MACHINE LEARNING BASED AVERAGE PRESSURE COEFFICIENT PREDICTION FOR ISOLATED HIGH-RISE BUILDINGS

SHIVA SAI REDDY MARAPAKALA

Bachelor of Technology in Mechanical Engineering

Manipal Academy of Higher Education

June 2020

Submitted in partial fulfillment of requirements for the degree

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

August 2023

We hereby approve this Thesis for

SHIVA SAI REDDY MARAPAKALA

Candidate for the Master of Science degree

for the Department of Mechanical Engineering

And

CLEVELAND STATE UNIVERSITY'S

College of Graduate Studies by

Committee Chairperson, Prof. Dr. Navid Goudarzi

Department & Date

Committee Member, Prof. Dr. Mustafa Usta

Department & Date

Committee Member, Prof. Dr. Prabaha Sikder

Department & Date

Student's Date of Defence: August 15, 2023

ACKNOWLEDGMENT

I am profoundly grateful to all those who have supported me throughout this journey, enabling me to complete this master's thesis. I extend my sincere gratitude to my parents for their unwavering support and encouragement. Your belief in me has been, and continues to be, my driving force.

I would like to thank my Thesis mentor, Dr. Navid Goudarzi for his guidance and expertise which were instrumental in shaping this work. I would also like to thank Dr. Mustafa Usta and Dr. Prabaha Sikder for their invaluable support as my committee members.

I also extend my appreciation to Parva Jain and Shivesh N Sharma whose valuable insights greatly enriched the quality of this work. Lastly, I am grateful to the supportive academic community at Cleveland State University for fostering a conducive learning environment, which significantly contributed to the development of this work.

MACHINE LEARNING BASED AVERAGE PRESSURE COEFFICIENT PREDICTION FOR ISOLATED HIGH-RISE BUILDINGS

SHIVA SAI REDDY MARAPAKALA

ABSTRACT

In structural design, the distribution of wind-induced pressure exerted on structures is crucial. The pressure distribution for a particular building is often determined by scale model tests in boundary layer wind tunnels (BLWTs). For all combinations of interesting building shapes and wind factors, experiments with BLWTs must be done. Resource or physical testing restrictions may limit the acquisition of needed data because this procedure might be time- and resource-intensive. Finding a trustworthy method to cyber-enhance data-collecting operations in BLWTs is therefore sought.

This research analyzes how machine learning approaches may improve traditional BLWT modeling to increase the information obtained from tests while proportionally lowering the work needed to complete them. The more general question centers on how a machine learning-enhanced method ultimately leads to approaches that learn as data are collected and subsequently optimize the execution of experiments to shorten the time needed to complete user-specified objectives. 3 Different Machine Learning models, namely, Support vector regressors, Gradient Boosting regressors, and Feed Forward Neural networks were used to predict the surface Averaged Mean pressure coefficients c_p on Isolated high-rise buildings.

The models were trained to predict average c_p for missing angles and also used to train for varying dimensions. Both global and local approaches to training the

models were used and compared. The Tokyo Polytechnic University's Aerodynamic Database for Isolated High-rise buildings was used to train all the models in this study. Local and global prediction approaches were used for the DNN and GBRT models and no considerable difference has been found between them. The DNN model showed the best accuracy with ($R^2 > 99\%$, MSE $< 1.5\%$) among the used models for both missing angles and missing dimensions, and the other 2 models also showed high accuracy with ($R^2 > 97\%$, MSE $< 4\%$).

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

The pressure distribution for a specific building is generally collected from scale-model experiments in a boundary layer wind tunnel (BLWT). Experiments in BLWTs must be conducted for all combinations of building shape and wind parameters of interest. This process can be time- and resource-intensive, and resource or physical testing constraints may prevent collecting desired data. Therefore, it is hoped to find a reliable approach to cyber-enhance data collection procedures in BLWTs.

This research analyzes how machine learning approaches may improve traditional BLWT modeling to increase the information obtained from tests while proportionally lowering the work needed to complete them by using ML models to predict local wind pressures c_p.

c_p prediction, as indicated by various research studies is a Non-linear problem. ML models like Neural Networks, Decision Trees, Support Vector Machines

(SVMs), Random Forests, Gradient Boosting, and K-nearest neighbors (KNN) can all capture Non-linear relationships. But this work focuses primarily on Gradient Boosting Regression Trees (GBRT), Support vector regression (SVR), and Deep neural network models and, uses high-rise buildings using wind tunnel test data for prediction.

ML models have previously been used for c_p predicting low-rise buildings. *Weng et al.*[32] used Gradient boosting decision trees to predict average and RMS c_p of non-isolated low-rise buildings. *Li et al.*[18] used four ML models including Ridge regression, Decision Trees, Random Forests, and GBRT; they determined that GBRT is the best-performing model. Neural Networks have also been used to predict c_p for low-rise buildings. *Tian et al.*[28] used Deep neural networks to predict mean and RMS c_p values for isolated low-rise gable roof buildings. *Huang et al.*[13] also used DNN to predict mean and RMS c_p values for low-rise buildings. Little research on c_p prediction for High-rise buildings and a lack of comparison between the best-performing models motivates this research.

1.2 Thesis Objectives

This thesis demonstrates the development and applicability of ML models to numerically predict average wind-induced pressure coefficients for a high-rise building model using BLTW test data. Three ML models are trained and tested. Prediction of average c_p values are done under these conditions:

a) Use input data from various wind angles for a given test model to predict c_p values for an unseen wind angle for that test model.

b) Use input data from various building heights for a given incident angle to predict c_p values for an unseen building Height for that wind angle.

This work also explores two approaches for prediction, local and global bases, and determines the best approach for prediction accuracy.

1.3 Thesis Outline

1. **Chapter II** presents a theoretical review of the ML models used in the study.

2. **Chapter III** presets the Dataset used, ML model implementation, and results of predictions using the developed models.

3. **Chapter IV** presents the conclusions found in the present study and recommendations for future work.

4. **Appedices A, B and C** The Python code of the developed ML models.

CHAPTER II

MACHINE LEARNING

Chapter II reviews the basics and mathematical foundation of the Machine learning models used in this thesis.

2.1 Introduction

Machine learning, as a broad subset of artificial intelligence, is concerned with algorithms and techniques that enable computers to learn and improve their performance on a specific task without being explicitly programmed. In other words, instead of following predefined rules, ML models learn from data and experiences to make predictions, recognize patterns, or perform other tasks[1]. The advent of the information age along with the availability of big data has paved the way for new and innovative ML applications in various fields like Physics (*Karniadakis et al.[16]*), Economics (*Athey et al.[2]*), Medicine (*Rajkomar et al.[23]*), and Biology (*Jones et al.[15]*), and Engineering domains like Fluid Mechanics (*Brunton et al. [4]*), Energy (*Forootan et al.[8]*), Manufacturing (*Razvi et al.[24]*), etc. ML approaches are divided into supervised learning, unsupervised learning, and re-

inforcement learning. This MSc thesis primarily focuses on building supervised learning models.

2.1.1 Supervised Learning

In supervised learning, the algorithm is trained on a labeled dataset, where each input is associated with a corresponding target or output label. The goal is to learn a mapping from inputs to outputs so that the model can make accurate predictions on new, unseen data. Figure 2.1 shows the general supervised learning paradigm. Examples of supervised learning models are Linear Regression, Logistic Regression, Support Vector Machines (SVM), Decision Trees, Random Forest, K-Nearest Neighbors (KNN), Naive Bayes, Neural Networks, Gradient Boosting Machines (GBM), and Long Short-Term Memory (LSTM)[17]. In the present thesis, three supervised learning models were used for c_p prediction on high-rise buildings, Neural networks (Section 2.2), Support vector regressors (Section 2.3), and Tree-Based Gradient boosted regressors (Section 2.4).

2.2 Neural Networks

Deep Learning is a subset of machine learning that involves training deep neural networks with multiple layers. Deep learning has revolutionized various fields, especially computer vision, natural language processing, and speech recognition. The cornerstone of deep learning is the neural network. The name neural network was derived initially from thinking of these hidden units as analogous to neurons in the brain — values of the activations $A_k = h_k(X)$ close to one are firing, while those close to zero are silent (using the sigmoid activation function).[12]

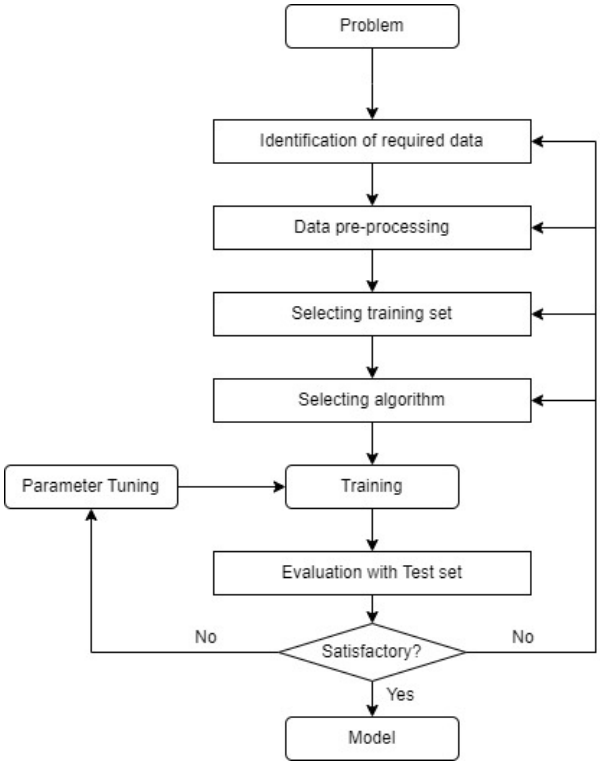


Figure 2.1: Supervised Machine Learning Process [17]

2.2.1 Single Layer Neural Network

A neural network takes an input vector of p variables $X = (X_1, X_2, ..., X_p)$ and builds a nonlinear function $f(X)$ to predict the response Y. It is also a nonlinear prediction model like trees, boosting, and generalized additive models. What distinguishes neural networks from other methods is the particular structure of the model.

Figure 2.2 shows a simple feed-forward neural network for modeling a quantitative response using $p = 4$ predictors. The four features $X_1, ..., X_4$ make up the units in the input layer. The arrows indicate that each of the inputs from the input layer feeds into each of the K hidden units. The neural network model is

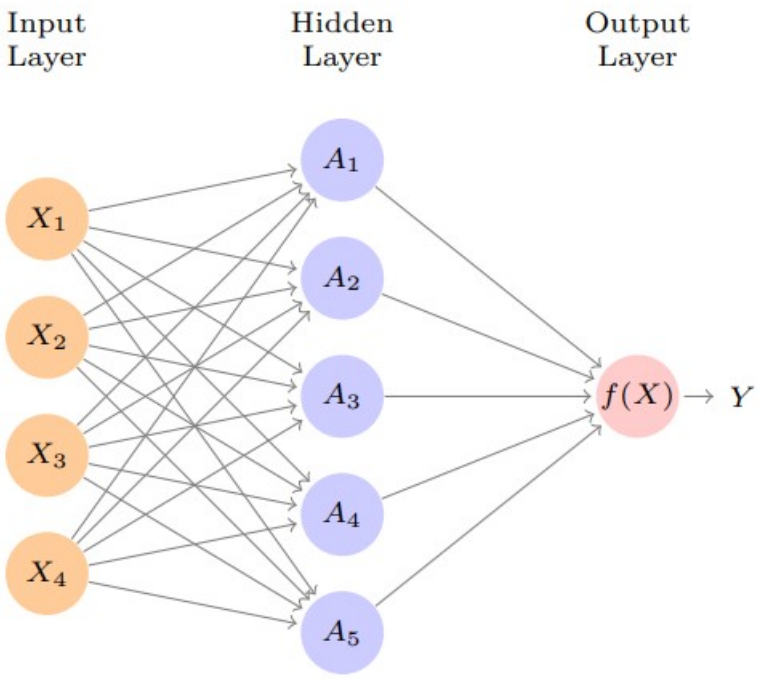


Figure 2.2: A Neural Network with a single hidden layer

given by

$$\begin{aligned} f(X) &= \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) \\ &= \beta_0 + \beta_0 g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right) \end{aligned} \tag{2.1}$$

Where β are constants and w are the weights at each node. It is built up here in two steps. First the K activations $A_k, k = 1, ..., K$, in the hidden layer are computed as functions of the input features $X1, ..., Xp$,

$$A_k = h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right) \tag{2.2}$$

where $g(z)$ is a nonlinear activation function that is specified in advance. Each A_k function can be thought of as a different transformation $h_k(X)$ of the original features. These K activations from the hidden layer then feed into the output

layer, resulting in

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k \tag{2.3}$$

In the early instances of neural networks, the sigmoid activation function was favored which is given by

$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}} \tag{2.4}$$

which is the same function used to convert a linear function into probabilities between zero and one. The preferred choice in modern neural networks is the ReLU (rectified linear unit) activation function, which takes the form

$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{else} \end{cases} \tag{2.5}$$

A ReLU activation can be computed and stored more efficiently than a sigmoid activation The nonlinearity in the activation function g(·) is essential, since without it the model f(X) in (10.1) would collapse into a simple linear model in X1,...,Xp. Moreover, having a nonlinear activation function allows the model to capture complex nonlinearities and interaction effects.[14]

2.2.2 Multilayer Neural Networks

Figure 2.3 shows an example of a Multi-layer Neural network that frequently includes multiple hidden layers with numerous units per layer. Theoretically, most functions can be approximated by a single hidden layer with a lot of units. However, many layers, each of modest size, greatly increase the efficiency of learning and finding a viable answer.

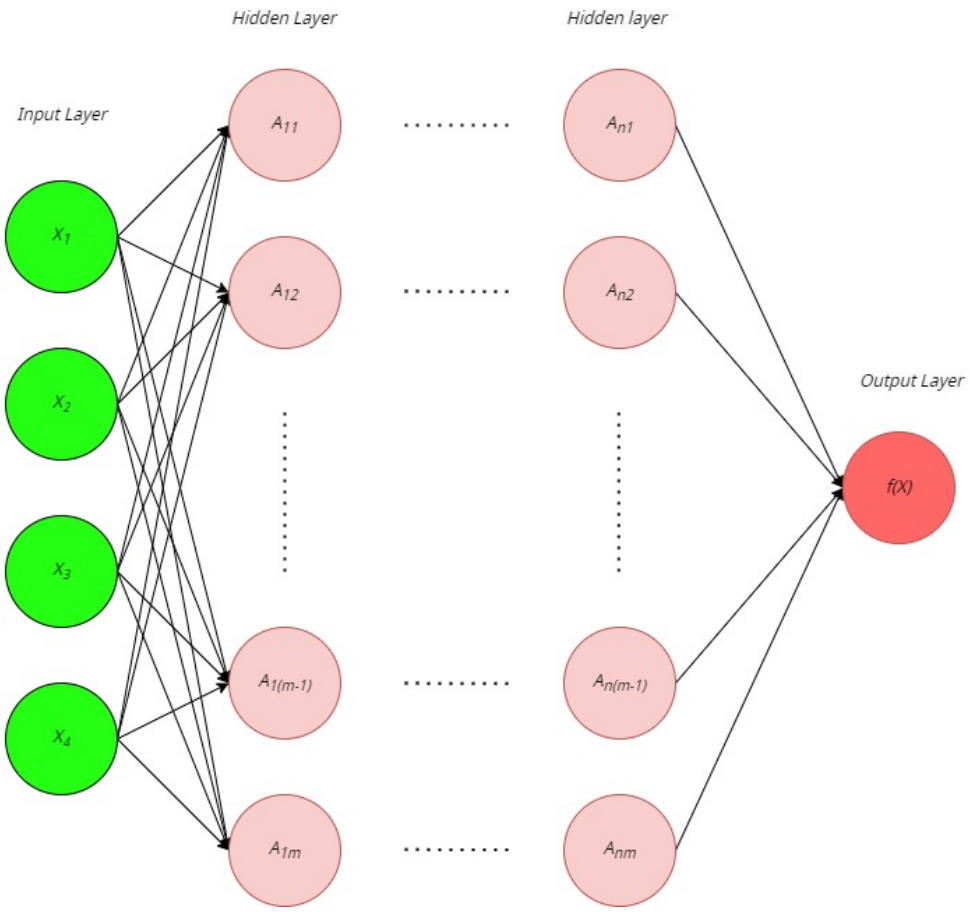


Figure 2.3: A neural network with multiple hidden layers

In multilayer Neural Networks each layer can have its own activation function or all layers can have the same activation functions. As discussed earlier in section 2.2.1 activation functions introduce non-linearity to the model, allowing it to learn complex patterns and relationships in data. They achieve this by determining the output of a node or a neuron in a neural network based on the weighted sum of its inputs. Below are some common activation functions used to train neural networks.

RELU: One of the most prevalent activation functions is the Rectified Linear Unit (ReLU) (bottom left in fig 2.4). If the input is positive it returns the value; else, it returns zero. ReLU is effective in terms of computing and aids in addressing

the vanishing gradient issue. It is given by

$$f(x) = max(0, x) \tag{2.6}$$

Sigmoid Inputs are translated by the sigmoid activation function into values between 0 and 1 (top left in fig 2.4). It is helpful when modeling probabilities in binary classification tasks or other situations. Sharp damp gradients during backpropagation from deeper hidden layers to inputs, gradient saturation, and sluggish convergence are some downsides of this activation. It is given by eqn 2.4.

Tanh The tanh function translates inputs to values between -1 and 1, much like the sigmoid function does (top right in fig 2.4). Additionally, it is helpful in situations where the data is ambiguous or contradictory. Tanh function became more popular than the sigmoid function historically because it performed better for multi-layer neural networks. However, it did not resolve the sigmoids' vanishing gradient issue. It is given by

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.7}$$

Leaky ReLU Leaky ReLU is a ReLU variant that permits a small gradient while the unit is not in use (bottom right in fig 2.4). By doing this, the "dying ReLU" problem, where a unit never activates, may be avoided. It is given by

$$f(x) = \begin{cases} x & \text{if } z < 0 \\ scale * x & \text{else} \end{cases} \tag{2.8}$$

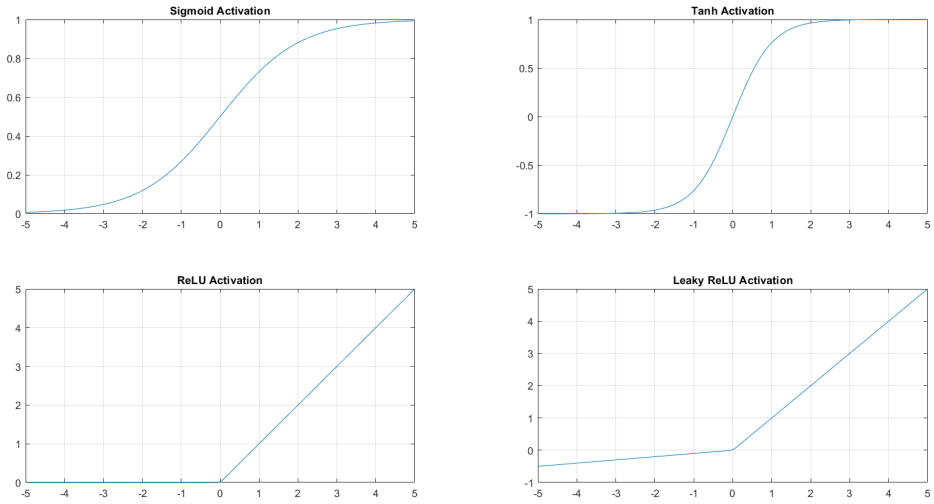


Figure 2.4: Activation functions

ELU Exponential Linear Unit (fig 2.5) is a type of linear unit that is similar to ReLU but has a smooth curve for negative values, which can aid in faster convergence and lessen the issue of vanishing gradients. It is given by

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \geq 0 \\ x & \text{else} \end{cases} \tag{2.9}$$

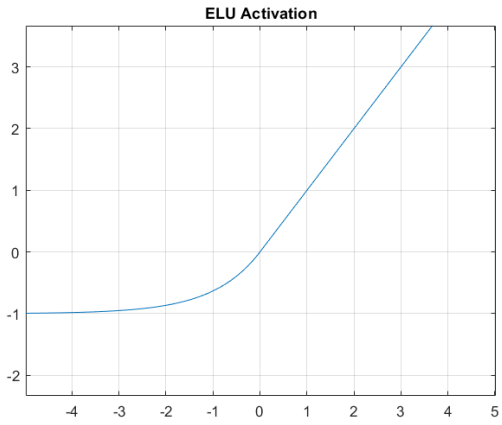


Figure 2.5: The Elu activation function

2.3 Support Vector Machines

Support vector machines are a type of supervised learning technique that is used for classification and regression. The optimum separation principle, which states that a successful classifier seeks the greatest gap between data points, serves as the foundation for prediction in SVMs.

2.3.1 Maximum Margin Classifiers

A hyperplane is a flat affine subspace with dimensions $p-1$ in a p -dimensional space. In p-dimensions and given parameters $(\beta_0, \beta_1, ...\beta_p)$ it is given by:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0 \tag{2.10}$$

a classification can be made for a point not on the hyperplane by checking the sign of LHS of eq 2.10.

A test observation is assigned a class based on which side of the separating hyperplane it is positioned on if such a hyperplane exists. This is a fairly natural classifier. Generally speaking, an endless number of these hyperplanes will exist if our data can be properly segregated by utilizing them.

The separating hyperplane that is furthest from the training observations is the maximal margin hyperplane, sometimes referred to as the best-separating hyperplane, and it is a logical option. In other words, we may calculate the (perpendicular) distance between each training observation and a specified separating hyperplane; the margin is the lowest such distance, which is the minimum distance between the observations and the hyperplane. The dividing hyperplane for which the margin is the maximum is known as the maximal margin hyperplane.

For a given data set with class labels $y_1, ..., y_n \in -1, 1$ and a set of n training observations $x_1, ..., x_n \in R^p$.

$$\text{maximize}\{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p, M\} \ \mathbf{M} \tag{2.11}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{2.12}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + +\beta_p x_{ip}) \tag{2.13}$$

This optimization problem's solution is the maximal margin hyperplane. A limitation of the maximal margin hyperplane is that it will inevitably categorize every training observation exactly, which may cause sensitivity to certain data. i.e., a single observation causes the maximal margin hyperplane to drastically alter, which is not always ideal.

To address this a *support vector classifier* is used. A support vector classifier accurately divides the majority of training data into two classes; nevertheless, a small number of observations may be misclassified and is given by the optimization problem

$$\text{maximize}\{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p, M\} \ \mathbf{M} \tag{2.14}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{2.15}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + +\beta_p x_{ip}) \geq M(1 - \varepsilon_i), \tag{2.16}$$

$$\varepsilon_i \geq 0, \sum_{i=1}^{n} \geq C, \tag{2.17}$$

where the tuning parameter C is non-negative. The width of the margin, denoted by M, is the value we aim to maximize. Slack variables $\epsilon_1, ..., \epsilon_n$ allow individual

observations to lie outside the hyperplane or edge.

2.3.2 Support Vector Machines

The support vector machine (SVM) is the advancement of the support vector classifier that emerges when the feature space is expanded using kernels in certain ways. It is based on the fact that the solution to equations 2.11 - 2.13 relies only on the inner products of the observations. The inner product of 2 observations $x_i, x_i^{'}$ is given by

$$\langle x_i, x_i^{'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i^{'}j} \tag{2.18}$$

A linear support vector classifier can be shown as :

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha \langle x, x_i \rangle \tag{2.19}$$

where $\alpha_i, i = 1, .., n$ are n parameters and only the support vectors in the solution have α_i that is nonzero. Then the solution is of the form

$$f(x) = \beta_0 + \sum_{i \in S} \alpha \langle x, x_i \rangle \tag{2.20}$$

where S is the collection of support points. Now if we replace the inner product in eqn 2.18 with a generalization $K(x, x_i^{'})$ where K is a function generally referred to as a *kernel*. The kernel function K estimates the similarity of two data points. For example, if we take

$$K(x, x_i^{'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i^{'}j} \right)^d \tag{2.21}$$

this is the polynomial kernel of degree d and is an example of a non-linear kernel. Non-linear kernels greatly improve the flexibility of the decision boundary. Another popular choice of K is the radial kernel given by

$$K(x, x_i^{'}) = e^{\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i^{'}j})^2\right)} \tag{2.22}$$

where γ is a positive constant. When a support vector classifier is combined with a non-linear kernel such as the polynomial or radial kernels it is known as a support vector machine(SVM). [14]

2.3.3 Support Vector Regression

For a set of data including predictor variables and observed response values, the general goal of linear support vector regression is to find a function $f(x)$ that deviates from y_n by a value no greater than ϵ for each of the training points. If x_n and y_n are the observed and response values respectively, then the goal is to find the linear function $f(x) = x^{'}\beta + b$ with a norm value $(\beta^{'}\beta)$ to be minimal. To minimize, this is expressed as a convex optimization problem

$$J(\beta) = \frac{1}{2}(\beta^{'}\beta) \qquad \text{subject to} \qquad \forall n : |y_n - (x_n^{'}\beta + b)| \leq \epsilon \tag{2.23}$$

for cases where no such function exists, slack variables ξ and ξ^* are introduced for each point. Slack variables allow the existence of regression errors and yet satisfy the other conditions. Including the slack variables the problem becomes [20]

$$J(\beta) = \frac{1}{2}(\beta^{'}\beta) + C\sum_{n=1}^{N}(\xi + \xi^*) \tag{2.24}$$

subject to the conditions :

$$\forall n : y_n - (x_n^{'}\beta + b) \leq \epsilon + \xi_n$$

$$\forall n : (x_n^{'}\beta + b) - y_n \leq \epsilon + \xi_n^*$$

$$\forall n : \xi_n^* \geq 0$$

$$\forall n : \xi_n \geq 0$$

where the constant C is a box constraint, a penalty control for the observations that lie outside the epsilon margin ϵ. C helps in regularization and prevents overfitting by determining the tradeoff between the slimness of $f(x)$ and the tolerance of deviations beyond ϵ.

the loss function ignores errors within the ϵ distance and is written as

$$L_\epsilon = \begin{cases} 0 & \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & \text{otherwise} \end{cases} \tag{2.25}$$

2.4 Regression Trees and Gradient Boosting

In tree-based approaches, the feature space is divided into several rectangles, and a straightforward model—such as a constant—is then fitted into each rectangle. They are a powerful prediction tool despite their conceptual simplicity. Here we discuss, in brief, the working of a regression tree algorithm. Consider our data consists of p inputs and a response, for each of N observations.

$$(x_i, y_i) \text{ for } i = 1, 2, .., N, \text{ with } x_i = (x_{i1}, x_{i2}...x_{ip}) \tag{2.26}$$

A tree-based algorithm needs to automatically decide on the splitting variables and split points, and also what topology the tree should have. Suppose first that we have a partition into M regions $R_1, R_2, .., R_M$, and we model the response as a constant $\hat{c}_m$ in each region given by :

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{2.27}$$

taking the minimization criterion as the sum of squares $\sum(y_i - f(x_i))^2$ it can be seen that the $\hat{c}_m$ that is the best is just the average of y_i in region R_m:

$$\hat{c}_m = ave(y_i | x_i \in R_m) \tag{2.28}$$

due to the computational infeasibility of finding the best binary partition in terms of the minimum sum of squares, an alternate approach is necessary. Starting with the entire data, we consider a splitting variable j and split point s and define the

pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\} \tag{2.29}$$

then we find the splitting variable j and split point s that solve

$$\min_{j,\,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right] \tag{2.30}$$

for a given choice j and s , the minimization inside can be solved by

$$\hat{c}_1 = ave(y_i|x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = ave(y_i|x_i \in R_2(j,s)) \tag{2.31}$$

The best pair (j, s) may be determined by scanning over all of the inputs because it is possible to quickly determine the split point s for each splitting variable. The data is divided into the two resulting regions after the optimal split has been determined, and the splitting process is then repeated on each of the two regions. Then, this procedure is done on every region that results. This approach is more commonly known as the "greedy" algorithm. A very large tree might overfit the data, while a small tree might miss the key structure. The appropriate tree size should be determined adaptively from the data since tree size is a tuning parameter that controls the complexity of the model.

2.4.1 Boosting

Boosting is a technique that creates a powerful "committee" by combining the results of several "weak" classifiers. A brief discussion of the popular boosting algorithm by Freund and Schapire called the "AdaBoost.M1." is presented here.

Consider a two-class problem whose output variable is written as $Y \in \{-1, 1\}$. A classifier $G(X)$ generates a prediction using one of the two values $\{-1, 1\}$ given a vector of predictor variables X. The training sample's error rate is

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)) \tag{2.32}$$

where $E_{XY} I(Y \neq G(X))$ is the anticipated error rate for consequent predictions. When a classifier's error rate is only somewhat lower than that of random guessing, it is regarded as a weak classifier. With the use of boosting, a series of weak classifiers $G_m(x), m = 1, 2, .., M$ are created by progressively applying the weak classification method to variously modified versions of the data. The resulting predictions are then combined through a weighted majority vote for the final prediction given by

$$G(x) = \textbf{sign} \left(\sum_{m=1}^{M} \alpha_m G_m(x) \right) \tag{2.33}$$

$\alpha_1, \alpha_2, ..., \alpha_M$ here are computed by the boosting algorithm, and weight the contribution of each respective $G_m(x)$. This results in more accurate classifiers in the sequence having higher influence. At each boosting step, the data are modified by applying weights $(w_1, w_2, ..., w_N)$ to each of the training observations (x_i, y_i), where $i = 1, 2, ..., N$. To simply train the classifier on the data in the first stage, all of the weights are initially set to $w_i = 1/N$. The weights of the individual observations are changed for each subsequent iteration, where $m = 2, 3, ..., M$, and the classification method is then repeated to the weighted data. At step m, the weights of the observations that were incorrectly categorized by the classifier $G_{m-1}(x)$ produced at the previous step are raised, while the weights of the

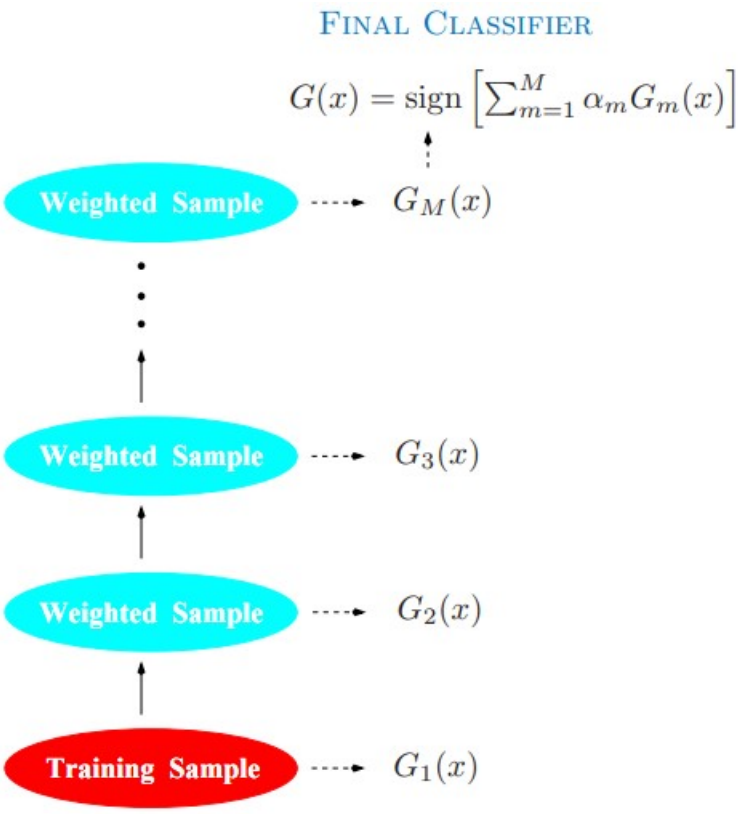


Figure 2.6: The AdaBoost algorithm[12]

observations that were successfully classified are dropped. Figure 2.6 shows the approach of the AdaBoost classification algorithm.

2.4.2 Gradient Boosting and its implementation

The generic gradient tree-boosting algorithm for regression as explained in [12] is shown here. We first initialize the single terminal node tree that represents the best constant model.

$$Initialize f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma) \tag{2.34}$$

where $L(y_i, \gamma)$ is a differentiable loss function. Common Loss functions and their Gradients are shown in table I

Table I. Common regression Loss Functions [12].

Application	***Loss function***	***Gradient***
Regression	$\frac{1}{2}\left[y_i - f(x_i)\right]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$sign[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i) for \lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m sign[y_i - f(x_i)] for \lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha th - quantile\{\lvert y_i - f(x_i)\rvert\}$

Next, calculate the generalized or pseudo residuals r :

For m = 1 to M:

For i = 1,2,...,N

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_m-1} \tag{2.35}$$

then we give terminal regions R_{jm},$j = 1, 2, ..., j_m$ to the targets r_{im} and Fit a regression tree and compute γ_{jm} for $j = 1, 2, ..., J_m$ given by

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma) \tag{2.36}$$

γ_{jm} is then used to update $f_m(x)$ in eqn 2.35 to

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{j_m} \gamma_{jm} I(x \in R_{jm}) \tag{2.37}$$

where I is the indicator function. This process is then repeated K times for each iteration m. Two basic tuning parameters for this algorithm are size $J_m, m = 1, 2, ..M$ of each constituent tree (Max. Depth) and number of iterations M(Number of Trees)

A visualization of a gbr algorithm is shown in figure 2.7. It can be seen that the prediction gets better with each iteration as more weak learners(Trees) are added.

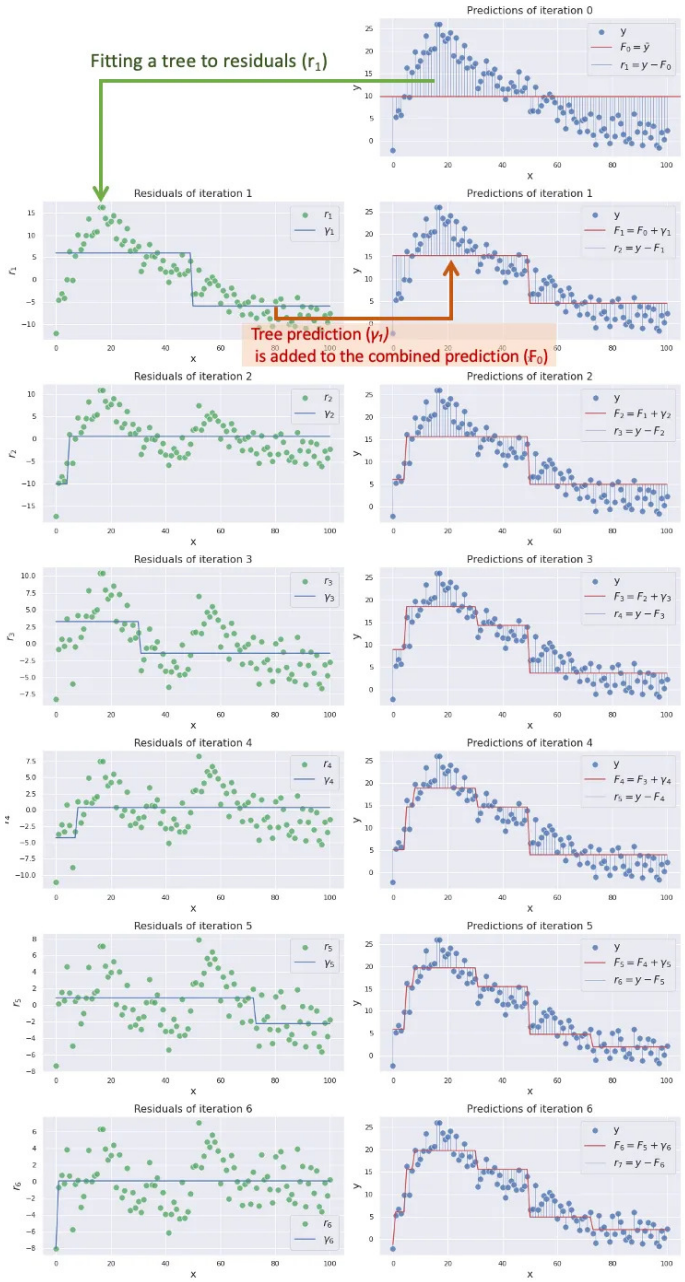


Figure 2.7: Schematic of a GBR algorithm achieving a complex prediction using multiple weak learners. [19]

CHAPTER III

Case Study: Isolated Building

3.1 The TPU Dataset

As a component of the wind effects on buildings and the urban environment, the 21st Century Center of Excellence Program, Japan, the Tokyo Polytechnic University (TPU) has created an aerodynamic database. This dataset consists of pressure coefficient time series data on various types of building models, i.e., Isolated High-Rise Buildings, Two Adjacent Tall Buildings, Isolated Low-Rise Buildings Without Eaves, Isolated Low-Rise Buildings With Eaves, and Non-isolated Low-Rise Buildings[30].

The high-rise building section of the aerodynamic database served as the basis for the dataset used in this study. Its goal is to deliver wind tunnel test data of wind loads on tall structures to structural design experts. There were 22 high-rise building models examined. On their webpage, graphs of area-averaged wind pressure coefficients on the wall surfaces, contours of local wind pressure coefficient statistics, and time series data of point wind pressure coefficient statistics for 394

test instances are hosted[31]. Figure 3.1 shows the BLTW used at TPU for their experiments.

The area-averaged wind pressure coefficient on wall surfaces, local wind pressures, and even the wind-induced dynamic responses of tall buildings can all be calculated using this data. The database was constructed in a boundary layer wind tunnel, 2.2m wide by 1.8m high. The model scale for the experiments was 1:400[29].

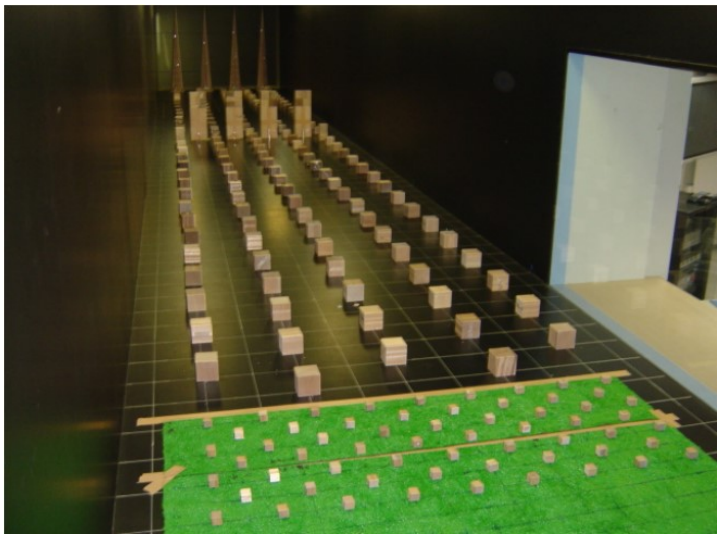


Figure 3.1: The Boundary Layer Wind tunnel at TPU [29]

The data files contain 13 unique breadth:depth:height configurations as shown in fig 3.3 and wind angles from $0 - 100°$ with $5°$ increments along with 2 different exposure factors $\alpha = 1/4 \ \& \ 1/6$. where α is the exponent assuming the velocity profile of the flow follows a power law given by :

$$\frac{u}{u_h} = \left(\frac{z}{z_h}\right)^{\alpha} \tag{3.1}$$

where u is the wind speed at height z, and u_h is the known wind speed at a reference height z_h. Figure 3.2 shows the comparison between the experimental

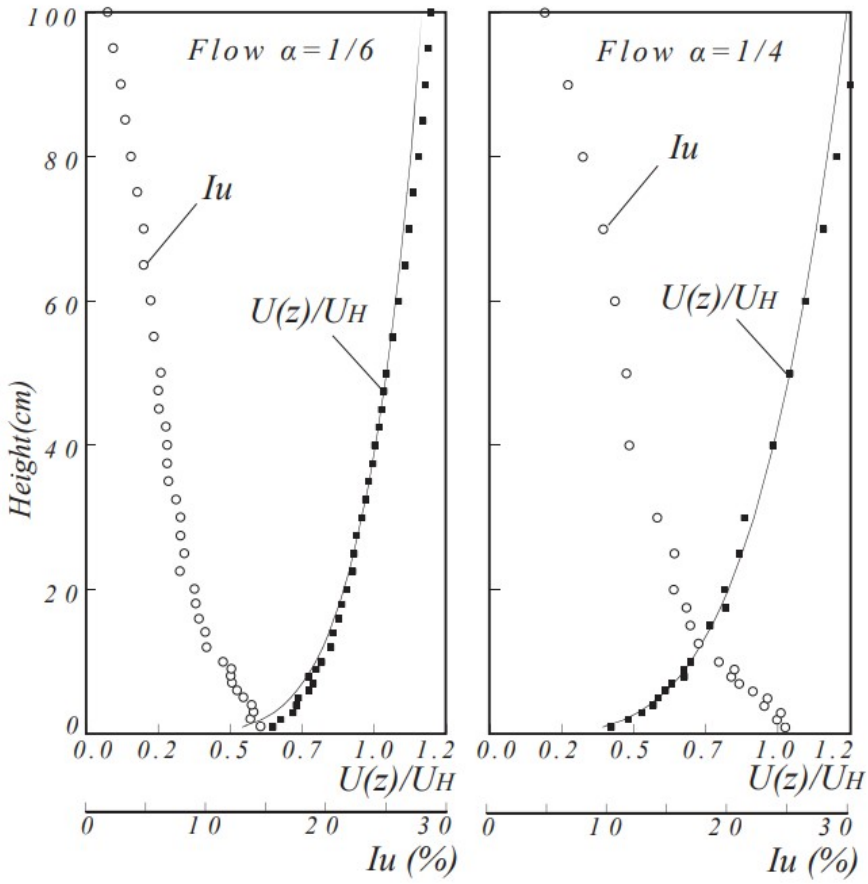


Figure 3.2: The Incoming velocity profile(u_z) and turbulence intensity profile(I_u) Used for the experiment [31]

velocity profile used at TPU and the power law profile.

These total 362 unique files each of which, again, contains the time series data of normalized c_p, x, y, and face number of the pressure taps (location of pressure taps), sample frequency, sample period, and reference velocity u_h. An example of the location of pressure taps hosted on the website is shown in fig 3.4 and the summary of the datafiles hosted on the website is shown in table II. It is worth noting that each of the wind angles available for each configuration in table II have different u_h but they vary by a minimal amount (+-.1m/s) from the value shown in the table.

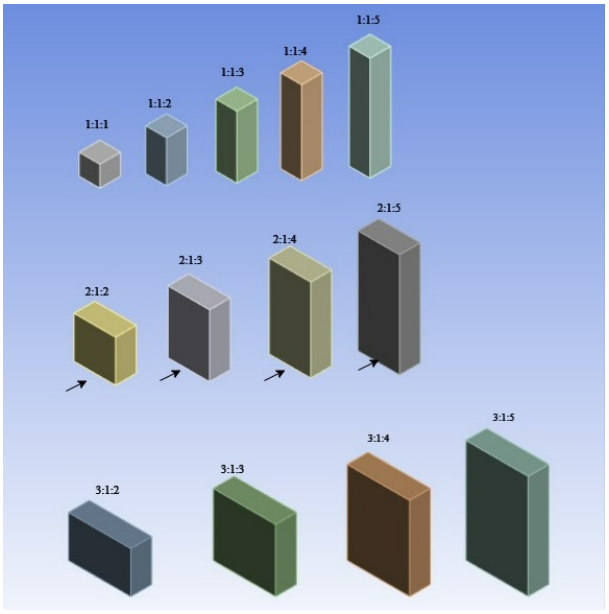


Figure 3.3: The Geometric configuration available for the Isolated high-rise buildings in the TPU Aerodynamic Dataset. The arrows indicate the 0-degree incident angle for all configurations.

Table II. Summary of the Data Hosted on the TPU Isolated High rise Building Database

No.	B:D:H	Wind angles Available	u_h	
			$\alpha = 1/4$	$\alpha = 1/6$
1	1:1:1	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50°	11.0	10.7
2	1:1:2	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50°	10.9	10.8
3	1:1:3	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50°	10.7	11.1
4	1:1:4	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50°	10.9	11.3
5	1:1:5	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50°	11.1	11.2
6	2:1:2	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	10.9	NA
7	2:1:3	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	10.7	NA
8	2:1:4	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	10.9	NA
9	2:1:5	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	11.0	NA
10	3:1:2	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	10.9	10.8
11	3:1:3	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	10.7	11.0
12	3:1:4	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	10.9	11.3
13	3:1:5	0°, 5°, 10°, 15°, 20°, 25°, 30°, 35°, 40°, 45°, 50° 55°, 60°, 65°, 70°75°, 80°, 85°, 90°, 95°, 100°	11.1	11.2

The pressure coefficient c_p is a dimensionless number used to describe the

relative pressure distribution in a flow field. It is generally defined as :

$$C_p = \frac{p - p_0}{\frac{1}{2}\rho_0 u_0^2} \tag{3.2}$$

where p is the static pressure at the point of interest p_0 is the free stream static pressure, ρ is free stream fluid density and u is the free stream velocity or the velocity of the body through the fluid whereas the pressure coefficient used in the TPU data is the "normalized" pressure coefficient defined as :

$$c_p = \frac{p}{q_{ref}} \tag{3.3}$$

where p is the static pressure at the point of interest and q_{ref} is the reference dynamic pressure measured at the reference height given by $1/2\rho_h^2$

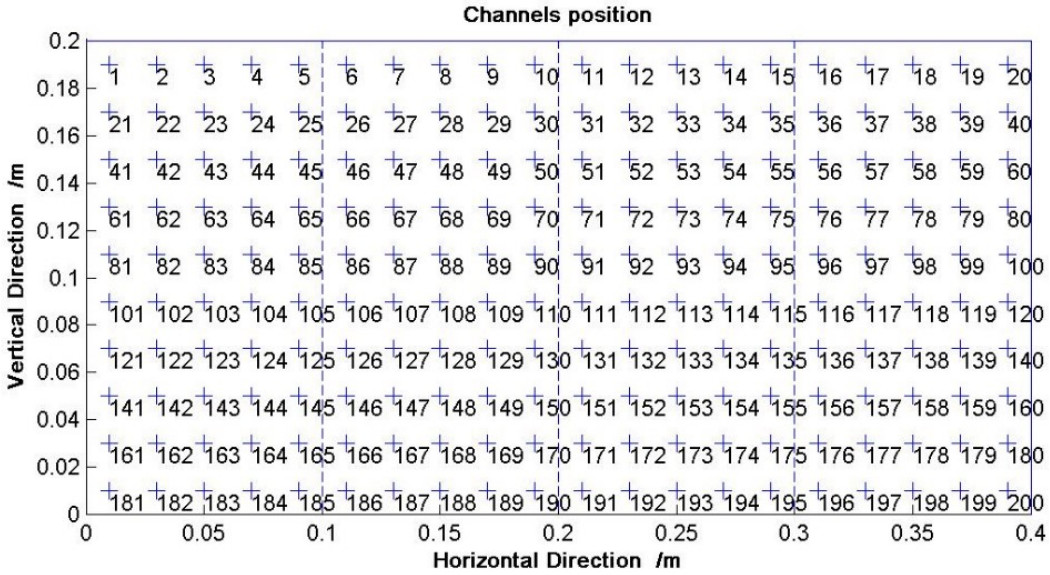


Figure 3.4: Pressure Tap locations for the configuration 1:1:2 [31]

3.2 Machine Learning Model Implementation

All the ML models used in this work were constructed on Python 3.9.15 [25] by using Python packages Numpy[11]and Pandas[27]. Furthermore, the sklearn.ensemble[26]

package was used for the gradient-boosting regression model and the Support Vector regressor model was constructed using the sklearn.svm[21] package. Both of these are from the scikit-learn[22] machine learning library for the Python programming language. It features various classification, regression, and clustering algorithms. The Neural Network was developed using the keras[6] package. All models were run on a computer with 64 bit-system of Win10, RAM 16GB, processor of 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz with a NVIDIA GeForce RTX 3060 Laptop GPU.

3.2.1 Hyperparameters

A hyperparameter in machine learning is a parameter whose value is utilized to regulate the learning process. Other factors, such as node weights, however, have values that are obtained by training.

Model hyperparameters, which refer to the model selection task and cannot be inferred while fitting the machine to the training set, and algorithm hyperparameters, which in theory have no bearing on the performance of the model but affect the speed and quality of learning, are two categories of hyperparameters.[33]

Table III. ML model Hyperparameters

Model	***parameter***	***Function***
SVR	kernel	Function Used to map the input data into a higher-dimensional space
	C	controls the trade-off between achieving a low training error and a low testing error
	γ	It defines the smoothness of the decision boundary. A small γ will create a broader decision region
GBR	N	also called n-estimators is the number of weak learners added to the ensemble
	ν	Learning rate which determines the contribution of each weak learner to the final prediction
	d	The maximum depth of each decision tree in the ensemble. It restricts the depth of individual trees to prevent overfitting.
DNN	L	Number of layers of Neurons between the input and the output (hidden layers)
	n	Number of neurons in each layer
	$f_a(x)$	Activation functions. Responsible for the non-linear behaviour exhibited by the Neural Networks
	η	The step size used in adjusting the weights of the network during training also called the learning rate
	m	The number of training examples used in each iteration of training also called the batch size
	N	The number of times the entire training dataset is passed through the network during training known as epochs
	optimizer	Algorithms that update the weights of the neural network during training based on the computed gradients

Hyperparameters vary based on the ML algorithm. Table III shows ML models and their common tuning parameters for learning. It should be noted that the parameters shown in table III are not the only parameters used for their respective models but the ones most used to tune the models.

Table IV. Hyperparameters Used for the NN model	
Hyperparameter/dataset	***Value***
Number of Hidden layers	3
Neurons per layer	64,64,16
Learning rate	0.01
Batch size	32
Training epoch	600
Loss Function	MSE
Activation Function	eLU
Optimizer	Adam
Input neurons	Global =x,y,z,α
	local = x,y,α
Output Neurons	Mean c_p
Training set	$\alpha = 0° - 100°$(Except 25,60)
	H = 1,2,4,5
Testing Set	$\alpha = 25°, 60°$
	H = 3

3.2.2 Parameter Optimization

Hyper-parameter Optimization (HPO) refers to the process of finding the best combination of hyperparameters for a machine learning model to achieve optimal performance on a given task. Due to the recent advent of deep learning networks, it has lately gained popularity. The Generalized approach for HPO includes choosing the hyper-parameters to tune them and their search space, changing them from coarse to fine, assessing how the model performs with various parameter sets, and choosing the best combination. The most common search algorithms employed for HPO are Grid search, Random Search, Bayesian Optimization and Tree Parzen Estimators [34]. In this work the Grid search algorithm is used for hyperparameter optimization for all the models.

Grid search is the most straightforward search algorithm. Its basic approach is to perform an exhaustive search on the set of hyperparameters specified by users. Let $n_1, n_2..n_m$ denote number of all possible values for hyperparameters $p^1, p^2...p^m$ then all the possible values of the parameters are $\{p_i^1\}_{i=1}^{n_1}$,$\{p_j^2\}_{j=1}^{n_2}$.....$\{p_k^m\}_{k=1}^{n_m}$ then

given an evaluation metric such as Mean square Error(MSE), the grid search algorithm computes MSE for all possible combinations of $\{p_i^1\, p_j^2 .... p_k^m\}$ and the best-performing combination is selected[32]. Table IV shows the parameters of the NN algorithms used in this work after they were selected by the grid search algorithm and Figure 3.5 shows the implementation of the ML models using the grid search algorithm.

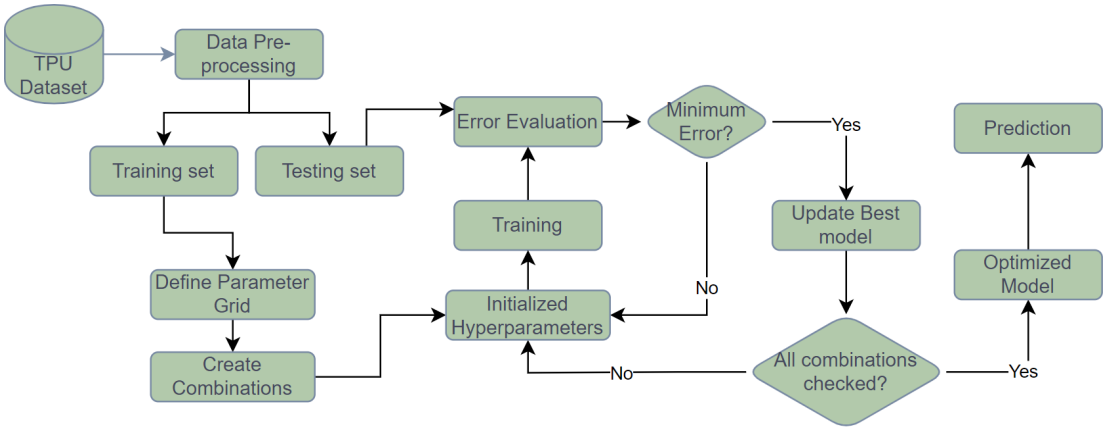


Figure 3.5: Implementation of ML models using grid search

3.2.3 Error metrics

Several error metrics are used to evaluate a given model's performance or to measure its prediction accuracy. These include Root-Mean-Square Error (RMSE), Mearn-Square Error (MSE) and correlation coefficients, R [5][35][9][10][7][3].

Mean square error (MSE) is defined as [5]:

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\frac{p_i - m_i}{m_i})} \tag{3.4}$$

where, N is the number of pressure sensors; p_i is the $i-th$ data in a predicted dataset; m_i is the $i-th$ data in a measured dataset.

Root mean square Error (RMSE) is defined as [9] :

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - m_i)^2} \tag{3.5}$$

where, N is the number of pressure sensors; p_i is the $i-th$ data in a predicted dataset; $m_i$is the $i-th$ data in a measured dataset.

Correlation Coefficient R is defined as [9]:

$$R = \frac{\sum_{i=1}^{N}((m_i - \overline{m}) \times (p_i - \overline{p}))}{\sqrt{\sum_{i=1}^{N}(m_i - \overline{m})^2} \times \sqrt{\sum_{i=1}^{N}(p_i - \overline{p})^2}} \tag{3.6}$$

Tian et.al. [28], pointed out that for the case of pressure distribution prediction, MSE and RMSE are unreliable approaches, and that the correlation coefficient is a much more reliable approach. However, this work presents both the correlation coefficient and the mean squared error as the error metrics for prediction accuracy.

3.2.4 Global vs Local Prediction

To predict the mean c_p at a point, two approaches are generally used. A prediction can be made for a point p on the surface based on the data from *(a) within the neighborhood* (Local hand-picked neighborhood of a few points up to the whole surface)[5][35][7] and *(b) From all the points on the building* (Global)[28]. The local approach will require training a series of models, only using the subset of local inputs(2 dimensions), whereas, in the Global Approach a single model uses all the tap locations on the building (3 dimensions). Figure 3.6 shows the Prediction basis for local and global predictions.

The Major Drawback for Local prediction is the information beyond the selected region is not utilized. Whereas Global Prediction might add unnecessary

parameters to the prediction. A comparison of local and global prediction for the 2:1:5 building is shown in table V.

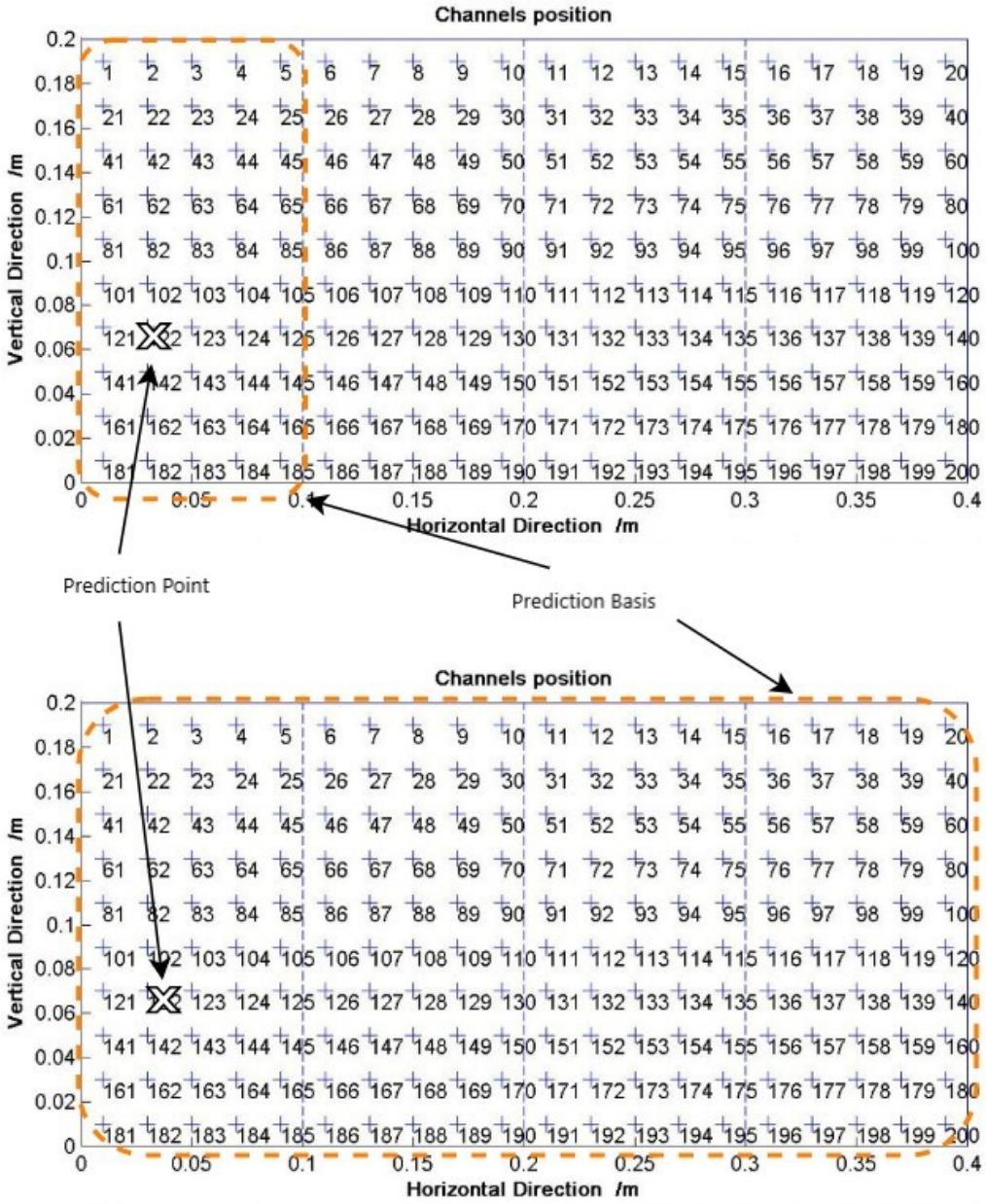
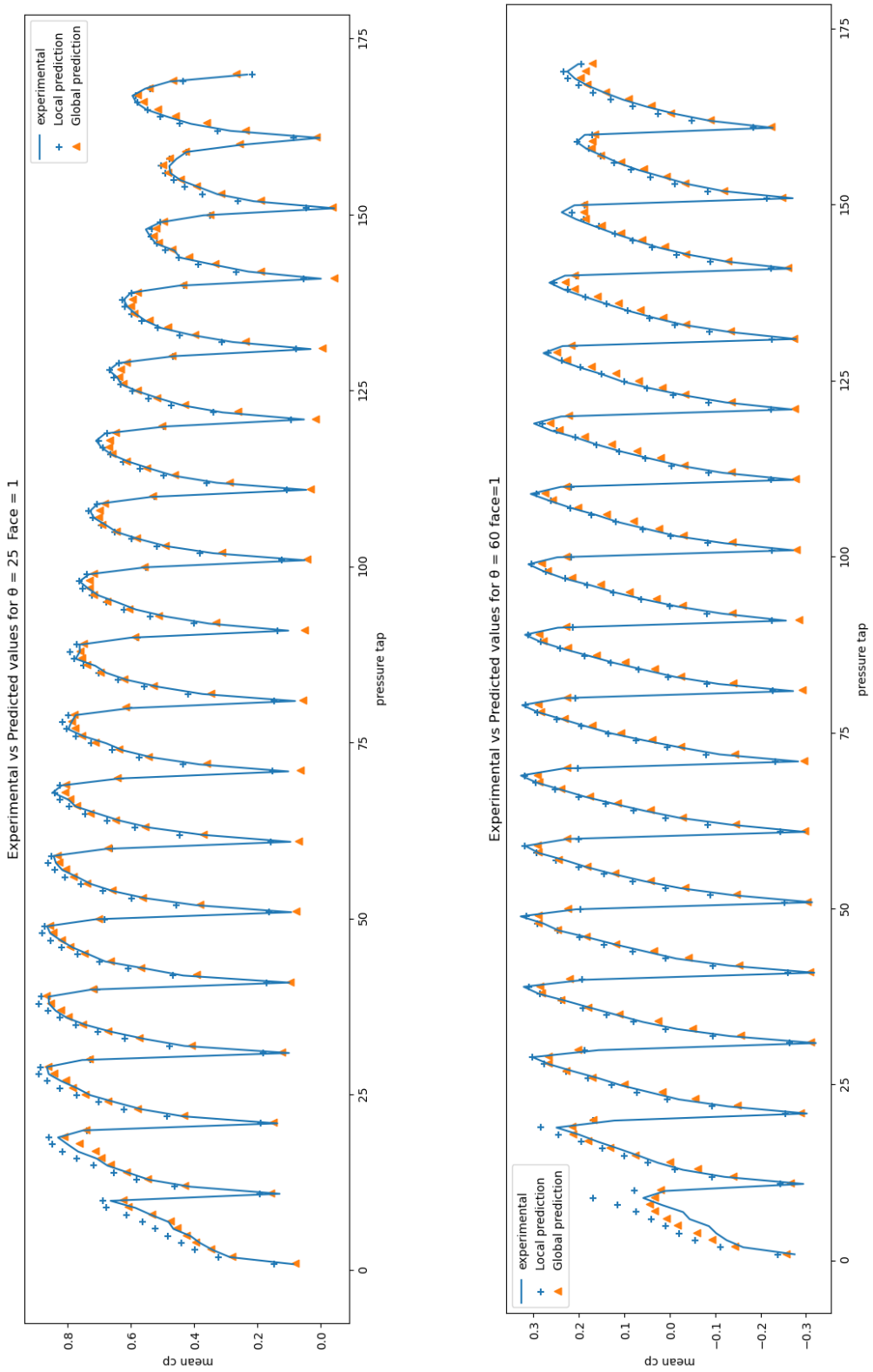


Figure 3.6: Prediction Basis for Local Prediction (Top) vs Global Prediction (bottom)

Table V. Local Vs Global Predictions comparison for 2:1:5 building		
	Local Prediction	***Global Prediction***
Input	x,y,z, wind direction α	x,y,wind direction α
Training Dataset	170 taps on surface 1	510 on whole building
	$\times$ 19 wind directions	$\times$ 19 wind directions
Test Dataset	170 taps on face 1	170 taps on face 1
	$\times$ 2 unseen wind directions	$\times$ 2 unseen wind directions
Prediction Accuracy	$R^2 = 0.9868$	$R^2 = 0.9951$
- DNN	$MSE = 0.00124$	$MSE = 0.00045$
Prediction Accuracy	$R^2 = 0.9774$	$R^2 = 0.9775$
- GBR	$MSE = 0.00335$	$MSE = 0.00334$

It can be seen in table V and Figure 3.7 that the Global approach for prediction has minor improvement in prediction over the local approach. Although this difference is not as much as reported by *Tian et. al* [28] who reported an R^2 value of .9995 and .9444 for global and local respectively. As for the other ML models, No significant difference was seen in Local v Global for the GBR model.

It is, however, beneficial to use a global approach because of its added benefit of requiring only one model for prediction on the whole building. In contrast, the local model requires one model for each surface that requires prediction for little to no improvement over the global approach. Therefore for the rest of this work, a global basis for prediction is used.



(a) $\alpha = 25°$ (b) $\alpha = 60°$

Figure 3.7: Local vs Global Prediction for NN for 2:1:5 building

3.3 Results

The ML models were used to predict mean c_p for (a) Unseen wind angles for a given building dimensions using the rest of the wind angles as the basis for predictions. and (b) Unseen Building height for a given angle and width using the rest of the data of the buildings as a basis for prediction.

3.3.1 Predicting for an Unseen Angle

To demonstrate the feasibility to use ML models to predict an unseen wind angle, 2 wind angles $\alpha = 25°$ and $\alpha = 60°$ were removed from the training data set and the models were trained for the rest of the wind angles. Table shows the results of the global prediction for the DNN models for the 2:1:5 configuration building. The details of the test and train datasets used are shown in table

Figure 3.8 shows the correlation coefficients obtained after training for each of the models globally and figures 3.9a and 3.9b shows the Truth vs Predicted for all models on all points of the 2:1:5 building.

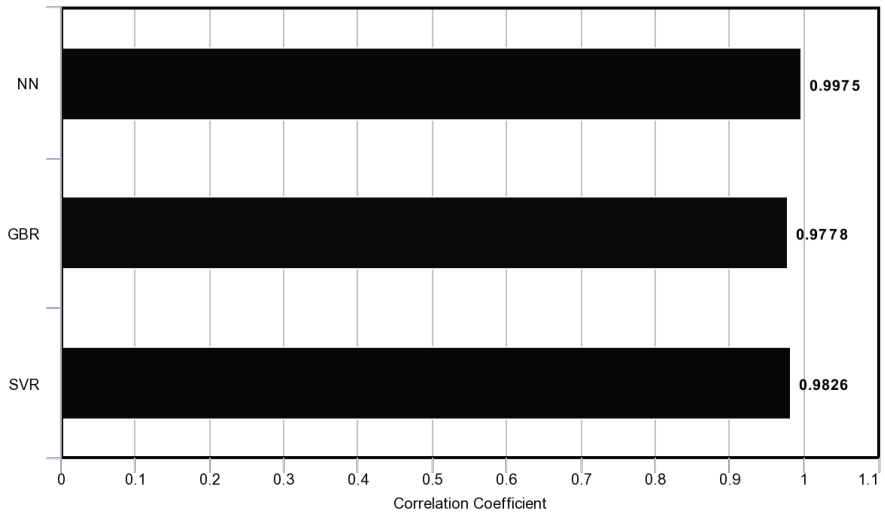
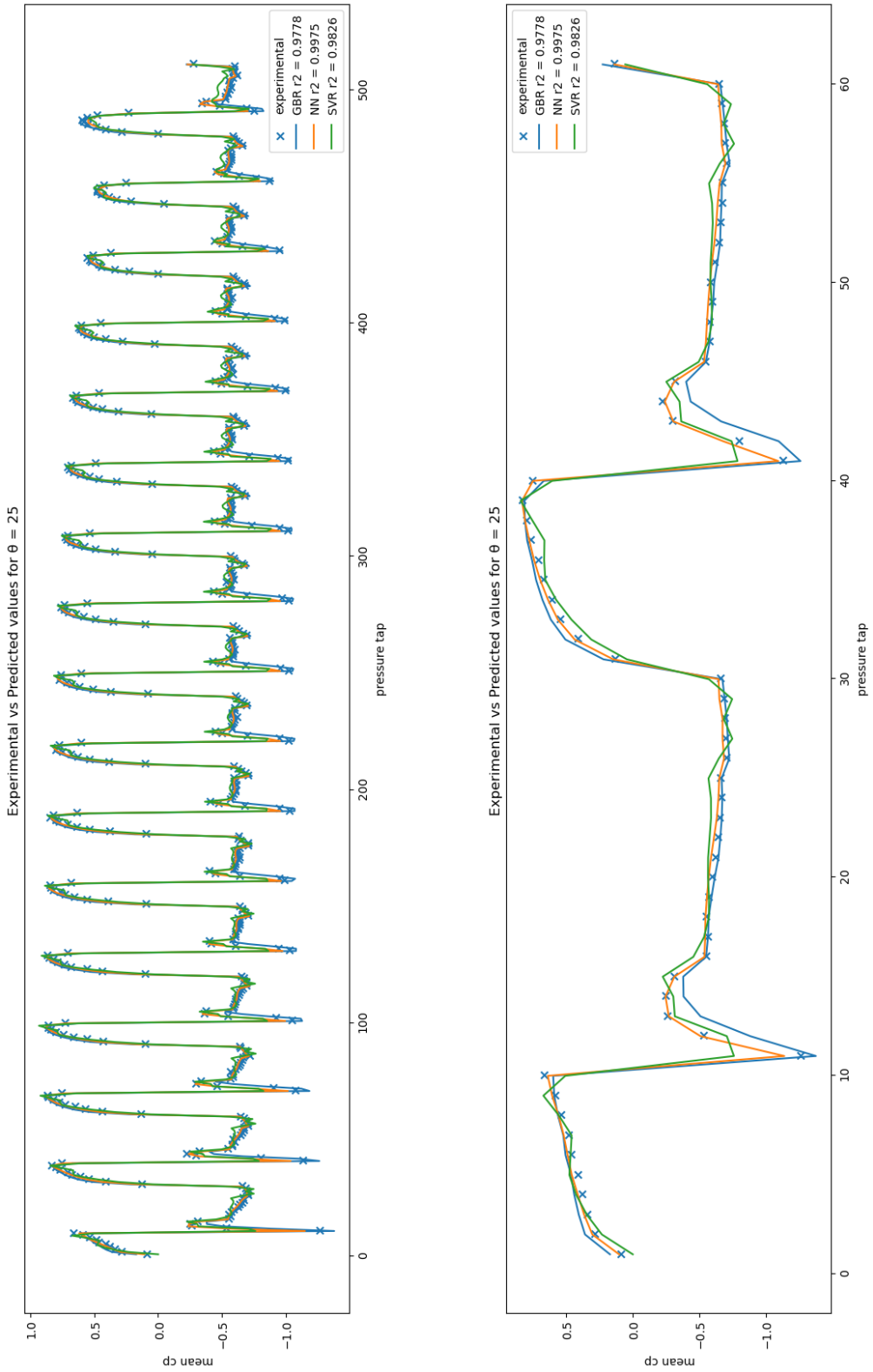


Figure 3.8: Correlation Coefficient for 3 models for predicting mean c_p

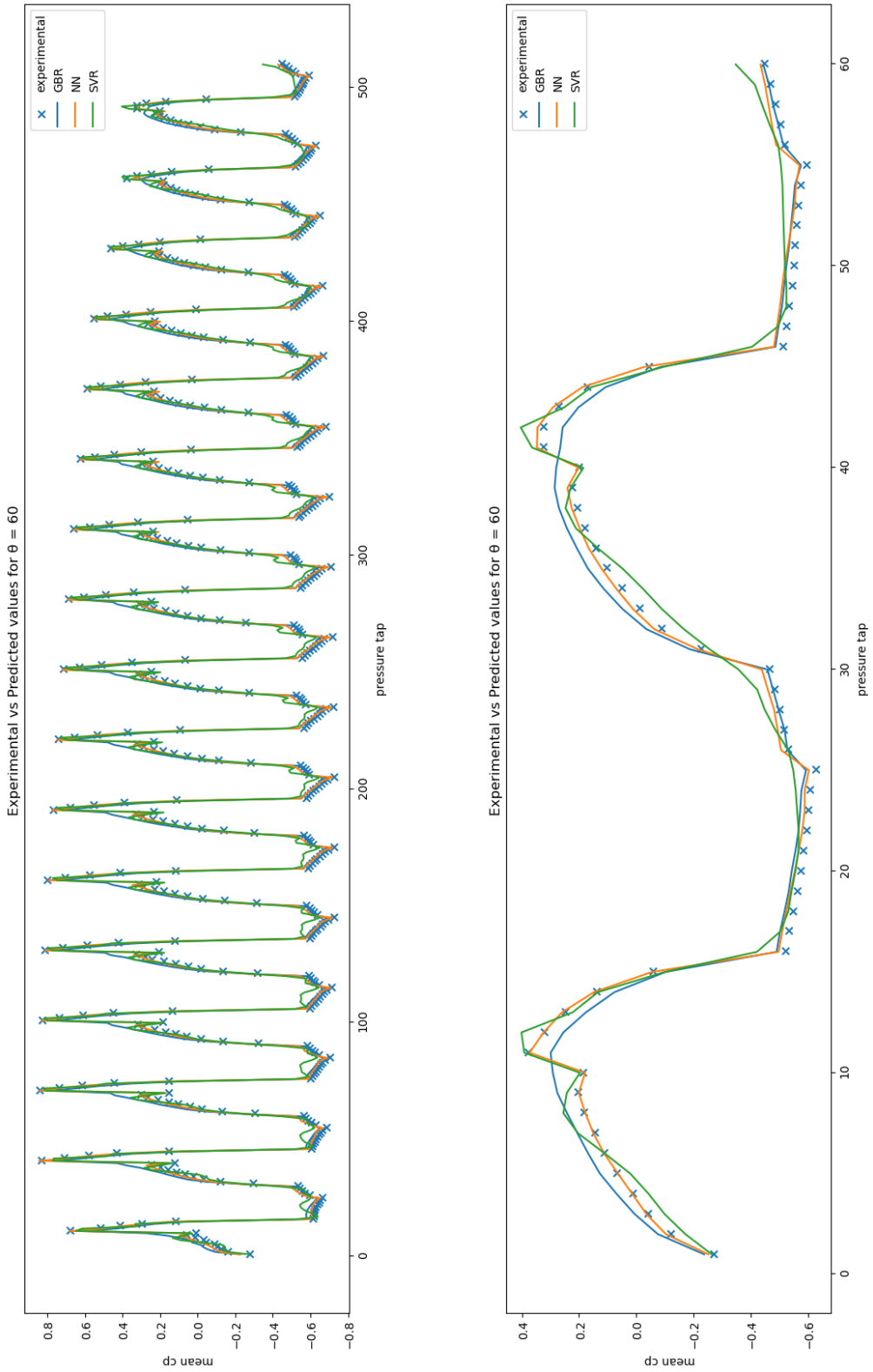
It can be seen from figures 3.9 and 3.10 that all the models performed well



(a) All taps

(b) first 60 taps

Figure 3.9: Experimental vs Predicted for 2:1:5 building $\alpha = 25°$



(a) All taps

(b) First 60 taps

Figure 3.10: Experimental vs Predicted for 2:1:5 building $\alpha = 60°$

with the GBR model having $R^2 = 0.9778$ and $MSE = .0057$, the SVM model with $R^2 = 0.9826$ and $MSE = .0044$ but the Neural Network model performed the best with $R^2 = .9975$ and $MSE = .00062$. Although the models have relatively high R^2 and MSE values the NN also showed a very good fit with the truth compared to the other models. Therefore, The Neural Network is the obvious best choice for missing angle prediction for the selected models.

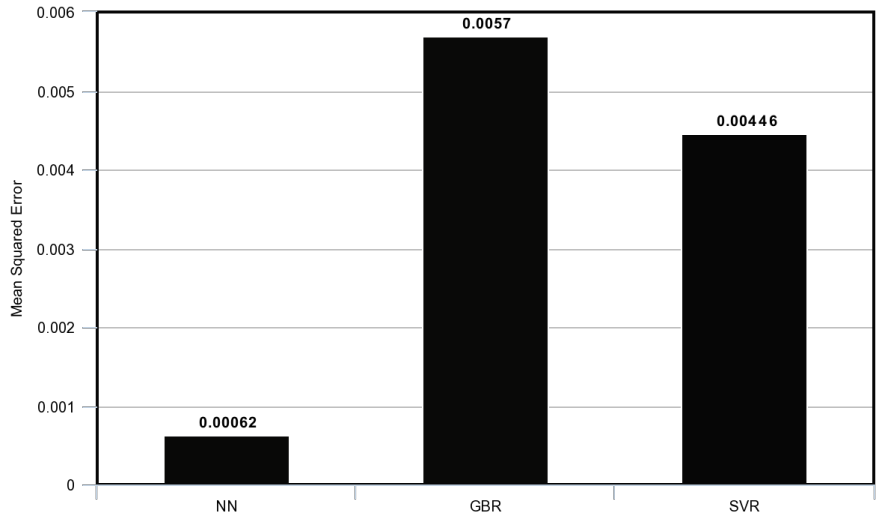
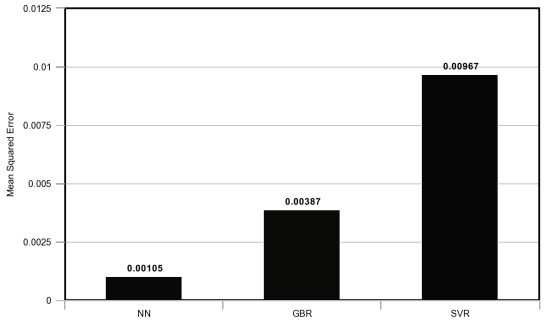


Figure 3.11: Mean squared error of ML models for predicting mean c_p

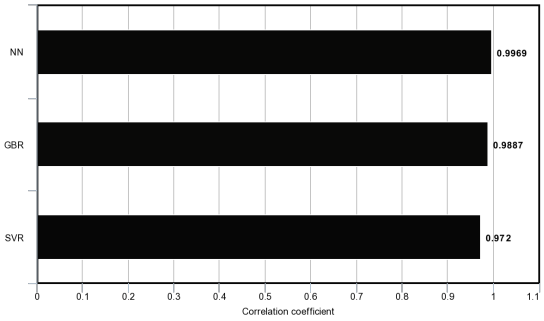
3.3.2 Predicting for an Unseen Building Height

ML models were also used to train and predict mean c_p for all tap locations for various dimensions of the building models. For this approach data from 5 building models, with B/H ratios $1/1, 1/2, 1/3, /1/4/, 1/5$ with constant depth were chosen as the training dataset. The only varying parameter for these buildings is their height and the incoming wind angle is fixed.

From the 5 models selected one model (B/H = 1/3) has been removed from the training dataset and used for testing. Figure 3.12 shows the results and comparison for prediction of mean c_p for the unseen building(B/H = 1/3). Again, the DNN model showed the best performance compared to the other models. A lower



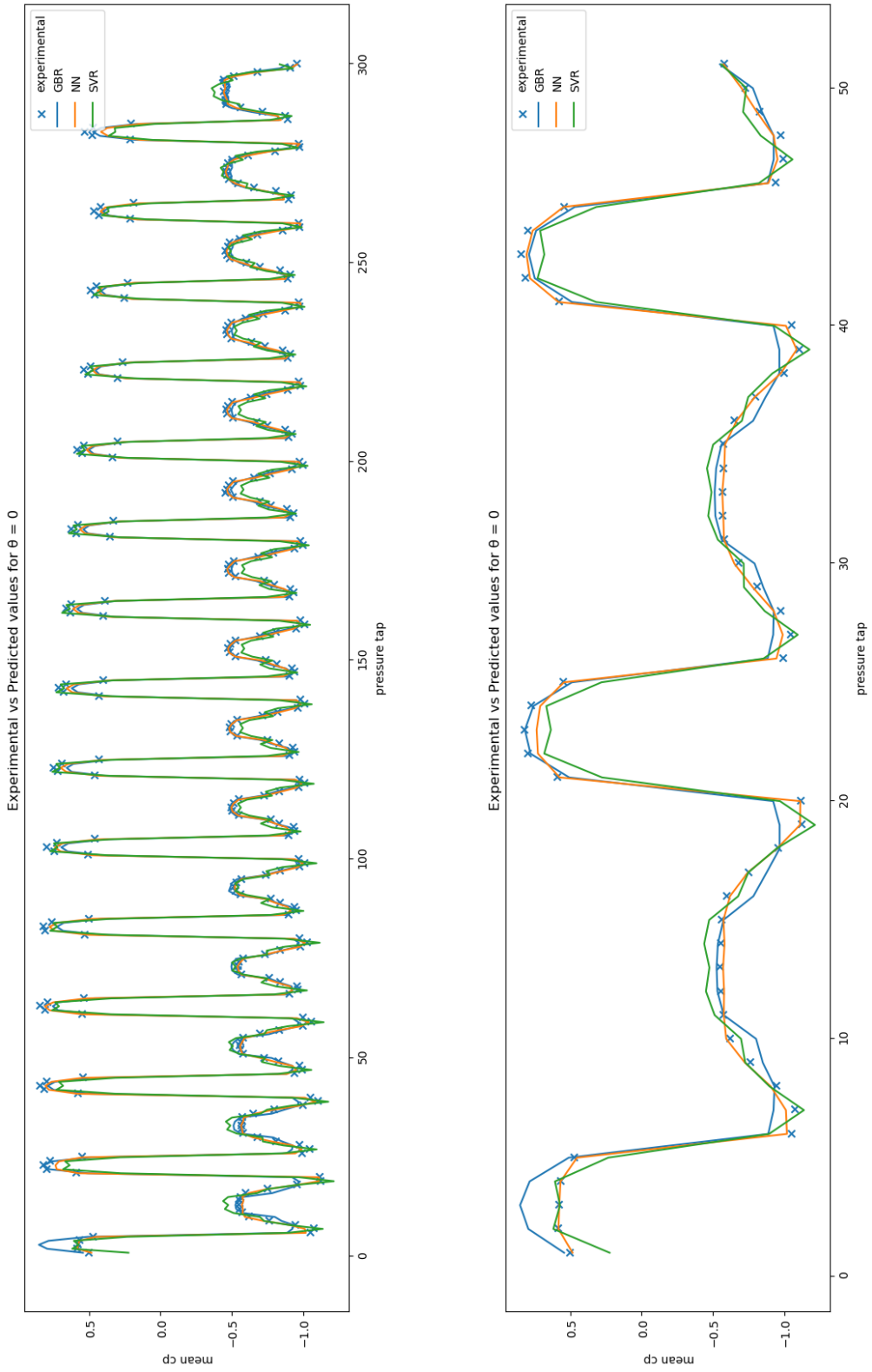
(a) Mean Squared Error



(b) Correlation coefficient

Figure 3.12: Error Metrics for unseen building height prediction

amount of overall data might be the reason for overall lower MSE and R^2 as compared to predicting for a missing angle. Figure 3.13 shows results for prediction of all models on all points on the 1/3 B/H ratio building predicted by the ML models using other buildings of various heights.



(a) (b)

Figure 3.13: ML model comparison for 1:1:3 building

CHAPTER IV

CONCLUSIONS

4.1 Conclusions

3 ML algorithms (NN, GBRT and SVR) have been trained to predict the Surface-averaged wind pressure coefficients c_p on Isolated high-rise buildings. The Training dataset chosen for this work is the Tokyo Polytechnic University's Aerodynamic Database for Isolated High rise buildings. The database was used to predict for an unseen wind angle and for an unseen building height. The Models were constructed in Python and the ML algorithms were implemented using the Sci-kit learn (SVM and GBRT) and keras (NN) modules. Grid Search was used as the hyperparameter tuning algorithm for this study. An Additional investigation on the effect of using a local vs global prediction approach for an unseen wind angle was conducted. The global approach used all the taps on the building as the prediction basis whereas the local approach only used the pressure taps on the surface of the point of investigation. It has been shown that there is very little to no difference in prediction accuracy between using a Local or global prediction

basis and either approach can be used to predict Average c_p for isolated High rise buildings. While predicting for an Unseen wind angle, the 2:1:5 building and its 21 different incoming wind angles were chosen for prediction. 2 wind angles $\alpha = 25°, 60°$ were removed from the training dataset and used for testing. For the case of prediction for an unseen building height, Data for 5 building models (B/H ratio 1/1,1/2,1/3,1/4,1/5) at an incident wind angle of $\alpha = 0°$ were chosen out of which the 1/3 building data was excluded from the dataset and used for testing. The correlation coefficient and Mean squared error were chosen as the error metrics for this study. Of the models studied the Neural Network with 3 hidden layers showed the best accuracy for avg c_p prediction although the GBRT and SVM models were not further behind the NN model and showed relatively similar accuracy to each other.

4.2 Remarks and Future work

Although it has been shown that ML models used in the study for predicting surface average c_p showed very high accuracy, the hyperparameters used in the models were selected using a grid search algorithm that relies on the user input of the hyperparameter space. The hyperparameters used in GSA in this work were selected based on the previous literature, personal experience, and observation. A Future work would therefore be to use algorithms like Random Search and Gaussian optimization that are much more computationally heavy but do give the best hyperparameters for a given model.

In this work ML models were used to predict only the mean c_p whereas in general, variation in c_p is transient in nature. Also, a combined model that can take the incoming wind angle and building dimensions together as inputs to predict

the surface c_p in the building is another potential extension to this work. The TPU dataset facilitates investigation into both the transient nature of the problem (the Data Set hosted is transient) and the combined model(13 models, up to 21 wind angles), however, using other Datasets can help in building a more robust ML model.

Another potential tangent is to use ML models to predict for an unseen Pressure tap location which can be used to determine the resolution of pressure taps required to achieve a target accuracy when using a global basis for prediction and also to establish the relative importance of pressure taps on the building models.

Bibliography

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[2] Susan Athey. "The impact of machine learning on economics". In: *The economics of artificial intelligence: An agenda*. University of Chicago Press, 2018, pp. 507–547.

[3] Facundo Bre, Juan M. Gimenez, and Víctor D. Fachinotti. "Prediction of wind pressure coefficients on building surfaces using artificial neural networks". In: *Energy and Buildings* 158 (2018), pp. 1429–1441. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2017.11.045. URL: https://www.sciencedirect.com/science/article/pii/S0378778817325501.

[4] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. "Machine learning for fluid mechanics". In: *Annual review of fluid mechanics* 52 (2020), pp. 477–508.

[5] Y. Chen, G.A. Kopp, and D. Surry. "Interpolation of wind-induced pressure time series with an artificial neural network". In: *Journal of Wind Engineering and Industrial Aerodynamics* 90.6 (2002), pp. 589–615. ISSN: 0167-6105. DOI: https://doi.org/10.1016/S0167-6105(02)00155-1. URL: https://www.sciencedirect.com/science/article/pii/S0167610502001551.

[6] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[7] Huang Dongmei et al. “Prediction of wind loads on high-rise building using a BP neural network combined with POD”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 170 (2017), pp. 1–17. ISSN: 0167-6105. DOI: https://doi.org/10.1016/j.jweia.2017.07.021. URL: https://www.sciencedirect.com/science/article/pii/S016761051730003X.

[8] Mohammad Mahdi Forootan et al. “Machine learning and deep learning in energy systems: A review”. In: *Sustainability* 14.8 (2022), p. 4832.

[9] J.Y. Fu, S.G. Liang, and Q.S. Li. “Prediction of wind-induced pressures on a large gymnasium roof using artificial neural networks”. In: *Computers and Structures* 85.3 (2007), pp. 179–192. ISSN: 0045-7949. DOI: https://doi.org/10.1016/j.compstruc.2006.08.070. URL: https://www.sciencedirect.com/science/article/pii/S0045794906003075.

[10] X. Gavalda et al. “Interpolation of pressure coefficients for low-rise buildings of different plan dimensions and roof slopes using artificial neural networks”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 99.5 (2011), pp. 658–664. ISSN: 0167-6105. DOI: https://doi.org/10.1016/j.jweia.2011.02.008. URL: https://www.sciencedirect.com/science/article/pii/S0167610511000432.

[11] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[12] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[13] Youqin Huang et al. "Prediction of mean and RMS wind pressure coefficients for low-rise buildings using deep neural networks". In: *Engineering Structures* 274 (2023), p. 115149.

[14] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[15] David T Jones. "Setting the standards for machine learning in biology". In: *Nature Reviews Molecular Cell Biology* 20.11 (2019), pp. 659–660.

[16] George Em Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[17] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. "Supervised machine learning: A review of classification techniques". In: *Emerging artificial intelligence applications in computer engineering* 160.1 (2007), pp. 3–24.

[18] Yi Li et al. "Machine learning based algorithms for wind pressure prediction of high-rise buildings". In: *Advances in Structural Engineering* 25.10 (2022), pp. 2222–2233.

[19] Tomonori Masui. *All You Need to Know about Gradient Boosting Algorithm Part 1. Regression*. Accessed on Aug 5, 2023, 4:23 pm. 2023. URL: https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502.

[20] Mathworks. *Understanding Support Vector Regression*. Accessed on Aug 15, 2023, 4:43 am. 2023. URL: https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html.

[21] F Pedregosa. *Support Vector Machines*. URL: https://scikit-learn.org/stable/modules/svm.html#svm.

[22] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[23] Alvin Rajkomar, Jeffrey Dean, and Isaac Kohane. "Machine learning in medicine". In: *New England Journal of Medicine* 380.14 (2019), pp. 1347–1358.

[24] Sayyeda Saadia Razvi et al. "A review of machine learning applications in additive manufacturing". In: *International design engineering technical conferences and computers and information in engineering conference*. Vol. 59179. American Society of Mechanical Engineers. 2019, V001T02A040.

[25] Guido van Rossum. *Python Release Python 3.9.15*. URL: https://www.python.org/downloads/release/python-3915/.

[26] scikit-learn. *Ensemble Methods*. URL: https://scikit-learn.org/stable/modules/ensemble.html.

[27] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[28] Jianqiao Tian et al. "Low-rise gable roof buildings pressure prediction using deep neural networks". In: *Journal of Wind Engineering and Industrial Aerodynamics* 196 (2020), p. 104026.

[29] TPU. *Aerodynamic Database for Low rise buildings*. Accessed on August 03, 2023, 3:44 am. 2007. URL: http://www.wind.arch.t-kougei.ac.jp/info_center/windpressure/lowrise/Introductionofthedatabase.pdf.

[30] TPU. *Tokyo Aerodynamic Database*. Accessed on August 03, 2023, 2:02 am. 2007. URL: http://www.wind.arch.t-kougei.ac.jp/system/eng/contents/code/tpu.

[31] Tokyo Polytechnic University. *Aerodynamic Database of High-rise Buildings*. Accessed on August 03, 2023, 1:44 am. URL: http://www.wind.arch.tkougei.ac.jp/info_center/windpressure/highrise/Homepage/homepageHDF.htm.

[32] Yanmo Weng and Stephanie German Paal. "Machine learning-based wind pressure prediction of low-rise non-isolated buildings". In: *Engineering Structures* 258 (2022), p. 114148.

[33] Li Yang and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice". In: *Neurocomputing* 415 (2020), pp. 295–316.

[34] Tong Yu and Hong Zhu. "Hyper-Parameter Optimization: A Review of Algorithms and Applications". In: *CoRR* abs/2003.05689 (2020). arXiv: 2003.05689. URL: https://arxiv.org/abs/2003.05689.

[35] Aishe Zhang and Ling Zhang. "RBF neural networks for the prediction of building interference effects". In: *Computers and Structures* 82.27 (2004), pp. 2333–2339. ISSN: 0045-7949. DOI: https://doi.org/10.1016/j.

compstruc.2004.05.014. URL: https://www.sciencedirect.com/science/article/pii/S0045794904002007.

APPENDIX A

PYTHON CODE FOR LOCAL VS GLOBAL PREDICTION (NN only)

```
#Please keep in mind that all the code presented in this work are not perfectly efficient.
# For this code to work all the files from the high-rise building section of the TPU aerodynamic database have to be in the working folder and the filenames have to remain unchanged.
# The main objective for this code is to make a feed-forward Neural Network (ANN) and train it using the TPU dataset We want to train a model that can learn how the cp values on a building will change with changing incident angles. This means that the shape of the building will remain constant for a given model along with the incoming velocity profile curve.
# The model will have the following properties: predict the cp value at a given point on a building for a given incidence wind angle and a given dimensional configuration. After training the model will take the location (x,y,z) and incident wind angle (a) as inputs and produce the mean cp as output.
# necessary imports
import numpy as np
import pandas as pd

from pymatreader import read_mat # the TPU dataset available on the website is in the .mat format. This import allowes to read .mat files in Python.


import os.path#import necessary to look for files in the working directory

import matplotlib.pyplot as plt # for graphs and plots

b=2
h=5
p=4
facenum=1 # defining breadth(b), height (h) and velocity profile(p) these values will remain the same throughout this code. These can be changed according to the model to be investigated. face number is defined for local prediction
# loading a datafile to check the kind of information
df = read_mat('T114_4_000_1.mat')
# since the datafile contains different types of information it generally contains several variables. calling the "keys()" function shows the variables available in the .mat files.
df.keys()
# the variable of interest for us is the 'Wind_pressure_coefficients' variable.
dataframe = pd.Series([],dtype=float)# defining an empty dataframe for use later. dtype has to be defined for empty dataframes
```

```
26
27 # *****TPU FILE NOMENCLATURE****: It is important to understand the
       nomenclature behind the files downloaded from the TPU website in
       order to understand the code. For example: if the file name is
       'T112_4_010_1.mat' it would mean the file corresponds to the
       breadth:depth:height ratio of 1:1:2 with a velocity profile of 1/4
       power law and an incident wind angle of 010 degrees tilted from the
       normal.
28
29 # looping strategy : the looping contains 3 nested loops m,n,k which
       correspond to the 100s, 10s, and 1s, place for the incoming angle in
       the filename. since b,h,p are already defined the resulting filename
       should look like this : 'T114_4_mnk_1.mat'. this will result in
       loading different files based on the values of m,n and k.
30
31 # loop to merge all files of similar configuration
32 for m in range(2):# m is the 100s place in the angle. the TPU data set
       has angles varying from 0 to 150 degress with a 5 degree increment.
33 # this mean the 100s place value can vary only between 0 and 1 hence
       range is 2
34     for n in range(10):# n is 10s place and can range from 0 to 9 hence
           range 10
35         for k in range(6):# k is the 1s place and can only be either 0
               or 5 hence the range 6
36             filename = "T{}1{}_{}_{}{}{}_1.mat".format(b,h,p,m,n,k)#
                   after the values set in the loop '.format' allows us to
                   place variable in a string. each {} is a variable.
37
38             if os.path.isfile(filename) == True:# this ensures the loop
                   proceeds for only existing filenames. The '.isfile'
                   checks for existance of the file in the working directory
39                 df = read_mat(filename)
40
41                 avgspeed= df['Uh_AverageWindSpeed']
42                 position =
                       pd.DataFrame(df['Location_of_measured_points']).transpose()#
                       the position of points is an important variable.
                       Transpose is done for later.
43                 # the position variable has 4 columns. "0,1,2,3"
                       corresponding to x,y,pointnum and facenum
44                 breadth = df['Building_breadth']
45                 depth = df['Building_depth']
46                 height = df['Building_height']
47 # the TPU dataset position of pressure taps is presented in the
       following way. each point has a x,y coordinate along with the face
       number. faces 1,2,3,4 correspond to windward,right,leeward,left
       respectively. the x,y coordinates are the coordinates resulting
       after the model is "unwrapped" and the point of origin being the
       bottom right corner of face1. please check
       http://www.wind.arch.t-kougei.ac.jp/info_center/windpressure/highrise
       for futher clarification.
```

```
48
49
50              #changing the data from 2-D points and faces to 3-D
                    points.
51
52              # to achieve this add additional columns x,y,z to the
                    position dataframe and generate x,y,z values
53              # based on the x,y,face values using the below strategy.
54              position.columns = position.columns.map(str) # this
                    ensures the column values are converted to strings
55              position['x']=0 # initializing x
56              position['y']=position['1'] # the y coordinate will be
                    unchanged be in 2-d or 3-d to we just equate existing
                    y to the new y
57              position['z']=0 # initializing z
58
59
60              for index, row in position.iterrows(): # iterrating
                    through each row.T
61
62                  if row['3'] == 1:
63                         # conversion for face 1
64                         position['x'][index]= row['0']
65                  elif row['3'] == 2:
66                         # conversion for face 2
67                         position['z'][index]= row['0']-breadth
68                         position['x'][index]= breadth
69                  elif row['3'] == 3:
70                         # conversion for face 3
71                         position['z'][index]= depth
72                         position['x'][index]=
                               (breadth+breadth+depth)-row['0']
73                  elif row['3'] == 4:
74                         # conversion for face 4
75                         position['z'][index]=
                               (breadth+breadth+depth+depth)-row['0']
76
77              # dropping the position values and renaming the columns
78              xyz = position.drop(['0','1'],axis=1)
79              xyz.rename(columns={"2":"num"},inplace=True)
80
81              #adding mean cp data
82              cp = pd.DataFrame(df['Wind_pressure_coefficients'])
83              # .mean() creates a mean of the entire column
84              meancp = cp.mean()
85              # merging the cp data to the position data
86              dat = pd.concat([xyz,meancp],axis=1)
87
88              a = m*100+n*10+k#adding the corresponding angle column.
                    the angle will be same for a given datafile hence
89              # we create a column with the same angle value with
```

```
                    length of the dataframe.
                angle = pd.Series([a]*(len(dat)))

                #adding angle value
                meandat = pd.concat([dat,angle],axis=1,ignore_index=True)

                #merging old and new dataframes for each loop.
                dataframe = pd.concat([dataframe,meandat],axis=0)

dataframe = dataframe.reset_index(drop=True)

#preparing dataset for training

traindat = dataframe[[2,3,4,5,6]] # training data will only have
    x,y,z,meancp and angle
traindat1=traindat[~traindat[6].isin([25,60])]
testdat1=traindat[traindat[6].isin([25,60])]
# imports error metrics
from sklearn.metrics import r2_score, mean_squared_error
X_train = traindat1[[2,3,4,6]] # defining inputs
y_train = traindat1[5] # defububg outputs
X_test = testdat1[[2,3,4,6]]
y_test = testdat1[5]

# creating the NN architecture

# we use the keras python package for constructing the NN.
from keras.models import Sequential
from keras.layers import Dense

# sequential is the call for a FeedForward NN.
gmodel = Sequential()

# adding the first layer in 4 inputs and relu activation function
gmodel.add(Dense(128, input_shape=(4,), activation='elu'))

# adding a hidden layer
gmodel.add(Dense(128, activation='elu'))
# adding a hidden layer
#model.add(Dense(128, activation='elu'))
# adding a hidden layer
gmodel.add(Dense(16, activation='elu'))
# adding the output layer. 1 output and linear activation function
gmodel.add(Dense(1,activation='linear'))


# defining the loss function and optimizer
gmodel.compile(loss = 'mse',optimizer='adam')


gmodel.fit(X_train,y_train, epochs=2000, batch_size=32)
```

```
139
140 y_pred = gmodel.predict(X_test)
141 nn_r2 = r2_score(y_test, y_pred)
142 nn_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
143 print(nn_r2 ,nn_rmse) #checking the error metrics
144
145 #preparing datafiles for local prediction
146 faces1= dataframe.groupby(dataframe[1])
147 dfs1={}
148 for name,group in faces1:
149     dfs1[name]=group
150 face1 = dfs1[facenum]
151 facexyz = face1[[2,3,4,6]]
152 facexyz.reset_index(inplace=True,drop=True)
153 faceglobal=facexyz[facexyz[6].isin([25,60])]
154 globalpred = gmodel.predict(faceglobal)
155 dataframe = pd.Series([],dtype=float)
156
157 for m in range(2):
158     for n in range(10):
159         for k in range(6):
160             filename = "T{}1{}_{}_{}{}{}_1.mat".format(b,h,p,m,n,k)
161             if os.path.isfile(filename) == True:
162                 df = read_mat(filename)
163                 avgspeed= df['Uh_AverageWindSpeed']
164                 location =
                        pd.DataFrame(df['Location_of_measured_points']).T
165                 cp = pd.DataFrame(df['Wind_pressure_coefficients'])
166                 meancp = cp.mean()
167                 dat =
                        pd.concat([location,meancp],axis=1,ignore_index=True)
168                 a = m*100+n*10+k
169                 angle = pd.Series([a]*(len(dat)))
170                 meandat = pd.concat([dat,angle],axis=1,ignore_index=True)
171                 dataframe = pd.concat([dataframe,meandat],axis=0)
172
173 dataframe = dataframe.reset_index(drop=True)
174 faces = dataframe.groupby(dataframe[3])
175 dfs = {}
176 for name, group in faces:
177     dfs[name] = group
178 face = dfs[facenum]
179 face.reset_index(inplace=True,drop=True)
180 dataframe = face.copy()
181 #training and testing data for local prediction
182 traindat = dataframe[[0,1,4,5]] # training data will only have
    x,y,z,meancp and angle
183 traindat # training data check
184
185 traindat1=traindat[~traindat[5].isin([25,60])]
186 testdat1=traindat[traindat[5].isin([25,60])]
```

```
X_train = traindat1[[0,1,5]] # defining inputs
y_train = traindat1[4] # defining outputs
X_test = testdat1[[0,1,5]]
y_test = testdat1[4]

from keras.models import Sequential
from keras.layers import Dense

# sequential is the call for a FeedForward NN.
model = Sequential()

# adding the first layer in 4 inputs and relu activation function
model.add(Dense(128, input_shape=(3,), activation='elu'))

# adding a hidden layer
model.add(Dense(128, activation='elu'))
# adding a hidden layer
#model.add(Dense(128, activation='elu'))
# adding a hidden layer
model.add(Dense(16, activation='elu'))
# adding the output layer. 1 output and linear activation function
model.add(Dense(1,activation='linear'))

#NN model for local prediction
# defining the loss function and optimizer
model.compile(loss = 'mse',optimizer='adam')

# training the model with the stopping criterion being the number of
    epochs
model.fit(X_train,y_train, epochs=1000, batch_size=32)

y_pred = model.predict(X_test)

nn_r2 = r2_score(y_test, y_pred)
nn_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(nn_r2 ,nn_rmse) #checking the error metrics

#code below is for the graphs
modelpred = model.predict(X_test)
graphdat = testdat1[[4]].copy()

tap_no = pd.DataFrame(range(1,len(modelpred)+1,1))
pred_dat = np.transpose(modelpred)
global_dat = np.transpose(globalpred)
flattened_array = pred_dat.flatten()
global_array = global_dat.flatten()

pred_ser = pd.Series(flattened_array)
global_ser = pd.Series(global_array)

graphdat = graphdat.reset_index(drop=True)
```

```
graph1 = pd.concat([tap_no,graphdat],axis=1,ignore_index = True)
graph =
    pd.concat([graph1,pred_ser,global_ser],axis=1,ignore_index=True)#
    this is the dataframe showing the tap no. , true value and predicted
    value.

tap = graph.loc[:(len(graph)/2)-1,0]
y1 = graph.loc[:(len(graph)/2)-1,1]
y2 = graph.loc[:(len(graph)/2)-1,2]
y3 = graph.loc[:(len(graph)/2)-1,3]
plt.figure(figsize=(20,5))
plt.title('Experimental vs Predicted values for theta = 25 Face = 1')
plt.plot(tap, y1, label='experimental')
plt.scatter(tap, y2, label='Local prediction',marker='+')
plt.scatter(tap, y3, label='Global prediction',marker='^')
plt.legend()
plt.xlabel('pressure tap')
plt.ylabel('mean cp')
plt.show()
```

APPENDIX B

PYTHON CODE FOR ML MODELS USED TO PREDICT c_p for AN UNSEEN WIND ANGLE

```
# This code compares the performance of 3 ML models to predict average cp for an unseen wind angle
#Please keep in mind that all the code presented in this work are not perfectly efficient.
# For this code to work all the files from the high-rise building section of the TPU aerodynamic database have to be in the working folder and the filenames have to remain unchanged.
#grid search has been omitted for SVM and the hyperparameter value used here is the best among the selected hyperparameters. The code for grid-search algo used is presented in the GBRT model.
#comments have been omitted because of the repeated nature of the code in the other appendices
import numpy as np
import pandas as pd
from pymatreader import read_mat
import os.path
import matplotlib.pyplot as plt
b=2
h=5
p=4
facenum = 1
df = read_mat('T114_4_000_1.mat')
dataframe = pd.Series([],dtype=float)
for m in range(2):
    for n in range(10):
        for k in range(6):
            filename = "T{}1{}_{}_{}{}{}_1.mat".format(b,h,p,m,n,k)
            if os.path.isfile(filename) == True:
                df = read_mat(filename)
                avgspeed= df['Uh_AverageWindSpeed']
                position =
                    pd.DataFrame(df['Location_of_measured_points']).transpose()
                breadth = df['Building_breadth']
                depth = df['Building_depth']
                height = df['Building_height']
                position.columns = position.columns.map(str)
                position['x']=0
                position['y']=position['1']
                position['z']=0
                for index, row in position.iterrows():
                    if row['3'] == 1:
                            position['x'][index]= row['0']
                    elif row['3'] == 2:
                            position['z'][index]= row['0']-breadth
                            position['x'][index]= breadth
                    elif row['3'] == 3:
```

```
                            position['z'][index]= depth
                            position['x'][index]=
                                (breadth+breadth+depth)-row['0']
                        elif row['3'] == 4:
                            position['z'][index]=
                                (breadth+breadth+depth+depth)-row['0']
                    xyz = position.drop(['0','1'],axis=1)
                    xyz.rename(columns={"2":"num"},inplace=True)
                    cp = pd.DataFrame(df['Wind_pressure_coefficients'])
                    meancp = cp.mean()
                    dat = pd.concat([xyz,meancp],axis=1)
                    a = m*100+n*10+k
                    angle = pd.Series([a]*(len(dat)))
                    meandat = pd.concat([dat,angle],axis=1,ignore_index=True)
                    dataframe = pd.concat([dataframe,meandat],axis=0)
dataframe = dataframe.reset_index(drop=True)

from sklearn.preprocessing import StandardScaler
traindat = dataframe[[2,3,4,5,6]].copy()
scaler = StandardScaler()
traindat_scaled =
    pd.DataFrame(scaler.fit_transform(traindat[[2,3,4,6]]))
traindat1=traindat[~traindat[6].isin([25,60])]
testdat1=traindat[traindat[6].isin([25,60])]
testdat1_scaled = scaler.transform(testdat1[[2,3,4,6]])
traindat1_scaled = scaler.transform(traindat1[[2,3,4,6]])
from sklearn.metrics import r2_score, mean_squared_error
X_train = pd.DataFrame(traindat1_scaled)
y_train = traindat1[5]
X_test = pd.DataFrame(testdat1_scaled)
y_test = testdat1[5]

#NN model
from keras.models import Sequential
from keras.layers import Dense
gmodel = Sequential()
gmodel.add(Dense(128, input_shape=(4,), activation='elu'))
gmodel.add(Dense(128, activation='elu'))
gmodel.add(Dense(16, activation='elu'))
gmodel.add(Dense(1,activation='linear'))
gmodel.compile(loss = 'mse',optimizer='adam')
history = gmodel.fit(X_train,y_train, epochs=1000, batch_size=32)
y_pred = gmodel.predict(X_test)
nn_r2 = r2_score(y_test, y_pred)
nn_rmse = mean_squared_error(y_test, y_pred)
print(nn_r2 ,nn_rmse) #checking the error metrics

plt.figure(figsize=(10,5))
plt.plot(history.history['loss']) # 'loss' key contains the training
    loss values
plt.title('Model Loss')
```

```
plt.ylabel('Loss = MSE')
plt.xlabel('Epoch')
plt.show()

#GBRT model with gridsearch
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

gbr = GradientBoostingRegressor()
# you can add more parameters here for a bigger hyperparameter space.
param_grid = {
    'n_estimators':[500,550],
    'learning_rate':[.1],
    'max_depth': [7]
}

grid_search = GridSearchCV(gbr,
    param_grid,cv=5,scoring='neg_mean_squared_error')
grid_search.fit(X_train,y_train)

y_pred = grid_search.predict(X_test)

gbr_r2 = r2_score(y_test,y_pred)
gbr_mse= mean_squared_error(y_test,y_pred)

params_gbr = grid_search.best_params_

print( gbr_r2,gbr_mse)
print(params_gbr)

#SVR model
from sklearn.svm import SVR
# Train the model on the training data
svr_model = SVR(
    C=10.0,
    kernel='rbf',
    gamma=1, # or a specific value
)
svr_model.fit(X_train, y_train)

# Predict on the test data
y_pred = svr_model.predict(X_test)

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test,y_pred)

#plotting graphs
modelpred = grid_search.predict(X_test)

graphdat = testdat1[[5]].copy()
```

```
136
137 tap_no = pd.DataFrame(range(1,len(modelpred)+1,1))
138 pred_dat = np.transpose(modelpred)
139 globalpred = gmodel.predict(X_test)
140 global_dat = np.transpose(globalpred)
141 svrpred = svr_model.predict(X_test)
142 svr_dat = np.transpose(svrpred)
143 flattened_array = pred_dat.flatten()
144 global_array = global_dat.flatten()
145 svr_array = svr_dat.flatten()
146
147 pred_ser = pd.Series(flattened_array)
148 global_ser = pd.Series(global_array)
149 svr_ser = pd.Series(svr_array)
150 graphdat = graphdat.reset_index(drop=True)
151 graph1 = pd.concat([tap_no,graphdat],axis=1,ignore_index = True)
152 graph =
    pd.concat([graph1,pred_ser,global_ser,svr_ser],axis=1,ignore_index=True)
153
154 #for angle = 25 degrees
155 tap = graph.loc[:(len(graph)/2),0]
156 y1 = graph.loc[:(len(graph)/2),1]
157 y2 = graph.loc[:(len(graph)/2),2]
158 y3 = graph.loc[:(len(graph)/2),3]
159 y4 = graph.loc[:(len(graph)/2),4]
160 plt.figure(figsize=(20,5))
161 plt.title('Experimental vs Predicted values for angle = 25 ')
162 plt.scatter(tap, y1, label='experimental',marker= 'x')
163 plt.plot(tap, y2, label='GBR ')
164 plt.plot(tap, y3, label='NN ')
165 plt.plot(tap, y4, label='SVR ')
166 plt.legend()
167 plt.xlabel('pressure tap')
168 plt.ylabel('mean cp')
169 plt.show()
170
171 #for angle = 60 degrees
172
173 tap = graph.loc[:(len(graph)/2)-1,0]
174 y1 = graph.loc[len(graph)/2:,1]
175 y2 = graph.loc[len(graph)/2:,2]
176 y3 = graph.loc[len(graph)/2:,3]
177 y4= graph.loc[len(graph)/2:,4]
178 plt.figure(figsize=(20,5))
179 plt.title('Experimental vs Predicted values for angle = 60 ')
180 plt.scatter(tap, y1, label='experimental',marker = 'x')
181 plt.plot(tap, y2, label='GBR')
182 plt.plot(tap, y3, label='NN')
183 plt.plot(tap, y4, label='SVR')
184 plt.legend()
185 plt.xlabel('pressure tap')
```

```
186 plt.ylabel('mean cp')
187 plt.show()
```

APPENDIX C

PYTHON CODE FOR ML MODELS USED TO PREDICT c_p for AN UNSEEN BUILDING HEIGHT

```
1
2  # This code compares the performance of 3 ML models to predict average
       cp for an unseen Building Height
3  #Please keep in mind that all the code presented in this work are not
       perfectly efficient.
4  # For this code to work all the files from the high-rise building
       section of the TPU aerodynamic database have to be in the working
       folder and the filenames have to remain unchanged.
5  #grid search has been omitted for SVM and the hyperparameter value used
       here is the best among the selected hyperparameters. The code for
       grid-search algo used is presented in the GBRT model.
6  #comments have been omitted because of the repeated nature of the code
       in the other appendices
7
8  import numpy as np
9  import pandas as pd
10 from pymatreader import read_mat
11 import os.path
12 import matplotlib.pyplot as plt
13 from sklearn.svm import SVR
14
15 a = '000'
16 p = 4
17
18 dataframe = pd.Series([],dtype=float)
19
20 #loop to merge all files of similar configuration
21
22
23 for n in range(6):
24
25         filename = "T11{}_{}_{}_1.mat".format(n,p,a)
26         if os.path.isfile(filename) == True:
27             df = read_mat(filename)
28             avgspeed= df['Uh_AverageWindSpeed']
29             position =
                   pd.DataFrame(df['Location_of_measured_points']).transpose()
30             breadth = df['Building_breadth']
31             depth = df['Building_depth']
32             height = df['Building_height']
33
34             position.columns = position.columns.map(str)
35             position['x']=0
36             position['y']=position['1']
37             position['z']=0
38             for index, row in position.iterrows():
```

```
                if row['3'] == 1:
                        position['x'][index]= row['0']
                elif row['3'] == 2:
                        position['z'][index]= row['0']-breadth
                        position['x'][index]= breadth
                elif row['3'] == 3:
                        position['z'][index]= depth
                        position['x'][index]=
                            (breadth+breadth+depth)-row['0']
                elif row['3'] == 4:
                        position['z'][index]=
                            (breadth+breadth+depth+depth)-row['0']

            xyz = position.drop(['0','1','3'],axis=1)
            xyz.rename(columns={"2":"num"},inplace=True)

            #adding mean cp data
            cp = pd.DataFrame(df['Wind_pressure_coefficients'])
            meancp = cp.mean()
            dat = pd.concat([xyz,meancp],axis=1)

            #adding the corresponding agle


            h = pd.Series([n]*(len(dat)))
            meandat = pd.concat([dat,h],axis=1,ignore_index=True)

            #merging loop
            dataframe = pd.concat([dataframe,meandat],axis=0)

dataframe = dataframe.reset_index()

from sklearn.preprocessing import StandardScaler
traindat = dataframe[[1,2,3,4,5]].copy()
scaler = StandardScaler()
traindat_scaled =
    pd.DataFrame(scaler.fit_transform(traindat[[1,2,3,5]]))

traindat1=traindat[~traindat[5].isin([3])]
testdat1=traindat[traindat[5].isin([3])]
testdat1_scaled = scaler.transform(testdat1[[1,2,3,5]])
traindat1_scaled = scaler.transform(traindat1[[1,2,3,5]])

from sklearn.metrics import r2_score, mean_squared_error
X_train = pd.DataFrame(traindat1_scaled)
y_train = traindat1[4]
X_test = pd.DataFrame(testdat1_scaled)
y_test = testdat1[4]

from keras.models import Sequential
from keras.layers import Dense
```

```
model = Sequential()
model.add(Dense(64, input_shape=(4,), activation='elu'))
model.add(Dense(64, activation='elu'))
model.add(Dense(16, activation='elu'))
model.add(Dense(1,activation='linear'))
model.compile(loss = 'mse',optimizer='adam')
model.fit(X_train,y_train, epochs=600, batch_size=16)
y_pred = model.predict(X_test)
nn_r2 = r2_score(y_test, y_pred)
nn_rmse = mean_squared_error(y_test, y_pred)
print(nn_r2 ,nn_rmse)

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

gbr = GradientBoostingRegressor()

param_grid = {
    'n_estimators':[450,500,550,600,650],
    'learning_rate':[0.01,.05,.1,.5,1],
    'max_depth': [3,4,5,7,8]
}

grid_search = GridSearchCV(gbr,
    param_grid,cv=5,scoring='neg_mean_squared_error')
grid_search.fit(X_train,y_train)

y_pred = grid_search.predict(X_test)

gbr_r2 = r2_score(y_test,y_pred)
gbr_mse= mean_squared_error(y_test,y_pred)

params_gbr = grid_search.best_params_

print(gbr_r2,gbr_mse)
print(params_gbr)

from sklearn.svm import SVR
svr_model = SVR(
    C=50,
    kernel='rbf',
    gamma='scale', # or a specific value
)
svr_model.fit(X_train, y_train)
y_pred = svr_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test,y_pred)
print(r2,mse)

modelpred = grid_search.predict(X_test)

```

```
#plotting
graphdat = testdat1[[4]].copy()
tap_no = pd.DataFrame(range(1,len(modelpred)+1,1))
pred_dat = np.transpose(modelpred)
globalpred = model.predict(X_test)
global_dat = np.transpose(globalpred)
svrpred = svr_model.predict(X_test)
svr_dat = np.transpose(svrpred)
flattened_array = pred_dat.flatten()
global_array = global_dat.flatten()
svr_array = svr_dat.flatten()

pred_ser = pd.Series(flattened_array)
global_ser = pd.Series(global_array)
svr_ser = pd.Series(svr_array)
graphdat = graphdat.reset_index(drop=True)
graph1 = pd.concat([tap_no,graphdat],axis=1,ignore_index = True)
graph =
    pd.concat([graph1,pred_ser,global_ser,svr_ser],axis=1,ignore_index=True)
tap = graph.loc[:len(graph),0]
y1 = graph.loc[:len(graph),1]
y2 = graph.loc[:len(graph),2]
y3 = graph.loc[:len(graph),3]
y4 = graph.loc[:len(graph),4]

plt.figure(figsize=(20,5))
plt.title('Experimental vs Predicted values for angle = 0 ')
plt.scatter(tap, y1, label='experimental',marker= 'x')
plt.plot(tap, y2, label='GBR')
plt.plot(tap, y3, label='NN')
plt.plot(tap, y4, label='SVR')
plt.legend(loc='upper right')
plt.xlabel('pressure tap')
plt.ylabel('mean cp')
plt.show()
```