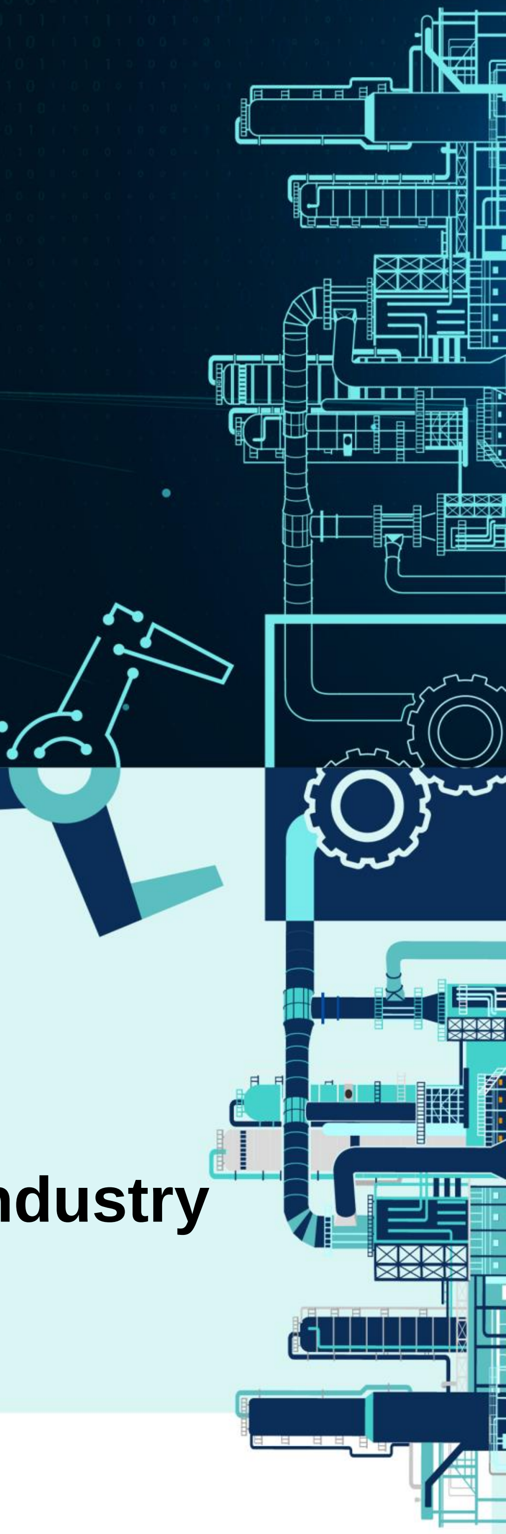
FACTLOG
www.factlog.eu

ENERGY-AWARE FACTORY ANALYTICS FOR PROCESS INDUSTRIES



Deliverable D2.2

Analytical Platform for Process Industry

Version
Version 1.0

Lead Partner
JSI

Date
30/10/2021

Project Name
FACTLOG – Energy-aware Factory Analytics for Process Industries

FACTLOG has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement Nº 869951

Call Identifier H2020-NMBP-SPIRE-2019	**Topic** DT-SPIRE-06-2019 - Digital technologies for improved performance in cognitive production plants
Project Reference 869951	**Start date** November 1st, 2019
Type of Action IA – Innovation Action	**Duration** 42 Months

Dissemination Level

X	**PU**	Public
	CO	Confidential, restricted under conditions set out in the Grant Agreement
	CI	Classified, information as referred in the Commission Decision 2001/844/EC

Disclaimer

Executive Summary

This document presents the analytics platform developed in the FACTLOG project for use in the process industry. The platform analyses the data from the manufacturing systems and produces insights and predictions which inform the other components. This includes predicting future states of systems from past sensor readings and machine settings information; computing the most likely values of missing readings, specialised models of key infrastructure assets from the pilots such as distillation columns; and identifying and analysing unusual situations in complex multi-variate datasets.

The methodology section (Section 2) introduces the main concepts on which the platform tools are based. The stream forecasting problem is framed, where future values of timeseries is predicted from past observations and possibly contextual data. The approach based on Artificial Neural Networks is introduced and extended with the capability to impute missing values in the timeseries in a pre-processing layer. A specialised approach for modelling distillation columns is presented as they are a common asset in two of the pilots. Finally, methodology for detecting and analysing unusual situations for the cognitions process is described.

Implementation details are given in Section 3. The focus is on the description of the API structure of the Batch Learning Forecasting component. The description includes building forecasting models and data imputation as well as details regarding deployment of multiple instances and attaching the models to a messaging queue (i.e. a Kafka stream). A similar technical description is given for the distillation columns model.

Finally, demonstration scenarios are presented in Section 4. These include example tool deployments from the Tupras, JEMS and Continental pilots with their input data, model setup, evaluation setup, and results. Though these deployments are still in active development, the results achieved show promise.

Revision History

Revision	Date	Description	Organisation
0.1	29/09/2021	Table of contents	JSI
0.2	15/10/2021	Sections 2.1, 3 and 4.1.1	JSI
0.3	22/10/2021	Sections 2.2 and 4.2	JSI
0.4	25/10/2021	Sections 2.4 and 4.3	NISSA
0.5	26/10/2021	Sections 2.3, 3.2 and 4.2	JSI
0.6	27/10/2021	Introduction and Executive Summary	JSI
1.0	30/10/2021	Final version after internal review	NISSA, JSI

Contributors

Organisation	Author	E-Mail
JSI	Aljaž Košmerlj	aljaz.kosmerlj@ijs.si
JSI	Beno Šircelj	beno.sircelj@ijs.si
JSI	Bor Brecelj	bor.brecelj@gmail.com
JSI	Jože Rožanec	joze.rozanec@ijs.si
NISSA	Nenad Stojanovic	Nenad.Stojanovic@nissatech.com
NISSA	Branislav Jovicic	Branislav.Jovicic@nissatech.com
NISSA	Jelena Jakimov	Jelena.Jakimov@nissatech.com

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

This deliverable accompanies the release of the FACTLOG analytics platform and details the tools included in the platform. Analytics act as the eyes and ears of the FACTLOG cognitive digital twin, with its tools detecting anomalies in the data and raising alerts when unusual situations arise. The tools are built using a wide array of machine learning methodology used to clean, analyse and process the data from manufacturing systems into actionable alerts and insights. Through those the other components, such as the optimisation and process modelling services, can be invoked to address the anomalies and return the system into a desired state.

The document explains the theoretical background for the analytics methods as well as the technical implementation details and the resulting API structure. The use of the methods and tools is demonstrated through pilot scenarios where the platform has already been tested. These are demo deployments and are not yet ready for full production operation but already show the impact the analytics can have. Future work will improve upon these initial results.

1.2 Relation with other Deliverables

The most related other deliverable to this one is D2.1, Analytics System Requirements and Design Specification, where the requirements for the analytics platform were formulated and the platform architecture was specified.

In future, the platform will be extended with additional capabilities and integrated with other components, most of which will be covered in Deliverable D3.2, Data Analytics as a Cognitive Service, and Deliverable D4.4, KG-based Analytics for Process Optimization.

The tools listed in this deliverable will be, to some extent, used in all the pilots and therefore also have some relation to all the deliverables describing the final deployments.

1.3 Structure of the Document

After the document introduction in Section 1, the theoretical background of the methods used in the analytics platform are explained in Section 2, with each subsection covering one of the main methods. Section 3 then explains the implementation details and the API structure of the forecasting component as well as the model specialised for distillation columns. Finally, demonstration scenarios for use of the analytics platform tools from a selection of pilots (Tupras, JEMS and Continental) is detailed in Section 4.

2 Methodology

This section introduces the methodological details of the analytics tools in the FACTLOG analytics platform.

2.1 Stream Forecasting

The idea of performing forecasting on a stream of data is that we are trying to make predictions based on a continuous flow of temporally sequential data. Models that make predictions based on such data must consider enough input from the past to make good predictions for some time window into the future. The model that performed best in our tests is an artificial neural network that can be trained to take as input all sensor readings from a desired time window in the past and predict all sensor readings for a given time in the future. Such a model can only work if we already have a sufficient learning set of data from the past and will not work at the very beginning of the data stream.

There is a possibility of concept drift, i.e., an unforeseen change in system dynamics over time, which can lead to less accurate predictions. In such cases, it would be good if the model could detect this and update itself. One possibility for such a model would be to use online machine learning models. This type of model is typically used when the initial data is unknown or too large to be processed by a batch model such as the feedforward artificial neural network mentioned earlier. We did not choose to use such a model because our experiments have shown it is unnecessary and because such models commonly sacrifice a portion of performance for their capability to learn online. Instead, the model can simply be re-trained after a period of time with the most recent data.

2.2 Imputation

In both the Tupras and the JEMS use-case the data contains missing values. The most basic method to solve this issue is to fill all missing values of each feature with its mean or median value. In Python's scikit-learn library this method is called SimpleImputer. A more advanced method from the scikit-learn library is IterativeImputer, which models each feature as a function of others in a round-robin fashion. More precisely, it learns a regressor for each feature where input data are the other features. Although IterativeImputer performs better than SimpleImputer, it is slow which means that it is not suitable for real-time data streams.

We implemented a method described in the paper "*Processing of missing data by neural networks*" [1] which we will call GMM Layer. It is a modification of the first layer of a feedforward neural network. Instead of assigning exact values to missing values in the data, missing values are modelled with a probability distribution. A mixture of Gaussians (GMM) is used to model all data points. A data point is represented as a conditional probability distribution of missing features given known values. The first layer of a feedforward neural network is modified so that it gets a (possibly degenerate) mixture of Gaussians as an input and returns expected activation of neurons in that layer (as shown in Figure 1). The remaining layers are not modified. All parameters of the mixture of Gaussians are trained together with the neural network.

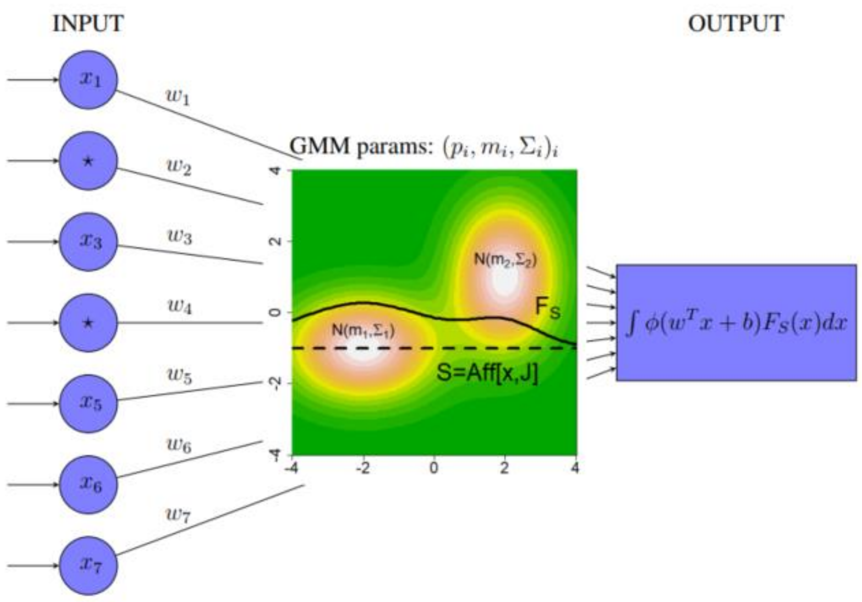


Figure 1: The GMM layer.

Since the method is a modified first layer of the artificial neural network, it can be included in a neural network that makes predictions directly. Alternatively, it can be used as an autoencoder that does only imputation. Autoencoder is a feedforward neural network, which consists of two parts, encoder, and decoder. Encoder transforms input data into data points in fewer dimensions, whereas decoder reconstructs the original input. Autoencoder is trained to produce the output as close to input data as possible. Therefore, its output is imputed values that can then be used by any method.

The method adds a new hyperparameter, the number of Gaussians which has to be carefully chosen. Too small values might mean the layer cannot model the data. On the other hand, too high values may result in a high number of model parameters which increases learning time. Even though it needs time to learn, the method is suitable for real-time stream setting, because it can make quick predictions on new data.

2.3 Modelling Distillation Columns

Crude oil is sourced to petroleum refineries from different provenances. Crude oil characteristics vary from source to source. Thus, different configurations are required through the oil refinery pipeline to ensure the quality standards are met. One of such processes is the distillation that treats the inlet to create the Liquefied Petroleum Gas (LPG). The final mixture such a product usually contains 48% propane, 50% butane, and up to 2% pentane. Among the processes applied to LPG to remove impurities we find the debutanization, which removes pentanes. To ensure the final mixture meets the specification standards, laboratory analysis is performed on samples taken in various stages of the purification process. Such analyses are performed a few days a week and can take several hours to complete. This fact favours scenarios where off-specs situations can spread over time before being detected and adequate measures taken. It is thus important to provide means towards early pentanes excess detection. A possible solution is the use of machine learning models to forecast pentane content or off-specification events based on real-time streamed sensor data.

When modelling the debutanizer columns, we took into account debutanizer unit diagrams on which details regarding sensors were provided. In particular, we gained insights regarding their position, type of sensor, and sensor reading values at a particular point in time. While the debutanizer columns differed regarding their design, we identified two pairs of temperature and pressure sensors located at the top of the debutanizer unit, close to the exit and before the condensation unit. Not all data regarding these sensors was available to us, and thus we limited our models to the input of two sensors: a pressure and temperature sensor, located at two separated points of the distillation column. We overcome the lack of multiple sensor inputs with careful featurisation and the results so far indicate this is sufficient for an operational model. While the sensor readings inform changes in the process, relationships between pressure and temperature at both points are governed by physical principles and laws. Additional insights can be obtained given such principles and laws, and data obtained from the diagrams informing sensor readings for all sensors at a specific point in time.

Multiple features were created based on physical and chemical principles and laws related to the distillation process. In particular, we considered the Ideal Gas law equation, combined gas law equation, the Clausius-Clapeyron relation (to understand the relation between temperature and pressure at two different points), the Antoine's equation (to compute the expected saturation pressure of an approximate LPG mixture), and Raoult’s law and Molar weight equation (to hint on the LPG mixture composition). A full explanation of how these first-principle laws are used in computing the features and the references to papers explaining the physics background can be found in an upcoming paper available as a preprint [6].

We created two models: a regression model to predict pentane content, and a classification model to predict whether the pentane content was above allowed levels. The regression model was a Voting Regressor[1] that learned from two Catboost[2] models with different optimization objectives: a first model optimizes against the root mean square error (RMSE), penalizing large errors, while the second one optimizes against mean absolute error (MAE) to achieve best median performance. Outputs from both models are weighted by the voting regressor, to create the final forecast. The classification model, on the other side, was a Catboost classifier with a focal loss, which guides the model's learning process focusing on such instances that are harder to learn, to achieve superior results. Full details are described in an upcoming paper currently available as pre-print [6].

2.4 Unusuality Detection

As already described in deliverable D1.2 [3], variation detection is one of the main processing steps in the realization of the cognition process.

There are mainly two types of variations in multivariate time-series; one type is abnormal observation values or unusual subsequences within an individual variable, and the other type is unexpected changes of relationships among multiple variables. The second one is very important for describing/understanding the behaviour of a system (and cognition). We consider these variations as unusualities, or unusual system behaviour, as illustrated in

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html
[2] https://catboost.ai/

Figure 2. These are situations where the cognition should be activated since in a general case, an unusual behaviour must be further processed in the cognition process in order to know how to react on.

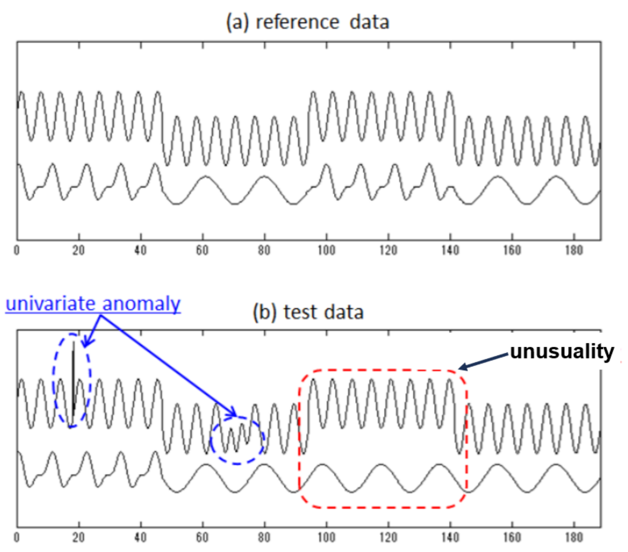


Figure 2: Unusuality in complex signals.

2.4.1 Basic Approach

In this section we present the approach for unusuality detection based on Neural Networks.

An Autoencoder is a type of Neural Networks used to learn efficient data coding in an unsupervised manner. The aim of an Autoencoder is to learn a representation (*encoding*) for a set of data, by training the Neural Network to ignore *signal noise*. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input. Said characteristics make Autoencoders applicable to dimensionality reduction, information retrieval, anomaly detection, image processing, machine translation, etc.

Autoencoders consist of three parts – encoder, used to transform input data to a corresponding code; decoder, used to transform code to its corresponding input; code, functioning as a *gate* between encoder and decoder parts (see Figure 3).

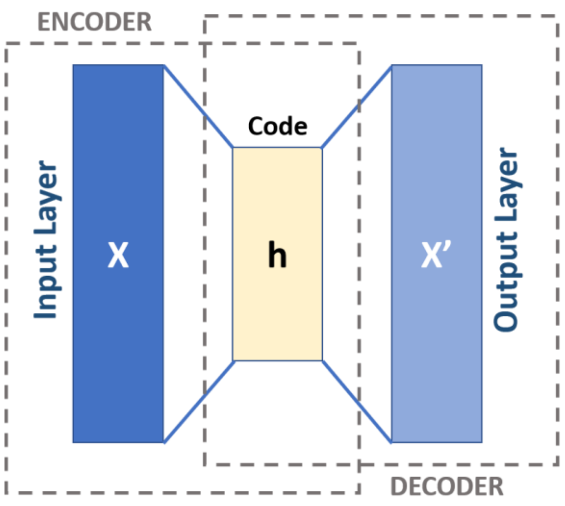


Figure 3: An Autoencoder architecture.

This idea for the approach is based on the working principle of Autoencoders and the fact that Neural Networks fail to properly learn representation of under-balanced data [7].

In essence, if there is significantly lower number of instances of particular class (*data type*) in comparison to other classes, a Neural Network will learn its characteristics and representation poorly, which will result in wrong classifications. In the context of Autoencoder Neural Networks, this means that the Autoencoder will fail to properly learn how to encode and decode (*reconstruct*) its input.

This means that if we use an entire dataset and do not remove anomalous instances, the Autoencoder should automatically learn the representation of usual data and later reconstruct it well, because under-balanced anomalous data should not impact training greatly. On the other hand, anomalous data will not be learned properly and will result in greater reconstruction errors.

The above stated would work under the assumption that the process from which data originated is stable and that there is significantly lower number of unusual data compared to the usual. This stability could be checked with *Stability Index* method[3]. If the process is stable, we can assume there will be few unusual data points and vice versa.

Another way we can check whether the process is stable is to use one of methods for anomaly detection. If a large percentage (*above 15%*) of data is anomalous, then we would not recommend using this method. It can indicate that the process is very unstable and we cannot be certain of proposed method's usefulness.

2.4.2 Validation Approach

As we do not have any information which data points (*i.e., training instances*) are unusual prior to the training of neural networks, we do not have a proper way of validating our model.

Therefore, we came up with several "semi-validation" methods:

[3] https://www.listendata.com/2015/05/population-stability-index.html

1. Comparing the results of our model to the results of outlier detection algorithms. We do not expect a great matching level of outputs of these algorithms and our model, because they function differently. However, we expect that there would be a certain matching level, as outliers fall into the category of something unusual;
2. Manual validation – we would visualize the data set and the results of our model and perform visual inspection, if that is possible. Usually, there are a lot of features in the dataset, so manually performing multivariate validation to such extent is nearly impossible, but some conclusions may be drawn out with this method;
3. Compare multiple models of the same architecture trained on different training and validation parts of the same data set and see whether their results match, and to what extent. If the results match, we can assume, and with fair certainty, that our models perform well. Otherwise, we can assume that there are issues, either with the dataset or with the model, which should be further inspected.

2.4.3 Threshold

Since we use our model to reconstruct instances, how well the reconstruction was performed is inferred from comparison of the actual and reconstructed instances. If they differ greatly, we assume that the instance is unusual, and vice-versa. This raises the question of where to draw the line between usual and unusual – i.e., how to set up a threshold based on which some instance is classified as unusual, and make it broadly applicable, as well?

Our approach uses several thresholds. In its essence, this method calculates the standard deviation of the reconstruction errors, and any instance with error that is located within *X* or more standard deviations from the errors mean is considered unusual. In many disciplines if something is located within 3, or even 2, standard deviations from the mean it is considered an outlier. Therefore, we decided to use both of these thresholds, and to add one in between, as well.

This way, we ended up with 3 “unusuality zones”. In essence, we can look at instances located within the first zone as potentially unusual (*that may need further inspection*), while the ones in the last zone as extremely rare, and thus very unusual (*from loose to strict classification*).

Since reconstruction errors can follow several different distributions (*other than normal*), we cannot approximate the percentage of the data that we classify as unusual, as it can vary greatly depending on the data set (see 68–95–99.7 rule[4]). Therefore, we decided to define percentage zones; i.e., 10%, 5% and 1% of greatest errors will be used to define the respective thresholds.

2.4.4 Feature Selection/Reduction

The datasets can have a considerable number of features. This number can vary significantly from case to case. As a consequence, this can make implementation of general unusuality detection method with neural networks very difficult. Additionally, it makes visual inspection/manual validation impossible.

[4] https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule

Therefore, we are considering performing feature selection/reduction[5] in order to reduce the dimensionality and complexity of the problem that needs to be solved.

2.4.5 Our Approach – Ensemble Learning Supported Unusuality Detection

This approach is an extension of the 3rd numbering in Validation approach section. It implies joint usage of multiple models in order to increase confidence in the final model/algorithm.

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone[6].We assume that this would be especially useful for our method, as we have no prior knowledge about the data sets that this method might be used for. So, it would be helpful to cover and use entire historical data that we get with multiple models and implement voting or an average model.

According to a review of the field, we singled out three ensemble learning methods that, can improve basic approach:

1. Bootstrap aggregating (*Bagging*) – Using bootstrap sampling[7], create n samples with m instances and train n models with these samples. Validation is performed on the instances that were left out of the sample. Aggregation is performed either as averaging or voting;
2. K-fold aggregating – Using k-fold sampling[8], create and train k models. Aggregation is performed either as averaging or voting;
3. Boosting – Run initial training with entire training dataset. Then, based on reconstruction errors from validation dataset calculate threshold using proposed method. Use calculated threshold to separate instances with small reconstruction errors in training dataset. Use said instances to further train the model, i.e., to boost it, in order to be able to better differentiate between usual and unusual instances.

[5] https://en.wikipedia.org/wiki/Feature_selection
[6] https://en.wikipedia.org/wiki/Ensemble_learning
[7] https://machinelearningmastery.com/a-gentle-introduction-to-the-bootstrap-method/
[8] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

3 Implementation

This section presents the implementation details of the FACTLOG analytics platform. Note that all the tools follow the loosely coupled architecture presented in the specification in deliverable D2.1 [4] and can integrate with the other components through the project messaging queue and orchestration services.

3.1 Batch Learning Forecasting Component

The component enables using external predictive models from PyTorch[9] and Scikit Learn[10] library (for example, models such as the Random Forest Regressor[11]) implementation in a streaming scenario. Fitting, saving, loading and live prediction are enabled. Live predictions work via Kafka streams in such a way that that feature vectors are read from a Kafka stream and predictions are written to a Kafka[12] stream. In FACTLOG a neural network model was added enabling prediction of multiple streams using a single model.

The layout of the neural network used in the component is shown in Figure 4. The structure is a typical feedforward network. An additional hidden layer can be added that can impute missing data by replacing typical neuron's response in by its expected value using a Gaussian mixture model (GMM) as presented in Section 2.2.

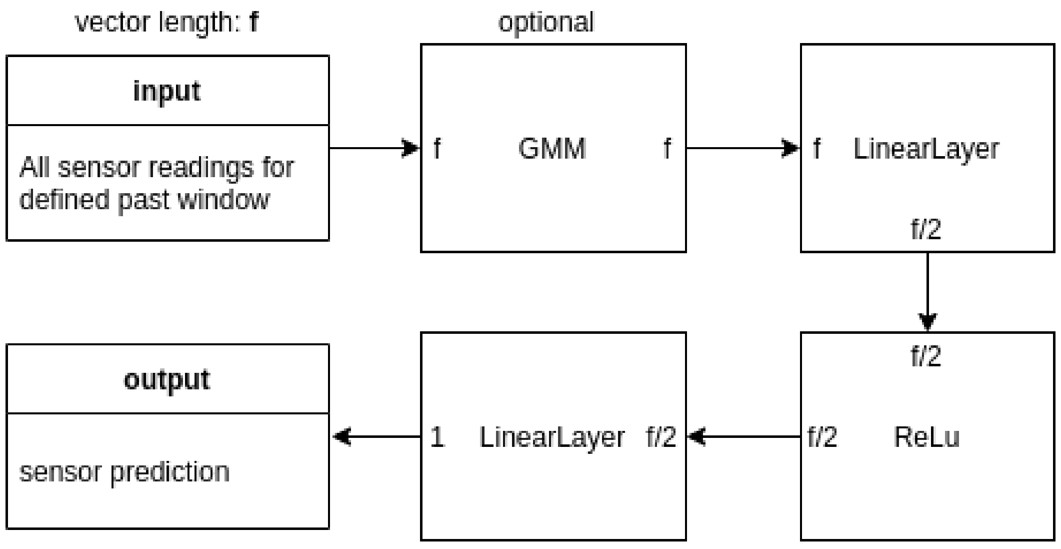


Figure 4: Structure of the neural network used in the component

3.1.1 Architecture

Separate models are trained for each individual horizon (for how much time in the future do we want to make predictions). If a different model than the neural network is used, a separate model needs to be trained for each sensor. These models should be trained first with the

[9] https://pytorch.org
[10] https://scikit-learn.org
[11] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[12] https://kafka.apache.org

previously available data and can be updated periodically with a certain number of the latest measurements. An overview of this component architecture setup is shown in Figure 5.

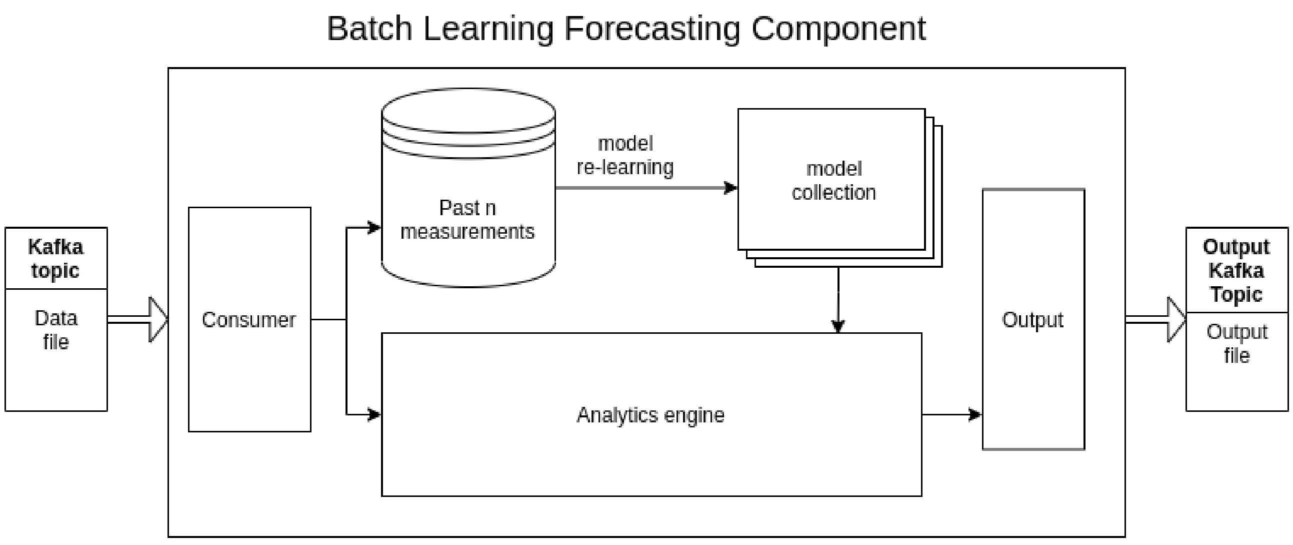


Figure 5: Batch learning forecasting component architecture

3.1.2 API Specification

The forecasting library is open-source and is freely available on GitHub[13].The predictive model is designed in a decentralized fashion, meaning that several instances (submodels) will be created and used for each specific sensor and horizon (`#submodels = #sensors * #horizons`). Such a decentralized architecture enables parallelization.

Usage:

```
python main.py [-h] [-c CONFIG] [-f] [-s] [-l] [-p]
```

Optional parameters:

Short	Long	Description
`-h`	`--help`	show help
`-c CONFIG`	`--config CONFIG`	path to config file (example: config.json)
`-f`	`--fit`	learning the model from dataset (in /data/fused)
`-s`	`--save`	save model to file
`-l`	`--load`	load model from file
`-p`	`--predict`	start live predictions (via Kafka)
`-w`	`--watchdog`	start watchdog pinging

Configuration file:

Configuration file specifies the Kafka server address, which algorithm to use, prediction horizons and sensors. Configuration files are stored in `src/config/`.

[13] https://github.com/JozefStefanInstitute/forecasting

General Parameters:

Name	Type	Default	Description
prediction_horizons	list(integer)		List of prediction horizons (in units specified in time_offset) for which the model will be trained to predict for.
time_offset	string	H	String alias[14] to define the data time offsets. The aliases used in training and topic names are lowercase for backwards compatibility.
sensors	list(string)		List of sensors for which this specific instance will train the models and will be making predictions.
bootstrap_servers	string or list(string)		String (or list of host[:port] strings) that the consumer should contact to bootstrap initial cluster metadata.
algorithm	string	torch	String as either a scikit-learn model constructor with initialization parameters or a string torch to train using a pre-defined neural network using PyTorch with architecture: [torch.nn.Linear, torch.nn.ReLU, torch.nn.Linear],
evaluation_period	integer	512	Define time period (in defined time offset that is hours by default) for which the model will be evaluated during live predictions (evaluations metrics added to output record).
evaluation_split_point	float	0.8	Define training and testing splitting point in the dataset, for model evaluation during learning phase (fit takes twice as long time).
retrain_period	integer	None	The number of received samples after which the model will be re-trained. This is an optional parameter. If it is not specified no re-training will be done.
samples_for_retrain	integer	None	The number of samples that will be used for re-training. If retrain_period is not specified this

[14] https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

			parameter will be ignored. This is an optional parameter. If it is not specified (and retrain_period is) the re-train will be done on all samples received since the component was started.
watchdog_path	string	None	Watchdog path.
watchdog_interval	integer	60	Delay in seconds between each Watchdog ping.
watchdog_url	string	`localhost`	Watchdog url.
watchdog_port	integer	3001	Watchdog port.

PyTorch parameters:

Name	Type	Default	Description
learning_rate	float	4E-5	Learning rate for the torch model.
batch_size	integer	64	Size of training batches for torch model.
training_rounds	integer	100	Training rounds for torch model.
num_workers	integer	1	Number of workers for torch model.

GMM layer parameters:

Name	Type	Default	Description
gmm_layer	boolean	False	If True, the GMM layer is added to the model.
initial_imputer	string	simple	Options are simple or iterative which uses either sklearn SimpleImputer[15] or IterativeImputer[16].
max_iter	integer	15	If the iterative imputer is chosen, this argument defines the maximum number of iterations for it.
n_gmm	integer	5	Number of components of GaussianMixture. If n_gmm is set to -1, then all values between min_n_gmm and max_n_gmm are checked and the one with the best BIC score is chosen.
min_n_gmm	integer	1	Minimum number of components for GMM if search is enabled.
max_n_gmm	integer	10	Maximum number of components for GMM if search is enabled.
gmm_seed	integer	None	Random state seed for GMM.
verbose	boolean	False	If set to True, the progress and results of n_gmm parameter search is displayed.

[15] https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html
[16] https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html

Example of config file:

```
{
    "bootstrap_servers": "127.0.0.1:9092",
    "algorithm":
"sklearn.ensemble.RandomForestRegressor(n_estimators=100, n_jobs=16)",
    "evaluation_period": 72,
    "evaluation_split_point": 0.8,
    "prediction_horizons": [1, 2, 3],
    "sensors": ["test", "test2"],
    "retrain_period": 100,
    "samples_for_retrain": 5000
}
```

3.1.3 Running multiple instances:

The forecasting instance is loosely coupled to the system via Kafka streaming API, therefore it can be started as multiple processes (simple parallelization). For this purpose, we can use `start_cluster.sh` script with the same input parameters as `main.py`. Cluster is defined in a separate config file `cluster.json`. The script runs several instances of `main.py` in a tmux session (named `modeling_cluster`), each under a different window.

Usage:

```
bash start_cluster.sh [-f] [-s] [-l] [-p]
```

Configuration file:

Specify which sensors should be processed by a specific instance in a separate line.

Example of cluster configuration file:

```
["N1", "N2"]
["N3", "N4"]
["N5", "N6"]
["N7", "N8"]
```

Alternatively, process managers like `PM2` or `pman` would be a better fit for the task than `tmux`.

3.1.4 Model training and naming

- **Training data**: all the training files should be stored in a subfolder called `/data/fused`. Data should be stored as json objects per line (e.g. `{"timestamp": 1459926000, "ftr_vector": [1, 2, 3]}`). Separate file for each sensor and prediction horizon. Files should be named the same as input Kafka topics, that is `{sensor}_{horizon}{time_offset.lower()}` (e.g. `sensor1_3h.json`).The target sensor is the first element of `ftr_vector`.

  If we want to train a model that uses data from the last 60 minutes, we should make sure that the first reading is the target sensor at the specified timestamp and all other values are all sensor readings for every minute up to 60 minutes in the past. When we make predictions later, we also need to provide a vector with such a structure to the component.

- **Re-training data**: all the re-training data (if re-training is specified) will be stored in a subfolder called `/data/retrain_data` in the same form as training data. Separate files will be made for each sensor and prediction horizon. The names of the files will be in the following form: `{sensor}_{horizon}{time_offset.lower()}_retrain.json` (eg. `sensor1_3h_retrain.json`).
- **Models**: all the models are stored in a subfolder called `/models`. Each sensor and horizon has its own model (with the exception of the neural net model, which covers multiple sensors. The name of the models is composed of sensor name and prediction horizon, `model_{sensor}_{horizon}{time_offset.lower()}` (e.g. `model_sensor1_3h`).

3.1.5 Kafka specifications

- **Input Kafka topic**: The names of input Kafka topics on which the prototype is listening for live data should be in the same format as training data file names, that is `features_{sensor}_{horizon}{time_offset.lower()}.`
- **Output Kafka topic**: Predictions are sent on different topics based on a sensor names, that is `{sensor}` (e.g. `sensor1`).
- **Input data structure:** JSON file with the following structure. The feature vector specifies the composition of the data row.

```
{
    "ftr_vector":  array (a feature vector) of values,
    "timestamp": timestamp as strings in the Unix timestamp format
}
```

- **Output data structure:** JSON file with following structure

```
{
    "stampm":  timestamp as strings in the Unix timestamp format,
    "value":  prediction,
    "sensor_id": target sensor,
    "horizon": target horizon,
    "predictability": sklearn r2 score
}
```

3.2 Modelling Distillation Columns

To implement the models forecasting pentanes content at the end of the LPG debutanization process we used the scikit-learn (sklearn) and Catboost[17] [5] libraries. In the deployment, the models consume real-time sensor data streams published to a Kafka topic. Each Kafka message contains information regarding the sensor ID, and based on metadata, it is known

[17] https://catboost.ai/

to which debutanizer unit it corresponds. When a new sensor reading arrives, a forecast is made and published with metadata regarding the latest timestamp of the sensor reading, and the debutanizer unit to which the forecast corresponds.

The models were developed with widely used open source libraries given the flexibility and quality of implementation they provide for particular machine learning models. These libraries are widely used in the industry, enhancing the trust regarding their implementation in light of the final product. We did not implement feature creation with the library developed by JSI (described above) due to domain-specific requirements.

In Figure 6 we depict the models' integration. Sensor readings are obtained from the debutanizer column and streamed to a specific Kafka topic, with metadata regarding the timestamp of the reading, the particular sensor ID, and value. On the deployment instance, those values are fed to the model to produce a forecast. A lookup is performed to retrieve meaningful metadata and enrich the message before publishing it to the forecasts' Kafka topic.

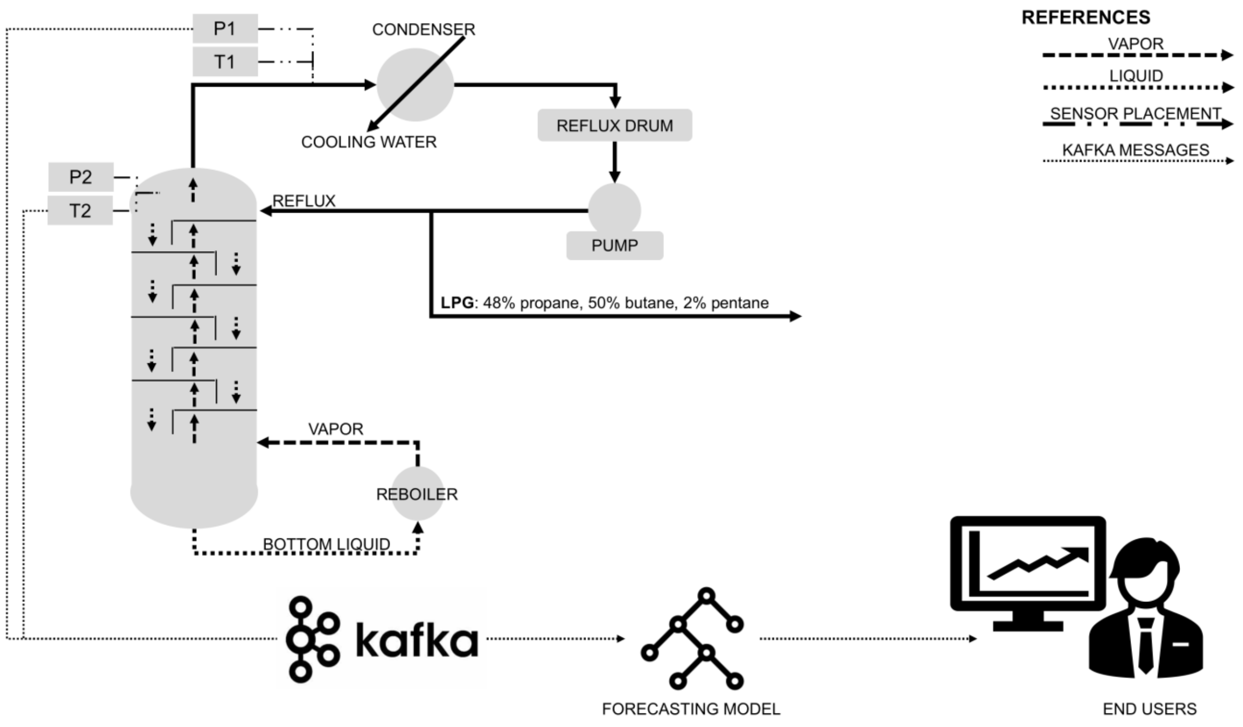


Figure 6: Schematic diagram of the debutanizer column, its sensors, and their integration towards the forecasting model and the end users.

4 Scenarios

This Section presents scenarios from the FACTLOG pilots demonstrating the use of analytics tools. Note that these are demo cases and do not represent final production versions. Details on the pilots themselves can be found in FACTLOG deliverable D1.1 [2].

4.1 Tupras

4.1.1 Predicting Unit State

The data on which we trained and evaluated the models came from Crude Distillation Unit 2 in the Tupras pipeline. The data had any duplicated time readings removed and missing values were imputed using the sklearn iterative imputer[18]. The generated dataset was split into the first 80%, which was used to train the model, and the last 20%, which was used to evaluate performance. The models we created had different number of past values used as the input range and predicted for different prediction horizons in the future.

The input value in the table indicates for how many minutes in the past the data was given to the model. Data was collected in minute intervals, so the value 60 means that all sensor readings for every minute in the last hour were used as input. The prediction horizon indicates for how many minutes in the future all sensor readings are predicted.

The metrics displayed are MAE (Mean Absolute Error) and MSE (Mean Square Error).

Table 1: Model prediction accuracy for different input ranges and different horizons.

Input range [min]	Prediction horizon[min]	MAE	MSE
5	1	3.703	179.67
20	1	4.891	204.54
5	15	6.684	399.55
20	15	9.105	571.04
5	60	8.246	552.97
20	60	9.676	594.82
60	60	9.844	638.30

Results shown in Table 1 are as can be expected – the best accuracy is achieved by models predicting for the near future and giving the model more information by increasing input range also increases the prediction accuracy. The impact of both factors is illustrated in

[18] https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html

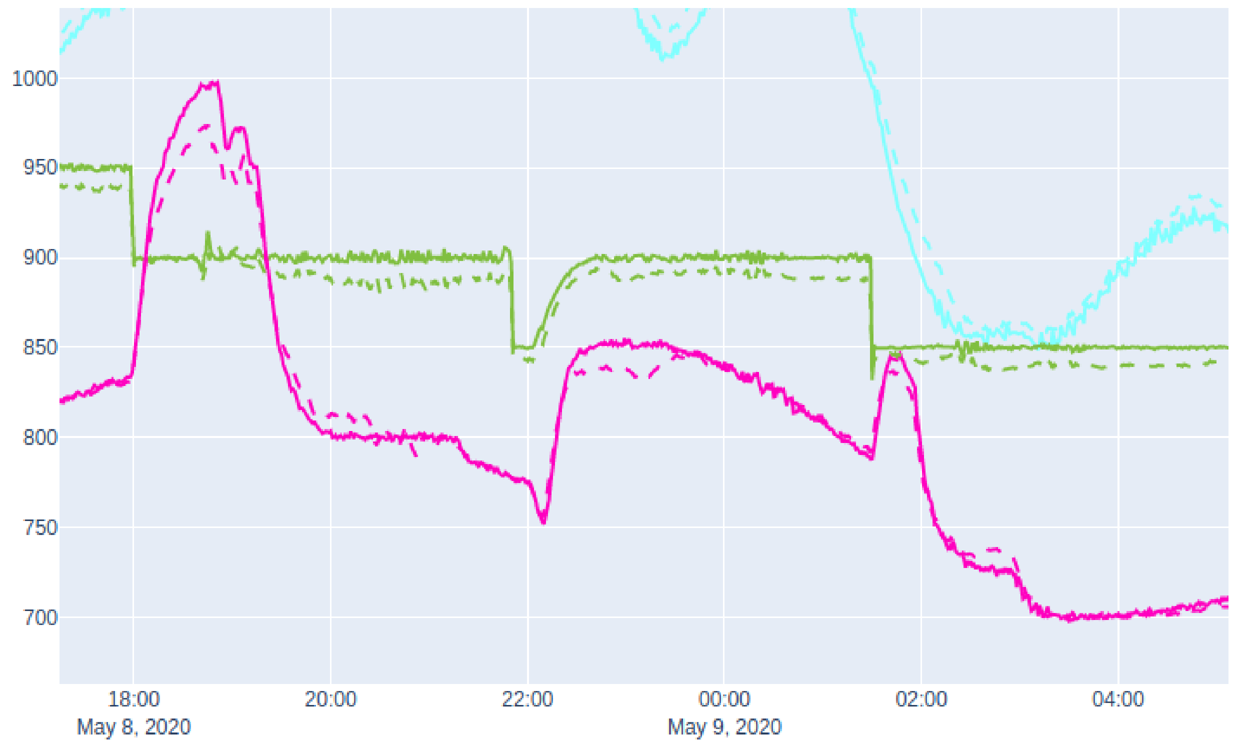


Figure 7: Predictions using the past 60 minutes as input and predicting 60 minutes in the future for some of the sensors in Crude Distillation Unit 2. Full line is the true value while the dotted line represents the prediction.

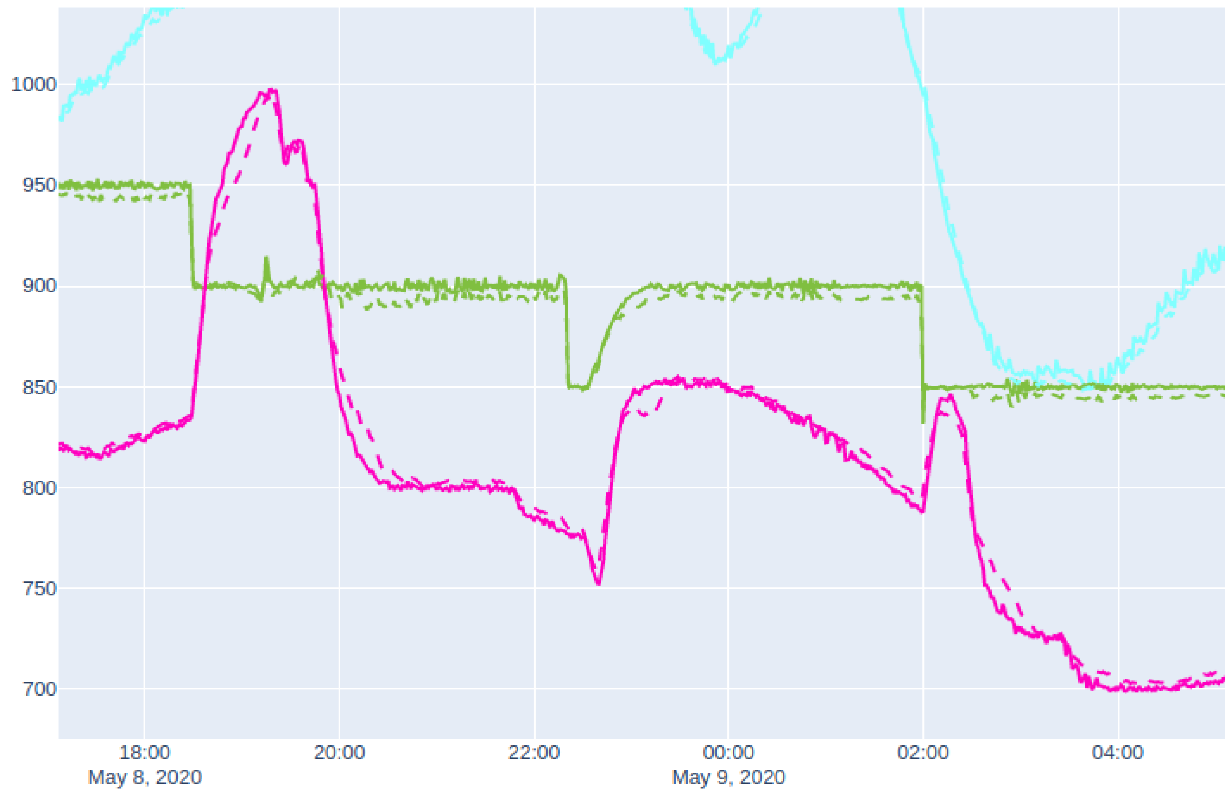


Figure 8: Predictions using the past 30 minutes as input and predicting 30 minutes in the future for some of the sensors in Crude Distillation Unit 2. Full line is the true value while the dotted line represents the prediction.

4.1.2 Predicting LPG Impurities

The Tupras refinery is located in Izmit. It began oil production in 1961 and achieved a design capacity to process 11.3 million tons of crude oil per year. It receives crude oil from four countries: Iran, Iraq, Russia, and Saudi Arabia. The refinery complies with Euro 5 standards, producing diesel, gasoline, and LPG. Our work focuses on the LPG debutanizer units. Given the previous purification steps, the experts consider there is no much variability in light hydrocarbon content regardless the crude oil provenance, and thus changes regarding the feedstock provenance were not subject to modelling.

In order to model the pentanes content at the end of the debutanization process, we created a total of 198 features, and then performed feature selection based on features' mutual information, to avoid overfitting. To measure the models' performance, we executed a ten-fold cross-validation, fifty times. Data we used to train the models was provided for two debutanizer units. In Figure 9 we show measured pentanes content over time for two debutanizer units: Unit A (Fig. A), and Unit B (Fig. B). While the Unit A presents several out-of-spec events, Unit B has no such events, and thus the classifiers' performance was assessed only on events from Unit A. These units have been added to the pipeline at different times and are not identical, so the different behaviour is not entirely surprising and enforces the conclusion that unit-specific models are needed for successful modelling.

Best results were obtained when considering the data from both debutanization units to build a single model. We attribute such result to the fact that more data provided a greater amount of information, enhancing the models' learning process, and thus achieving greater results

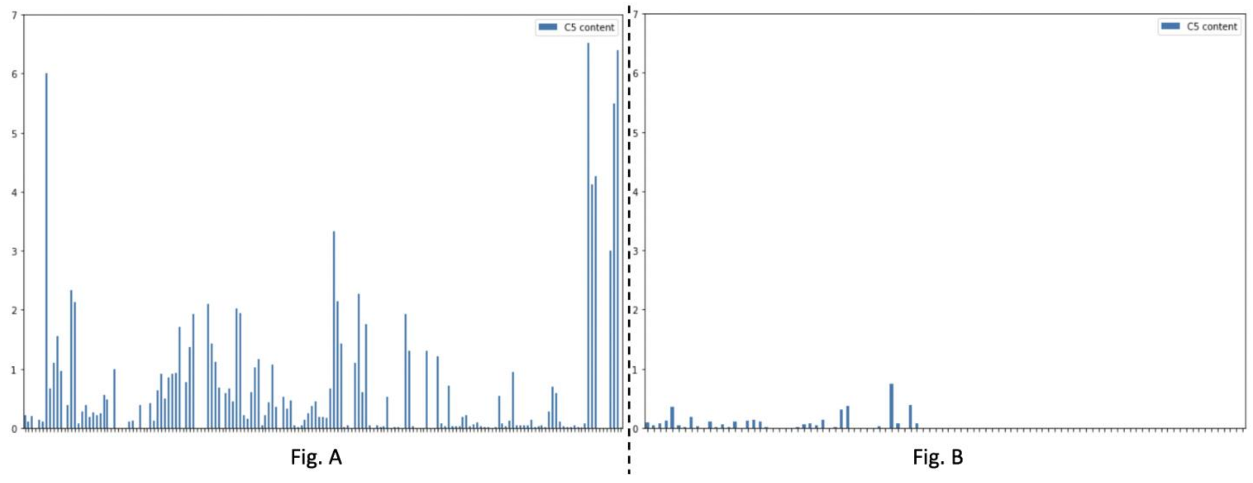


Figure 9: Pentanes content measured in debutanizer units.

We compared the performance of our model’s regression against a baseline forecasting pentanes content forecast as the median of past pentane measurements. We found our model had a stronger performance, surpassing it for Unit A, and achieving almost the same forecast quality for Unit B, when measuring RMSE and MAE.

When measuring the discrimination power of our classification model, we found it achieved a value of 0,7670 for the Area Under the Curve Receiver Operating Characteristics curve, with best results for the models built with data from Unit A and Unit B.

4.2 JEMS

For each of the chambers (B100, B200, and B300) we trained a feedforward neural network predicting the dynamics of sensor measurements. The model predicts values of all sensors in the next hour based on the data from the last 5 hours. Before the training of the models we imputed datasets using linear interpolation. We have not used the GMM layer for imputation yet and that will be our next step.

We tested each model against the baseline model which always predicts the values from the last hour. The mean squared error (MSE) and the mean absolute error (MAE) of each of the models are shown in Table 2. We can see that the mean squared error of a feedforward neural network is smaller than the baseline's MSE. That is not the case for the mean absolute error because the models were built to optimise the MSE. More experiments are still needed to improve the models.

Table 2: Results of feedforward neural network and baseline models for each chamber.

	MAE	MSE
B100 neural network	1.186	14.138
B100 baseline	1.092	18.456
B200 neural network	2.005	30.591
B200 baseline	1.340	37.402
B300 neural network	0.962	21.336
B300 baseline	0.696	21.971

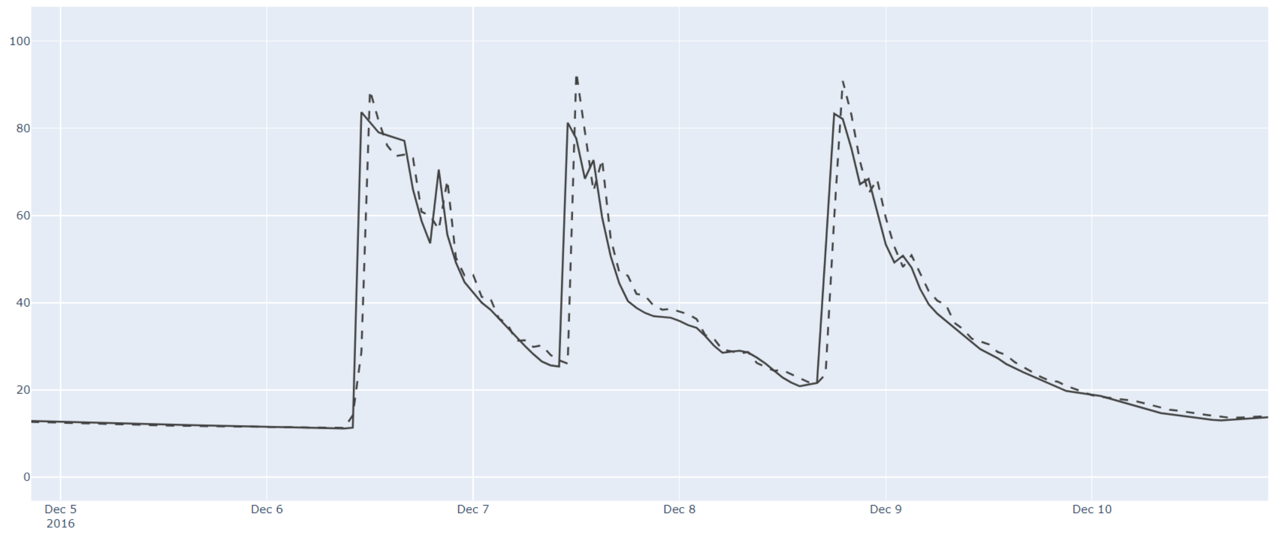


Figure 10: Measured values of sensor 62 from chamber B100 and its predicted values.

In Figure 10 we plotted measured sensor values (solid line) and our model's prediction (dashed line). The model was not able to predict the spikes, because they are result of an external influence that cannot be predicted (e.g. a change in the settings of the plant). On the other hand, in Figure 11 and Figure 12 the model appears to be more successful in predicting sensor values. The exact reason why it does better for these sensors is still investigated, but chambers B200 and B300 are later in the pipeline and so more stable.

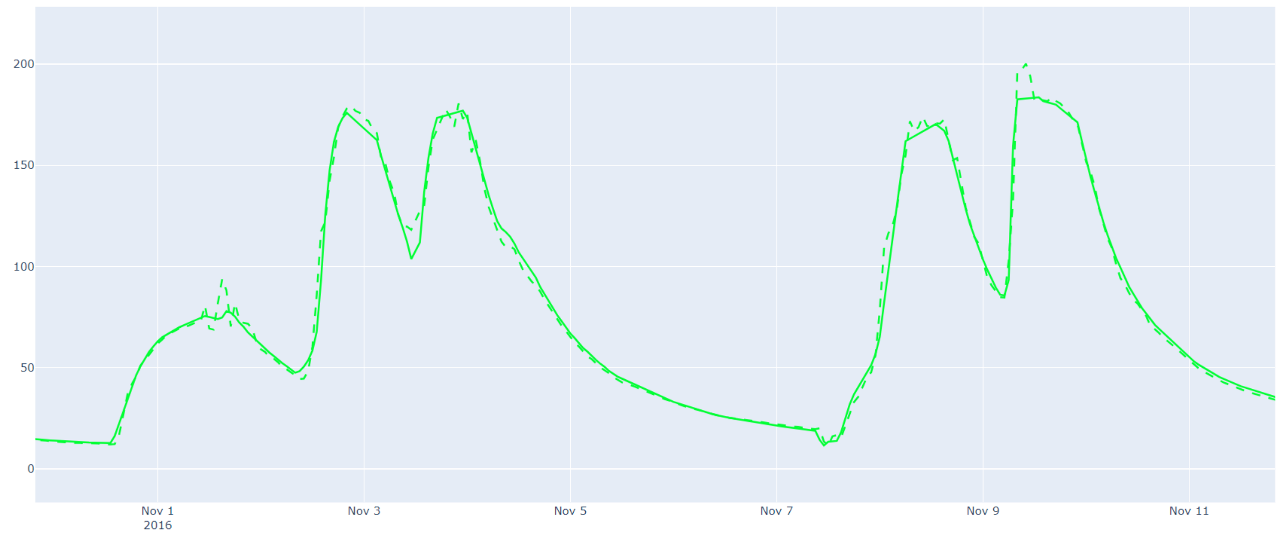


Figure 11: Measured values of sensor 49 from chamber B200 and its predicted values.

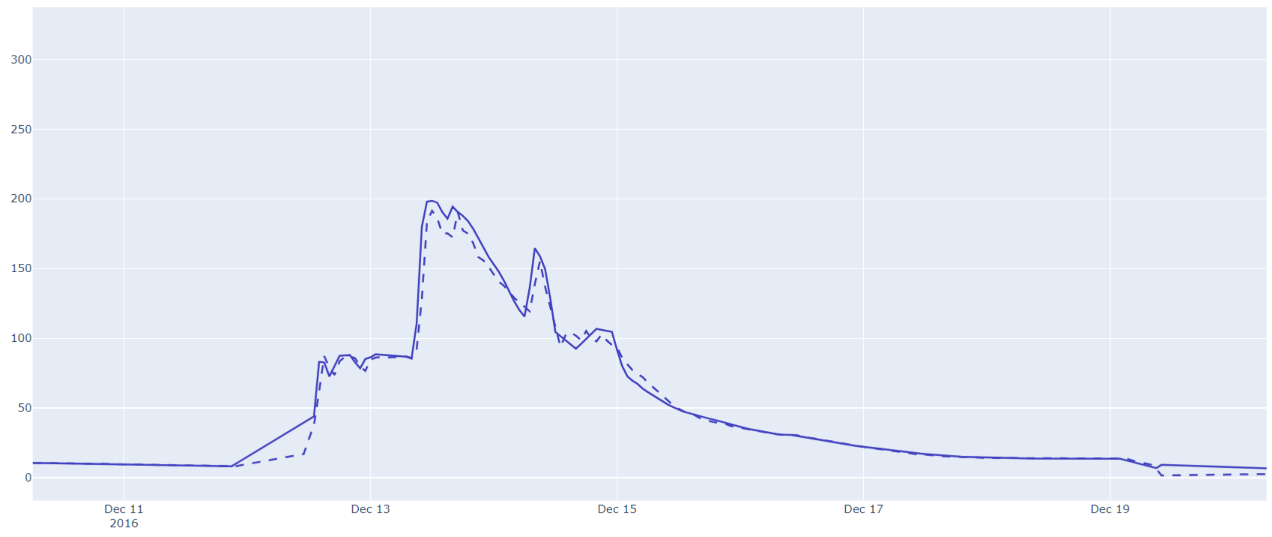


Figure 12: Measured values of sensor 79 from chamber B300 and its predicted values.

We will continue working on the JEMS use-case. As mentioned already, first step will be to use the GMM Layer for imputation.

4.3 Continental

Experimentation related to the unusuality detection (Section 2.4) is currently being performed upon Continental data sets, described in deliverable D1.1 [2].

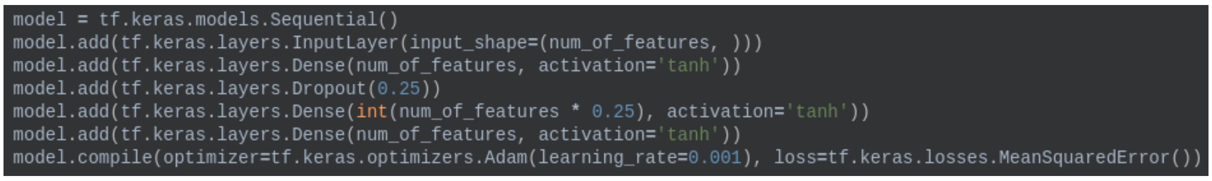
Experimentation is conducted on following parameters from Continental dataset – *Height 1 value*, *Height 2 value*, *Height 3 value*, *Height 4 value*, *Angle 1 value*, *Angle 2 value*, *Angle 3 value* and *Angle 4 value*.

Data is being segmented based on time-gap between each segment. In essence, there are some process parameters that get changed occasionally, and that change introduces said time-gaps, as well as changes in process behaviour.

Parameter standardization of each segment is performed, as well, to ensure that values of one segment do not affect model training and inference on other segments.

4.3.1 Settings

For the experimentation we used the Neural Network with the structure shown in Figure 13 below.



```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(num_of_features, )))
model.add(tf.keras.layers.Dense(num_of_features, activation='tanh'))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Dense(int(num_of_features * 0.25), activation='tanh'))
model.add(tf.keras.layers.Dense(num_of_features, activation='tanh'))
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss=tf.keras.losses.MeanSquaredError())
```

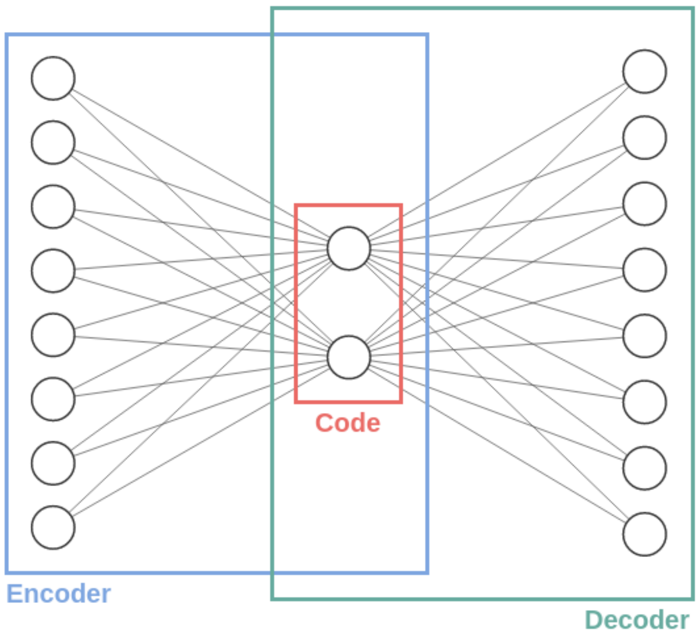


Figure 13: Neural Network model for the num_of_features = 8

The process of basic experimentation (*training and validation of single model*) can be split into following tasks (shown also in Figure 14):

- Split the data into training, validation and test parts;
- Standardize the training data set;
- Standardize the validation data set, using mean and standard deviation of the training data set;
- Standardize the test data set, using mean and standard deviation of the initial data;
- Train the model on training data set, validating it with validation data set;
- Calculate the thresholds based on the reconstruction errors of validation data set;
- Compare the results of validation/test data set with the results of MEWMA performed on the same data set.
- Use test data set to compare the results of ensemble learning models and basic model.

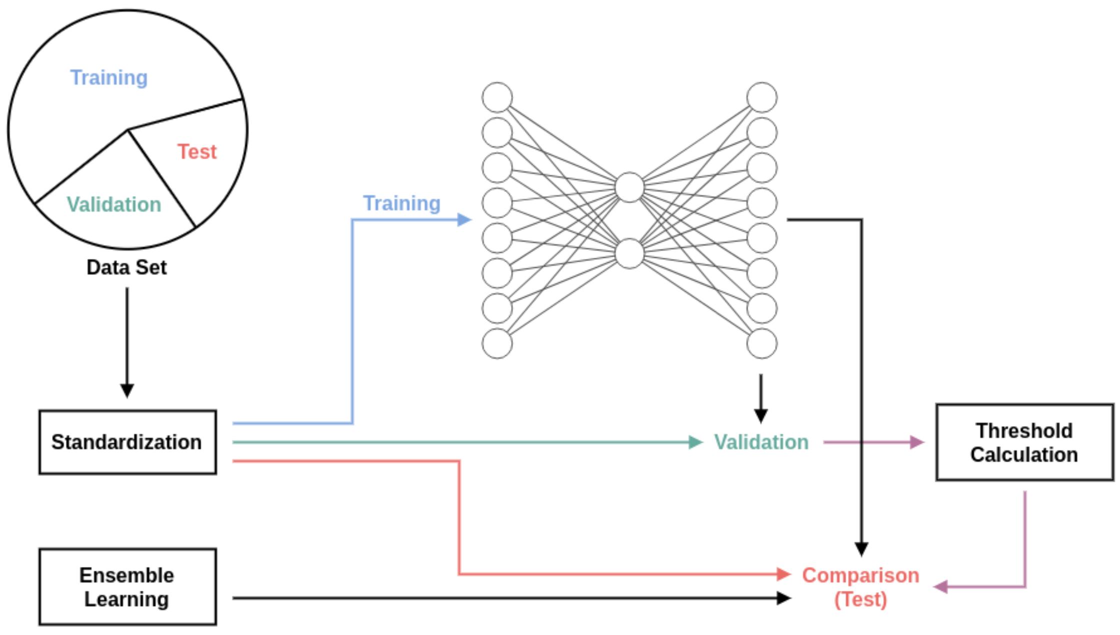


Figure 14: Continental Experimentation process.

In addition to the basic model training, we are employing ensemble learning techniques, explained in Section 2.4.5, and comparing them to the basic model using the test data set.

4.3.2 Results

Depending on the unusuality zone, we get between 30% and 70% match between our model and MEWMA.

Regarding comparison of multiple models trained on different parts of the data set, we get low number of unique unusual instances from all models, relative to the average number of unusual instances each model outputs individually. This means that there is a high percentage of matches from model to model, which looks promising.

For one of the tests, the number of unique instances classified as unusual was 350, with an average number of instances classified as unusual per model being 240 for threshold of 2.4 standard deviations. Out of these 350 instances, 138 were classified as unusual by each model and 237 were classified as unusual by at least 6 of them. In addition, comparison between pairs of models showed that, on average, there was 70-95% match between instances classified as unusual. Total number of models used for said test was 11 and total number of outliers in said test found by MEWMA was 181.

An example of error curve and unusuality zones (*2, 2.4 and 3 standard deviations from the mean*) for one of the validation parts of data set is presented in Figure 15 below.

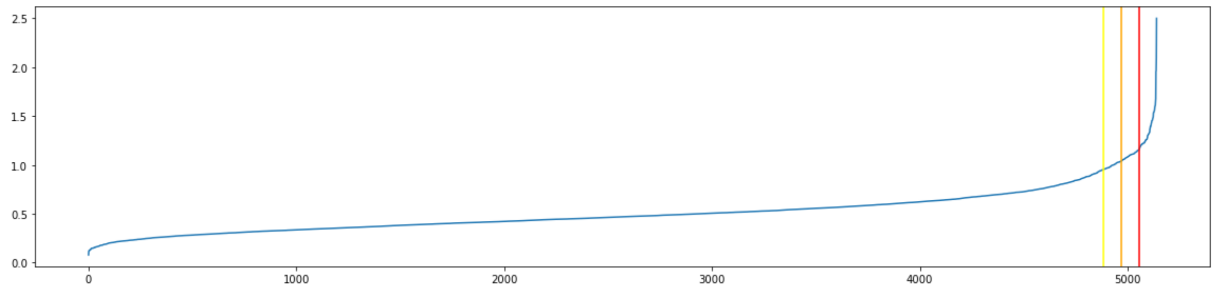


Figure 15: Error curve and unusuality zones.

Regarding the ensemble learning techniques, we have tried all of them which are mentioned in section 2.4.5:

- Bootstrap aggregating (*Bagging*) – There is a 1-5% higher matching rate between this method and MEWMA, than there is between basic model and MEWMA, with voting usually giving slightly better results. Even though there are slight variations when it comes to matching rate (*+1-5% comparing to the basic model*) because of the different test data sets being used and the randomness of training/validation data set split, the results of an ensemble model are always better than the ones from basic model;
- K-fold aggregating – There is a 1-5% higher matching rate between this method and MEWMA, than there is between basic model and MEWMA, with voting usually giving slightly better results. Even though there are slight variations when it comes to matching rate (*+1-5% comparing to the basic model*) because of the different test data sets being used and the randomness of training/validation data set split, the results of an ensemble model are usually better than the ones from basic model. This method usually has slightly worse results than the Bootstrap aggregating method;
- Boosting – There is an occasional improvement, relative to the basic model, regarding the comparison with MEWMA, of up to 2%. However, it can happen that there is no improvement. Cases where boosting made the model worse are rare and can happen.

Averaging was performed on the reconstruction outputs – i.e., one instance is fed to every model and the final result is the average of all models' outputs for that instance. Threshold is calculated based on these average results on the validation data set.

Voting is implemented such as for each model a threshold is determined that is used to classify instances, and if instance is classified/voted for enough times (*N/2 + 1, N = number of models*), then it is anomalous.

References

References are listed in a numbered list, ordered alphabetically as shown in the Reference Section. References are denoted in the text as cross links, e.g. Berners-Lee, T. (1999). *Weaving the Web*. San Francisco: Harper. or "Free/Libre and Open Source Software: Survey and Study - FLOSS" Deliverable D18: FINAL REPORT FLOSS project, June 2002, . **Be careful when updating your reference list because Word sometimes destroys the cross links*.***

[1] Marek Smieja, Łukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. 2018. Processing of missing data by neural networks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 2724–2734.

[2] FACTLOG – Energy-aware Factory Analytics for Process Industry, Deliverable D1.1 "*Reference Scenarios, KPIs and Datasets*".

[3] FACTLOG – Energy-aware Factory Analytics for Process Industry, Deliverable D1.2 "*Cognitive Factory Framework*".

[4] FACTLOG – Energy-aware Factory Analytics for Process Industry, Deliverable D2.1 "*Analytics System Requirements and Design Specification*".

[5] Prokhorenkova, Liudmila, et al. "CatBoost: unbiased boosting with categorical features." arXiv preprint arXiv:1706.09516 (2017).

[6] Rožanec, J.M.; Trajkova, E.; Lu, J.; Sarantinoudis, N.; Arampatzis, G.; Eirinakis, P.; Mourtos, I.; Onat, M.K.; Ataç Yilmaz, D.; Košmerlj, A.; Kenda, K.; Fortuna, B.; Mladenić, D. Cyber-Physical LPG Debutanizer Distillation Columns: Machine Learning-Based Soft Sensors for Product Quality Monitoring. Preprints 2021, 2021100364 (doi: 10.20944/preprints202110.0364.v1).

[7] Welzer, T., Eder, J., Podgorelec, V., & Kamišalić Latifić, A. (Eds.). (2019). Advances in Databases and Information Systems. Lecture Notes in Computer Science. doi:10.1007/978-3-030-28730-6.