit, where the gasoline is separated into heavy naphta, hexane, n-pentane, isopentane and unstabilized C_3 / C_4 fraction. Figure 6.5 shows a flowsheet of the HDS process.

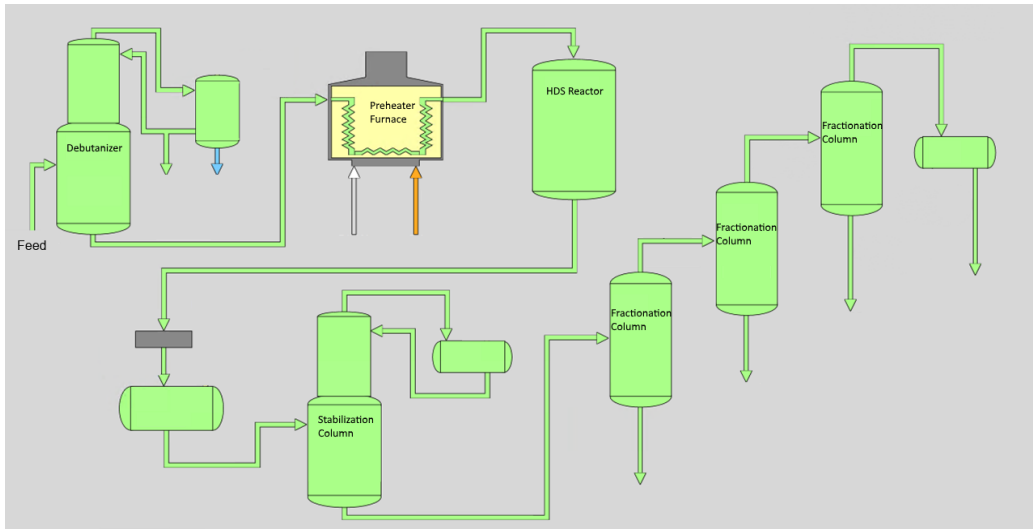


Figure 6.5: Hydrodesulfurization process including the debutanizer column

The task of the debutanizer is to remove butane and lighter hydrocarbon compounds from the feed. The debutanizer column contains 40 trays. The feed enters at tray 17.

A multivariate controller is used to control the top and bottom product quality of the debutanizer column. Currently, pressure compensated temperature of tray 36 is used to control the concentration of butane in the bottom product. A small amount of butane should be allowed to enter the bottom product so that harmful substances such as dimethyl sulfide (DMS) would not enter the distillate. Pressure compensated temperature of tray 10 is used to control the concentration of pentane in the distillate.

The concentration of butane in the bottom product of the column is proportional to the pressure compensated temperature of tray 36. The temperature should be low enough to minimize the concentration of DMS in

the distillate, but high enough to contain a buffer for changing feed quality. The concentration of pentane and DMS in the distillate is proportional to the pressure compensated temperature of tray 10. In order to minimize the DMS concentration, the temperature should be as low as possible. The optimal temperatures for the trays were determined in test done on the real unit.

6.4 Gathering training and test data

Instead of using real process data to train the machine learning model, data from a simulator is used. Using a simulator gives the possibility of easily running the process in different conditions, and the simulator always knows the butane concentration in all streams in real time, unlike in the real process where butane concentration is measured with a considerable delay.

Simulations were carried out in NAPCON ProsDS dynamic process simulator, proprietary of Neste Engineering Solutions. ProsDS is a dynamic simulation software intended for rigorous simulation of process and automation models.

In real world, the butane content would have to come from laboratory results. This obviously gives a much lower sampling rate and less data to use for training, but over time, depending on how often samples are taken, the number of measured samples would grow to be sufficient. The laboratory sample information model in OPC UA should contain a timestamp to when the sample was taken.

6.4.1 ProsDS process model

The process model used is large and contains the whole hydrodesulfurization plant. Its intended use is as a large scale operator training simulator.

The debutanizer column is modeled as a normal distillation column. Distillation columns in ProsDS are modeled with stirred tanks for each tray. Appendix A shows a screenshot of the model inside ProsDS.

The original model was modified by adding an chemical composition analyzer to the bottom product line, which would measure the weight-% of butane and send it to the OPC UA server. This does not exist in the actual process unit.

A list of process variables that were gathered from the simulator are shown in Table 6.1. They were chosen with the help from an operator of the actual unit.

Table 6.1: List of process variables in the debutanizer column (anonymized)

Tag	Description
FC10	Feed flow rate t/h
FC11	Top return flow rate t/h
TC10	Tray 10 pressure compensated temperature °C
TC11	Tray 36 pressure compensated temperature °C
TC12	Feed temperature °C
TI10	Bottom product temperature °C
PC10	Top pressure kPa

6.4.2 Test run

Test runs were performed by programming the simulator to make very small changes to the setpoints of feed temperature, column top pressure, and tray 36 temperature in order to gather data at different operating conditions. As the process took very long time to settle between even small setpoint changes, and the simulator could only be run at approximately 1.5 times faster than real-time, test runs has to be performed over weekends. The simulator machine was also used for other development purposes during the week. Some overnight / weekend test run results were ruined by misconfiguration of the simulator system (not historizing important variables, historizing only 2 hours of data before overwriting), others by too drastic process changes which completely destabilized the simulator. Eventually, a test run containing mostly acceptable data was managed. Figure 6.6 shows the butane concentration of the debutanizer on a logarithmic scale from start to finish of the simulation.

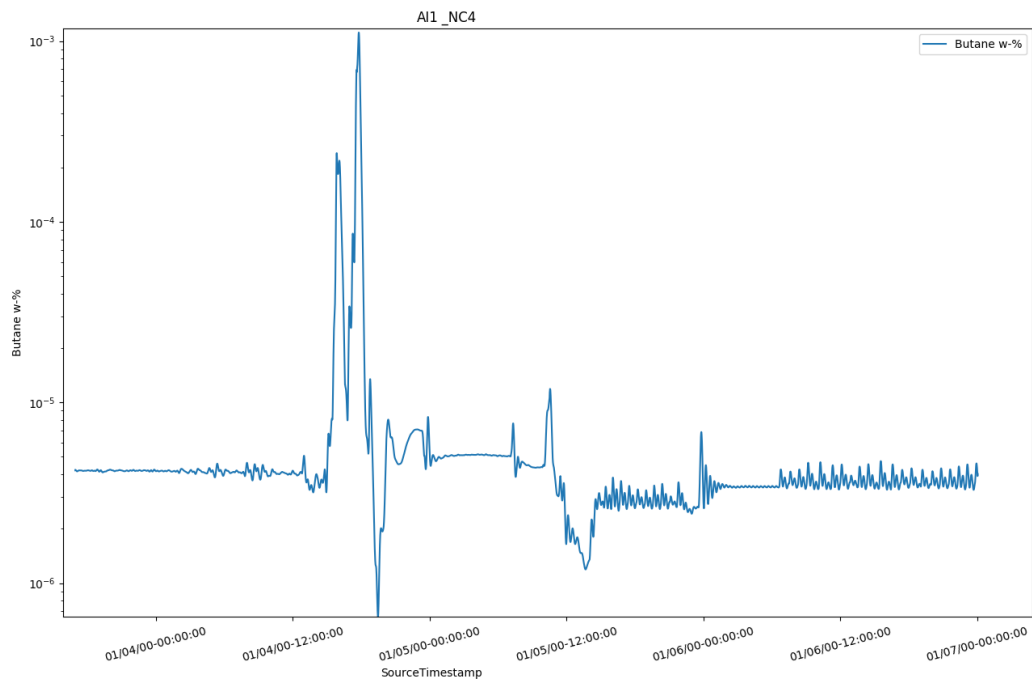


Figure 6.6: Debutanizer column output butane concentration on a logarithmic scale.

Some data between days 01/04 and 01/05 (SourceTimestamp) was unusable due to how much the butane concentration fluctuated. This can be seen more easily in Figure 6.7 if the butane concentration is plotted on a linear scale.

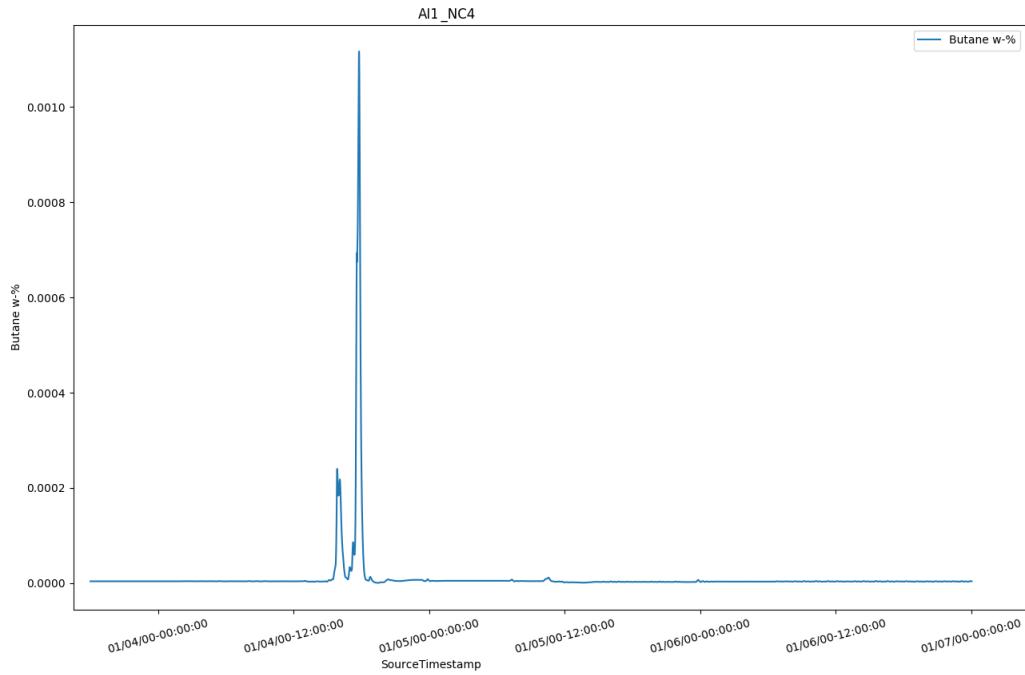


Figure 6.7: Debutanizer column output butane concentration on a linear scale.

In the end, the simulation produced approximately 55-60 hours of usable data for machine learning purposes.

The OPC UA server state including all history was saved and imported for use in the experimental parts of this thesis.

Chapter 7

Results and Discussion

A proof-of-concept framework was implemented in Chapter 6. The results will be evaluated in two ways, mainly on how well the requirements specified in Chapter 5 were met, and how well the machine learning models produced perform.

7.1 Framework requirements results

This section compares the implementation from Chapter 6 to the requirements and goals laid out in Chapter 5.

7.1.1 Process data acquisition and storage

Process data acquisition and storage were handled by NAPCON Informer OPC UA server. The model training and deployment programs communicate with the OPC UA server using OPC UA protocol, which means they should technically work with all standard conforming OPC UA servers.

Some problems with standard OPC UA features `ReadAtTime` and `ReadRaw` were discovered and had to be manually patched. Data acquisition from the OPC UA server for model training was slow, taking over a minute to fetch three days worth of process data (around 17000 samples). The slowness was most probably caused by bad programming on the client side and not the server side, as NAPCON Informer has been used and proved to function in applications requiring much higher data throughput.

7.1.2 Data preprocessing

Scikit-learn implements a lot of data preprocessing tools, but only standardization and normalization were implemented for usage through the configura-

tion file. Other scikit-learn features, such as feature binarization, categorical encoding and custom transformations can be easily configured with some programming knowledge.

7.1.3 Algorithms

All supervised learning regression algorithms available in scikit-learn can be used with the model training program. Large portion of the algorithms discussed in the requirements chapter are built-in to scikit-learn. All supported methods can be found in scikit-learn supervised learning API reference. Support for ones own machine learning algorithms could be added, they just need to follow the same API as scikit-learn. (Anonymous, 2017m).

7.1.4 Model Deployment

The model deployment program provides bare minimum command line implementation. None of the "extra" features discussed for model deployment, such as version control or automatic checks against older models or laboratory data.

The implementation had problems with relatively slow generation of predictions. We could not determine the reason for this, but it probably has something to do with writing to OPC UA server and threading, because predictions were generated much faster in the model training program for model validation. The slow prediction, combined with generating predictions every time a measurement changes, would definitely lead to problems with more measurements and fast updating of process variables.

7.1.5 User Interface

The developed used interface is crude but contains a decent amount of functionality for training machine learning soft sensor models. It succeeds in having a simple but powerful UI. The used model can be changed very fast, but choosing process variables and what data points are used for training (start and end times) is cumbersome.

What caused the most problems with the user interface was that the model training program had to be restarted every time the user wants to make changes to the machine learning model configuration. With large amounts of data and slow history read times, this often meant waiting around for the data to load. This could be fixed by allowing the user to change the configuration while the program is running and the data is already in memory.

7.2 Model performance evaluation

Machine learning model performance relies on many factors, but most importantly on algorithm choice, and the quality and quantity of training data. Performance evaluation is very important for delivering the best possible machine learning model. Model evaluation is used for determination of generalization performance, tweaking of model hyperparameters and comparing different machine learning algorithms.

Machine learning models are scored on a test dataset, which is totally separate from the training dataset and ideally represents the real world use case of the model well. Several mathematical methods are available in scikit-learn for model evaluation.

All estimators (trained machine learning models) in scikit-learn define a score method which calculates a default evaluation criteria for that specific model. For most supervised learning regressions models this is coefficient of determination R^2 . There are multiple definitions for R^2 . The version used in scikit-learn is shown in 7.1 and it is generally accepted as valid for evaluating model performance on a test dataset (Alexander et al., 2015).

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \quad (7.1)$$

where SS_{res} is the residual sum of squares:

$$SS_{res} = \sum_i (y_i - f_i)^2 = \sum_i e_2^i \quad (7.2)$$

and SS_{tot} is the total sum of squares:

$$SS_{tot} = \sum_i (y_i - \bar{y})^2 \quad (7.3)$$

where \bar{y} is the mean of data. R^2 is a statistical tool which gives indication on how well the model fits the data. Care should be taken when using R^2 with nonlinear models, as most nonlinear models can be made to follow training data arbitrarily close, producing R^2 values as close to 1 as wanted (overfitting).

7.3 Model performance

This section presents how the models trained with the framework performed depending on which algorithm was chosen and how much training data was used.

Generally model performance ranged from mediocre to bad. With non-linear models (Neural Networks, SVMs) training scores were often quite close to one but validation scores were never very good. Linear models often produced worse training scores, but better validation scores, which could be explained by overfitting of the nonlinear models.

A common problem encountered with nonlinear models in scikit-learn was offset from the validation data. Figure 7.1 shows how a neural network with a single hidden layer with 100 neurons follows the shape of the test data accurately, but has an almost constant offset, which causes the validation score to be poor. Similar effects can be seen with non-linear Support Vector Regression in Figure 7.2

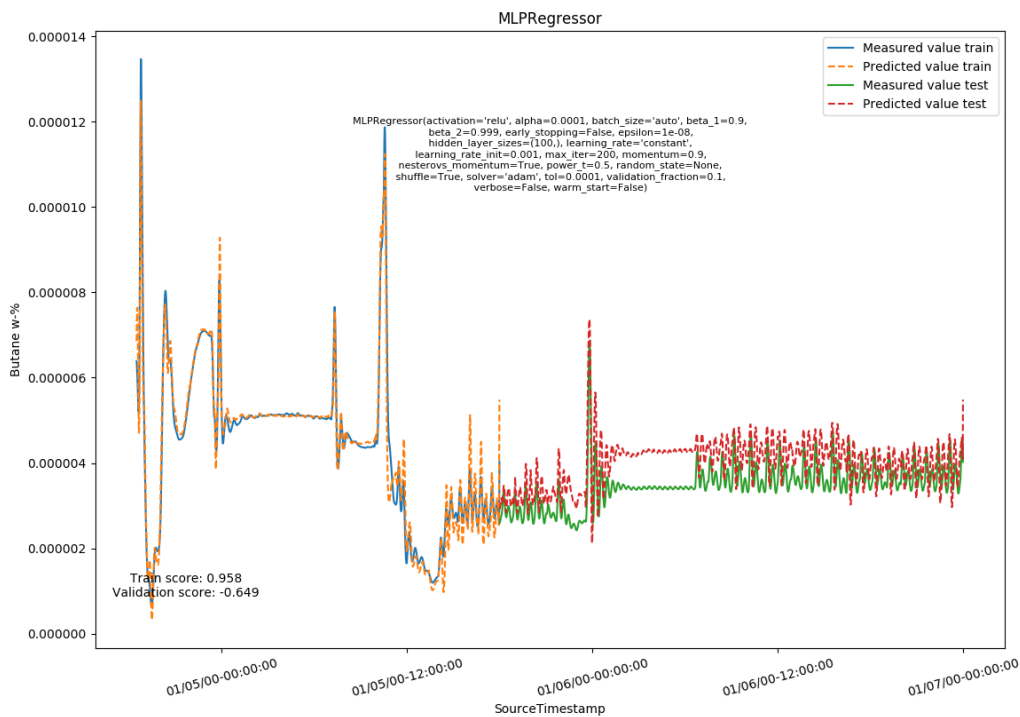


Figure 7.1: An MLPRegressor (Neural Network) trained with approximately 24 hours worth of simulation data.

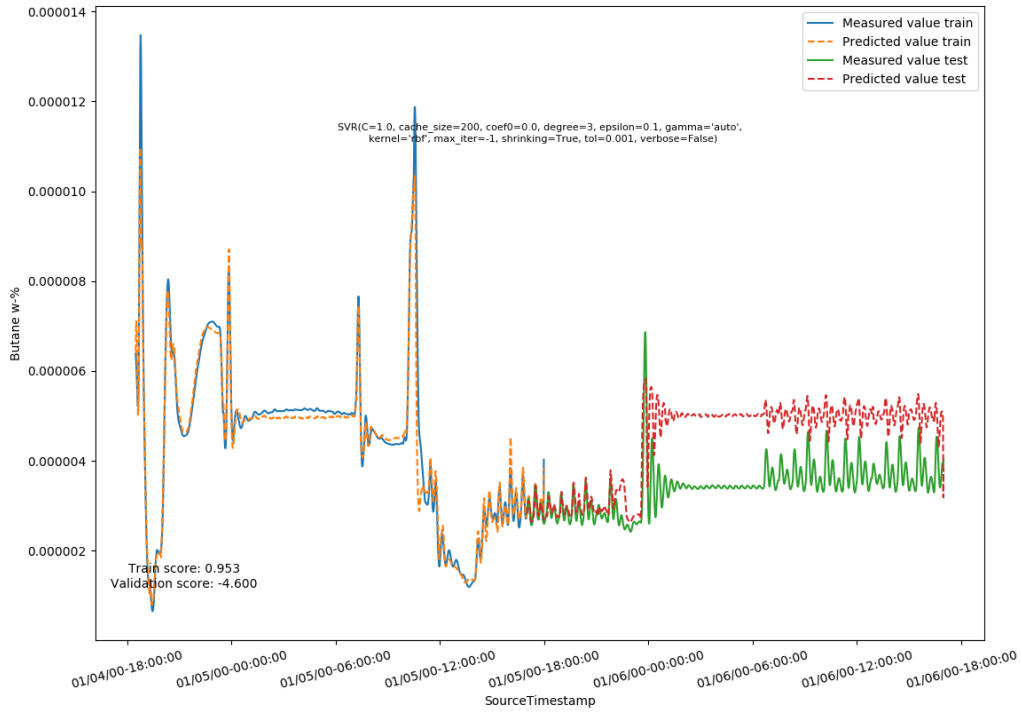


Figure 7.2: An SVR (Regression Support Vector Machine) trained with approximately 24 hours worth of simulation data.

On the other hand, linear models generally showed better performance with both following the data and smaller offsets with validation data, example shown in Figure 7.3 using scikit-learns HuberRegressor which is a form of robust linear regression, designed to control the effects of outliers.

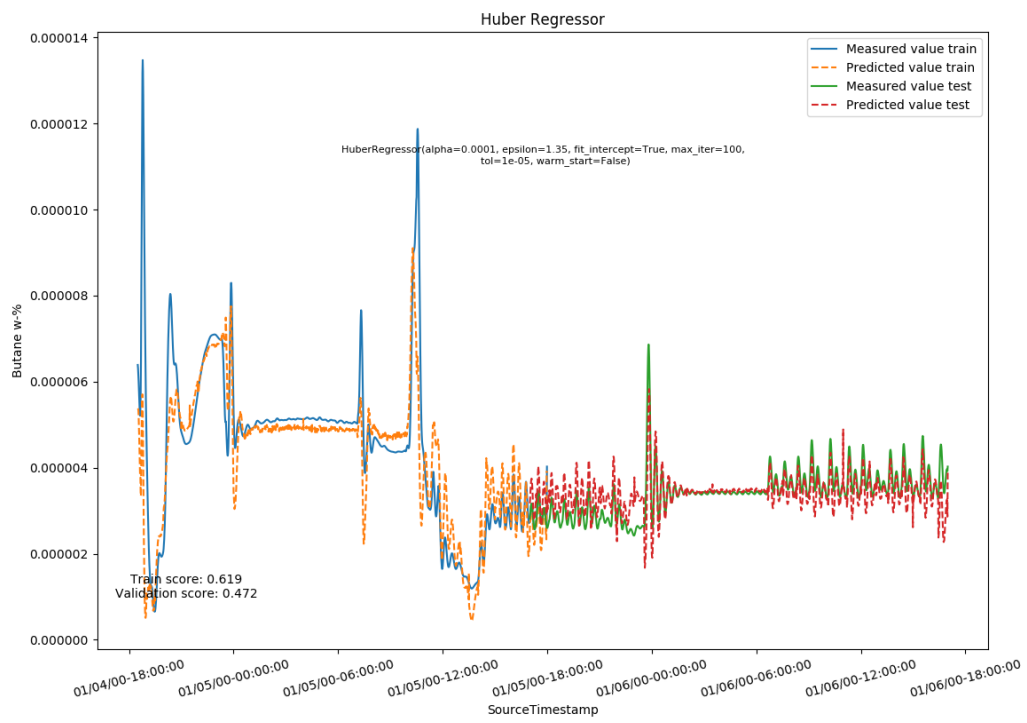


Figure 7.3:

Chapter 8

Conclusions and future work

The aim of the study was to research how machine learning methods could be used to gain added value from the growing amount of measurement data gathered in the process industries. The open source scikit-learn machine learning library for Python was used to create an user-configurable application which can generate data driven soft sensors from data stored in an OPC UA server. The application provides an interface for specifying the soft sensor input and output variables, model type and configuration parameters for the model. For testing the application, a dynamic process simulator was used to gather data from a debutanizer column with multiple operating conditions over a 60 hour period.

The results suggest that the OPC UA specification provides a good interface for both gathering data for offline training of models and publishing model predictions in real time. The soft sensor performance varies a lot depending on the type of model, model parameters and the quality of training data used.

The amount of machine learning methods is vast and their optimal use cases can vary a lot depending on the dynamics of the process being studied and the type of data available from the process. Scikit-learn provides a large set of easy to use tools that can be integrated with relatively little work, but more specialized tools and methods can provide better results in their optimal use cases.

Many free for commercial use open source libraries with large community and company support behind them exist that make implementing machine learning models much easier. The choice between them is not straightforward and depends a lot on the intended use case and skills of the user. Many of the projects are young but have proved themselves in many applications. The landscape of machine learning changes very quickly compared to that of the process industries.

Future work related to the proof-of-concept framework presented in this thesis can be divided to few different categories which better fit under many branches of science. Most important category regarding process control is studying model performance with quantitative methods and real world process data. Understanding the Big Data capabilities of the OPC UA protocol is important for many machine learning tasks, especially deep learning, which has shown promising results in many applications.

Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <http://tensorflow.org/>. Software available from tensorflow.org.
- Aldrich, C. and Auret, L. *Unsupervised Process Monitoring and Fault Diagnosis with Machine Learning Methods*. Springer Publishing Company, Incorporated, 2013. ISBN 1447151844, 9781447151845.
- Alexander, D. L. J., Tropsha, A., and Winkler, D. A. Beware of $r(2)$: simple, unambiguous assessment of the prediction accuracy of qsar and qspr models. *J Chem Inf Model*, 55(7):1316–1322, Jul 2015. ISSN 1549-9596. doi: 10.1021/acs.jcim.5b00206. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4530125/>. 26099013[pmid].
- Anonymous. Apache license 2.0, 2004. URL <https://www.apache.org/licenses/LICENSE-2.0>.
- Anonymous. Gnu general public license, 2007a. URL <http://www.gnu.org/licenses/gpl.html>.
- Anonymous. Gnu lesser general public license, 2007b. URL <https://www.gnu.org/licenses/lgpl-3.0.en.html>.
- Anonymous. File:svm max sep hyperplane with margin.png — wiki-media commons, 2008. URL https://commons.wikimedia.org/w/index.php?title=File:Svm_max_sep_hyperplane_with_margin.png&oldid=225162099. [Accessed 2017-10-02].

- Anonymous. Opc ua standard part 11: Historical access. Industry standard specification, 2015.
- Anonymous. File:kernel machine.png — wikimedia commons, 2016. URL <https://commons.wikimedia.org/w/index.php?title=File:Kernel.Machine.png&oldid=215993536>. [Accessed 2017-10-02].
- Anonymous. Aws documentation, 2017a. URL <https://aws.amazon.com/documentation/>. [Accessed 2017-09-04].
- Anonymous. Amazon machine learning developers guide. 2017b. URL <http://docs.aws.amazon.com/machine-learning/latest/dg/what-is-amazon-machine-learning.html>. [Accessed 2017-06-20].
- Anonymous. Azure machine learning documentation. 2017c. URL <https://docs.microsoft.com/en-us/azure/machine-learning/>. [Accessed 2017-06-21].
- Anonymous. Bigml machine learning documentation, 2017d. URL <https://bigml.com/>. [Accessed 2017-09-26].
- Anonymous. H₂O, sparkling water, steam, & deep water documentation, 2017e. URL <http://docs.h2o.ai/h2o/latest-stable/index.html>. [Accessed 2017-07-12].
- Anonymous. Ibm watson machine learning documentation, 2017f. URL <https://console.bluemix.net/docs/services/PredictiveModeling/index.html#WMLgettingstarted>. [Accessed 2017-09-26].
- Anonymous. Mathworks, 2017g. URL <https://se.mathworks.com/>. [Accessed 2017-07-07].
- Anonymous. Opc ua standard part 1: Overview and concepts. Industry standard specification, 2017h.
- Anonymous. Python language reference, version 3.6, 2017i. URL <https://docs.python.org/3.6/>. [Accessed 2017-12-13].
- Anonymous. What is r?, 2017j. URL <https://www.r-project.org/about.html>. [Accessed 2017-07-06].
- Anonymous. Tensorflow, 2017k. URL <https://www.tensorflow.org/>. [Accessed 2017-10-12].

- Anonymous. Wolfram language & system documentation center, 2017l. URL <http://reference.wolfram.com/language/>. [Accessed 2017-07-11].
- Anonymous. scikit-learn api reference, 2017m. URL <http://scikit-learn.org/0.18/modules/classes.html>.
- Bacon, P. Agent-environment interaction in tikz, 2013. URL <https://gist.github.com/pierrelux/6501790>. [Accessed 2017-08-02].
- Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.
- Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. *CoRR*, abs/1309.0238, 2013. URL <http://arxiv.org/abs/1309.0238>.
- Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. Recurrent neural networks for multivariate time series with missing values. *CoRR*, abs/1606.01865, 2016. URL <http://arxiv.org/abs/1606.01865>.
- Chen, J. and Wang, F. Cost reduction of co2 capture processes using reinforcement learning based iterative design: A pilot-scale absorption-stripping system. *Separation and Purification Technology*, 122:149 – 158, 2014. ISSN 1383-5866. doi: <http://dx.doi.org/10.1016/j.seppur.2013.10.023>. URL <http://www.sciencedirect.com/science/article/pii/S1383586613006163>.
- Cortes, C. and Vapnik, V. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL <http://dx.doi.org/10.1023/A:1022627411411>.
- Fauske, K. Tikz example: Neural network, 2006. URL <http://www.texample.net/tikz/examples/neural-network/>. [Accessed 2017-06-19].
- 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*, PP(99): 1–1, 2017. doi: 10.1109/ACCESS.2017.2756872.

- Ge, Z. and Song, Z. A comparative study of just-in-time-learning based methods for online soft sensor modeling. *Chemometrics and Intelligent Laboratory Systems*, 104(2):306 – 317, 2010. ISSN 0169-7439. doi: <http://dx.doi.org/10.1016/j.chemolab.2010.09.008>. URL <http://www.sciencedirect.com/science/article/pii/S0169743910001759>.
- George, S. Microsoft introduces new open-source cross-platform opc ua support for the industrial internet of things, 2016. URL <https://blogs.microsoft.com/iot/2016/06/23/microsoft-introduces-new-open-source-cross-platform-opc-ua-support-for-the-industrial-internet-of-things/>. [Accessed 2017-06-21].
- Gonzaga, J., Meleiro, L., Kiang, C., and Filho, R. M. Ann-based soft-sensor for real-time process monitoring and control of an industrial polymerization process. *Computers & Chemical Engineering*, 33(1):43 – 49, 2009. ISSN 0098-1354. doi: <http://dx.doi.org/10.1016/j.compchemeng.2008.05.019>. URL <http://www.sciencedirect.com/science/article/pii/S0098135408001142>.
- Gudmundsson, S., Runarsson, T. P., and Sigurdsson, S. Support vector machines and dynamic time warping for time series. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2772–2776, June 2008. doi: 10.1109/IJCNN.2008.4634188.
- Haimi, H., Mulas, M., Vahala, R., and Corona, F. Soft-sensors in wastewater treatment: the benefits of the data-driven approach. 2011. URL https://www.researchgate.net/publication/287211149_Soft-sensors_in_wastewater_treatment_the_benefits_of_the_data-driven_approach.
- Harding, J., Shahbaz, M., and Kusiak, A. Data mining in manufacturing: A review. *Journal of Manufacturing Science and Engineering-transactions of The Asme - J MANUF SCI ENG*, 128, 11 2006.
- Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer New York, 2009. ISBN 9780387848587.
- Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL <http://dx.doi.org/10.1162/neco.2006.18.7.1527>.

- Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.
- Hoskins, J. and Himmelblau, D. Process control via artificial neural networks and reinforcement learning. *Computers & Chemical Engineering*, 16(4):241 – 251, 1992. ISSN 0098-1354. doi: [http://dx.doi.org/10.1016/0098-1354\(92\)80045-B](http://dx.doi.org/10.1016/0098-1354(92)80045-B). URL <http://www.sciencedirect.com/science/article/pii/009813549280045B>. Neural network applications in chemical engineering.
- Hothorn, T. Cran task view: Machine learning & statistical learning, 2017. URL <https://cran.r-project.org/web/views/MachineLearning.html>. [Accessed 2017-07-06].
- Jang, H., Park, A., and Jung, K. Neural network implementation using cuda and openmp. In *2008 Digital Image Computing: Techniques and Applications*, pages 155–161, Dec 2008. doi: 10.1109/DICTA.2008.82.
- Johnson, W. K-means clustering - what is it, and how it works, 2017. URL <http://www.learnbymarketing.com/methods/k-means-clustering/>. [Accessed 2017-08-10].
- Kadlec, P., Gabrys, B., and Strandt, S. Data-driven soft sensors in the process industry. *Computers & Chemical Engineering*, 33(4):795 – 814, 2009. ISSN 0098-1354. doi: <http://dx.doi.org/10.1016/j.compchemeng.2008.12.012>. URL <http://www.sciencedirect.com/science/article/pii/S0098135409000076>.
- Kadlec, P., Grbić, R., and Gabrys, B. Review of adaptation mechanisms for data-driven soft sensors. *Computers & Chemical Engineering*, 35(1):1 – 24, 2011. ISSN 0098-1354. doi: <http://dx.doi.org/10.1016/j.compchemeng.2010.07.034>. URL <http://www.sciencedirect.com/science/article/pii/S0098135410002838>.
- Kohonen, T. *Self-organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 3-540-62017-6.
- Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. Data preprocessing for supervised learning. *International Journal of Computer Science*, 1:111–117, 01 2006.
- 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*, 2:24, 2015. ISSN 2196-1115. doi: 10.1186/s40537-015-0032-1. URL <http://www.journalofbigdata.com/content/2/1/24>.
- Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2nd edition, 2014. ISBN 1107077230, 9781107077232.
- McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL <http://dx.doi.org/10.1007/BF02478259>.
- Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., Xin, D., Xin, R., Franklin, M. J., Zadeh, R., Zaharia, M., and Talwalkar, A. Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.*, 17(1):1235–1241, January 2016. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=2946645.2946679>.
- Murphy, K. P. *Machine learning : A Probabilistic Perspective*. The MIT Press, Cambridge, Mass. [u.a.], 2012. ISBN 9780262018029 0262018020.
- Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.
- Opfell, J. B. and Sage, B. H. Applications of least squares methods. *Industrial & Engineering Chemistry*, 50(5):803–806, 1958. doi: 10.1021/ie50581a038. URL <http://dx.doi.org/10.1021/ie50581a038>.
- Pani, A. K., Amin, K. G., and Mohanta, H. K. Soft sensing of product quality in the debutanizer column with principal component analysis and feed-forward artificial neural network. *Alexandria Engineering Journal*, 55(2):1667 – 1674, 2016. ISSN 1110-0168. doi: <http://dx.doi.org/10.1016/j.aej.2016.02.016>. URL <http://www.sciencedirect.com/science/article/pii/S1110016816000697>.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

- Polikar, R. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, Third 2006. ISSN 1531-636X. doi: 10.1109/MCAS.2006.1688199.
- Qin, S. J. Survey on data-driven industrial process monitoring and diagnosis. *Annual Reviews in Control*, 36(2):220 – 234, 2012. ISSN 1367-5788. doi: <https://doi.org/10.1016/j.arcontrol.2012.09.004>. URL <http://www.sciencedirect.com/science/article/pii/S1367578812000399>.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL <http://dl.acm.org/citation.cfm?id=104279.104293>.
- Sato, K., Young, C., and Patterson, D. An in-depth look at google’s first tensor processing unit (tpu), 2017. URL <https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu>.
- Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. ISSN 0893-6080. doi: <https://doi.org/10.1016/j.neunet.2014.09.003>. URL <http://www.sciencedirect.com/science/article/pii/S0893608014002135>.
- Shang, C., Yang, F., Huang, D., and Lyu, W. Data-driven soft sensor development based on deep learning technique. *Journal of Process Control*, 24(3):223 – 233, 2014. ISSN 0959-1524. doi: <http://dx.doi.org/10.1016/j.jprocont.2014.01.012>. URL <http://www.sciencedirect.com/science/article/pii/S0959152414000365>.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, 2017. URL <https://arxiv.org/abs/1712.01815>.
- Smith, M. J. S. *Application-specific Integrated Circuits*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997. ISBN 0-201-50022-1.
- Souza, F. A., Araújo, R., and Mendes, J. Review of soft sensor methods for regression applications. *Chemometrics and Intelligent Laboratory Systems*, 152:69 – 79, 2016. ISSN 0169-7439. doi: <http://dx.doi.org/10.1016/>

j.chemolab.2015.12.011. URL <http://www.sciencedirect.com/science/article/pii/S0169743915003263>.

Spielberg, S. P. K., Gopaluni, R. B., and Loewen, P. D. Deep reinforcement learning approaches for process control. In *2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, pages 201–206, May 2017. doi: 10.1109/ADCONIP.2017.7983780.

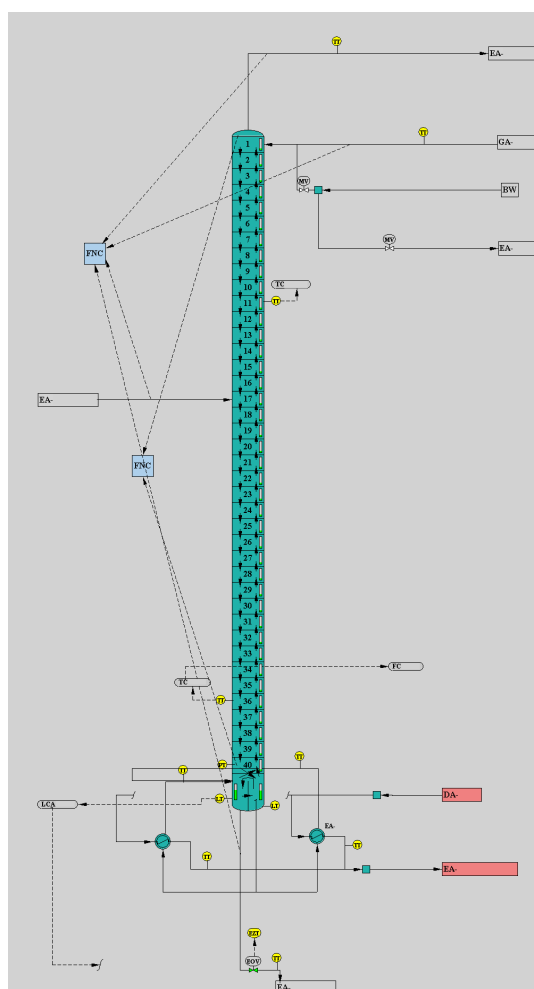
Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., and Keogh, E. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2): 275–309, Mar 2013. ISSN 1573-756X. doi: 10.1007/s10618-012-0250-5. URL <http://dx.doi.org/10.1007/s10618-012-0250-5>.

Yan, W., Tang, D., and Lin, Y. A data-driven soft sensor modeling method based on deep learning and its application. *IEEE Transactions on Industrial Electronics*, 64(5):4237–4245, May 2017. ISSN 0278-0046. doi: 10.1109/TIE.2016.2622668.

Appendix A

Debutanizer column model in ProsDS





Appendix B

Data visualization tools in Azure Machine Learning Studio

Clustering: Group Iris Data > Metadata Editor > Results dataset

rows
150

columns
5

view as  

F1	F2	F3	F4	Label
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa
5.7	3.8	1.7	0.3	Iris-setosa
5.1	3.8	1.5	0.3	Iris-setosa
5.4	3.4	1.7	0.2	Iris-setosa
5.1	3.7	1.5	0.4	Iris-setosa
4.6	3.6	1	0.2	Iris-setosa

Statistics

Mean	5.8433
Median	5.8
Min	4.3
Max	7.9
Standard Deviation	0.8281
Unique Values	35
Missing Values	0
Feature Type	Numeric Feature

Visualizations

F1

ScatterPlot

compare to

