Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datateknik
2021 | LIU-IDA/LITH-EX-A--2021/001--SE

Metamodeling-Based Design Optimization of Gas Turbine Blade Cooling Channels

Namita Sharma

Supervisor : Joel Oskarsson
Examiner : Anders Grimvall

External supervisor : Edgar Bahilo Rodriguez, Ahmed Mohamed

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

The aim of this paper is to investigate the use of Machine Learning to accelerate the process of gas turbine design optimization. Traditionally, the process simulations involved in optimization are performed using commercial Computational Fluid Dynamics (CFD) software tools to find the best combinations of design parameters under specific conditions and operational constraints. However, such simulations can become computationally too complex and slow for routine analysis of component designs. To address this challenge, a meta-modeling approach is developed in this study that uses "computer simulation data" collected from a small number of simulation runs. The trained metamodel is then used to predict the metal temperatures of the turbine blades to approximate the complex thermodynamic simulations. The focus of this research is to accurately model the cooling process of a turbine by analyzing the effect of changing the diameters of the cooling holes drilled into the gas turbine blades on the output metal temperatures. It is concluded that neural networks can estimate the metal temperatures several orders of magnitude faster than a parallelized CFD software at a cost of mean error of $5.44°C$. This approach can provide immediate feedback for real-time design iterations at the early stages of design.

Acknowledgments

Throughout the writing of this dissertation, I have received a great deal of support and assistance.

I would like to express my sincere gratitude to everyone at Siemens Energy who made it possible for me to work on a cutting-edge research topic. My manager, **Ronny Norberg** for giving me the opportunity and making me feel like a part of the team form day one, my supervisors **Edgar Bahilo Rodríguez** and **Mohamed Ahmed** for providing me continued guidance despite their busy schedules, **Olle Sriknar** for his valuable expertise in mechanical engineering aspects of the project. Special thanks to Edgar for providing me with the tools, opportunities, and support to explore topics far beyond the scope of this work. I was lucky to be a part of such a motivating and inspiring team.

I would like to acknowledge my colleagues from my internship at Siemens Energy for their wonderful collaboration. Many thanks to **Gabriel Lindman** and **Margo Van Schaik** for generating data and that was vital for the work done in this thesis, **Marina Gómez Estévez** for open collaboration and useful discussions related to the work. A special mention to **Noufal Muhammad** for all the fascinating technical discussions and lending a helping hand in tackling challenging implementation issues.

I would like to thank my examiner **Anders Grimwall** and opponent **Erik Rosendal** for their valuable feedback on my thesis, which improved the overall quality of my report. I cannot begin to express my thanks to my university supervisor **Joel Oskarsson** for his insightful and invaluable guidance throughout my work. I am deeply grateful for his patience and expert inputs which always gave me a better sense of the direction I was headed in and helped me explain my findings better.

Last but not the least, my sister, brother, mother, and father who take a keen interest in the work that I do and give me unwavering support and encouragement.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Siemens Industrial Turbomachinery (SIT AB), Finspång is one of the leading manufacturers of medium industrial gas turbines for power generation in the world. Gas turbines are designed to operate for long periods of time and in tough environments, of which the turbine blades are often the limiting component [Figure 1.1]. According to [1], modern gas turbine vanes and blades are exposed to gas at temperatures well above the melting point of the component material. Therefore, the blades and vanes have to be cooled in order to lower the temperature and avoid creating too large temperature gradients. Large temperature gradients cause excessive thermal stresses and significantly reduce the component life, leading to premature failures [1]. The use of effective turbine blade cooling technologies is imperative since gas turbine performance and efficiency is highly dependent on turbine rotor inlet temperatures. Inadequate turbine blade cooling can jeopardize safe turbine operation, making it critical to optimize the life cycle of turbine components.

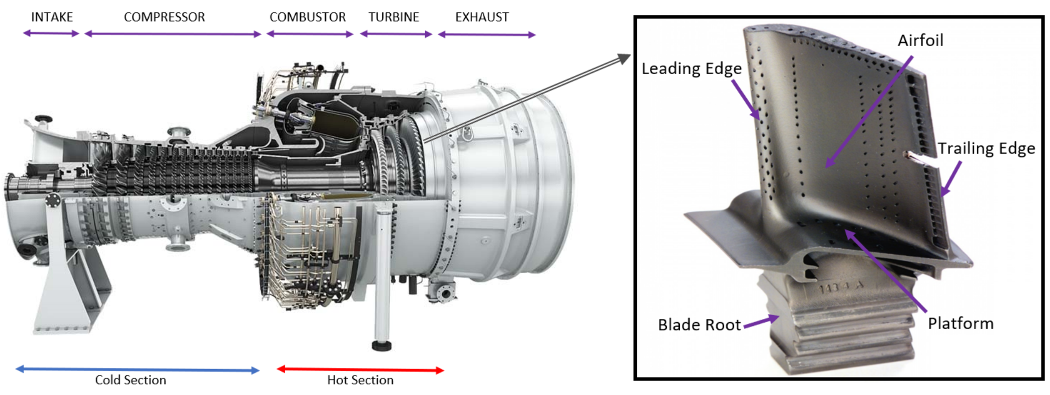


Figure 1.1: Siemens Gas Turbine (SGT) 800. Image downloaded from [2], [3] © SIT AB

Significant advances in science and technology have revolutionized the design of gas turbines. Turbine blades are usually made of super alloys that can operate at a high fraction of their melting point, have excellent mechanical strength and resistance to oxidation. Due to the high cost associated with conducting physical test experiments, many industries rely heavily on virtual prototyping tools to perform process simulations. For example, the cooling process of a turbine blade can be analyzed over a complete range of operating conditions, and different blade designs can be digitally studied to optimize the life cycle. SIT AB uses three-dimensional CFD simulations to accurately predict the complex flow-field and metal temperatures of the turbine blade to make engineering design decisions [4]. However, a general problem with complex computer simulations is that they are time consuming and computationally intensive. In addition, design optimization is an iterative process that often involves a loop of design-simulation-redesign. To overcome this challenge, the metamodeling technique has been recognized in the literature as a useful approach to reduce the computational effort involved [5], [6], [7]. It is also known as surrogate modeling, fast-approximation modeling, or data-driven modeling. It is a concept that introduces an explicit and simplified model based on the empirical relationship between the inputs and the corresponding outputs in a data set generated from the detailed simulation model.

The current design process at SIT AB is shown in Figure 1.2. It is an iterative loop of five different stages. First, a parametric Computer-Aided Design (CAD) model is created for the component geometry of interest, which in this study is the blade of an SGT 800 gas turbine [Figure 1.1]. The cooling inside the blade is simulated on the discretized mesh model using the CFD software called C3D, which performs complex thermodynamic calculations iteratively until convergence. In the next stage, the metal temperature from the C3D output are used to predict the mechanical stresses and strains using Finite Element Method (FEM). The fourth and final stage uses the thermal and mechanical integrity outputs to calculate the expected life of the component in terms of load cycles (number of on-off cycles before component failure) and hours (time for which the model can run at operating threshold speeds).

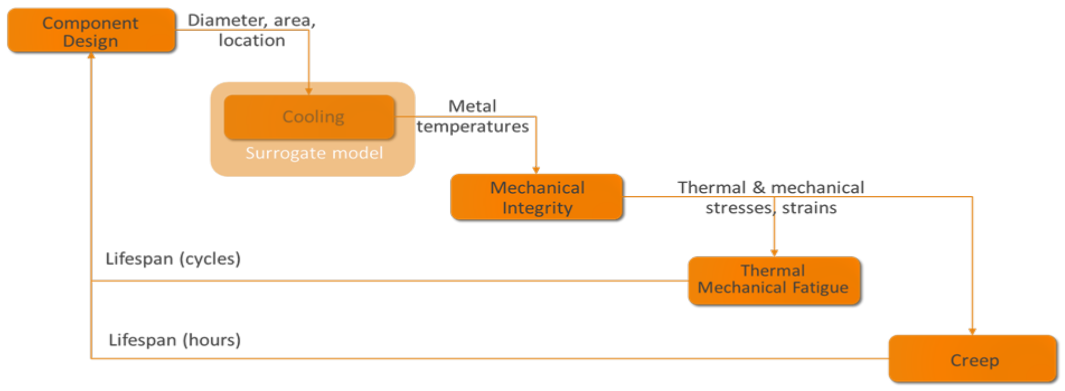


Figure 1.2: Design Cycle at Siemens Energy

If a particular simulation results in poor stress distributions or expected life, the model must be redesigned and the entire cycle restarted. To speed up the design process, each stage must be replaced by a fast approximation model. The focus of this study is to replace the cooling model in the second stage of the design cycle using a surrogate model built on empirical simulation data. A linear kriging surrogate [8] is inbuilt in the optimization software presently used which serves as an existing baseline for this study. Since the empirical model performs a fixed number of computations, it runs much faster than the original simulation model. The computational cost is mainly determined by the number of runs of the original model to obtain the simulation data, which can be decided depending on the availability of time and resources. Surrogate modeling is an efficient way to explore input parameter ranges, analyze the sensitivity of design variables and response variables [9], warm-start high-accuracy CFD simulations to speed up design cycles [10].

The type of metamodel depends on the intended use and the underlying physics that the model aims to capture. The complex simulation model used to study turbine blade cooling implies an implicit input/output function that maps the simulation parameters (e.g., the diameter of the cooling holes, the mass flow in the cooling channel of the blade, geometry of the blade), onto the response metal temperatures. The metamodel learns this underlying function and is thus able to predict the output temperature for any value of the inputs.

Based on previous work, it has not been possible to establish one surrogate which always yields the best result, instead it depends strongly on the problem it is designed to solve. Studies have shown that the best performing surrogate models for high-dimensional data sets are deep learning models and ensemble methods. In [11], regression forest is used to approximate the behaviour of fluid particles with a feature vector designed based on the differential equations governing the flow. The promising results in the study suggest the applicability of machine learning methods such as regression forests to physics-based fluid simulations. The research in [7] also uses an ensemble approach taking advantage of multiple neural networks to model transient gas-solid flow simulations in a rectangular 3D fluidized bed. To the best of the author's knowledge, metamodeling has not been investigated with respect to its application to CFD simulation of a gas turbine blade cooling, which is the primary objective of this work. Gradient Boosted Decision Trees (GBDT) and Fully Connected Feedforward Deep Neural Networks (FDN) are chosen as the surrogate models for this study and are evaluated against linear baselines. The report also addresses how to handle high dimensional response variables in metamodeling by exploiting their correlation structure. Principal Component Analysis (PCA) is used to reduce the high-dimensional response to low-dimensional score vector. The motivation behind using PCA is that the temperature response variables spatially close to each other are highly correlated and thus can be efficiently represented by a low-dimensional score vector [12]. According to the study in [12], PCA was successfully used for a small-scale CFD simulation with up to 182,250 output variables.

1.2 Aim

The main objective of this paper is to investigate Machine Learning methods to build a metamodel that can help speed up the design optimization process of a gas turbine, in particular the cooling process of the turbine blades. The data obtained from CFD simulations is used to train and evaluate the model. The metamodel will solve the problem of simulating the effect on turbine blade cooling when the diameter of the cooling holes is changed. The boundary conditions for the cooling process and the parametric CAD mesh model of the component are not changed in the computer simulation experiments. In fluid mechanics, boundary conditions are constraints necessary for the solution of a boundary value problem such as a system of differential equations and are important to model the physical phenomenon [13].

1.3 Research Aims

1. Explore Machine Learning approaches to predict the output metal temperatures using the input diameters of cooling holes, geometry of the component, and the boundary conditions for the cooling model such as mass flow. Create and validate metamodels which can be utilized by a mechanical engineer at Siemens in place of the original C3D model.

2. Investigate dimension-reduction techniques to handle the high dimensional input and response data.

3. Evaluate methods for multioutput predictions and draw comparisons with baseline models chosen in this study and the existing linear kriging baseline having a Root Mean Squared Error (RMSE) value of 98.2.

4. Compare and quantify the speed up gained from metamodel-based predictions over the simulations from traditional CFD calculations for the cooling process of gas turbine blade.

2 Theory

2.1 Sampling

2.1.1 Design of Experiments

The metamodel tries to approximate a black-box function that maps input diameters to output metal temperatures as represented by the simulation model. All information about this unknown function is in the generated design samples. However, it is not possible to know in advance what the optimal distribution of designs is and how many designs are needed to achieve a good modeling solution [14]. To determine the input settings for the design samples, generally, a process called the design of experiments (DOE) is used. It is a systematic and rigorous statistical sampling approach to control where the design points are placed in the design space to effectively evaluate the response function. In the context of computer experiments, the main objective of DOE is to select the values of simulation inputs such that the simulation data obtained are representative of the input space being explored and informative for predicting the process outputs.

The classical DOE methods [15], e.g., factorial designs, typically assign two or three predetermined levels for each process input, and then conduct experiments at all combinations of the different input levels. Obviously, the more variables to investigate, the more combinations are required. Due to the limited levels of the factors being studied, this strategy may not have an optimal coverage of the design space and thus may result in a less reliable empirical model [16]. The drawbacks of classical DOEs have motivated the concept of "space-filling" designs, where the design points are uniformly distributed in the domain of each input [17].

2.1.2 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) is one of the most common methods used today for space-filling designs [18], which tries to minimize the number of data points needed to achieve high accuracy for surrogate models. In the context of statistical sampling, a square grid containing sample positions is a latin square if there is only one sample in each row and each column [Figure 2.1b]. Latin square corresponds to a 2D LHS where stratified samples are drawn from two features. A latin hypercube is the generalization of this concept to an arbitrary number of dimensions, where each sample is the only one in each axis-aligned hyperplane that contains it [19]. This is the latin hypercube requirement. In random sampling, new sample points are generated without taking into account the previously generated sample points. Latin hypercube sampling first specifies the number of samples and also stores which row and column each sample was taken from to satisfy the Latin hypercube requirement.

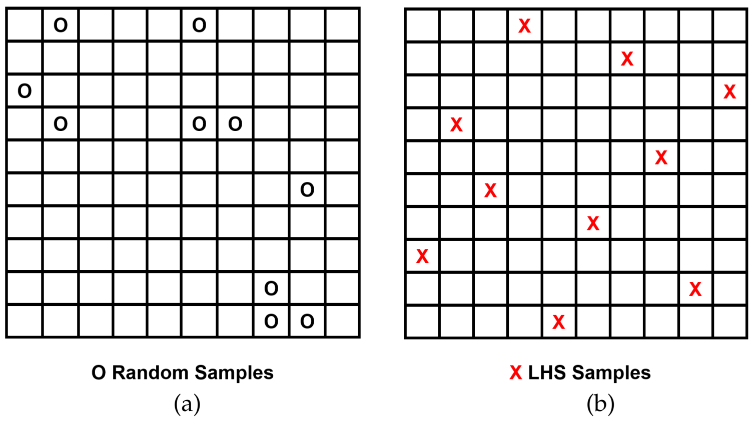


Figure 2.1: (a) 2D random sampling (b) 2D LHS

The key principle behind this method is to stratify the input probability distribution to avoid sampling all points from the same local region. In one-dimensional LHS, the cumulative density function (if no prior exists, a uniform distribution is assumed) is divided into n equal intervals as shown in Figure 2.2. Then, a sample is randomly selected from each interval. In two-dimensional LHS, the one-dimensional method is applied to the marginal distributions to obtain one-dimensional samples for each variable separately. Once we have the two lists of samples, they are randomly combined, into two-dimensional pairs to get n two-dimensional samples [18], [20].

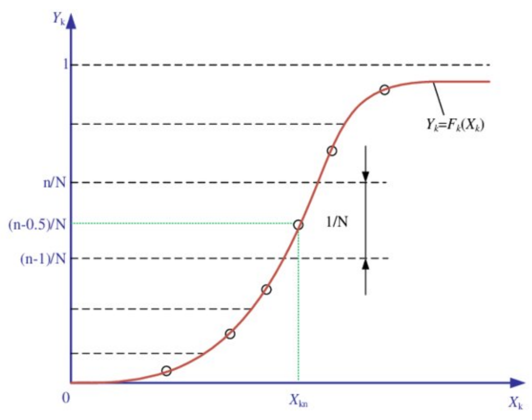


Figure 2.2: 1-Dimensional Stratified Sampling

Extending the same principle to k dimensions, when LHS is performed on k continuous design variables to achieve n samples, the chosen range of values for each variable is first divided into n intervals. This creates a total of n x k distinct intervals. Once we have k lists of n intervals each, we randomly combine them into k-dimensional groups. Then, within each interval of each pair, we randomly select a value according to a defined probability distribution. In the absence of a prior, we assume a uniform distribution. Thus, we obtain n samples (groups) of k-dimensions. In design optimization, design variables are referred to as factors and the intervals in which their input domains are stratified are called the factor levels. LHS assumes independence between the factors, but is still relevant for the data in this study because the diameters of the cooling holes can be varied independently of each other.

A basic LHS does not guarantee qualitative space-filling properties [Figure 2.3(a)]. There are several variations of the method to achieve more optimal LHS and better space-filling. The Optimal LHS technique is a modified Latin Hypercube where the combination of factor levels for each factor is optimized, rather than randomly combined [21]. The goal of this

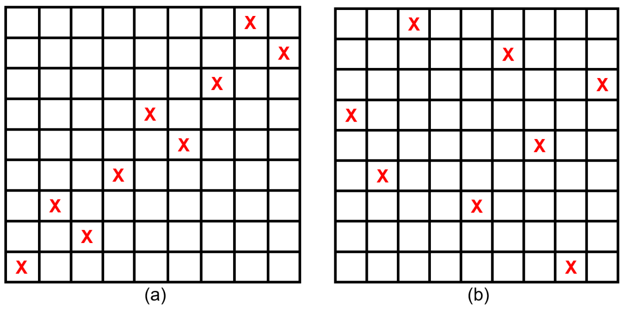


Figure 2.3: (a) 2D LHS with bad space-filling properties (b) 2D LHS with good space-filling properties

optimization process is to maximize distances between data points in order to achieve more space-filling data.

2.2 Principal Component Analysis

In theory, when modelling the cooling process of the turbine blade, the temperature values at different spatial locations can be directly used as output variables. However, the dimension of the temperature outputs from CFD simulation can be extremely high. As the granularity of the mesh increases, the output dimension drastically increases since the temperature is predicted at every node in the mesh. For example, the 3D simulation in this work gives approximately 284,553 metal temperature values which can make the meta model complex. This problem can be addressed by using dimensionality reduction techniques, such as principal component analysis (PCA) [22], [23].

PCA is calculated on the covariance matrix or the correlation matrix of the data. PCA finds a linear transformation of data such that the directions where data varies the most are preserved. This transformation is orthogonal and converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. PCA uses singular value decomposition to calculate the eigenvectors and their eigenvalues for the covariance matrix [24]. The eigenvectors form the basis for transformed data, these are called principal components or principal axes and form the new dimensions or a new coordinate system. The eigenvalues represent the variances in the data captured by the principal component directions. The original data projected onto the new principal component axes are called the scores.

Let $\mathbf{y_i}(i = 1, \ldots, N)$ be a vector of dimension P containing the metal temperatures from the i-th simulation run. Then, the data from all the N runs are concatenated into a matrix of order $N \times P : \mathbf{Y} = [y_\mathbf{1}, \ldots, y_\mathbf{N}]'$. In PCA, the temperature matrix is projected into scores and loadings as

$$\mathbf{Y} = \mathbf{TP}' \tag{2.1}$$

where:
$\mathbf{T}(N \times Q)$ is the scores;
$\mathbf{P}(P \times Q)$ the loadings or principal components;

If the original variables are highly correlated, the dimension of T can be dramatically smaller than that of $\mathbf{Y}$ (i.e. $Q << P$), while still containing the essential information of $\mathbf{Y}$. Hence, a large number of temperature values are represented with very few score variables, which are then used as the outputs for metamodeling. The model prediction is thus also in the form of PCA scores, which can be transformed, according to 2.2, to get the prediction for the original temperatures. For brevity, a single score vector from the matrix **T** is denoted $\mathbf{t_{*,j}} (j = 1, \ldots, Q)$, which is of the order $(N \times 1)$ (N is the number of simulation runs).

$$\hat{\mathbf{Y}}_* = \left[\hat{\mathbf{t}}_{*,1}, \ldots, \hat{\mathbf{t}}_{*,Q}\right] \mathbf{P}' \tag{2.2}$$

where $\hat{\mathbf{t}}_{*,j} (j = 1, \ldots, Q)$ is the prediction from metamodel for the j-th score vector and $\hat{\mathbf{Y}}_*$ is the reconstructed original temperature prediction. After dimensionality reduction, regression models are developed to relate the input variables **x**, to the PCA scores as output. Since the scores are of dimension Q, this is a multioutput regression problem.

2.3 MultiOutput Regression

Regression refers to a predictive modelling that typically involves predicting a single numerical value. Multioutput regression are regression problems where multiple numerical values are predicted given input variables. Often the outputs are not independent of each other and may require a model that predicts all outputs together, or each output dependent upon the other outputs.

The task of multioutput regression can be divided into multiple sub-tasks. For example, to predict three values y_1, y_2 and y_3 given an input X, three single-output regression models can be built where one model is fit to each output. The simple regression models can be implemented using two approaches [25], [26]. First, by developing a separate model for each target value to be predicted directly. These models are independent of each other.

Model1: Given X, predict y_1
Model2: Given X, predict y_2
Model3: Given X, predict y_3

Second, by creating a sequence of dependent models where the first model uses the input to predict one output. The second model uses the input and the output from the first model to predict the second output and so on [27].

Model1: Given X, predict y_1
Model2: Given $X, y1$, predict y_2
Model3: Given $X, y1, y2$, predict y_3

The second approach assumes an implicit order in the y's which may be justified in some problems. But also since the models are chained together and every model depends on the output from its previous model, the computational time complexity can be quite high. The first approach is preferred in the experiment performed in this study where the y's are independently modelled in parallel because of reasons explained in Section 3.2.

2.4 Evaluation Metrics

2.4.1 Mean Squared Error

If a vector of n predictions is generated from a sample of n data points, and $\boldsymbol{y}$ is the vector of observed values, with $\hat{\boldsymbol{y}}$ being the predicted values (e.g. as from a least-squares fit), then the MSE of the predictor is computed as

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.3}$$

The only issue with MSE is that we cannot directly correlate it with the data as the order of the error metric is more than that of the data. Hence, we take the root of the MSE which is called the Root Mean Squared Error. Here, we are not changing the error metric but only reducing its order to match that of the data and make it more interpretable.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.4}$$

2.4.2 Mean Absolute Error

Mean absolute error (MAE) is another common loss function used for regression. The loss is the average of the absolute differences between true and predicted values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y - \hat{y}_i| \tag{2.5}$$

MAE is a popular metric because, like RMSE, the units of the error score match the units of the target value that is being predicted. But unlike the RMSE, the changes in MAE are linear and therefore intuitive. That is, MSE and RMSE punish larger errors more than smaller errors, inflating or magnifying the mean error score due to squaring of the error. The MAE does not give unequal weights to different types of errors and instead increases linearly with increases in error.

2.4.3 R-Squared

R-squared (R^2), also known as the coefficient of determination, is the proportion of variance of a dependent variable that's explained by an independent variable in a regression model. Whereas correlation represents the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of the features explains the variance of the target variable. For a data set $\boldsymbol{y} = [\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}]^T$ with fitted (or modeled, or predicted) values $\hat{\boldsymbol{y}} = [\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_n]^T$, the variability of the data set can be measured with two sums of squares formulas:

The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2 = n\sigma^2 \tag{2.6}$$

The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = n\text{MSE} \tag{2.7}$$

where $\bar{y}$ and σ^2 are the mean and the variance of the observed data respectively, given by:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{2.8}$$

The most general definition of the coefficient of determination is

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} = 1 - \frac{\text{MSE}}{\sigma^2} \tag{2.9}$$

Best possible score is 1 and it can be negative because the model can be arbitrarily worse, i.e. $R^2 \in [1, -\inf)$. A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0. In a multi-target regression setting, R2 is first computed as a vector on the individual targets and then averaged out over all the m targets.

$$\frac{1}{n}\sum_{j=1}^{m} R_j^2 = 1 - \frac{1}{n}\sum_{j=1}^{m} \frac{\mathrm{MSE}_j}{\sigma_j^2} \tag{2.10}$$

where $\sigma_j^2(j = 1, \ldots, m)$ are the variances of the different target variables.

The mathematical framework for R-squared does not hold good when the regression model is non linear [28]. In non-linear models such as neural networks, the features get combined in different ways, where the variance of data SS_{tot} is not a sum of variance of the errors SSres and the variance of the model $(= \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2)$. Still, R^2 as a function of the MSE and data variance σ^2 may be considered useful to interpret how well the model performs alongside other evaluation metrics like MSE and MAE.

2.5 Ridge Regression

Ridge estimation is carried out on the linear regression model

$$y = X\beta + \varepsilon \tag{2.11}$$

where:
y is the $N \times 1$ vector of observations of the dependent variable;
X is the $N \times K$ matrix of regressors (there are K regressors);
β is the $K \times 1$ vector of regression coefficients;
ε is the $N \times 1$ vector of errors.

The OLS estimator $\widehat{\beta}$ solves the minimization problem

$$\widehat{\beta} = \arg\min_b \sum_{i=1}^{N} (y_i - x_i b)^2 \tag{2.12}$$

where x_i is the i -th row of X and b and $\hat{\beta}$ are $K \times 1$ column vectors.

The ridge estimator $\hat{\beta}_\lambda$ solves the slightly modified minimization problem [29]

$$\widehat{\beta}_\lambda = \arg\min_b \sum_{i=1}^{N} (y_i - x_i b)^2 + \lambda \sum_{k=1}^{K} b_k^2 \tag{2.13}$$

where λ is a positive constant.

In ridge estimation a penalty is added to the least squares criterion. We minimize the sum of squared residuals plus the squared norm of the vector of coefficients. In other words, the linear regression model is penalized for having too many variables in the model. The ridge problem penalizes large regression coefficients, and the larger the parameter λ is, the larger the penalty. The consequence of imposing this penalty, is to reduce (i.e. shrink) the coefficient values towards zero. This allows the less contributive variables to have a coefficient close to zero or equal zero.

2.6 Neural Network

Neural networks are networks of simple processing elements operating on the data and communicating with one another. They are widespread in machine learning because of their ability to perform universal approximations [30]. The element in the network are called neurons. The illustration of a simple neural network is shown in Figure 2.4.

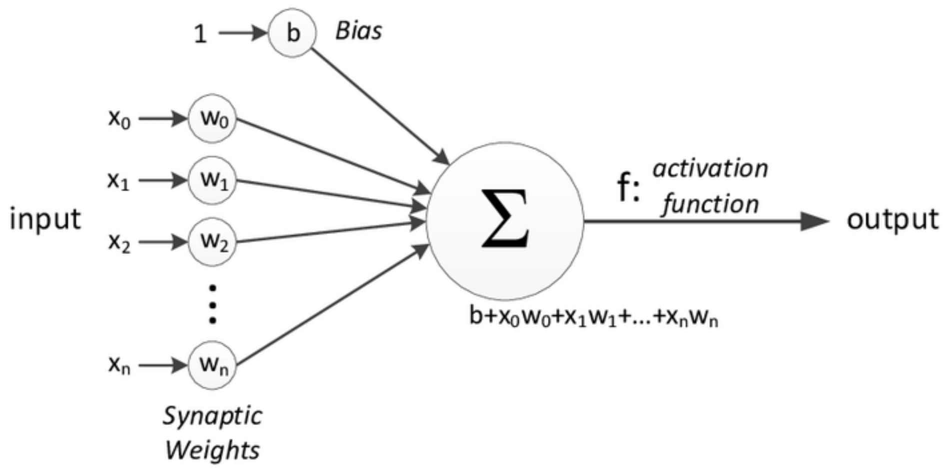


Figure 2.4: Mathematical model of a neural network

Activation functions are mainly used to add non-linearity to the model. Neural Network without non-linear activation simply acts as a Linear Regression. There are different activation functions suitable for different tasks. In this work, ReLU activation layer was used.

$$\mathrm{relu}(x) = \max(0, x)$$

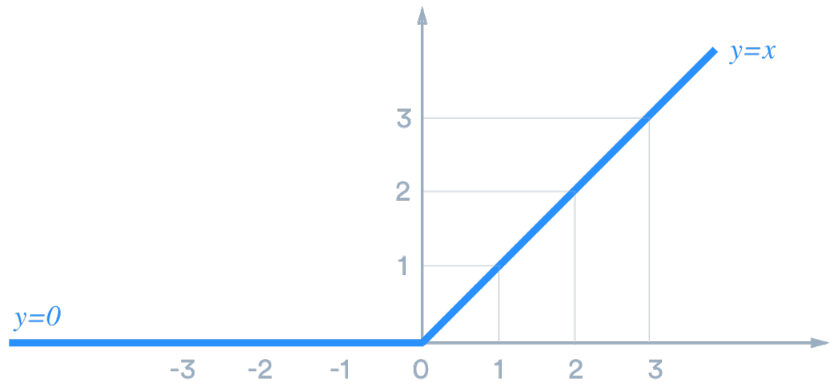


Figure 2.5: ReLU activation function

Its gradient is defined as:

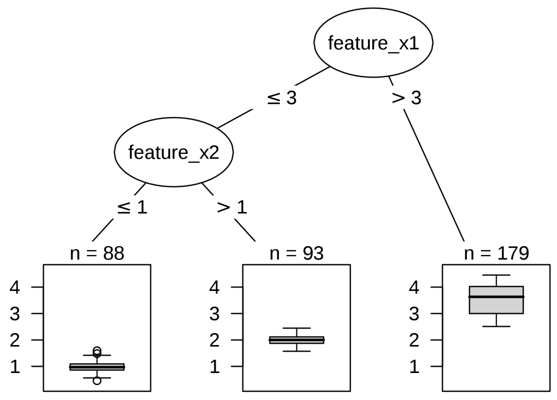
$$\frac{d}{dx}\,\mathrm{relu}(x) = \begin{cases} 0 & \text{if } x \leqslant 0 \\ 1 & \text{if } x > 0 \end{cases}$$

As part of network training, the gradient of the loss function to be minimized is repeatedly evaluated with the current weights of the network. The network weights are then updated in the opposite direction of this gradient by an amount determined by the learning rate. The most commonly used loss functions for regression are the mean squared loss and mean absolute loss. Different optimization algorithms are used to direct the search of optimal weights efficiently to minimize the loss.

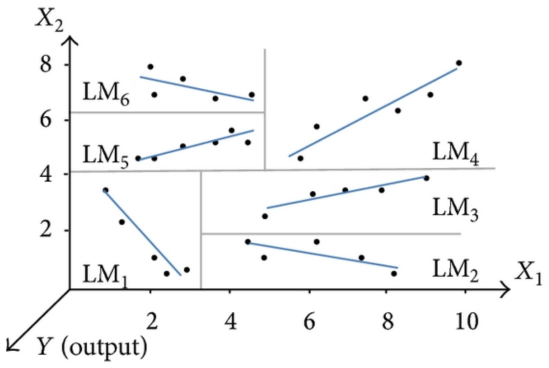
2.7 Decision Trees for Regression

Decision trees are predictive models that use a set of binary rules to calculate a target value. They split the data space multiple times according to certain cutoff values in the features.

Given any data point, a decision tree arrives at an estimate by passing it through a series of True/False questions on the input variables until it reaches a leaf node. The final prediction is the average of the values of the target variable in that leaf node [Figure 2.6]. The model learns in the training phase, what questions to ask and in which order, i.e., the features needed for the best splits and their thresholds in order to make the most accurate estimates possible. When the relationship between features and outcome is nonlinear or where features interact with each other, decision trees outperforms linear regression models. It is considered to be a non-parametric method. This means that decision tree makes no assumptions about the data distribution and the model structure [31]. While working with continuous numerical variables, it converts them into buckets and hence loses some information and predictive power when assigning different data points to the same bucket. The underlying model is simply the average of the data points in the terminal nodes of the tree [32]. Another possible option is to use median or even run a linear regression model instead of using the average [Figure 2.6a, 2.6b].



(a)



(b)

Figure 2.6: Decision Tree Regressor with (a) Median estimates (b) Least Squares (Linear Regression) estimates. (Images downloaded from [33], [34] under creative Commons Attribution License.)

A tree's accuracy is mainly affected by the decision of its splits. Decision tree for regression normally uses mean squared error (MSE) to decide to split a node in two or more sub-nodes. It picks features and their values for the splits such that the sub-groups created are as different from each other as possible. It calculates the MSE separately for each subset. The weighted average (MSE*n_samples) of the MSE of all the sub-groups created during a split represents how good that split is. The tree then chooses the best split which results in the smallest average MSE value, also known as the minimum impurity. A brute-force approach to find the best split is to try every variable and to try every possible value of that variable and see which variable and which value gives a split with the best score. While this kind of exhaustive search is possible in case of decision tree regressors with categorical features having finite number of factors, it is impossible in case of decision tree regressors with continuous input features. More commonly, some form of optimization algorithms are used to build the decision trees by controlling hyperparameters such as minimum impurity decrease or maximum depth [35]. The optimization process halts if the stopping conditions are reached in the tree. No more splits occur in the tree if the decrease in MSE induced by the split is less than the minimum impurity value specified, or when the tree reaches its specified maximum depth [Figure 2.7].

Decision trees and tree-based models in general are unable to extrapolate to any kind of data they haven't seen before as they are just averaging data points they have already seen.

They would fail to make accurate predictions if the new data provided are unrelated to the data that they have been trained on. In summary, decision tree is a type of supervised machine learning algorithm (having a pre-defined target variable) that can be used in regression problems if and only if the target variable is expected to be inside the range of values seen in the training dataset [36], [37]. One major drawback of Decision Trees is that they are prone to overfitting. Small variations in the data might result in a completely different tree being generated. Hence, they are often used in an ensemble such as random forest to decrease the variance in predictions. Random forest combines the predictions from multiple different trees trained on different subsets of the same data to make more accurate predictions than any individual model.

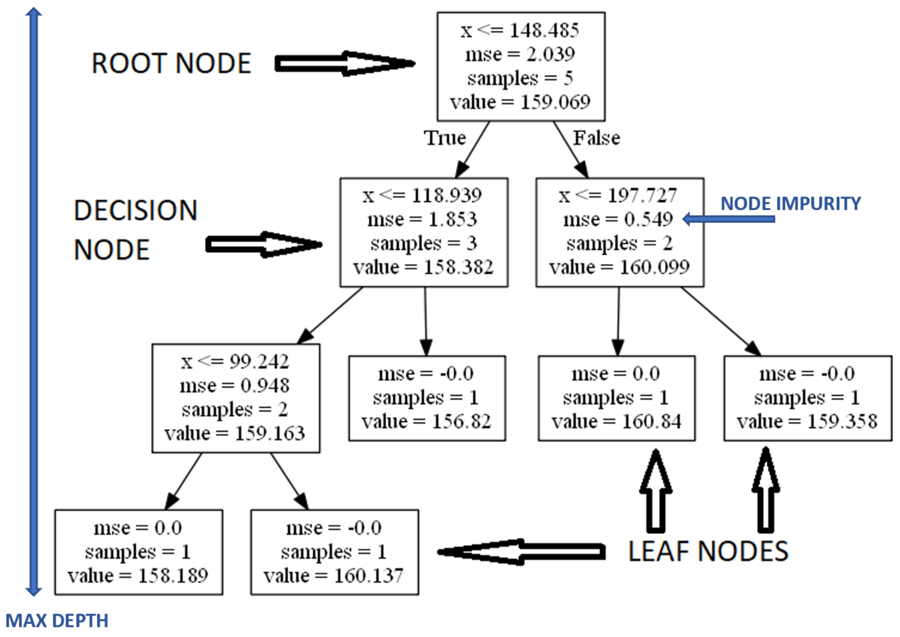


Figure 2.7: Regression Tree Hyperparameters

2.7.1 Gradient Boosted Decision Trees

The term boosting refers to a family of ensemble algorithms which converts weak learners to strong learners [31]. It is an ensemble of machine learning models, typically decision trees, aimed at primarily reducing bias, and also variance. Bagging ensemble techniques such as random forests [38] train multiple weak learners in parallel and “bag” their predictions to get a mean final prediction. Whereas, boosting is a sequential ensemble method where weak learners, are sequentially produced during the training phase. In each iteration, the performance of the present model is improved or “boosted” by assigning higher weights to the previous, incorrectly classified samples or by optimizing some loss function.

There are many ways through which boosting can be carried out. Two of the methods are discussed here. AdaBoost, short for "Adaptive Boosting" identifies the shortcomings of base learners by increasing weights of incorrectly predicted data points (and decreasing the weights of correctly predicted points) so that the next model will pay extra attention to get them right [Figure 2.8a]. It also assigns weights to each learner in a way where models with better predictions get more weight in the final prediction [39]. Gradient Boosting method does not adjust weights of the data points, instead it focuses on the residual errors made by the previous estimators and predict them. In other words, it optimizes the loss function of the previous learner by adding the prediction of a new model that reduces the overall

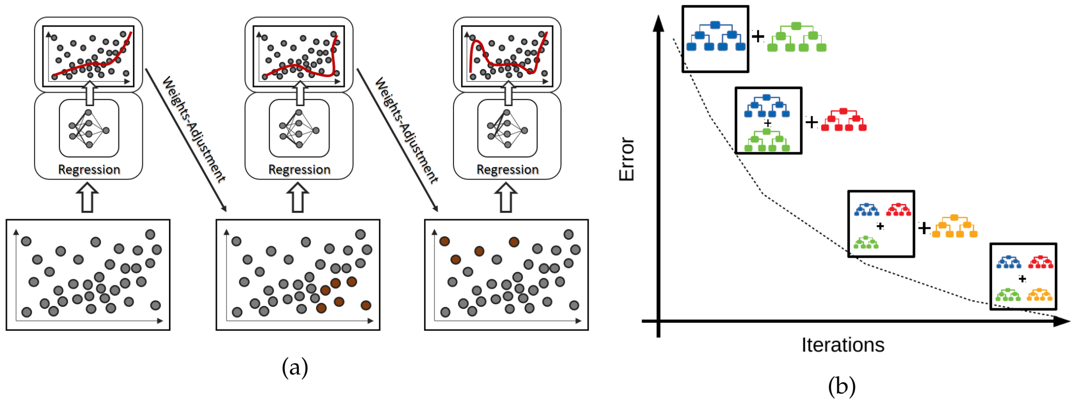


Figure 2.8: Ensemble of Decision Tree Regressors with (a) AdaBoosting (b) Gradient Boosting

loss. In this way, the present base learner is always more effective than the previous one. Gradient Boosted Regression Trees can be seen as a stage-wise additive modeling algorithm. In each iteration, it performs a steepest descent minimization for a given loss function, such as mean squared error or mean absolute error by adding a base learner to fit the residuals of the previous learner [Figure 2.8b]. The loss function is a measure indicating how good the model's coefficients are at fitting the underlying data. In case of regression tasks, the loss function is based on the error between true and predicted values of the target.

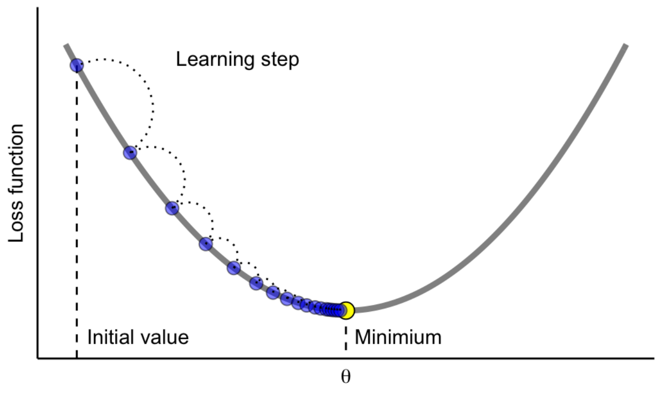


Figure 2.9: Gradient Descent. (© Taylor & Francis Group LLC - Books. Reprinted with permission.)

To improve a model F, we want to minimize the loss of the model for the entire training dataset of size n [40]

$$\mathrm{L}(Y, F(X)) = \mathrm{f}\left(F\left(X_1\right), F\left(X_2\right), \ldots, F\left(X_n\right), Y\right)$$

In a non-parametric approach, $F(X)$ evaluated at each point X is considered to be a "parameter". In the function space there are infinite number of such parameters, but in a given dataset, only a finite number are involved [41]. Minimization is performed by fitting an estimator H on $\left(X_i, \frac{\partial L}{\partial X_i}\right)$ $\forall i$ to approximate the gradient descent :

$$F\left(X_i\right) - \frac{\partial L}{\partial F\left(X_i\right)}$$

Algorithm 1: Gradient Boosting for regression

```
Fit estimator $F^1$ for $i$
for $i$ in $[1, M]$ //M weak estimators do
    Loss in $i^{th}$ iteration : $L^i = \sum_{j=1}^{n} \left(Y_j - F^i\left(X_j\right)\right)^2$
    Calculate negative gradient : $-\frac{\partial L^i}{\partial X_j} = -\frac{2}{n} * \left(Y_j - F^i\left(X_j\right)\right) \forall i$
    Fit a weak estimator $H^i$ on $\left(X, \frac{\partial L}{\partial X}\right)$
end
Prediction from the final estimator $F^M$ :
$F^M(X) = F^i(X) + \rho * H^i(X) = F^1 + \rho * \sum_{i=1}^{m} H^i(X)$ where $\rho$ is the learning rate which
  determines the size of the steps taken to reach a (local) minimum.
```

The key hyperparameters that can be tuned to optimize the performance of boosting algorithms are the number of estimators which controls the number of weak learners and the learning rate which controls the contribution of weak learners in the final combination. Apart from this, the hyperparameters of base learners discussed above [Figure 2.7] can also be optimized for maximum performance.

2.8 Repeated K-Fold Cross Validation

K-fold cross validation is a re-sampling procedure used to evaluate the performance of machine learning algorithms on a limited data set. Data is first shuffled and split into k equal and non-overlapping groups or folds. Each fold is held back as a validation set once while all the other folds are collectively used as a training set. A model is fit on the training set and evaluated on the hold-out validation set. The evaluated score is retained, and the model is discarded each time. Hence, a total of k models are fit and evaluated on the k unique hold-out test sets and the mean performance is reported [Figure 2.10]. K-fold cross-validation method is primarily used to estimate the generalization performance of machine learning models which is their ability to predict well on unseen data.

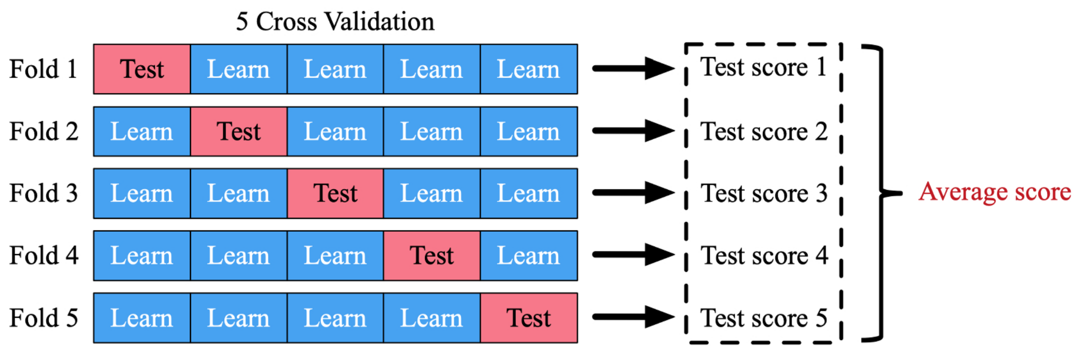


Figure 2.10: K-Fold Cross-Validation. Image downloaded from [42]. (© Springer Nature. Reprinted with permission.)

The cross validation algorithm is stochastic, different splits of the same dataset may result in very different results. Therefore, a single run of the k-fold cross validation procedure is likely to give a noisy estimate of the model performance which is either too optimistic or pessimistic. The difference in the estimated performance from one run of k-fold cv to another is dependent upon the model and on the dataset itself [43].

Repeated k-fold cross validation is one way to improve the estimated model performance. It involves repeating the k-fold cv procedure multiple times and reporting the mean result across all folds from all repeated runs. This mean is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset [44].

2.8.1 Asymmetric Loss for Regression

In regression one usually minimizes the mean squared error (MSE) for a sample:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(f\left(x_i\right) - \hat{f}\left(x_i\right)\right)^2 \tag{2.14}$$

where f is the function that maps input x to target y and $\hat{f}$ is the approximated function.

A way to define an asymmetric loss function would be to simply multiply by a weight:

$$\mathcal{L} : \left(f\left(x_i\right), \hat{f}\left(x_i\right), \alpha\right) \rightarrow \frac{1}{n}\sum_{i=1}^{n}\left(sign\left(f\left(x_i\right) - \hat{f}\left(x_i\right)\right) + \alpha\right)^2 \cdot \left(f\left(x_i\right) - \hat{f}\left(x_i\right)\right)^2 \tag{2.15}$$

with $\alpha \in (-1, 1)$ being the parameter that can be adjusted to change the degree of asymmetry, while maintaining the quadratic loss and $sign(x)$ is a function that returns -1 if x is negative and $+1$ if x is positive. It is used to trade off the penalty of underestimation against overestimation. Positive values of α penalize overestimation and negative values penalize underestimation [45].

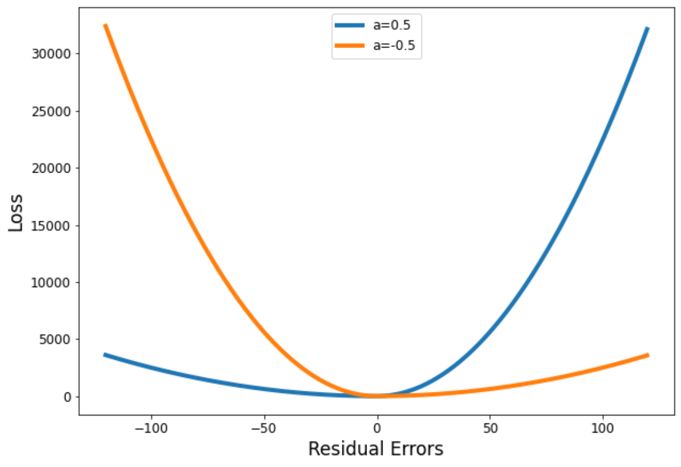


Figure 2.11: Custom Loss

2.9 Hyperparameter Optimization

There are some parameters in machine learning models that are not updated during the learning. These are called the hyperparameters and generally used to either control the model (e.g. number and depth of the decision trees, number of layers in a deep neural network, etc.) or the learning process (learning rate in gradient descent algorithm, number of epochs, etc.). These hyperparameters must be tuned to allow a machine learning model to be customized for a specific task or dataset. Hyperparameter tuning is an optimization problem where the objective function of optimization is unknown or a black-box function. Two of the simplest and most common optimization methods used are random search and grid search.

Random search defines a search space as a bounded domain of the various hyperparameter values and randomly samples points in that domain. Whereas grid search is more exhaustive, it defines a search space as a grid of hyperparameter values and samples every position in the grid. To find the hyperparameters that achieve the best performance, the machine learning model is trained with the sampled hyperparameter values on the training set and evaluated according to a predefined metric on a cross validation set. While both these algorithms can theoretically obtain the global optimal value of the optimization objective function, they suffer from the curse of dimensionality, i.e., the efficiency of the algorithms decrease rapidly as the number of hyperparameters being tuned and the range of values of hyperparameters increase [46].

2.9.1 Bayesian Optimization

Bayesian optimization is an effective method for finding exrema of objective functions that are computationally expensive to evaluate [47]. This objective function to be optimized is usually unknown and the optimization problem is akin to mining for gold in an unknown land [48]. Bayesian optimization uses a surrogate function (also called a response surface) with a prior over the space of objective functions to model the true underlying objective function. Gaussian Process (GP) is a common choice for surrogates because of its flexibility and ability to give uncertainty estimates [49]. It combines prior information about this unknown function with sample information, to obtain the surrogate posterior by using the Bayes' rule. The posterior captures the updated beliefs about the unknown objective function. Then, based on this posterior information, it gives a distribution over the "updated" function values from which we get a probabilistic estimate for where the maximum value can be found. The optimization goal is to find the maximum value at the sampling point for an unknown function f which has no analytical expression and/or is not differentiable:

$$x^{+} = \arg\max_{x \in A} f(x) \tag{2.16}$$

Bayesian optimization derives from Bayes' theorem [47], i.e., given evidence data E, the posterior probability $P(M \mid E)$ of a model M is proportional to the likelihood $P(E \mid M)$ of observing E given model M multiplied by the prior probability of $P(M)$:

$$P(M \mid E) \propto P(E \mid M) P(M) \tag{2.17}$$

A common use case of Bayesian Optimization is hyperparameter tuning to find the best performing hyperparameters on machine learning models using limited number of function evaluations. This is useful in case of models such as deep neural networks and gradient boosted regression trees that are relatively expensive to train and evaluate multiple times. Here, the unknown objective function is a function of all the hyperparameters to be optimized, and the data are the hyperparameter values where the machine learning models are evaluated. The data points are sampled from the search space sequentially according to an aquisition function $\alpha(x)$. The aquisition function $\alpha(x)$ chooses the next sample point as the one which has the highest probability of improvement over the current max $f\,(x^{+})$. Mathematically, the selection of next point can be written as follows [49],

$$x_t = \operatorname{argmax}_x \alpha(x) \tag{2.18}$$

The aquisition function is a function of the posterior and it trades off exploitation and exploration. Exploitation means sampling where the surrogate model predicts a high objective and exploration means sampling at locations where the prediction uncertainty is high. Both correspond to high acquisition function values and the goal is to maximize the acquisition function to determine the next sampling point [47]. Bayesian optimization is more suitable than grid search or random search for high dimensional search spaces but one disadvantage is that parallelization is not trivial [50].

3 Method

3.1 Data

This section will cover the source and description of the data, the sampling method used to generate the data and the generation process itself. Preliminary data exploration findings related to the data distribution and correlation analysis of input and target variables are also presented.

3.1.1 Source

This thesis work is based on the analysis of CFD simulation data generated in the design optimization process of a gas turbine blade within the R&D department at SIT AB, Finspång. The cooling simulation model in the second stage of the design cycle [figure 1.2] is the focus of this study. It takes as inputs the diameter (m), area (m^2), and the location of the cooling channels inside the blade and calculates as output, the steady-state metal temperatures (C) at each node in the CAD mesh of the component geometry [figure 3.1]. The cooling channels in the blade are also known as the flow network collectively. Some of the channels that are internal to the component are already casted when the component arrives in Finspång and their diameters cannot be changed during the optimization process. Only the cooling holes on the surface of the component can be optimized for improved lifespan of the blade. In total, 143 holes were found to be optimizable for which the diameters are varied across the experiments. The component mesh has 96,800 nodes in the airfoil and platform regions of the blade [figure 1.1] where the metal temperatures are analyzed. The cooling process occurs as a result of coupling between the flow network where the cooling air flows and the component mesh where the target metal temperatures are calculated. Therefore, each cooling hole (input diameter) is connected to multiple mesh nodes (target temperature) around it as visualized in [figure 3.1].

The location of the cooling channels and the component mesh i.e., the blade geometry is kept constant in this study. Apart from the CAD geometric input and the parametric inputs that are the diameters of the cooling channels, there are also boundary conditions specified on certain input nodes in the cooling channel of the blade known as ducts. These are the inlet Pressure Loss Coefficient (PLC) defined in the duct at the root of the blade, the inlet temperature (C), inlet pressure (Pa) and inlet mass flow (kg/s) of the cooling air. Mass flow

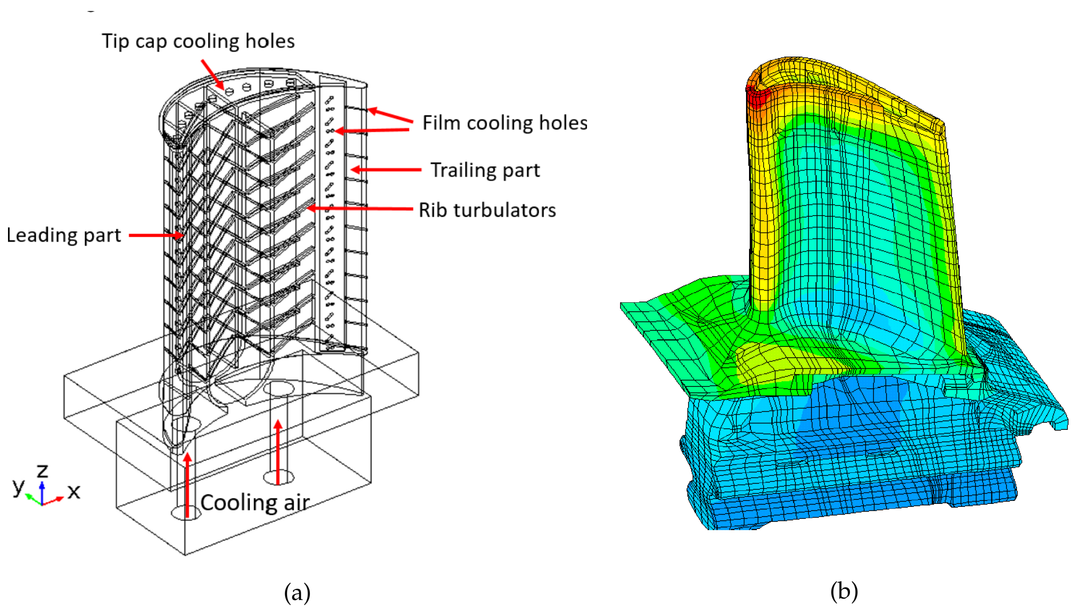


Figure 3.1: (a) Cooling Channels/Flow Network where the input diameters are defined (b) Metal Temperatures calculated on the nodes in the component mesh. Images downloaded from [51], [52]

also known as mass transfer is the movement of fluids down a pressure or temperature gradient and the mass flow of the cooling air influences the cooling capacity of the blade. The boundary conditions are also kept constant for the purpose of this study. For clarity, a set of input variables and output variables that can be used to train a surrogate cooling model is described as a data point or a design. The dataset for this study contains 1600 unique designs generated using the commercial CFD software called C3D. The input ranges for cooling channel diameters and the values for boundary conditions defining the flow of the cooling air inside the blade are decided by a team of experts at Siemens.

3.1.2 Sampling

For the input sampling process, the 143 optimizable cooling branches in the flow network of the blade are grouped into three flow groups called the Leading Edge (LE), Trailing Edge (TE) and the Centre (C) each having a further level of subgrouping of their own (start, middle and end). The diameters of the branches within a subgroup are varied together. Therefore, a total of 9 input factors are sampled from their pre-determined diameter ranges listed in Table 3.1 using the LHS technique that gives the best spread of the input factors [Section 2.1.2].

3.1.3 Generation

HEEDS is an automation program developed by Siemens, that is used to automate design generation. It allows users to create a customizable pipeline of different processes such as generating the CAD model and doing CFD simulations on it.

In this thesis, HEEDS is used to simulate the metal temperatures on a turbine blade under a range of diameter values for the optimizable cooling holes on the blade. A pipeline of three processes was set up [Figure 3.2]. The first process samples the 9 input factors from their chosen diameter ranges using the LHS technique to ensure proper coverage of the input space. A python script is used to write these values into an excel file. The second process maps the input diameter values from the excel into an .xml format which is the file format expected by

Number	Variable	Min	Baseline	Max
1	TE_start	0.00001	0.0007	0.001
2	TE_middle	0.00001	0.0007	0.001
3	TE_end	0.00001	0.0007	0.001
4	LE_start	0.00001	0.0006	0.001
5	LE_middle	0.00001	0.0006	0.001
6	LE_end	0.00001	0.0006	0.001
7	C_start	0.00001	0.0006	0.001
8	C_middle	0.00001	0.0006	0.001
9	C_end	0.00001	0.0006	0.001

Table 3.1: Input Factors

the simulation software. The final process in the data generation pipeline is running the commercial C3D toolkit to simulate the metal temperatures corresponding to the cooling channel diameters. The CAD mesh model of the blade used by the C3D software remains constant across all the generated designs. The computational time for a single simulation run is on average 15 minutes on a desktop computer running Windows 10 system with an Intel Core i7-9850H processor. HEEDS has support for extra features, such as cloud computing, different sampling techniques and surrogate models (linear and gaussian kriging [8], [53]).

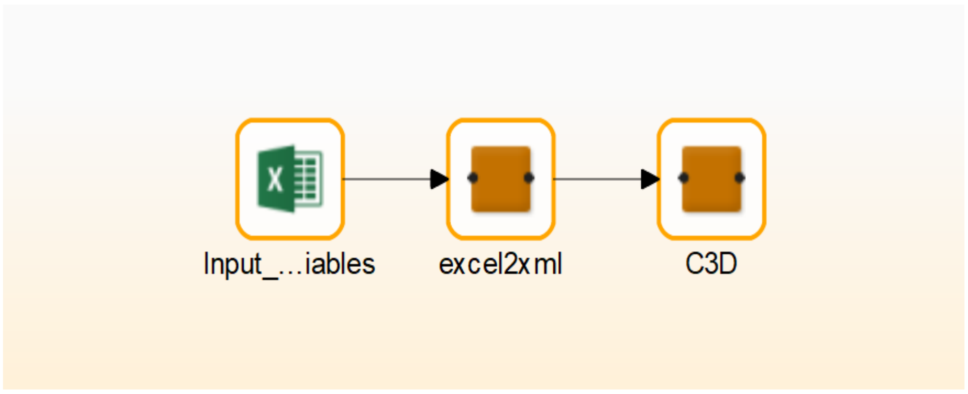


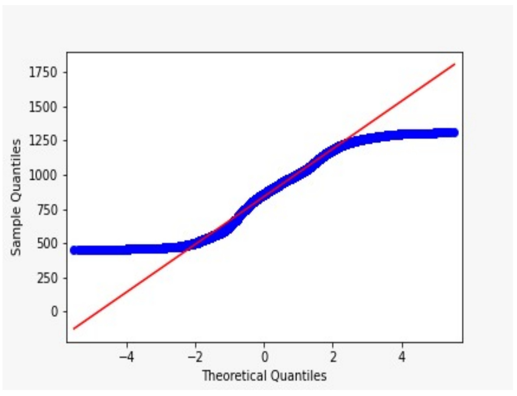
Figure 3.2: HEEDS Processes as shown in the graphical interface of HEEDS. © SIT AB

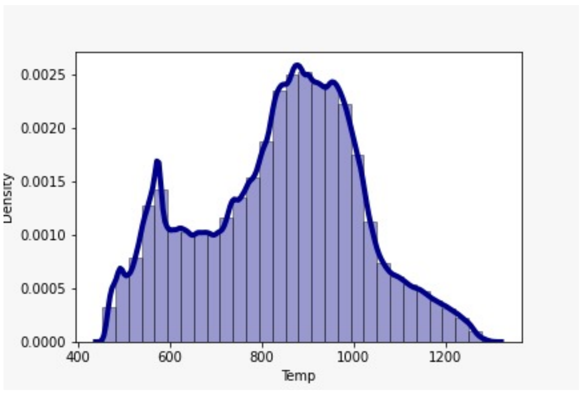
3.1.4 Data Distribution

Quantile-Quantile (Q-Q) plot is a probability plot, which is a useful graphical method for comparing two probability distributions by plotting their quantiles against each other. The Q-Q plot of the metal temperature response data [Figure 3.3] suggests that it is under-dispersed i.e., the distribution has thinner tails than a normal distribution. Under dispersion can occur when adjacent subgroups are correlated with each other, also known as autocorrelation. On a Q-Q plot under-dispersed data appears S shaped. In a histogram, it can be seen that such data has a reduced number of outliers [54], [55]. The data is bimodal (mixture of distributions) with one peak close to 900°C and another smaller peak at 500°C.

3.1.5 Correlation Analysis

The correlation between the cooling hole diameters and the output metal temperatures were studied using scatter plots for each hole and its neighbouring metal temperatures. Changing the diameter of a single cooling hole affects the temperature at multiple points on the blade

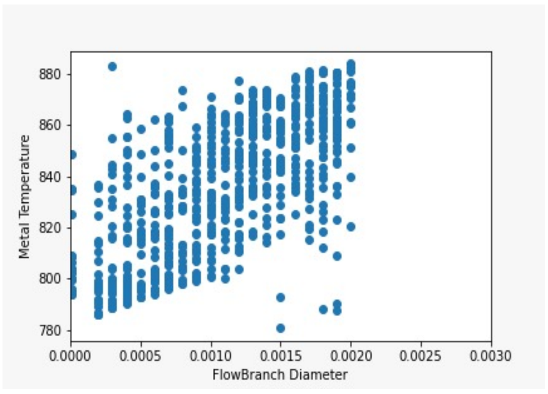


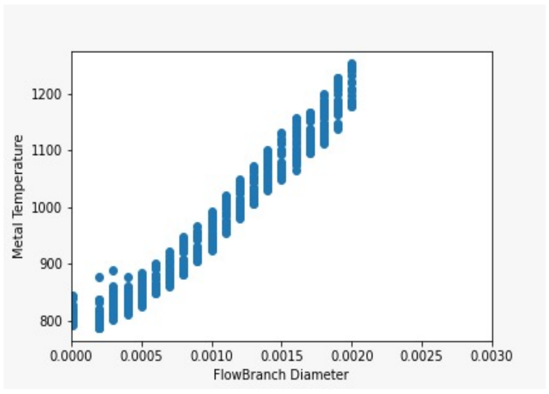


(a) (b)

Figure 3.3: (a) Q-Q plot of metal temperature response (b) Density distribution of metal temperatures

region close to it. Thus, there is a one-to-many mapping between the inputs and outputs. The diameters for two branches, one from the leading edge of the blade and one from the trailing edge of the blade are plotted in Figure 3.4 against the metal temperatures of their closest points on the blade across all the 1600 simulation runs. The cooling channel on the trailing edge is more correlated to its neighbouring node temperatures than the trailing edge is to its neighbouring temperatures. This means that not all the holes need to be changed to lead to the desired change in temperatures and the position of branches in the blade is essential in determining the metal temperatures.





(a) (b)

Figure 3.4: Correlation between input diameters and output metal temperatures in (a) Leading Edge (b) Trailing Edge of the blade

3.2 Implementation

This section will cover how the experiments performed in this thesis were carried out. As part of data pre-processing, data conversions and dimensionality reduction using PCA are performed on the data. Separate metamodels are then implemented to predict each score under the assumption that the score variables are independent. It is a reasonable assumption since the scores are linearly uncorrelated due to the application of PCA. The original response is then re-constructed from the predicted score vector.

3.2.1 Data Pre-Processing

Data Conversion Pipeline

The data for this research is a dump of .xml and .txt files exported from the CFD software. The data is hosted in the simple cloud storage offered by Amazon Web Services (AWS). Several pre-processing steps were involved in order to extract the relevant input and output information from the files. A simple pipeline was set up [Figure 3.5] to transform the raw data into suitable formats that can be used to train the surrogate models on.

The pipeline consists of three steps. In the first step, parsers built for each of the file formats are used to get the raw data into data frames. In the second step, data from multiple data frames are combined and merged to get the input and output variables of the model into desired shape. In the last step, the reshaped arrays are converted into pickle file format and stored in Amazon Simple Storage Service (Amazon S3) for performance reasons.

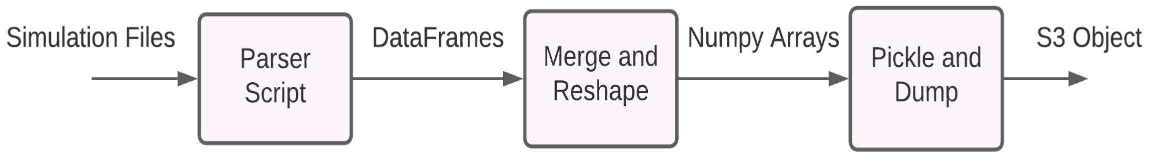


Figure 3.5: Data Processing Pipeline

Dimensionality Reduction using Cross Validated PCA

The most important parameter for the PCA algorithm is to choose the optimum number of Principal Components (PCs) that will be sufficient to describe the data effectively. For this study, it is not only enough to capture the maximum variance in the data but also reduce the reconstruction error on the validation set. Reconstruction error is the absolute difference between actual temperature values and the temperatures computed using the PCs [Equation 2.2]. When applying PCA to the 98,600-dimensional temperature response, 99.9 % of the variance in the original data is explained by just 50 principal components [Figure 3.6]. It indicates that 50 principal components are sufficient to capture the essential information of the high-dimensional temperature variables. However, the mean reconstruction error calculated on the validation set is not optimal for 50 components [Figure 3.7] and can be further lowered with the addition of more PCs.

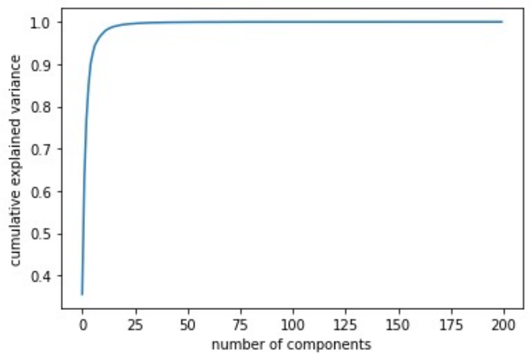


Figure 3.6: PCA Cumulative Variance Explained

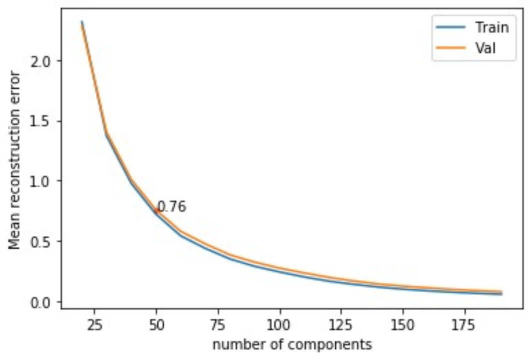


Figure 3.7: PCA mean reconstruction error for a particular train-validation split

Furthermore, the number of optimum PCs for low reconstruction error also depends on the particular split of train and validation data. These results hold good only for a particular train-validation split. In order to find the optimum number of PCs for any train-test split on the dataset, cross-validation (CV) is carried out on the PCA algorithm using the training dataset which consists of 80% of the total simulated designs. During the cross-validation procedure, the training dataset is further split into train-validation sets in the ratio of 80-20 containing 64% and 16% of the total designs respectively. For validation of the PCA algorithm, two error metrics are used to compute the cross-validation scores, the maximum and mean reconstruction error on the validation set. The maximum reconstruction error gives an estimate of the worst error in the temperatures reconstructed from PCs averaged over all the validation samples. Whereas the mean error gives an estimate of the mean absolute error in the reconstructed temperatures averaged over all the validation samples.

The method of repeated K-fold cross-validation is used with K=5 [Algorithm 2]. In each step of the repeated K-fold CV, the PCA algorithm is fit on 80% of the training data and the remaining 20% of the training set is used to compute the CV scores. The resulting graphs of cross validated PCA are presented and discussed in the results chapter. A comparative study of the metamodels trained using different number of principal components $(100, 200, 300 \ldots 1000)$ is also presented in the results chapter and the two approaches are compared with each other in order to select the optimum number of PCs.

Algorithm 2: Cross-Validation on PCA

```
Result: The optimum number of Principal Components
for n_components in range [20, 1000] with a step size of 20 do
    Initialize training and validation error metrics;
    Create repeated K fold cross validation splits;
    for TR, VA data sets in CV splits of temperature response do
        Scale the TR and VA sets to have zero mean and unit variance;
        Fit PCA to TR set and apply transformation on VA set;
        Apply inverse PCA to TR and VA sets to get back scaled temperatures;
        Apply inverse scaling to TR and VA sets to get back the original temperatures;
        Compute max reconstruction error for TR and VA sets;
        Compute mean reconstruction error for TR and VA sets;
    end
    Compute average of max and mean errors of all repeated folds;
end
```

3.2.2 Multi Output Regression Models

The dataset containing 1600 simulation designs is split into train-validation-test sets with a 64-16-20% ratio. When working with a regression problem, the most common baseline is to use a central tendency measure such as the mean or median as the result for all predictions. Several multi-output regression models [Section 2.3] including a mean, median and linear regression baseline are trained on the train and validation sets and evaluated on the test set. The mean and median baselines simply predict the target temperature at each point in the 3D mesh as the mean and median respectively of the corresponding temperature values in the training data set. Mainly, two models are compared in this study - Gradient Boosted Decision Trees (GBDT) and a Fully Connected Feedforward Deep Neural Network (FDN).

Linear Regression (LR) with L2 Regularization

The implementation of the ridge regression algorithm in the scikit-learn library supports multiple outputs directly. Thus, fitting the model to the multi dimensional temperature response makes it possible to predict multiple values at once. The linear regression model with L2 regularization strength $\lambda = 0.0001$ is used as a baseline in this study.

Gradient Boosted Decision Trees (GBDT)

The gradient boosted decision trees algorithm on the other hand does not support multi output regression directly and will raise an error when fit on multi dimensional response variables. A workaround for such models designed to predict only one value is to divide the multi output regression problem into multiple sub problems [section 2.3]. The MultiOutputRegressor wrapper implementation in scikit-learn, which uses the CART algorithm [56], takes a regression model as input and fits one regressor per target. In this experiment, an ensemble of gradient boosted decision trees was fit on each PCA score of the high-dimensional metal temperature response variables. The GBDT base learners were tuned on the hyperparameter ranges summarized in Table 3.2 using bayesian optimization [Section 2.9.1]. For comparison, GBDT was also fit on each of the original high-dimensional temperature responses. The objective function minimized by the optimizer is the mean squared loss and the model is evaluated on the validation set for the hyperparameter tuning.

Parameter	Range Low	Range High
Learning Rate	0.0003	0.1
Number of Estimators	10	200
Maximum Depth	3	20
Minimum Impurity	0.1	0.5

Table 3.2: GBDT - Hyperparameter Ranges

Fully Connected Feedforward Deep Neural Network (FDN)

To decide on the architecture of the deep neural network, a tuning job was carried out on parameters such as the number of dense layers and the number of neurons in the dense layers. The tuning job utilized 30 different networks with different number of hidden layers and neurons to find the optimum depth of the network and the optimum number of neurons in each layer using the bayesian optimization strategy [Section 2.9.1]. Other hyperparameters like the learning rate, number of epochs and dropout regularization were also optimized to achieve high model performance. Table 3.3 summarizes the parameter ranges used in the hyperparameter tuning of the fully connected feedforward deep neural network. Early stopping was used with a patience of 20 by monitoring the validation mean squared error loss. In this experiment, the output layer size was first fixed to 96800 for predicting the original high-dimensional temperatures and then to 700 for PCA score predictions.

Parameter	Range Low	Range High
Learning Rate	0.0003	0.01
Epochs	500	3000
Number of Dense Layers	2	6
Dropout	0.15	0.7
Number of Neurons in Dense layers	300	2000

Table 3.3: FDN - Hyperparameter Ranges

The proposed best architecture uses two fully connected dense layers, each with ReLU as an activation function, and an Adam optimizer that minimizes the mean squared loss objective function [Figure 3.8]

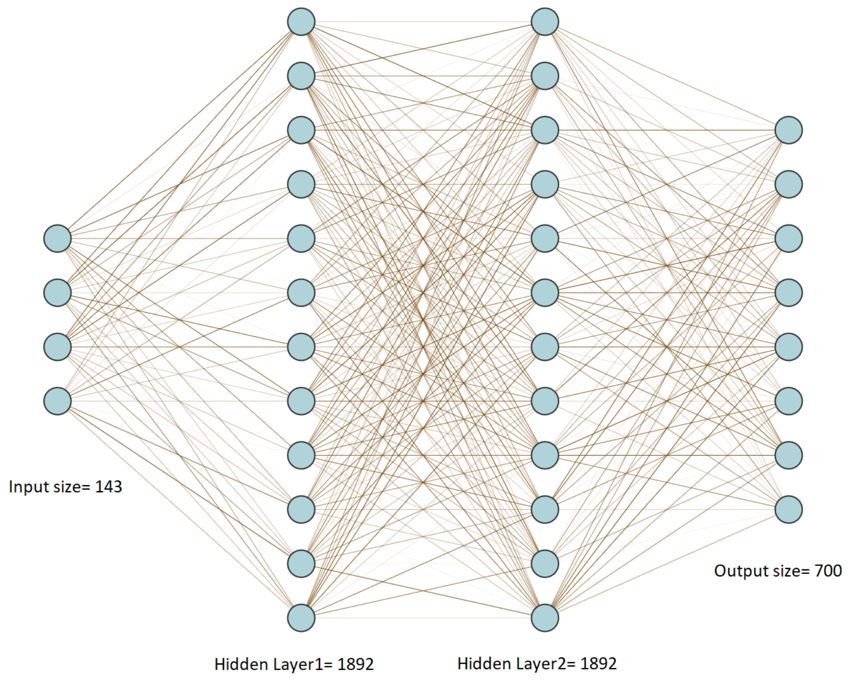


Figure 3.8: Architecture of NN Model used for training

3.2.3 Experimental Setup

AWS Sagemaker estimators are used to train the surrogate and baseline models. Tensorflow is used for deep learning models, while Sklearn is used for gradient boosted decision trees and linear regression models. The instances used for training the Tensorflow and Sklearn models are g4dn.2xlarge and ml.m4.2xlarge, respectively, as the former supports GPU while the latter does not. Both the instances have Intel Xeon Scalable processors as CPUs, 32GB of RAM, whereas g4dn.2xlarge has an additional 1 NVIDIA T4 Tensor Core GPU. The AWS Sagemaker tuner is used for hyperparameter tuning, which uses Bayesian optimization by default. The models are tuned and trained on 1280 simulation designs (train + validation sets) and evaluated on the remaining 320 designs (test set).

3.3 Evaluation

Before using the metamodels, it is important to know the accuracy of the model, i.e., how well the metamodel represents the underlying detailed simulation model. This could be done by studying different error measures. When the metamodel is found to be accurate enough, it can be used for metamodel-based optimization studies.

3.3.1 Standard Metrics

The evaluation metrics used to compare the performance and evaluate the predictive accuracy of the surrogate models are the Standard Mean Squared Error (MSE), coefficient of determination (R^2), Mean Average Error (MAE) and the Root Mean Squared Error (RMSE). The standard MSE does not distinguish between the two types of error i.e., under-predictions and overpredictions and penalizes both the same way. It is mainly concerned with penalizing large errors significantly (quadratically) more than the small ones.

3.3.2 Custom MSE

In determining the lifespan of the turbine blade, it is more favourable for the model to over-predict than underpredict the metal temperatures. This is because predicting temperatures higher than the actual values will result in a conservative life evaluation which can be regarded as a lower limit for the true life of the component computed from the CFD simulation. Hence, a custom MSE metric which is asymmetric is implemented for the evaluation of the model. The asymmetric loss function introduced in Section 2.8.1 is used to optimize the model and also used as an evaluation metric to assess its performance on the test set. As we want to err on the side of over-estimating, we trade off the penalty of over-estimation against under-estimation by assigning asymmetric weights to the two types of errors [section 2.8.1]. An α value of -0.49 was used to set the degree of asymmetry.

4 Results

This section presents and compares the results gathered in the experiments. Four different experiments were carried out in this study. First, performance of the baselines and the surrogate models were evaluated on the entire output dimension of the target. Second, optimum number of principal components was selected based on the results of CV PCA on the target variables. Third, baselines and the surrogates were evaluated on the optimum number of PCA scores of the target variables. Fourth, the FDN surrogate was trained and evaluated using an asymmetric regression loss. To compare the CFD results with the surrogate model results, the best and worst predictions by the surrogates are presented. The MSE and MAE of all the predictions on the test set are also visualized on the blade geometry. Finally, the time taken by the surrogate models for training and making predictions are compared.

4.1 Hyperparameter Optimization

Before any training, selection of model hyperparameters was done for all the models. Hyper parameter tuning jobs were run for both FDN models with fixed output layer size 96800 and 700, and GBDT model with output size 700. Whereas, the GBDT model with output size 96800 was trained with the default settings for the hyperparameters: *Number of Estimators*=10, *Maximum Depth*=4, *Minimum Impurity*=0.1 and *Learning Rate*=0.1. Because running a tuning job for GBDT models fit on 96800 response variables without GPU support was not practically feasible.

A total of 30 FDN models and 20 GBDT models with different hyperparameter settings were trained on 1024 simulation designs in the training set (60%) and evaluated on 256 designs in the validation set (16%). The optimal values of hyperparameters found by the bayesian optimization are listed in Table 4.1 for FDN and Table 4.2 for GBDT. As can be seen from plots in Figure 4.1 and Figure 4.2, the hyperparameter samples evaluated by the optimizer are chosen greedily to minimize the MSE loss of the response surface of the underlying objective function.

Parameter	Optimum Value FDN (700)	Optimum Value FDN (96800)
Learning Rate	0.006064	0.000988
Epochs	1985	2950
Number of Dense Layers	2	2
Dropout	0.424645	0.525365
Number of Neurons in Dense layers	1892	1692

Table 4.1: FDN - Optimal hyperparameters

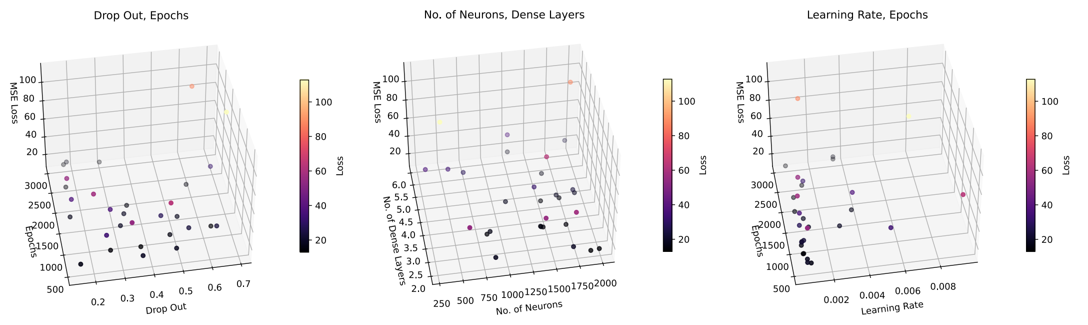


Figure 4.1: FDN - Hyperparameter Tuning using Bayesian Optimization (Drop out, Epochs, Learning Rate, No. of neurons, No. of dense layers)

Parameter	Optimum Value GBDT (700)
Learning Rate	0.095749
Number of Estimators	193
Maximum Depth	4
Minimum Impurity	0.209655

Table 4.2: GBDT - Optimal hyperparameters

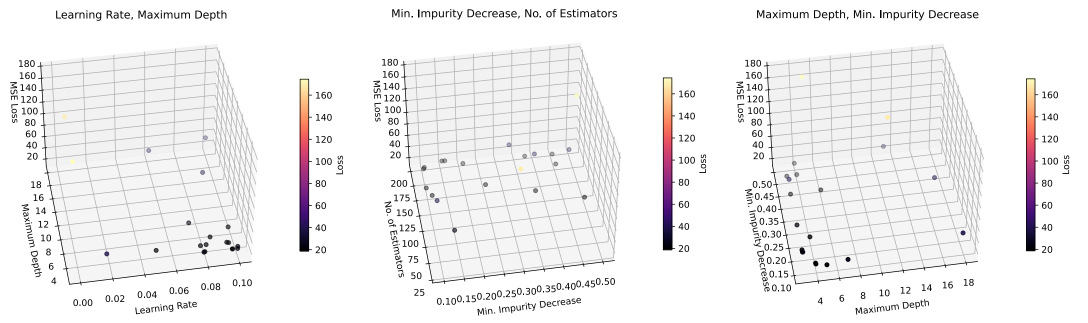


Figure 4.2: GBDT - Hyperparameter Tuning using Bayesian Optimization, Max Depth, Min Impurity, No. of Trees, Learning Rate

4.2 Training directly on high-dimensional target data

As a reference, all the models are first trained directly on the high dimensional metal temperature response variables. The performance of mean, median, and linear regression baseline models are compared with that of the two chosen models in this study [Table 4.3].

Model	MAE	MSE	RMSE	R2
Median	34.10	3699.45	60.82	-0.051
Mean	34.82	3626.04	60.22	-0.005
Ridge Regression	12.79	473.83	21.77	0.544
FDN	11.59	335.20	18.31	-6.605
GBDT	12.92	486.69	22.06	0.745

Table 4.3: Performance comparison of models trained on high-dimensional metal temperature response

4.3 Results for different amount of PCA components

It is observed that for the CV PCA experiments ran with the settings detailed in Section 3.2.1, the improvement in the maximum reconstruction error gained from adding additional PCs plateaus after 500 principal components [Figure 4.3]. The mean reconstruction error reaches its minimum value around 300 principal components, much before the maximum error reaches its minimum. According to these results, selecting 500-1000 principal components for building the metamodels will most likely keep the maximum error introduced due to dimensionality reduction below 10°C. It is an optimal choice considering how the max error doesn't improve much with the addition of more PCs beyond this point. The results of CV PCA can be generalized and said to be true for the underlying true distribution of the data.

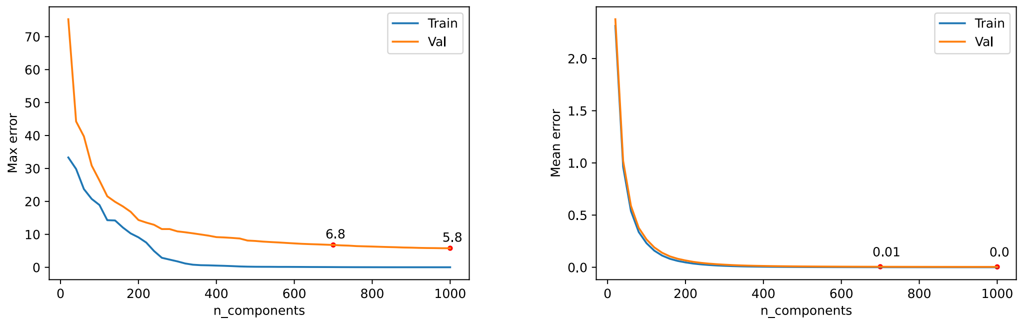


Figure 4.3: Cross-Validation Scores

To further validate if the choice of PCs was optimal, an alternative approach was considered. The two chosen models were trained on different number of PCs determined on the training set and evaluated on their predictions on the validation set. Here, the errors introduced due to the surrogate models are evaluated on top of the errors present from dimensionality reduction [Figure 4.4]. This is in contrast with the first part of this experiment where only the reconstruction errors due to the dimensionality reduction was considered to optimize the PCs. For both the models, validation MSE falls below 20°C when 700 principal components is selected. There is no decided value of MSE that is desired to be achieved here, only to minimize it as much as possible with low computational overhead. It is obvious from the results that adding more PCs drives the validation MSE further lower, but the improvement quite saturates after 500 components. This is in agreement with the results of CV PCA.

To minimize the validation MSE objective and maximize model performance, 700 PCs were selected as the optimum choice for training the surrogate models on. The FDN model was trained with the optimized architecture found in the previous experiment with the full output dimension. It was not feasible to run a hyperparameter tuning job for each selection of PCs before model training. Likewise, the GBDT model was trained on different number of PCs with the default hyperparameter settings.

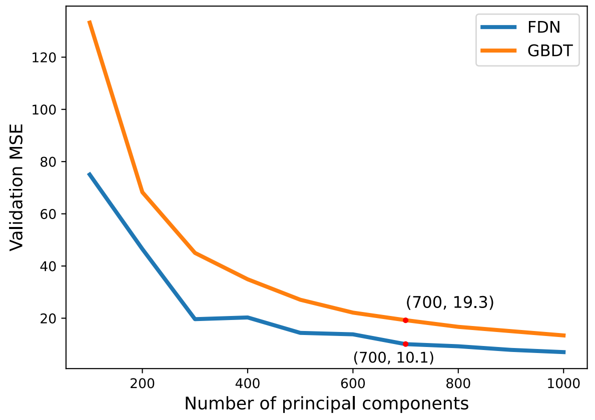


Figure 4.4: Test MSE of FDN and GBDT models trained on different number of principal components

4.4 Training on PCA scores

The performance of the baselines, FDN and GBDT models trained on (700) PCA scores of the high dimensional temperature response are presented in Table 4.4. The percentage improvement of the best performing model which is the FDN is also reported.

Model	MAE	MSE	RMSE	R2	% Decrease in MAE
Median	35.59	3826.07	61.86	-0.036	84.71%
Mean	34.90	3627.72	60.23	-0.004	84.40%
Ridge Regression	12.79	473.83	21.77	0.544	57.47%
GBDT	5.49	120.89	10.99	0.917	0.91%
FDN	5.44	92.76	9.63	0.925	-

Table 4.4: Performance comparison of models trained on low-dimensional PCA scores of temperature response

The mean squared error and the mean absolute error of predictions over the test samples are plotted at each point in the 3D mesh of the blade for both the surrogate models [Figures 4.8 - 4.11]. The MSE and MAE are high along the leading and trailing edges of the blade where the temperatures are harder to predict due to their wide ranges. Since, MSE is in squared units, its harder to interpret its values in terms of the temperature outputs that we are interested in. Whereas, MAE matches the units of the target value being predicted and is therefore more intuitive. The mean absolute error in both cases range between $0°C - 18°C$. There are two modes in the MAE distribution of all the temperature nodes on the blade [Figure 4.7], the first one is near $1°C$ and the second peak is around $7.5°C$. In other words, most of the bad predictions at the output nodes are off by less than $10°C$ while at some nodes (mostly concentrated around the leading and trailing edges of the blades), they are off by $10°C - 17°C$.

The mean over all the target points on the blade gives a single error metric for the model as presented in Table 4.4.

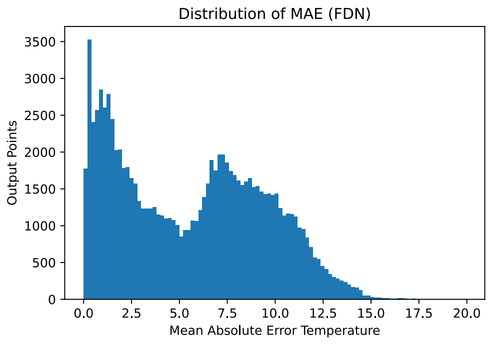


Figure 4.5

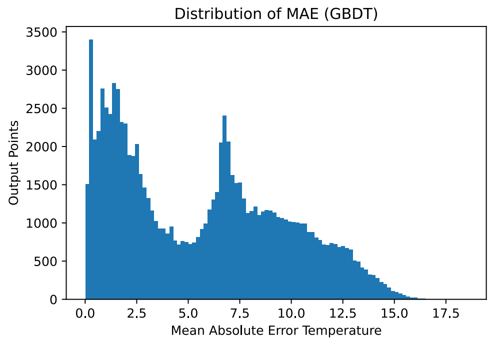


Figure 4.6

Figure 4.7: (a) Mean Absolute Error Distribution in FDN (B) Mean Absolute Error Distribution in GBDT

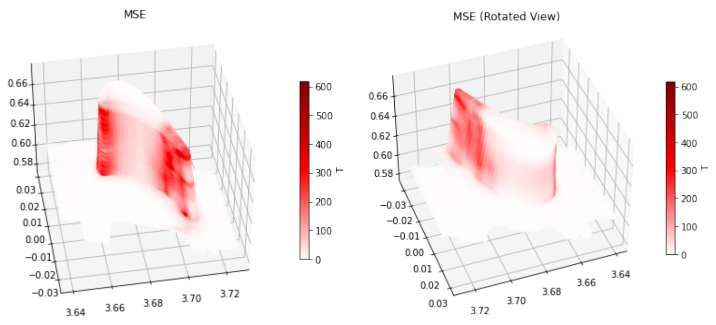


Figure 4.8: FDN Mean Squared Error

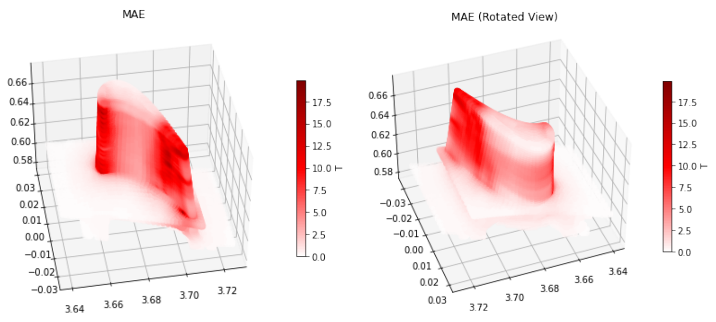


Figure 4.9: FDN Mean Absolute Error

The worst and best predictions of the metal temperatures from the two surrogate models- GBDT and FDN trained on PCA scores, as visualized in the form of the gas turbine blade geometry is presented in Figures 4.12 - 4.15. Each figure has three subplots, the left plot is the result of CFD simulation model, the middle plot is the prediction from the surrogate model, and the right plot is the error distribution which is the difference between CFD and the surrogate. The regions in red are the nodes where the predicted temperature values are lower than the actual temperatures, while the blue regions on the blade indicate overpredictions.

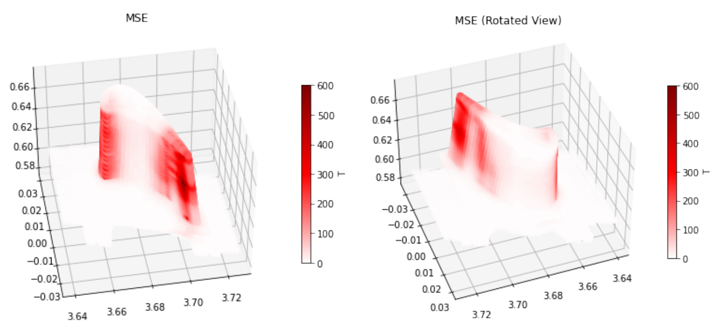


Figure 4.10: GBDT Mean Squared Error

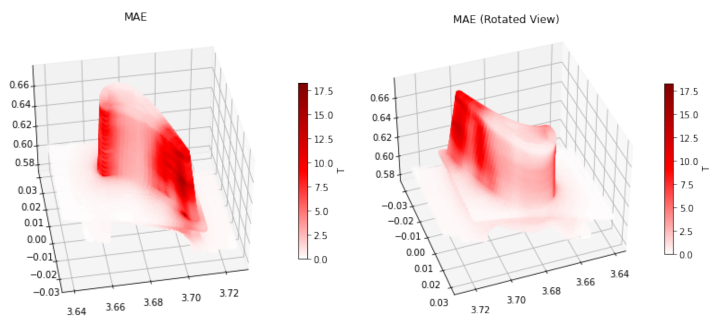


Figure 4.11: GBDT Mean Absolute Error

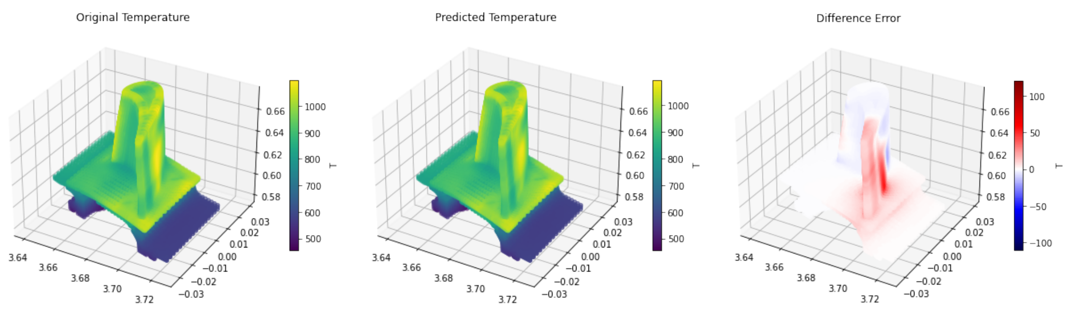


Figure 4.12: Worst prediction from Feedforward Deep Neural network

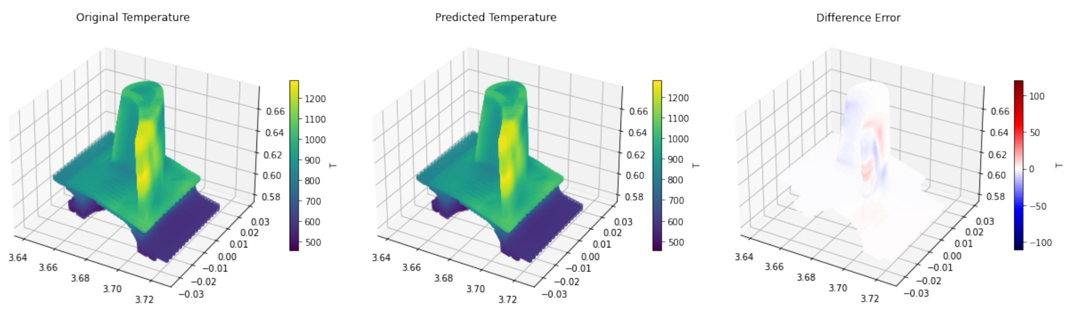


Figure 4.13: Best prediction from Feedforward Deep Neural network

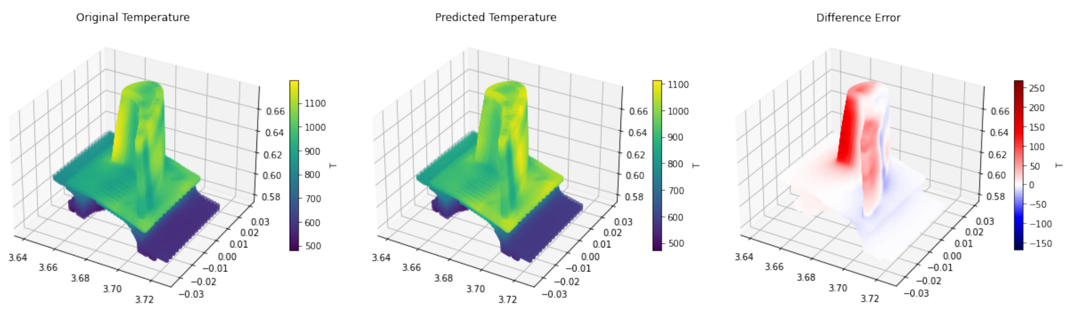


Figure 4.14: Worst prediction from Gradient Boosted Decision Trees

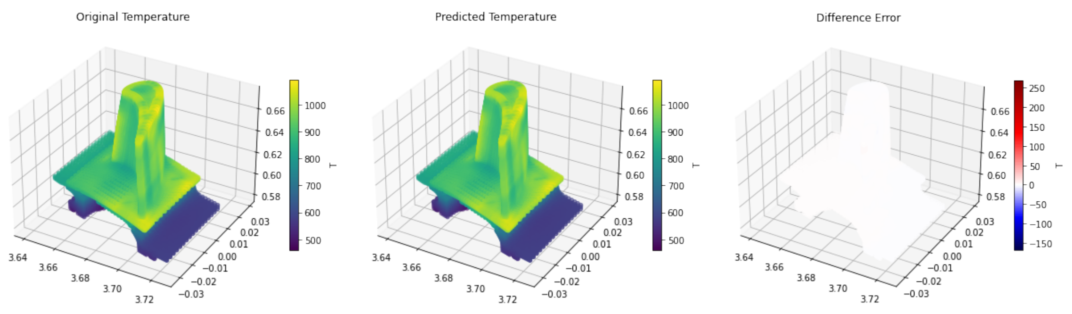


Figure 4.15: Best prediction from Gradient Boosted Decision Trees

The neural network model with an RMSE of 9.63 outperforms the linear kriging baseline having RMSE value of 98.2 with a relative improvement of 90%. The software script for the integrated surrogate model only supports RMSE which is why that is the metric used for comparison with the kriging baseline. Because the multi-target regression models compute the mean error metrics along two axes, i.e., mean over the samples in the test set and also over all the targets, we can compute the error metrics for each sample as well. Table 4.5 shows the metrics computed for the worst and best predictions of the best performing surrogate model which is the FDN. It can be seen that the worst RMSE is much lower than the RMSE of the kriging baseline.

Sample	**MAE**	**MSE**	**RMSE**	**R2**
Worst	10.33	319.07	17.86	0.93
Best	2.66	13.89	3.73	0.99

Table 4.5: Sample error metrics for the FDN model

4.5 Time Comparisons

The C3D simulation times are listed in Table 4.6. These times are approximate and not exact since the HEEDS software generates multiple designs at once and parallelizes the temperature calculates performed in C3D. At most 4 designs were run in parallel for the data generation in this study and the time taken for 320 designs is estimated by simple interpolation for fair comparison with the surrogate prediction times. The computation time for the speed up gained by the best performing surrogate models which are the FDN and GBDT models trained on 700 PCA scores are summarized in Table 4.7. The total time taken for tuning jobs are recorded in hours and the time taken for training the models with optimal hyperparameter values are recorded in seconds. The prediction times are over all the 320 designs in the test set.

Model	No. of Designs	Computation Time (h)	Computation Time (s)
C3D	4	0.25	900
C3D	320	20	72,000

Table 4.6: Temperature calculation times in C3D using HEEDS

Model	Tuning (h)	Training (s)	Prediction (s)	% Increase (over C3D)
FDN	0.92	129	0.137	99.9998
GBDT	1.13	124	7.843	99.9891

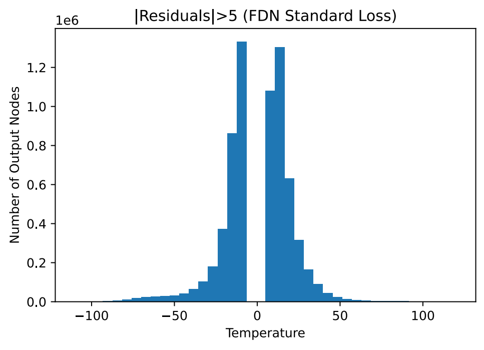
Table 4.7: Speed gain in predictions on 320 test designs using best performing surrogates trained on low-dimensional PCA scores of temperature responses.

4.6 Custom loss

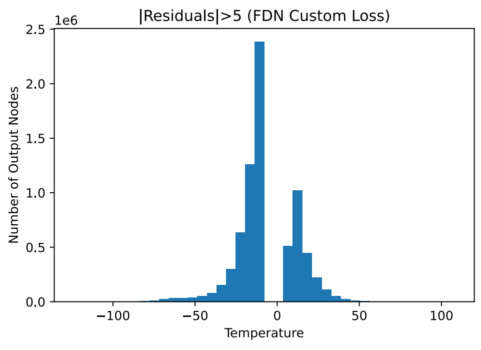
The residual errors from predictions on the test designs for the FDN model trained with standard MSE loss and the custom asymmetric MSE loss are compared in Figure 4.16. The majority of the residual errors close to zero are dropped to better visualize the effect of asymmetric loss. The skewness towards negative values of residuals has clearly increased in (b) compared to (a). But, the number of non-zero residuals (absolute value greater than 5) has increased along the y-axis from (a) to (b). The best and the worst predictions from FDN trained on the asymmetric loss is presented in Figure 4.17 - 4.18. Model metrics as computed in the previous experiments are listed in Table 4.8. It can be seen that while the custom MSE loss is minimized, the standard MSE loss of the model has increased as compared to the previous FDN model trained on standard MSE loss.

Model	MAE	MSE	RMSE	Custom MSE	R2
FDN (Custom Loss)	6.03	103.75	10.619	32.34	0.918

Table 4.8: Performance of FDN model trained using custom asymmetric MSE loss



(a)



(b)

Figure 4.16: Residuals from FDN trained using (a) standard MSE loss (B) custom MSE loss

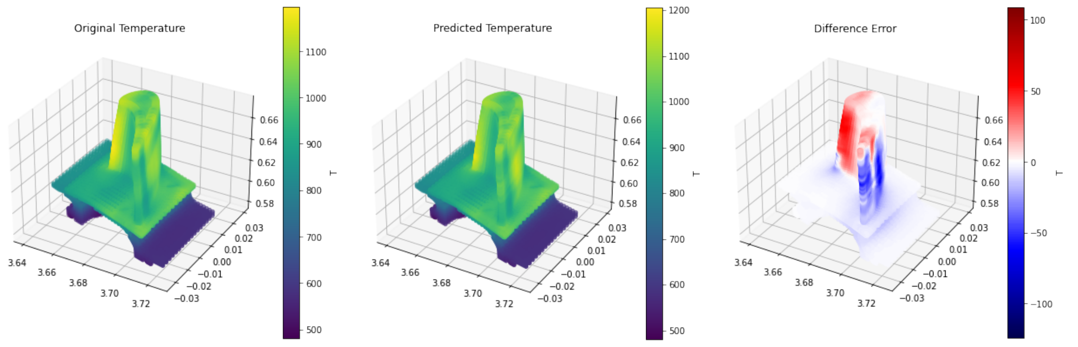


Figure 4.17: Worst prediction from Feedforward Deep Neural network with Custom MSE Loss

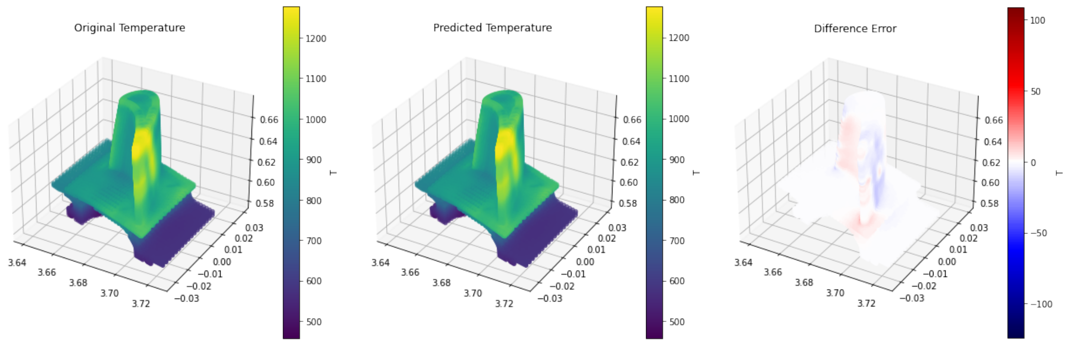


Figure 4.18: Best prediction from Feedforward Deep Neural network with Custom MSE Loss

5 Discussion

This thesis aims to use metamodeling in the life cycle optimization of a gas turbine. This chapter discusses the study results and the limitations of data and methods used to perform the experiments. Ideas about future work are presented to guide further studies and improvements within the subject. The work is also placed in the broader context and its impact on related areas are brought up.

5.1 Results

The results of the experiments presented in the previous chapter are discussed in this section. In particular, models trained on the high dimensional targets are compared with models trained on the low dimensional PCA scores of the targets in terms of their performance, training and prediction times.

5.1.1 Training directly on high-dimensional target data

The ridge regression baseline and gradient boosted decision trees perform considerably better than the FDN model on the full dimension of the target temperatures. At first glance, the results seem conflicting, both in terms of expected model performances and the observed values of evaluation metrics. One would expect the FDN model to generalize better than the simple linear regression baseline. Also, given the low value of MSE and MAE observed for the FDN, it is surprising to see a low R-squared value. The poor generalization of the model can be attributed to the large number of weights in the network. Due to the extremely high dimensional output layer in the network architecture, there are approximately 30 million trainable parameters in the model. The size of the data used in this study is insufficient to reasonably train such a vast network.

In a multi-target regression problem, the MSE, MAE and R-squared metrics are calculated for each target variable and averaged out over all of them to get a single estimate for the model. One should not be misled by the low MSE value while assessing the various model performances. A low MSE value or a high R-squared value on its own does not necessarily mean good output predictions. Considering the distribution of MSE, MAE and R-squared values over all the target variables gives a better insight into the goodness of fit of the models.

It is helpful to evaluate which target variables (for this study, the blade regions) are predicted well and which targets the model performs poorly on. The negative R-squared for FDN despite the low MSE can be explained by looking at how the metric is computed [Equation 2.10]. The mean R-squared is not a linear function of the mean MSE (mean over all targets) and is scaled by the variance of the targets. The average MSE of the FDN is comparable to that of the GBDT and linear regression models. Still, because the model performs poorly on most targets while performing exceptionally well on very few targets, it gets a low mean R-squared score. The GBDT and linear regression models perform reasonably well on all the targets and therefore have a low MSE complemented with a high mean R-squared score.

Due to the sampling methods used in the data generation process, there is an inherent grouping in the predictors of the models. Input diameters for a group of cooling branches are always varied together, which introduces collinearity among the predictors. Linear regression with L2 regularization and gradient boosting technique, in general, can handle multicollinearity between input features quite well [57]. Neural networks also tend to be fairly insensitive to multicollinearity because of their nonlinear projection methods and tendency to overparameterize [58]. However, multicollinearity affects the learning of an artificial neural network so that it will take more time to converge to its optimum weights.

5.1.2 Training on PCA scores

Two approaches were used to determine the optimum number of PCs in the target temperatures. In the CV PCA approach, reconstruction errors on validation data were considered to optimize the PCs. These are the errors introduced due to dimensionality reduction. In contrast, the learning curve approach optimized PCs based on the validation MSE errors in surrogate modeling. The learning curve shows the improvement in model performances with the addition of more PCs. Training the surrogates on 500-1000 principal components did not affect the model performances much. An optimum of 700 PCs were chosen to keep the maximum reconstruction error and also the validation MSE below $20°C$.

With the output layer size reduced to 700 dimensions instead of 96800 dimensions, the number of parameters in the FDN model reduces to almost 2 million. The metal temperature predictions improve considerably [Table 4.4]. The generalization performance of the FDN can to some extent be attributed to the high rate of dropout regularization used. Interestingly, the ridge regression model performs equally well when predicting the PCA scores of temperatures and the original temperatures. There is almost no increase in prediction errors introduced due to PCA for this model. Improvement of GBDT and FDN models is the most drastic after PCA is applied to the target variables. FDN somewhat outperforms the GBDT model on the test set in terms of the evaluation metrics and also its training and prediction times. The worst temperature predictions from FDN are $100°C$ off in magnitude from their true values while the worst predictions from GBDT are off by $250°C$. The high difference errors are generally concentrated along the leading and trailing edges of the blade as suggested by the MSE and MAE plots [Figure 4.8-4.11]. Both models are trained on 1280 training + validation samples and evaluated on 320 samples in the test set. Their training times are almost comparable [Table 4.7] even though FDN is trained using GPU support while GBDT is trained without it. Whereas, the prediction time of GBDT is about 100 times slower than that of the FDN which are both done on CPU. This is because the runtime cost for prediction in GBDT is linear in the number of trees and also in the number of targets, whereas with the FDN gives all outputs from one forward pass of the network.

The prediction results from FDN trained on asymmetric loss is unsatisfactory as suggested by the histogram of residuals in Figure 4.16. This could be because the predictions are in the domain of PC scores. And the asymmetric loss imposed on the scores does not translate back to the expected overpredictions in terms of the original temperatures. Overall, the number of non zero residuals have increased which is worse for the evaluation of life of the blade.Moreover, it is difficult to assess whether the goal of favoring overpredictions is achieved since it is not trivial to decide how much trade off is acceptable in terms of replacing the positive errors with negative errors. Ultimately, the output of the "thermal" surrogate model is only a step in the bigger process of design optimization of the blade. The model prediction errors themselves are not of high importance as long as the chain of surrogates [refer Figure 1.2] can guide the design optimization process in the right direction. This can only be validated when the optimum design of the blade (i.e., the diameter values of cooling holes when the geometry of the blade is considered constant) as determined by the surrogate models is also evaluated using the actual CFD simulations, which can be a week long process and is outside the scope of this study.

5.2 Method

This section discusses the methods used, its limitations in the context of its applicability to future and other similar works, and alternative approaches and recommendations for future research.

5.2.1 Data Sampling and Generation

Unless the data generation process is truly random, there is likely some amount of bias in the data used for training the surrogate models. One such constraint in the data sampling done for this study is that the diameters of the cooling holes are varied in groups [Section 3.1.2]. This introduces some sampling bias in the data, which propagates to the model training. The surrogate models are influenced towards patterns more commonly occurring in the training data and are thus less likely to be correct in predicting extreme variations. This bias in training data in regression problems is similar to the problem associated with minority classes in classification [59]. As a result, when test samples from a similar data generation process are presented to the models, predictions are within the well-represented region. The errors for such samples are low, while the predictions for test samples generated from a different process will lie outside this region resulting in a high prediction error.

The primary aim of surrogate models is to reduce computational cost by avoiding the solution of the full scale model. However, often, surrogate models do not exhibit the acceptable features over the entire domain, even after extensive sampling is carried out [60]. In [61], adaptive sampling or sequential sampling method is shown to require fewer observations thus reducing the total time and computational effort required for data. In adaptive sampling, a surrogate is fitted to the sample points that have already been generated. The fitted surrogate is then used for directing new sample points towards areas that will increase the surrogate accuracy. In other words, adaptive sampling relies on the prediction uncertainty of the surrogate models to generate new data. It is shown to be more efficient in accounting for the nature of the system being modelled [62].

5.2.2 Dimensionality Reduction

The number of nodes in the CAD mesh model of the blade where the metal temperatures are calculated is very large. It is in the order of tens of thousands and can increase further if the granularity of the mesh is increased. In this study, PCA is applied to the target temperature variables to reduce the number of points for output predictions. PCA projects the data into a lower-dimensional space where the variance of the projected data is greedily maximized along each axis. It does not take correlations between the inputs and the output into account. As a result, PCA may perform poorly if the input features are strongly correlated with target directions that have low variance. PCA will drop the directions with a lower variance, and the final regressor will not be able to leverage the predictive power of the input features on those target variables. Furthermore, PCA performs poorly when there are non linear correlations present in the data. Other dimensionality reduction techniques need to be employed to preserve the non-linear correlations between inputs and outputs. Because of the complex geometry of the turbine blade where the metal temperatures are calculated, they may be better represented using a non-linear dimension reduction method that can uncover both linear and nonlinear correlations in the data. Autoencoders are a family of neural networks that were first developed in the late 1980s for the purpose of non-linear dimensionality reduction [63]. It can give a latent space representation of the high dimensional targets and has been widely applied for this purpose in many studies. [64], [65]

5.2.3 Model Selection

The surrogate models used in this study fail to make accurate predictions along the leading and trailing edges of the blade consistently. This is mainly because the temperature ranges and distributions vary widely from one spatial region of the blade to another. They are not well modeled using a single surrogate without more physics-based feature engineering [66]. One way to improve the predictions is to model different regions of the blade separately. One can use a clustering method followed by a group of surrogate models for each cluster to model each region more accurately. Each model can take into account the local distribution of metal temperatures in that region leading to better predictions overall [67].

Furthermore, the models used in this work do not use the geometric structure of the blade explicitly as input features to predict the metal temperatures. The position of the cooling holes (input diameters) as well as the x, y, z coordinates of the nodes (output metal temperatures) are constant in this study and therefore implicitly encoded in the position of the input features and the target variables. However, if the blade's geometry were to change in the future, the models would still learn similar correlations between the cooling hole diameters and the metal temperatures without taking into account the effect of physical geometry on the input-output relation of variables. One way to encode this geometric information would be to use graph neural networks to capture the neighborhood information of the nodes on the blade. In [68], graph-based networks are used to learn accurate, long-term simulations of different fluids, and rigid solids. One of the more interesting claims in the study is that the graph network can generalize well beyond training to more challenging settings.

The use of ML models is particularly promising in scientific problems involving processes that are not completely understood, or where it is computationally infeasible to run mechanistic models in desired time. However, the application of even the state-of-the-art black box ML models often have limited success in scientific domains due to their large data requirements, inability to produce physically consistent results, and their lack of generalizability to out-of-sample scenarios [69]. Alternatively, in [66], a physics-guided machine learning (PGML) approach was used to leverage the knowledge of physical systems into the learning of a neural network. The features extracted from simplified physics-based models are combined with input features to learn physically consistent representation of the quantities of interest such as pressure, temperature and velocity field of fluid simulations. Integrating Physics-based modeling with machine learning to leverage their complementary strengths is highly active area of research [70], [71].

5.3 The work in a wider context

The long-term implementation of this research work involves two stages – first stage is to create surrogate models for each step of the design optimization cycle and the second stage is to use these surrogates in the optimization process. This project addresses the first stage i.e., to build a surrogate model for the cooling model of the turbine blade. The second stage of implementation is to integrate this surrogate model with other surrogate models created for stages further down in the design cycle. Ultimately, determining whether a mechanical engineer at SIT AB can use this pipeline of surrogates to replace the current design optimization is left as future work to be carried out after analysing the results of this thesis work. The best designs produced by the pipeline of surrogates will be validated by running numerical calculations on the actual CFD software. If the design is found to be optimal in the simulations, then the surrogates can be used in the optimization process of gas turbines blades. This is beneficial in many ways, both directly for Siemens in an economical aspect but also in an environmentally sustainable aspect. Higher efficiency of gas turbines lowers fuel costs, maintenance costs, and more crucially, in an age of escalating climate concerns, lowers carbon dioxide emissions per unit of electricity generated. Depending on the extent of reliance on the surrogate models and their predictions, this work is a first step in the direction of automating the design optimization process. This will in the near future reduce the human intervention required and relax the work loads of CFD engineers.

The ideas and techniques presented in this thesis apply to other related industries and not only Siemens. The surrogate modeling framework used in this study can be adapted to any field in engineering to optimize different design processes. If applied to the automobile and aviation industries, metamodeling-based optimization could provide better and safer alternatives quicker.

6 Conclusion

In this work, two meta-modeling methods using multi-target regression are applied to the cooling process of gas turbine blades. Through experiments, it is demonstrated that the predictions of blade metal temperatures by meta-modeling via fully connected feedforward neural network and gradient boosted regression trees are successfully correlated with the simulations by C3D in most situations. Training the models on the reduced dimension of the output temperature variables gives better prediction results than the models trained on the original temperature variables that are spatially correlated. The mean error of every prediction point from the neural network surrogate is below $20°C$ and the overall mean error of the model is about $5°C$. The model has an RMSE value of 9.63 which is a significant improvement over the existing kriging baseline model with an RMSE value of 98.2. Thus, given new diameters of the cooling holes, the metal temperatures can be estimated in a very short time. The computational time gain compared to traditional process is in the order of tens of thousands of seconds. Surrogates can make multiple predictions in under one-tenth of a second, while a set of four design runs in C3D takes around fifteen minutes. However, there is currently a lot of active research with deep learning methods and physics-informed neural networks (PINN) that could provide much more interesting results in this area in the future.

Bibliography

[1] J. Sundberg, *Heat transfer correlations for gas turbine cooling*, 2006.

[2] "Press release: Gas turbines from siemens energy are providing leipzig with a climate neutral power supply," Nov. 2020. [Online]. Available: `https://assets.siemens-energy.com/siemens/assets/api/uuid:2d9221e4-9bb9-426c-8596-347869d6e507/SE-PR-Order-SGT-800-Leipzig.pdf`.

[3] *Turbine blade pressure face*, 2010. [Online]. Available: `https://line.17qq.com/articles/whwrsttcx.html`.

[4] C. C. Jose Rodriguez Philipp Cavadini Marco Brunelli and C. Carpenter, "Siemens energy use a single integrated platform for gas turbine blade cooling prediction," Siemens Digital Industries Software, Tech. Rep., Jan. 2017.

[5] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 481–490, ISBN: 9781450342322. DOI: `10.1145/2939672.2939738`. [Online]. Available: `https://doi.org/10.1145/2939672.2939738`.

[6] M. D. Ribeiro, A. Rehman, S. Ahmed, and A. Dengel, *Deepcfd: Efficient steady-state laminar flow approximation with deep convolutional neural networks*, 2020. arXiv: `2004.08826 [physics.comp-ph]`.

[7] A. Ansari, S. Mohaghegh, M. Shahnam, J. F. Dietiker, A. Takbiri Borujeni, and E. Fathi, "Data driven smart proxy for cfd: Application of big data analytics machine learning in computational fluid dynamics, part one: Proof of concept; netl-pub-21574; netl technical report series; u.s. department of energy, national energy technology laboratory: Morgantown, wv, 2017.," Nov. 2017. DOI: `10.2172/1417305`. [Online]. Available: `https://www.osti.gov/biblio/1417305`.

[8] T. Simpson, F. Mistree, J. Korte, and T. Mauery, "Comparison of response surface and kriging models for multidisciplinary design optimization," in *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1998, p. 4755.

[9] O. Kajero, R. Thorpe, Y. Yao, D. Wong, and T. Chen, "Meta-model based calibration and sensitivity studies of cfd simulation of jet pumps," *Chemical Engineering Technology*, vol. 40, May 2017. DOI: `10.1002/ceat.201600477`.

[10] O. T. Kajero, "Meta-model assisted calibration of computational fluid dynamics simulation models.," 2017.

[11] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross, "Data-driven fluid simulations using regression forests," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, Oct. 2015, ISSN: 0730-0301. DOI: `10.1145/2816795.2818129`. [Online]. Available: `https://doi.org/10.1145/2816795.2818129`.

[12] T. Chen, K. Hadinoto, W. Yan, and Y. Ma, "Efficient meta-modelling of complex process simulations with time–space-dependent outputs," *Computers Chemical Engineering*, vol. 35, no. 3, pp. 502–509, 2011, ISSN: 0098-1354. DOI: `https://doi.org/10.1016/j.compchemeng.2010.05.013`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S009813541000195X`.

[13] A. H.-D. Cheng and D. T. Cheng, "Heritage and early history of the boundary element method," *Engineering Analysis with Boundary Elements*, vol. 29, no. 3, pp. 268–302, 2005.

[14] G. G. Wang and S. Shan, "Review of Metamodeling Techniques in Support of Engineering Design Optimization," *Journal of Mechanical Design*, vol. 129, no. 4, pp. 370–380, May 2006, ISSN: 1050-0472. DOI: `10.1115/1.2429697`. [Online]. Available: `https://doi.org/10.1115/1.2429697`.

[15] W. Smith, "Experimental design for formulation, vol. 15 of asa-siam series on statistics and applied probability," *Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa, USA*, 2005.

[16] K.-T. Fang, D. K. Lin, P. Winker, and Y. Zhang, "Uniform design: Theory and application," *Technometrics*, vol. 42, no. 3, pp. 237–248, 2000.

[17] A. Giunta, S. Wojtkiewicz, and M. Eldred, "Overview of modern design of experiments methods for computational simulations," vol. 0649, Jan. 2003. DOI: `10.2514/6.2003-649`.

[18] A. Olsson, G. Sandberg, and O. Dahlblom, "On latin hypercube sampling for structural reliability analysis," *Structural Safety*, vol. 25, no. 1, pp. 47–68, 2003, ISSN: 0167-4730. DOI: `https://doi.org/10.1016/S0167-4730(02)00039-5`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167473002000395`.

[19] M. D. Shields and J. Zhang, "The generalization of latin hypercube sampling," *Reliability Engineering System Safety*, vol. 148, pp. 96–108, 2016, ISSN: 0951-8320. DOI: `https://doi.org/10.1016/j.ress.2015.12.002`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0951832015003543`.

[20] S. Glen. (). "Latin hypercube sampling: Simple definition," [Online]. Available: `https://www.statisticshowto.com/latin-hypercube-sampling/` (visited on 04/26/2021).

[21] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[22] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1, pp. 37–52, 1987, Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, ISSN: 0169-7439. DOI: `https://doi.org/10.1016/0169-7439(87)80084-9`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/0169743987800849`.

[23] I. Jolliffe, "Principal component analysis. 2nd ed," *http://lst-iiep.iiep-unesco.org/cgi-bin/wwwi32.exe/[in=epidoc1.in]/?t2000=0177 16/(100)*, vol. 98, Oct. 2005. DOI: 10.1002/0470013192.bsa501.

[24] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal component analysis," in *A practical approach to microarray data analysis*, Springer, 2003, pp. 91–109.

[25] J. Brownlee. (Mar. 2020). "How to develop multi-output regression models with python," [Online]. Available: https://machinelearningmastery.com/multi-output-regression-models-with-python/ (visited on 03/08/2021).

[26] H. Borchani, G. Varando, C. Bielza, and P. Larranaga, "A survey on multi-output regression," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 5, pp. 216–233, 2015.

[27] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, and I. Vlahavas, "Multi-target regression via input space expansion: Treating targets as inputs," *Machine Learning*, vol. 104, no. 1, pp. 55–98, 2016.

[28] A.-N. Spiess and N. Neumeyer, "An evaluation of r 2 as an inadequate measure for non-linear models in pharmacological and biochemical research: A monte carlo approach," *BMC pharmacology*, vol. 10, no. 1, pp. 1–11, 2010.

[29] C. M. Theobald, "Generalizations of mean square error applied to ridge regression," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 1, pp. 103–106, 1974.

[30] "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[31] J. Elith, J. R. Leathwick, and T. Hastie, "A working guide to boosted regression trees," *Journal of Animal Ecology*, vol. 77, no. 4, pp. 802–813, 2008.

[32] E. Carrizosa, C. Molero-Rıo, and D. R. Morales, "Mathematical optimization in classification and regression trees," *Top*, vol. 29, no. 1, pp. 5–33, 2021.

[33] C. Molnar, *Interpretable machine learning*. Lulu. com, 2020.

[34] A. Alipour, J. Yarahmadi, and M. Mahdavi, "Comparative study of m5 model tree and artificial neural network in estimating reference evapotranspiration using modis products," *Journal of Climatology*, vol. 2014, 2014.

[35] S. Gey and E. Nedelec, "Model selection for cart regression trees," *IEEE Transactions on Information Theory*, vol. 51, no. 2, pp. 658–670, 2005.

[36] D. K. (Feb. 2020). "Implementing regression using a decision tree and scikit-learn," [Online]. Available: https://heartbeat.fritz.ai/implementing-regression-using-a-decision-tree-and-scikit-learn-ac98552b43d7 (visited on 03/10/2021).

[37] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[38] A. Criminisi and J. Shotton, "Regression forests," in *Decision Forests for Computer Vision and Medical Image Analysis*, A. Criminisi and J. Shotton, Eds. Springer London, 2013, pp. 47–58, ISBN: 978-1-4471-4929-3. DOI: 10.1007/978-1-4471-4929-3_5. [Online]. Available: https://doi.org/10.1007/978-1-4471-4929-3_5.

[39] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *ICML*, 1996.

[40] Z. Zhang. (Jun. 2019). "Boosting algorithms explained," [Online]. Available: `https://towardsdatascience.com/boosting-algorithms-explained-d38f56ef3f30` (visited on 04/26/2021).

[41] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[42] K. Fukami, K. Fukagata, and K. Taira, "Assessment of supervised machine learning methods for fluid flows," *Theoretical and Computational Fluid Dynamics*, vol. 34, no. 4, pp. 497–519, 2020.

[43] J. D. Rodriguez, A. Perez, and J. A. Lozano, "Sensitivity analysis of k-fold cross validation in prediction error estimation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 569–575, 2009.

[44] T.-T. Wong and P.-Y. Yeh, "Reliable accuracy estimates from k-fold cross validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 2019.

[45] Emre. (Mar. 2016). "Linear regression with non-symmetric cost function," [Online]. Available: `https://datascience.stackexchange.com/questions/10471/linear-regression-with-non-symmetric-cost-function` (visited on 04/01/2021).

[46] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[47] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[48] D. G. Krige, "A statistical approach to some basic mine valuation problems on the witwatersrand," *Journal of the Southern African Institute of Mining and Metallurgy*, vol. 52, no. 6, pp. 119–139, 1951.

[49] A. Agnihotri and N. Batra, "Exploring bayesian optimization," *Distill*, 2020, https://distill.pub/2020/bayesian-optimization. DOI: `10.23915/distill.00026`.

[50] J. Wang, S. C. Clark, E. Liu, and P. I. Frazier, "Parallel bayesian global optimization of expensive functions," *arXiv preprint arXiv:1602.05149*, 2016.

[51] P. Thongsanit and T. Mookum, "Numerical study of trailing edge cooling in a gas turbine blade," 2018.

[52] 2013. [Online]. Available: `https://web.kpi.kharkov.ua/gt/wp-content/uploads/sites/56/2018/07/new-14.gif`.

[53] R. Li and A. Sudjianto, "Analysis of computer experiments using penalized likelihood in gaussian kriging models," *Technometrics*, vol. 47, no. 2, pp. 111–120, 2005.

[54] "Q-q plot (quantile to quantile plot)," in *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 437–439, ISBN: 978-0-387-32833-1. DOI: `10.1007/978-0-387-32833-1_331`. [Online]. Available: `https://doi.org/10.1007/978-0-387-32833-1_331`.

[55] M. B. Wilk and R. Gnanadesikan, "Probability plotting methods for the analysis for the analysis of data," *Biometrika*, vol. 55, no. 1, pp. 1–17, 1968.

[56] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[57] V. Mahajan, A. K. Jain, and M. Bergier, "Parameter estimation in marketing models in the presence of multicollinearity: An application of ridge regression," *Journal of Marketing Research*, vol. 14, no. 4, pp. 586–591, 1977.

[58] R. D. De Veaux and L. H. Ungar, "Multicollinearity: A tale of two nonparametric regressions," in *Selecting models from data*, Springer, 1994, pp. 393–402.

[59] B. Mac Namee, P. Cunningham, S. Byrne, and O. I. Corrigan, "The problem of bias in training data in regression problems in medical decision support," *Artificial intelligence in medicine*, vol. 24, no. 1, pp. 51–70, 2002.

[60] S. S. Garud, I. A. Karimi, and M. Kraft, "Smart adaptive sampling for surrogate modelling," in *26th European Symposium on Computer Aided Process Engineering*, ser. Computer Aided Chemical Engineering, Z. Kravanja and M. Bogataj, Eds., vol. 38, Elsevier, 2016, pp. 631–636. DOI: `https://doi.org/10.1016/B978-0-444-63428-3.50110-7`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9780444634283501107`.

[61] J. Eason and S. Cremaschi, "Adaptive sequential sampling for surrogate model generation with artificial neural networks," *Computers & Chemical Engineering*, vol. 68, pp. 220–232, 2014.

[62] Y. Jin, J. Li, W. Du, and F. Qian, "Adaptive sampling for surrogate modelling with artificial neural network and its application in an industrial cracking furnace," *The Canadian Journal of Chemical Engineering*, vol. 94, no. 2, pp. 262–272, 2016.

[63] W. H. Lopez Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Chapter 11 - autoencoders," in *Machine Learning*, A. Mechelli and S. Vieira, Eds., Academic Press, 2020, pp. 193–208, ISBN: 978-0-12-815739-8. DOI: `https://doi.org/10.1016/B978-0-12-815739-8.00011-0`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B97801281573%2098000110`.

[64] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[65] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[66] S. Pawar, O. San, B. Aksoylu, A. Rasheed, and T. Kvamsdal, "Physics guided machine learning using simplified theories," *Physics of Fluids*, vol. 33, no. 1, p. 011 701, 2021.

[67] A. R. d. Silva, A. P. A. Silva, and L. J. Tiago, "A new local stochastic method for predicting data with spatial heterogeneity," *Acta Scientiarum. Agronomy*, vol. 43, 2020.

[68] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International Conference on Machine Learning*, PMLR, 2020, pp. 8459–8468.

[69] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, "Theory-guided data science: A new paradigm for scientific discovery from data," *IEEE Transactions on knowledge and data engineering*, vol. 29, no. 10, pp. 2318–2331, 2017.

[70] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, "Integrating physics-based modeling with machine learning: A survey," *arXiv preprint arXiv:2003.04919*, 2020.

[71] X. Jia, J. Willard, A. Karpatne, J. S. Read, J. A. Zwart, M. Steinbach, and V. Kumar, "Physics-guided machine learning for scientific discovery: An application in simulating lake temperature profiles," *ACM/IMS Transactions on Data Science*, vol. 2, no. 3, pp. 1–26, 2021.