Data-driven approaches for modelling of sub-critical coal-fired boiler

Valentin Formont [a,*], Vidar T. Skjervold [a], Lars O. Nord [a]

[a] *NTNU - Norwegian University of Science and Technology, Department of Energy and Process Engineering, Trondheim, Norway*
**valentin.formont@ntnu.no*

Abstract

Due to increasing shares of renewable electricity sources in the grid, thermal power plants need to operate in a more flexible manner in the future. This will involve more frequent startups, shutdowns, and load changes. A central part of a thermal power plant analysed in this study is the coal-fired boiler. In a previous study, a first-principle model of a sub-critical coal-fired boiler has been developed and validated with operational data from a Polish power plant. Based on this model, this work aims to develop a computationally efficient and sufficiently accurate data-driven model that is easy to implement in new software. A selection of multi-output algorithms was first compared using nonoptimised parameters, with very few adaptations to the data set. Then, each algorithm had undergone three different optimisation routines to tune the hyper-parameters. The results of the nonoptimised models were compared with the optimised ones, and then compared to the reference first-principle model using the average Mean Absolute Percentage Error as a score. The methods used comprise six base learners and three algorithms using ensemble methods. The optimisation routines were based on the Powell conjugate direction method, Bayesian optimisation and evolutionary algorithm. All the data-driven models had shown a lower percentage error than the first principle model, and optimisation had resulted in improved prediction capacity for every base learner, but not for ensemble method-based algorithms.

1. Introduction

It is expected that thermal power plants will need to operate more flexibly in the future due to the increased share of renewable energy sources in the grid. This will lead to more frequent startups, shutdowns and load changes. In order to aid this transition, there is a need for models that can describe the transient behavior of such power plants. As a result of the thermal inertia of e.g. boiler walls and heat exchanger surfaces, the boiler has a large influence on the dynamic behavior of a power plant. This unit is therefore an important part of a power plant model, and the aim of this work is to develop a computationally efficient and sufficiently accurate boiler model that is easily implemented in new software.

In the literature, several examples of data-driven modelling approaches applied to coal-fired power plants are available. Based on operational data of around 60 variables in a brown coal-fired power plant, Smrekar et al. developed two artificial neural network (ANN) models [1]. The aim of the work was to examine the feasibility of such models for coal-fired power plants. In Chandrasekharan et al. [2], separate models for the economizer, drum and superheater were developed by using a statistical approach based on the response surface methodology and design of experiments. The data used for model development was collected from a 210 MW power plant over a 2-3 hour period. In Zhu et al. [3], a local model network (LMN) was used to model the boiler-turbine unit of a sub-critical coal-fired power plant and applied as the prediction model in a non-linear model predictive controller. The model was validated with data from a 500 MW unit in China. With emphasis on cyclic operation, Navarkar et al. [4] built an ANN model of a coal-fired steam generator based on 10 years of operational data from a power plant in the USA. Manaf & Abbas [5] and Oko et al. [6] both applied non-linear autoregressive with exogenous input (NLARX) models to coal-fired power plants. In the former, a complete power plant divided into six sub-systems was modelled by using data from a 660 MW power plant. In the latter, the NLARX model was used to predict drum pressure and level in a sub-critical unit. The data used for model development were generated by a validated first-principle model. As indicated by the literature review, a comparison of several different static data-driven modelling approaches for a single coal-fired boiler has not yet been presented. This knowledge gap will be addressed by this work.

Using data-driven approaches has several advantages:

- It uses real data to make predictions. It makes a change from the first-principle models paradigm that simplifications must be made in order to be able to perform computations;
- It is a very active field of research and there is a big community of researchers developing and enhancing the different methods;
- Once a model is trained, it is very fast, within seconds, to get a prediction;
- The model can be updated by getting more data after it is trained, so it can have a better predictive ability.

2. Methodology

Measurements at different stages of a sub-critical coal-fired boiler at a Polish power plant have been performed, constituting the data set for this paper. The same observations are computed using a first principle model, which constitutes the reference data set. Different regressors have been chosen to compare to the reference model : Ordinary Least Square, Elastic Net, Support Vector Machine, Stochastic Gradient Descent, Nearest Neighbors, Decision Trees, Random Forest, AdaBoost and XGBoost. The first six model structures are called 'base learners' and the other are regressors based on ensemble methods. The ensemble methods use several weak learners to improve prediction capacity. These regressors allow the use of hyper-parameters, i.e. parameters independent from the observations, and the value of these are usually to be determined. First, the regressors are fit to the observations without prior hyper-parameter optimisation. The Mean Absolute Percentage Error (MAPE) of each model is kept for comparison. Then, using different optimisation routines the same methods are used combined with hyper-parameter optimisation. As a result, for each algorithm, one nonoptimised and three optimised models are created.

2.1. Data from industrial boiler

The data from the industrial boiler comprise of 480 observations including 5 inputs and 7 outputs each taken at a 1-minute interval. A sketch of the coal-fired boiler considered in this work is shown in Figure 1.

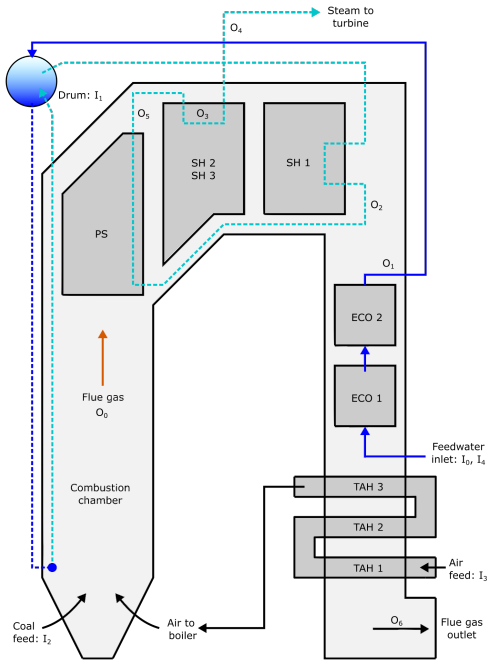

Figure 1: Sub-critical coal-fired boiler considered in this work. TAH stands for tubed air heater, ECO stands for economizer, SH stands for superheater and PS stands for platen superheater. Inputs and outputs, denoted by I and O, are described in Table 1.

The different variables are shown in Table 1. It is important to note that, for this paper, the time is not included in the set of inputs. The reason is that including time in the inputs shifts the analysis into a time-series analysis, and the target of this paper is to develop a multi-output model solely dependent on the state of the boiler. The time-series will be considered for future work.

Table 1: Monitored variables

Inputs	**Code**	**Unit**	**Bounds**
Time	None	min	[0, 480]
feedwater T	I_0	°C	[225, 236]
Pressure in the steam drum	I_1	MPa	[10, 11]
Fuel mass flow rate	I_2	kg/s	[6, 8]
Air flow rate	I_3	kg/s	[54, 65]
feedwater mass flow rate	I_4	t/s	[0, 51]
Outputs			
Flue gas T combustion chamber	O_0	°C	[977, 1033]
Water T from ECO2	O_1	°C	[307, 319]
Steam T I-st stage SH outlet	O_2	°C	[366, 387]
Steam T II-nd stage SH outlet	O_3	°C	[479, 497]
Steam T III-rd stage SH outlet	O_4	°C	[530, 544]
Steam T PS outlet	O_5	°C	[448, 475]
Flue gas T after TAH1	O_6	°C	[154, 173]

First, the Pearson correlation is calculated for each input variable. Given x_1 and x_2 two different variables in a data set comprising of n observations, it is possible to calculate the strength of association between these two variables by computing the Pearson's coefficient (Equation 1).

$$r = \frac{\sum_{i=1}^{n}(x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{\sqrt{\sum(x_{1i} - \bar{x}_1)^2 \sum(x_{2i} - \bar{x}_2)^2}} \tag{1}$$

For readability, the results of the correlation analysis are displayed in a heatmap in Figure 2. The input names are coded for visualisation purposes, and their code can be retrieved from Table 1. The data-driven models usually perform poorly when the data are not on the same scale, thus it is needed to have a step of feature scaling, Géron [7]. Standardisation offers a robust way of scaling the data by transforming the entire data properties so they follow an unknown distribution with $X \sim \mathcal{D}(0, 1)$. The formula used for standard scaling is given in Equation 2.

$$x_{i,\text{scaled}} = \frac{x_i - \mu}{\sigma} \tag{2}$$

with μ the mean of the observations and σ the associated standard deviation.

To assess the performance of the models, it is common to split the data set into a training set and a test set, Tan et al. [8]. The reason behind this first split is to keep an untouched data set. The test set is not used to train the model, nor to perform hyper-parameter tuning. After the model has been fit to the training set, it is rated using a score function calculated on the test set.

2.2. Data-driven modelling techniques

The regressors used to get predictive models are categorized into two groups: base learners and ensemble methods. The main difference between a base learner and one from ensemble method is that the ensemble method uses several base learners to make predictions, whereas a base learner is a single model. To assess the performance of each model the Mean Absolute Percentage Error (Equation 3) is computed. This choice of score function is justified by the noisy behaviour of the observations. The squared error metrics such as Mean Squared Error or Root Mean Squared Error penalize outliers by squaring the error, making them very sensitive to noise.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{3}$$

with y_i the real value and $\hat{y}$ the predicted value of the i_{th} observation.

As the model should be a multi-output regressor, there are actually 7 different MAPE per model. The error is computed for the 7 outputs, and then averaged to produce an average MAPE. Calculating the MAPE on the entire training set is not robust enough, because the model only sees one set of data, and there is no possibility to know how the model would behave with other samples from the same experiment. This is why the performance on the training set is assessed by implementing cross-validation. The cross-validation is performed along with K-fold sampling. Given a data set with n observations, the K-fold sampling method splits the data set randomly into k subsets. $k-1$ subsets are used for training the model and one subset acts as a test set. This is repeated until all the subsets acted once as a test set. Then, the outcome of the cross-validation is the mean of the k model scores. Performing cross-validation allows getting a robust estimation of both the bias and the variance of a model. The models will be fit to measurements of a real power plant, and the reference to outperform is a data set coming from the first principle model. The MAPE of the reference model compared to the real data model is 0.716%. All the hyper-parameter initial values are displayed in Table 2. Except where specified, these values are the values recommended by default by the different libraries used. Sci-kit learn is used for every model except XGBoost, which comes from the homonym library [9].

Table 2: Regressors' Hyper-parameters initial values

Algorithm	**Hyper-parameter**	**Value**
Elastic Net	λ	1
	α	0.5
SVR	Kernel	RBF
	σ	$\frac{1}{n_{\text{predictors}} \cdot \text{var}(X)}$
	L_2 penalty	1
	ε	0.1
SGD	Loss	huber
	iterations	10000
	ϵ	0.1
	η	adaptative
KNN	k	3
	weights	distance
	data structure	ball tree
	s_{leaf}	30
	p	2
Decision Tree	criterion	absolute error
	$d_{\max}$	$\varnothing$
	$ss_{\min}$	2
	$sl_{\min}$	1
	$f_{\max}$	$n_{\text{predictors}}$
Random Forest	N	100
	criterion	absolute error
AdaBoost	N	50
	η	1
XGBoost	N	100
	η	0.3
	$d_{\max}$	6
	sb_s	1
	cols	1
	γ	0
	$cw_{\min}$	1
	α	0
	λ	1

2.2.1. Ordinary Least Square Regression

Given a set of n observations $\{x_i, y_i\}_{i=1}^{n}$, each observation i comprises a column vector x_i of p predictors such as $x_i = [x_{0i}, x_{1i}, x_{2i}, ..., x_{pi}]^T$ and a response y. The simplest form of linear regression assumes that y is a function of x_i such as

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i \quad (4)$$

with ε_i accounting for the variations not explained by the regression model (bias) and β a column vector of unknown coefficients with the shape $p \times 1$. Finding these coefficients consists of optimizing a score P. Using the Mean Squared Error (MSE) as a score function, it is possible to estimate the unknown coefficients $\hat{\beta}$ using the matrix form of the *normal equation* displayed in Equation 5, Goodfellow et al. [10]

$$\hat{\beta} = (X^T X)^{-1} X^T y \quad (5)$$

$$\text{with } X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,n} \\ x_{2,1} & x_{2,2} & \dots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{p,1} & x_{p,2} & \dots & x_{p,n} \end{bmatrix}$$

The *normal equation* is obtained by minimizing the MSE in Equation 6

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{2n} \|Y - X\beta\|^2 \quad (6)$$

It is one of the simplest regression techniques, but falls short when the underlying data structure is complex, Zou and Hastie [11].

2.2.2. Elastic Net Regularisation

The Elastic Net Regularisation is a combination of two different regularisation techniques, the Least Absolute Shrinkage and Selection Operator (LASSO) known as L_1 regularisation and the Tikhonov or L_2 regularisation. These regularisation methods add a penalty function to the loss function in Equation 6. The penalty is based on the L_1 and L_2 norms. In Equation 7, it is possible to identify on the right-hand side the first term as the cost function of the Ordinary Least Square (OLS). The second term and the third term represent respectively the L_1 and L_2 penalties.

$$\begin{aligned} \hat{\beta}_{\text{net}} = \arg\min_{\beta} \frac{1}{2n} \|Y - X\beta\|^2 + \\ \lambda\alpha \|\beta\|_1 + \frac{1}{2}\lambda(1-\alpha) \|\beta\|^2 \end{aligned} \quad (7)$$

where λ is a parameter controlling the strength of regularisation, and α is strictly between 0 and 1. If $\alpha = 1$, the elastic net is the same as LASSO and as α tends toward 0, it gets closer to ridge regression. The Elastic Net is the same as an Ordinary Least Squares when $\lambda = 0$.

2.2.3. Support Vector Regression

Support Vector Machine (SVM) was initially used for classification, transforming a binary classification problem into a convex optimisation. The main principle behind SVM is to find a function $f(x)$ within a maximum deviation ε compared to the real response y. The function should be as flat as possible. To illustrate this method, $f(x)$ is given linear, and can be written as in Equation 8, Vapnik [12].

$$f(x) = < w, x > + b \quad (8)$$

with w a normal vector defining the hyperplane $< w, x >= b$
For this case, the problem lies in minimizing the loss function represented in Equation 9.

$$\arg\min_{w} \frac{1}{2} \|w\|^2 \tag{9}$$

with the following constraints to respect

$$C = \begin{cases} y_i - < w, x_i > -b \leq \varepsilon \\ < w, x_i > +b - y_i \leq \varepsilon \end{cases} \tag{10}$$

SVM for regression is shortened to SVR in this study. SVR combined with feature transformation allows modelling of complex data distribution. The underlying idea is that if a data structure is not linear in its original space, it might be linear in another space. The function used to change the space is a kernel function. Its general form is written

$$K(x, u) = \phi(x) \cdot \phi(u) \tag{11}$$

with x and u two independent vectors belonging to the same space. In the result section, the Radial Basis Function (RBF) is used as a kernel

$$K_{\text{RBF}}(x, u) = e^{\left(-\frac{\|x-u\|^2}{2\sigma^2}\right)} \tag{12}$$

σ is a free parameter called the length scale of the kernel. Its value is

$$\sigma = \frac{1}{n_{\text{predictors}} \cdot \text{var}(X)} \tag{13}$$

In addition, it is possible to add an elastic net regularisation to the cost function. Smola and Schõlkopf [13] made a tutorial about SVR, which details the full scope of this method.

2.2.4. Stochastic Gradient Descent Regression
Stochastic Gradient Descent is initially an iterative method used in optimisation to find an extremum. It inherits from Batch Gradient Descent (BGD), with the main difference that BGD computes the exact gradient of the loss function based on every single observation. On the other side, SGD uses an approximated gradient based on one point at a time. Considering an arbitrary loss function $J(\theta)$, it often takes the form of a sum which can be seen as

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\theta) \tag{14}$$

θ is obtained by iteration

$$\theta := \theta - \eta \nabla f_i(\theta) \tag{15}$$

with $f_i(\theta)$ a prediction of the i_{th} observation and η the learning rate of the gradient descent. The parameter η influences the distance between two steps, if it is too large the algorithm might never find an optimal solution. On the other hand, if the learning rate is too small, the algorithm will be very slow to converge. A common method is to use an adaptative learning rate which is set high then decreases as the optimisation converges to a minimum. SGD Regression uses this optimisation routine to fit a linear regression. For instance, if the squared error is used as a loss function, then performing a regression with SGD-based regression is equivalent to an OLS. The Huber loss (Equation 16) is a more robust loss function. It penalizes less the outliers by being quadratic when the error is smaller than ϵ and linear otherwise, Huber [14].

$$J_h(\theta) = \begin{cases} \frac{1}{2}(y - f(\theta))^2 \text{ for } y - f(\theta) \leq \epsilon \\ \epsilon \cdot (|(y - f(\theta)| - \frac{1}{2}\epsilon) \text{ for } y - f(\theta) > \epsilon \end{cases} \tag{16}$$

This method is very suitable for large-scale applications, but can be slower than BGD for convex functions with a single minimum, Bottou and Bousquet [15].

2.2.5. Nearest Neighbors Regression
Nearest Neighbors is an algorithm dealing with both classification and regression. The intuition behind this algorithm is that if a group of observations is close to each other, their responses should be close too. Given a query point q, a k nearest neighbors search consists of finding the k nearest observations to the query point. In its naivest implementation, the euclidean distance is computed using a brute force approach, meaning that the distance is calculated for each pair of points in the entire set of observations. It is then possible to generalize this process to any distance using the Minkowski distance (Equation 17)

$$D_{\text{mink}}(x_1, x_2) = \left(\sum_{i=0}^{n} (|x_{1i} - x_{2i}|^p \right)^{\frac{1}{p}} \tag{17}$$

This is very efficient for a small number of observations, but can reach life span calculation time very quickly as the size of the data set gets bigger. Instead of computing the proximity of every single pair of points with a brute force approach, modifying the data structure before applying a nearest neighbors search has led to more reasonable results for higher dimensional or bigger data sets. One of the data structures speeding up the nearest neighbors search is the Kd-tree data structure. For n observations with d dimensions, the i_{th} coordinate is split using the median value as a separator. It results in two partitions of the observations, L_1 and G_1, respectively the values lower and greater than the median. Then the process is repeated for the coordinate $i + 1$ and iterates once for each dimension. This process goes on until the leaves of the Kd-tree contain a maximum of s_{leaf} observations. An improvement of the Kd-Tree is to use round-shaped leaves instead of rectangles to create clusters. This is called a ball tree data structure. Andrew Moore [16] detailed how this data structure is created. As a result of this new data structure, the nearest neighbors search on the query point is compared only to a small subset of observations, lying within the same leaf or surrounding leaves. For regression, the label assigned to an observation is based on the weighted mean of the labels of its nearest neighbors, Pedregosa et al. [17]. The weights are inversely proportional to the distance between the query point and the neighbors. The determination of the optimal number of neighbors k is made by comparing the error of KNN models with different k values.

2.2.6. Decision Trees
A decision tree is initially a classification algorithm. It splits the observations into different leaves, just like Kd-tree. It comprises an initial node, including all the observations, internal nodes splitting the observation according to decision rules and leaf nodes, the final leaves of the tree. The decision criteria to split a node for a Decision Tree in the case of regression is usually the MSE, but will be set to MAE for this paper. The observations

are split such that the MSE within a leaf is minimized, and simple models can be fit to the different leaves. This partition continues until there is no possible point to split, or a termination criterion is reached. Common parameters to tune include $d_{\max}$ the maximum depth of the tree, $ss_{\min}$ the minimum number of observations that a node must contain to split, $sl_{\min}$ the minimum number of observations that a leaf must contain to be considered as a leaf node and $f_{\max}$the number of predictors to consider for the best split. Note that the maximum depth of the tree can be set to none, so that the tree expands until all leaves are pure. The main advantage of this algorithm, which is also its main drawback, is its simplicity. It is very sensitive to the data structure, and adding few observations can change drastically the structure of the tree. Thus, this model is associated with high variance and does not compare well to other more stable algorithms.

2.2.7. Random Forest

In 2001, Breiman [18] formally proposed an extension of the Decision Tree algorithm. The intuition behind this improvement is to use several subsets based on the original database to create different decision trees and then average their predictions. First, the observations are bagged N times, creating N random samples with replacement and N decision trees are fit to the different random samples. Then the final prediction for an unknown observation x is the average of the predictions of all the decision trees, such as

$$R(x) = \frac{1}{N}\sum_{i=1}^{N} F_i(x) \tag{18}$$

where F_i is the output of the i_{th} decision tree from the Random Forest.

This method reduces readability and has a small increase in the bias, but improves generalisation. Some hyper-parameters are unique to the forest, but many are shared between the trees and the forest. For instance, the number of trees is unique to the forest, but the maximum depth or the minimum samples split are parameters for the trees inside the forest.

2.2.8. AdaBoost

Combining several weak learners into a strong learner is called boosting, and the flagship of this method is the Adaptative Boosting algorithm, AdaBoost. The AdaBoost algorithm is very close to a random forest, but instead of having a uniform average of the decision trees, AdaBoost keeps weights associated with every single weak learner. Initially, all the weights are equal, but after each iteration, the weak learners with the worst performance get an increased weight. As a result, the algorithm will focus on minimizing the error on the weak learners with high weights. AdaBoost for regression, called AdaBoost.R2 is detailed by Drucker [19].

2.2.9. XGBoost

eXtra Gradient Boosting is one of the best implementations of gradient boosted trees. It has actually dominated machine-learning competition for a long time, and is recognized as one of the most accurate modeling methods. The main difference between XGboost and the other gradient boosting techniques lies in the optimisation of the algorithm structure. For instance, it treats calculations of base learners in parallel. Another example of improvement is to use the $d_{\max}$ as a stopping criterion instead of a score to compute, reducing the computation time. The hyper-parameters related to XGBoost are N the number of trees; η the learning rate or shrinkage factor to apply on the weights; sb_s the proportion of observations to shuffle and use from the training set, cols the fraction of predictors to use within each tree (feature reduction), γ the minimum loss reduction to split a leaf node, $wc_{\min}$ the minimum weight instance a child must carry in order to exist, and α and λ respectively the L_1 and L_2 regularisation factors. Other enhancements have been implemented. See Chen and Guestrin [9] for further details about Extreme Gradient Boosting.

2.3. Optimisation methods

All the methods evoked in the last sections include parameters independent from the data. These parameters are called hyper-parameters, and the optimisation of these values is still an active research topic. The naivest approach to optimize them is the brute force approach. Because it is not very realistic to do an exhaustive brute force search, a greedy search can be performed on manually specified values to narrow the search space. This method is called the Grid Search approach, but it requires to have an intuition about the parameters optimal values. The most promising way to find the best parameters is to use more complex optimisation methods. For this paper, the cost function is the result of the cross validation on the training set (Equation 19). It can be calculated given θ, an unknown vector of hyper-parameters, and $F(\theta)$, a model already fit to a data set.

$$J(\theta) = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - F_i^k(\theta)}{y_i}\right| \times 100 \tag{19}$$

Three different optimisation techniques will be used and compared on different data-driven models.

2.3.1. Powell Method

The Powell Method as stated by Powell in 1964 [20] is a gradient-free optimisation method derived from the conjugate direction method. It assumes that by combining several one-dimensional optimisations, it is possible to find the optimum of complex multidimensional functions, The optimisation starts with an initial point x_0 and a set of N guess vectors either manually given or parallel to each axis. A bi-directional search is performed along each vector, and a search vector is created based on the result of the search, which gives another search point. This process goes on until a criterion is met. The implementation of the Powell Method is made with SciPy [21].

2.3.2. Bayesian optimisation

The Bayesian optimisation comes from Bayesian Statistics. In Bayesian statistics, the inference of a model is a probabilistic approach based on a set of prior/posterior beliefs about an event. Given the event of fitting a group of observations X into a specific model, the prior probability distribution P(θ) is the initial belief about the distribution of the parameters θ. In the case of linear regression, the event would be to have the right distribution of β for X. After collecting new observations, an updated probability distribution is made based on the prior and the new observations: the posterior probability distribution, $P(\theta|X)$

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)} \tag{20}$$

with $P(X|\theta)$ the likelihood of X and $P(X)$ the marginal likelihood given by

$$P(X) = \int P(X|\theta)P(\theta)d\theta \tag{21}$$

The Bayesian optimisation method assumes that the unknown function is generated with a Gaussian Process. A Gaussian process is a stochastic process defined by its mean $\mu(x)$ and its covariance $k(x, x')$.

$$\begin{aligned} \mu(x) &= \mathbb{E}|f(x)|, \\ k(x, x') &= \text{cov}(x, x') \end{aligned} \tag{22}$$

such that

$$f(x) \sim \text{GP}(\mu(x), k(x, x')) \tag{23}$$

For a detailed explanation of how Gaussian Processes work, see Rasmussen and Williams [22]. After generating a prior over a Gaussian process, the posterior, also called acquisition function, is generated. Three main acquisition functions are mainly used in Bayesian optimisation; the Probability of Improvement, the Expected improvement and the Upper/Lower Confidence Bound. For this paper, the Upper Confidence Bound (UCB) is used as the acquisition function. Once the UCB function is evaluated, the next points for optimisation are chosen where the value of UCB is the highest, ie. where the uncertainty about the function estimation is at its highest. In the result section, 100 iterations were made with 5 initial random guess points. A practical guide for Bayesian optimisation has been written by Snoek et al. [23]. The Bayesian optimisation is implemented using the bayes_opt library [24].

2.3.3. Genetic Algorithm

The Genetic algorithm is a population-based metaheuristic optimisation method mimicking biological evolution. A first population p of chromosomes is randomly generated. Each chromosome comprises n_{genes} genes. Each gene has one random value attributed. The population comprises n_{chr} chromosomes. The cost function $J(\theta)$ is calculated for each chromosome, and a fitness function guides the evolution of the next population generation. The fitness function is a positive gain function. In this paper, the fitness function is the inverse of the cost function. To generate the next population $p + 1$, n_{parents} chromosomes are coupled together, and as a result, a child chromosome is created. Stronger chromosomes, with better fitness scores, are more likely to mate. The child is made of genes from the parents. This mating process is repeated until the next population is full. However, to avoid losing the best solutions in the mating process, k_{elit} genes with the best fitness function output will be kept in the population $p + 1$. At last, random mutations can happen during the process, changing spontaneously the value of one gene. These mutations have a probability of p_{mutation} to happen for each chromosome. The implementation of the Genetic Algorithm is made using PyGAD [25] and the values of the hyper-parameters are displayed in Table 3.
A detailed review of the Genetic Algorithm has been made by Katoch et al. [26].

3. Results and discussion

The first step before modeling is to perform the correlation analysis of the predictors. As shown in Figure 2, the feedwater temperature has a mild correlation coefficient with all the other inputs. These coefficients lie within an interval between [0.38,0.52]. However, the pressure in the drum; the fuel mass flow rate; air flow rate and feedwater mass flow rate are all highly correlated with each other [0.94,1]. It can be a possible lead for future improvement, because dropping highly correlated predictors within a model is usually associated with a low bias drop but a faster execution.

Table 3: Genetic Algorithm Parameters

Parameter	Value
$n_{\text{generation}}$	20
n_{parents}	4
n_{chr}	8
n_{genes}	$n_{\text{predictors}}$
k_{elit}	1
p_{mutation}	0.1

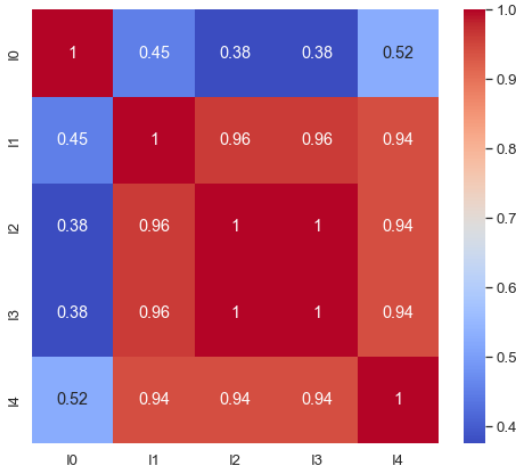

Figure 2: Pearson's correlation coefficients heatmap

For this paper, all the inputs, with the exception of time, are included in the models. Then the data set is split into a training set and a test set containing respectively 70% and 30% of the observations. The data set is randomly split into two subsets, therefore all the base learners do not result in the same outcome if the split happens in a different way. This is especially true for ensemble methods, which add even more randomness using bagging and other stochastic methods. However, cross-validation and several runs of the algorithms ensure that the results of this paper come with a minimized variance. As mentioned in the Methodology section, the reference model to outperform has a MAPE of 0.716%. Nonoptimised values are used to assess 'default' models. Some minor adaptations have been made. For instance, KNN uses the best K from the elbow method, and the SGD regressor minimizes the Huber loss. Table 4 shows the comparison of the average MAPE for the different models mentioned in the Methodology section, the hyper-parameter values associated with each model are in Table 2. Overall, every algorithm outperformed the reference model, by 15% for the worst performing model and 57% for the best performing model. The models do have a good generalisation capacity on the test set, ie. the test error is smaller than the training error. The two best algorithms are issued from ensemble methods. It follows a pattern, which is the more complex the algorithm is, the better the model prediction capacity will be, except for AdaBoost. Indeed very simple algorithms like OLS or Elastic Net are the worst-performing, whereas XGBoost and Random

Forest have the best prediction scores.

Table 4: Nonoptimised prediction models

Model	Training MAPE	Test MAPE
OLS	.472	.429
Elastic Net	.634	.609
SVR	.391	.361
SGD	.473	.437
K-nearest neighbors	.340	.324
Decision Tree	.393	.370
Random Forest	.328	.310
AdaBoost	.397	.390
XGBoost	.349	.317

Then every model, with the exception of OLS, goes through 3 different optimisation methods. The result of the optimisation is shown in Table 5 and the hyper-parameters values for the optimisation are shown in Table 6. In general, there is a net improvement after optimisation of the hyper-parameters. The best improvement was obtained on the worst-performing model. After optimisation, the worst-performing algorithm outperformed the reference by 40% and the best by 58%. It is important to notice that the strength of regularisation λ is set to 0 after optimisation, so regularisation is not necessary on the observations. Fine-tuning the parameters led to a net improvement of the prediction capacity. However, the Random Forest and the XGboost did not benefit from the optimisation. These regressors are generally optimised by default to general cases and it is possible that the bounds of hyper-parameters need to be shrunk, especially for XGBoost regularisation parameters. Out of eight regressors, only two did not benefit from optimisation. The Powell methods did not find the best solution in any cases; Bayesian optimisation 4 solutions; and the Genetic Algorithm 2. The results do not mean that Bayesian optimisation is a better optimizer, only that for this specific 'random state', it found better solutions. Another interesting finding is that the genetic algorithm seems to prioritize solutions close to the given bounds of the hyper-parameter values.

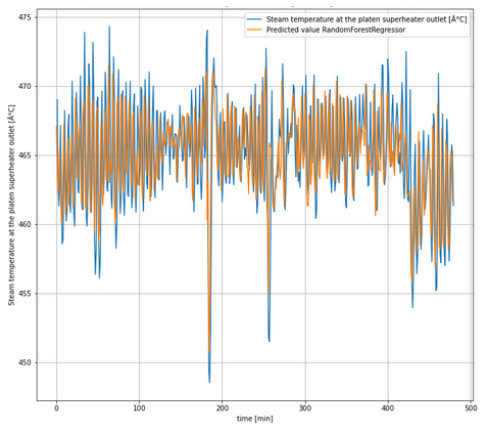

Figure 3: Univariate Time Serie representation of a Random Forest prediction of the Steam Temperature at the platen superheater outlet

These results can be criticized in several ways. Data-driven approaches come with their unique drawbacks. Compared to first-principle models, these algorithms are usually considered as black-box functions. They differ from the original 'laws' and must be represented with

Table 5: Nonoptimised and optimised predictions' MAPE on test set. NO stands for Not optimised, PWL for Powell method, BO for Bayesian Optimization and GA for Genetic Algorithm

Model	NO	PWL	BO	GA	Maximum gain
Elastic Net	.634	.428	.428	.428	29.77
SVR	.361	.301	.300	.299	17.04
SGD	.437	.905	.396	.401	9.38
K-nearest neighbors	.324	.307	.319	.304	6.14
Decision Tree	.370	.614	.327	.346	11.22
Random Forest	.310	.322	.315	.314	-1.51
AdaBoost	.390	.374	.353	.361	9.41
XGBoost	.317	.991	.565	.389	-22.75

Table 6: Regressors' Hyper-parameters optimised values. NO stands for Not optimised, PWL for Powell method, BO for Bayesian Optimization and GA for Genetic Algorithm

Model	Param.	Bounds	PWL	BO	GA
Elastic Net	λ	[0;1]	.01	.00	.00
	α	[0;1]	.94	.00	.00
SVR	σ	[0;1]	.74	.98	1.00
	λ	[0;10]	10.00	9.52	10.00
	ε	[0;5]	$1e^{-2}$	$1e^{-3}$	$1e^{-3}$
SGD	λ	[0;10]	6.18	.00	.00
	η_{ini}	$[1e^{-7};2]$	1.24	2	2
	α	[0;1]	.69	1	.00
	ε	[0;1]	.76	.36	1
KNN	k	[2;50]	4	5	2
	s_{leaf}	[1;100]	99	26	1
	p	[1;5]	4	4	5
Decision Tree	$d_{\max}$	[2;50]	38	6	50
	$ss_{\min}$	[0;1[	.11	.00	.00
	$sl_{\min}$	[0;1[	.17	.00	.00
	$f_{\max}$	]0; 1]	.71	1.00	1.00
Random Forest	N	[10;500]	197	434	500
	$d_{\max}$	[2;50]	31	18	50
	$ss_{\min}$	[0;1[	$1e^{-3}$	.00	.00
AdaBoost	N	[10;500]	207	126	10
	η	$[1e^{-7};1.5]$	.55	$2e^{-3}$	$1e^{-7}$
XGBoost	N	[10;500]	210	494	500
	η	$[1e^{-3};2]$	1.98	1.61	2
	$d_{\max}$	[2;50]	49	5	50
	sb_s	$[1e^{-3};1]$	.36	.67	1
	cols	$[1e^{-7};1]$	1	.30	1
	γ	$[1;1e^{4}]$	$1e^{4}$	2666	1
	$cw_{\min}$	[1;100]	62	57	1
	α	[0;100]	0	2	0
	λ	[0;100]	18	46	100

alternative forms. Another main issue is the bias-variance trade-off. Models depend on the observations they are trained on, and their capacity of generalisation is what determines which algorithm is good. If a model strongly captures the behaviour of a training set, it will probably perform poorly to predict unknown observations. This is a situation of low-bias/high-variance. Finding the right optimum is a complicated task. Also, Data-driven approaches are often stochastic by nature, and the score associated with each model should be interpreted as one value out of an unknown distribution of solutions. The models are static, as opposed to time-series. It is possible that taking away the time factor from the equation induces a higher bias. Using multivariate time-series forecasting methods is promising and some methods are already selected for future work, such as SARIMAX, Neural

Network, SVR and KNN with time. Regarding the optimisation, the optimisation ran on a restricted number of runs. Powell method did run until convergence was met, but Bayesian Optimisation and Genetic algorithm had limited iterations. It would be possible to change the termination criterion so that the optimization runs until there are no significant improvements on the loss function. This method could give better results, but there is a trade-off with the execution time. Figure 3 shows the plant observations compared to the entire set of observations predicted by a Random Forest. The observation from the power plant in blue seems a lot like random noise, and the model captured this noise. This is the most extreme example from the predictions, but it depicts probable overfitting. This could be confirmed/infirmed by predicting a totally new data set from the powerplant. The main solution to tackle the noise issue is to use time-series. Indeed the time-series focus on trend, seasonality and noise while static methods do not interpret noise. If static regressors are used, applying filters on the responses, especially low pass filters, might lead to a better generalization. Filtering data is a 'risky' process, because some important information might be lost in the process.

4. Conclusions and further work

A first-principle model with high prediction capacity and several data-driven models were compared to operational data from a coal-fired boiler. At first, the data-driven models were fit to the observation data with few hyper-parameter adaptations. Then regressors were fit to the data with optimised parameters using the same algorithms. The optimizations of these parameters have been performed with three different methods: Powell Conjugate Direction, Bayesian Optimisation and Genetic Algorithm. In general, the simpler models benefited more from the optimisation than ensemble methods. Data-Driven models outperformed the first principle model by at least 15%. Random Forest and XGBoost yielded the best results with a reduction of the error of 56% and 57% for nonoptimised models. After optimization, Support Vector Machine outperformed both with a 58% error reduction. The high accuracy of this model needs to be confirmed with other unknown observations, especially because the models can capture the random noise for outputs. The 'static' approach should then be compared to the dynamic one with the use of time-series analysis. Finally, artificial neural network structures should be added to the model list for both static and dynamic approaches.

Acknowledgement

We acknowledge financial support from the Polish-Norwegian Research Program for funding the InnCapPlant project (Grant NOR/POLNORCCS/0015/2019-00), and the NTNU Department of Energy and Process Engineering. We express our gratitude to Monika Rerak at the Cracow University of Technology for providing the data, which is essential to this work. A special thanks to Pr. Adil Rasheed for his precious advice for future work.

References

[1] J. Smrekar, M. Assadi, *et al.*, "Development of artificial neural network model for a coal-fired boiler using real plant data," *Energy*, vol. 34, no. 2, pp. 144–152, 2009.

[2] S. Chandrasekharan, R. C. Panda, and B. N. Swaminathan, "Statistical modeling of an integrated boiler for coal fired thermal power plant," *Heliyon*, vol. 3, no. 6, p. e00322, 2017.

[3] H. Zhu, G. Zhao, *et al.*, "Nonlinear predictive control for a boiler-turbine unit based on a local model network and immune genetic algorithm," *Sustainability (Switzerland)*, vol. 11, no. 18, 2019.

[4] A. Navarkar, V. R. Hasti, *et al.*, "A data-driven model for thermodynamic properties of a steam generator under cycling operation," *Energy*, vol. 211, p. 118973, 2020.

[5] N. Abdul Manaf and A. Abbas, "Dynamic modelling and simulation of clean coal power generation," *Journal of Physics: Conference Series*, vol. 1447, no. 1, 2020.

[6] E. Oko, M. Wang, and J. Zhang, "Neural network approach for predicting drum pressure and level in coal-fired subcritical power plant," *Fuel*, vol. 151, pp. 139–145, 2015.

[7] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras and TensorfFlow*. O'Reilly, 2019.

[8] J. Tan, J. Yang, *et al.*, "A critical look at the current train/test split in machine learning," 2021.

[9] T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, aug 2016.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] H. Zou and T. Hastie, "Regression shrinkage and selection via the elastic net, with applications to microarrays," 01 2004.

[12] V. Vapnik, *The Nature of Statistical Learning Theory*. Q325.7.V37, Springer-Verlag, 2nd ed., 1999.

[13] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, Aug. 2004.

[14] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73 – 101, 1964.

[15] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems 20 (NIPS 2007)* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 161–168, NIPS Foundation (http://books.nips.cc), 2008.

[16] A. Moore, "The anchors hierarchy: Using the triangle inequality to survive high dimensional data," *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, 01 2013.

[17] F. Pedregosa, D. Cournapeau, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] L. Breiman, "Random Forests," *Machine learning*, 2001.

[19] H. Drucker, "Improving Regressors using Boosting Techniques," p. 9, 1997.

[20] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, vol. 7, pp. 155–162, 01 1964.

[21] E. Jones, T. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[22] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*. Adaptive computation and machine learning, Cambridge, Mass: MIT Press, 2006. OCLC: ocm61285753.

[23] J. Snoek, Larochelle, *et al.*, "Practical Bayesian Optimization of Machine Learning Algorithms," Aug. 2012. Number: arXiv:1206.2944 arXiv:1206.2944 [cs, stat].

[24] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014–.

[25] A. F. Gad, "Pygad: An intuitive genetic algorithm python library," 2021.

[26] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, pp. 8091–8126, Feb. 2021.