

**Prediction of Axial Bearing Capacity of Piles using Sophisticated ML  
Algorithms Tuned through Random and Grid Search**



By

**Syed Jamal Arbi**

(Registration No: 00000328433)

Department of Geotechnical Engineering

NUST Institute of Civil Engineering

School of Civil and Environmental Engineering

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2024)

# **Prediction of Axial Bearing Capacity of Piles using Sophisticated ML**

## **Algorithms Tuned through Random and Grid Search**



By

**Syed Jamal Arbi**

(Registration No: 00000328433)

A thesis submitted to the National University of Sciences and Technology, Islamabad, in partial

fulfillment of the requirements for the degree of

**Master of Science in Geotechnical Engineering**

Thesis Supervisor: **Dr. Tariq Mahmood Bajwa**

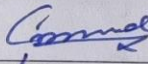
NUST Institute of Civil Engineering

School of Civil and Environmental Engineering

National University of Sciences & Technology (NUST) Islamabad, Pakistan

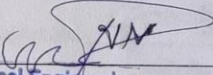
## THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS Thesis written by Mr. Syed Jamal Arbi  
(Registration No. 328433), of Geotechnical Engineering (NICE/SCEE/NUST)  
has been vetted by undersigned, found complete in all respects as per NUST Statutes/  
Regulations/ MS Policy, is free of plagiarism, errors, and mistakes and is accepted as  
partial fulfillment for award of MS degree. It is further certified that necessary  
amendments as point out by GEC members and foreign/ local evaluators of the scholar  
have also been incorporated in the said thesis.

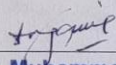
Signature: 

Name of Supervisor Dr. Tariq Mahmood Bajwa

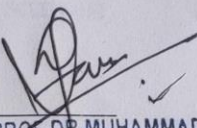
Date: 27/03/2024

Signature (HOD): 

**HOD Geotechnical Engineering**  
NUST Institute of Civil Engineering  
School of Civil & Environmental Engineering  
National University of Sciences and Technology  
Date: 27/03/2024

Signature (Associate Dean): 

**Dr. S. Muhammad Jamil**  
Associate Dean  
NICE, SCEE, NUST  
Date: 01/4/24


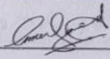
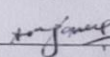
Signature (Principal & Dean) 

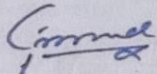
**PROF. DR. MUHAMMAD IRFAN**  
Principal & Dean  
SCEE, NUST  
Date: 02 APR 2024

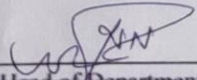
**National University of Sciences and Technology**  
**MASTER'S THESIS WORK**

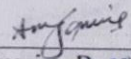
We hereby recommend that the dissertation prepared under our  
Supervision by: **Syed Jamal Arbi**, Regn No. **00000328433**  
Titled: **"Prediction of Axial Bearing Capacity of Piles using ML Algorithms  
Tuned Through Random and Grid Search"** be accepted in partial fulfillment  
of the requirements for the award of degree with B+ Grade

**Examination Committee Members**

- |    |                               |  |
|----|-------------------------------|--|
| 1. | Name: Dr. Numan Khurshid      | Signature:   |
| 2. | Name: Dr. Umar Saeed          | Signature:   |
| 2. | Name: Dr. Syed Muhammad Jamil | Signature:  |

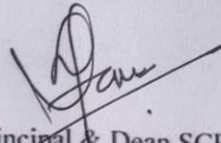
Supervisor's name: Dr. Tariq Mahmood Bajwa      Signature:  27/03/2024

  
Head of Department  
**HoD Geotechnical Engineering**  
NUST Institute of Civil Engineering  
School of Civil & Environmental Engineering  
National University of Sciences and Technology

  
(Associate Dean)  
**Dr. S. Muhammad Jamil**  
Associate Dean  
NICE, SCEE, NUST

**COUNTERSIGNED**

Date: 02 APR 2024

  
Principal & Dean SCEE  
PROF DR MUHAMMAD IRFAN  
Principal & Dean  
SCEE, NUST

## CERTIFICATE OF APPROVAL

This is certified that the research work presented in this thesis, entitled "Prediction of axial bearing capacity of pile using sophisticated ML algorithms tuned through Random and Grid Search" was conducted by Mr./Ms. Syed Jamal Arbi under the supervision of Dr. Tariq Mahmood Bajwa.

No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the Department of Geotechnical Engineering in partial fulfilment of the requirements for the degree of **Master of Philosophy in Geotechnical Engineering**. NUST Institute of Civil Engineering (NICE), School of Civil & Environmental Engineering, National University of Science & Technology, Islamabad.

Student Name: Syed Jamal Arbi

Signature: Jamal

Examination Committee:

- a) Dr. Syed Muhammad Jamil  
(Associate Dean, SCEE, NUST)
- b) Dr. Numan Khurshid  
(Assistant Professor, SEecs, NUST)
- b) Dr. Umer Saeed  
(Associate Professor, SCEE, NUST)

Signature: Syed Muhammad Jamil

Signature: Dr. Numan Khurshid

Signature: Dr. Umer Saeed

Dr. Tariq Mahmood Bajwa

(Supervisor)

Head of Department: Dr. Badee Alshameri

Associate Dean: Dr. Syed Muhammad Jamil

Principal & Dean: Dr. Muhammad Irfan

Signature: Dr. Tariq Mahmood Bajwa

Signature: Dr. Badee Alshameri  
**HoD Geotechnical Engineering**  
NUST Institute of Civil Engineering  
School of Civil & Environmental Engineering  
National University of Sciences and Technology

Signature: Dr. S. Muhammad Jamil  
**Associate Dean**  
**NICE, SCEE, NUST**

Signature: \_\_\_\_\_

Dr. Muhammad Irfan  
**PROF DR MUHAMMAD IRFAN**  
Principal & Dean  
SCEE, NUST

**Author's Declaration**

I Syed Jamal Arbi hereby state that my MS thesis titled

"PREDICTION OF AXIAL BEARING CAPACITY OF PILES USING

SOPHISTICATED ML ALGORITHMS TUNED THROUGH RANDOM AND

GRID SEARCH"

is my own work and has not been submitted previously by me for taking any degree from this University **National University of Sciences and Technology, NUST, Islamabad**, or anywhere else in the country/ world.

At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

Name of Student: Syed Jamal Arbi

Date: 27/03/24



*DEDICATED*

*To*

*MY MOTHER*



## **ACKNOWLEDGEMENTS**

All praise to Almighty Allah, who gave me the courage and power to complete this research work and gratitude to the last Prophet MUHAMMAD (P.B.U.H).

I extend my wholehearted gratitude to my revered supervisor, Dr. Tariq Mahmood Bajwa, for his technical guidance, indelible help and valuable feedback throughout this study. I also express my utmost gratitude to GEC members Dr. Nauman Khurshid, Dr. Umer Saeed, and Dr. Syed Muhammad Jamil for providing helpful feedback on various aspects of this study from time to time.

I would like to acknowledge my colleagues Nazeer Alam, Muhammad Rashid, Waqar Saleem, Muhammad Shahroz Khalid, and Rana Noman for their support and for making my stay at NUST pleasant and memorable. I also thank my life partner, Syeda Arifa, for her continuous support in completing my studies.

## ABSTRACT

Pile foundations support structures by transferring loads to deep sub-surface strata, designed to bear the maximum design load without failure. Recent studies are focused on developing innovative models to estimate the pile capacity on efficient ground in less time. The pile load tests are difficult to perform and time-consuming. So, this study aims to address existing gaps in geotechnical engineering research, specifically in pile strength estimation, by deploying advanced machine learning algorithms, namely Random Forest (RF), Support Vector Regression (SVR), and Xtreme Gradient Boost (XG Boost), which are meticulously fine-tuned using hyperparameter optimization techniques, such as Grid Search (GS) and Random Search (RS). The models were formulated in a high-level programming language, namely Python. The model's efficacy was assessed through Root Mean Square Error (RMSE), Coefficient of Determination ( $R^2$ ), and Standard Deviation (SD). The test results show that each model performs well; however, the XGBoost algorithm shows higher efficacy, with high accuracy on the data sets ( $R^2 = 0.933$ ).

**Keywords:** Machine learning, Pile bearing capacity, Driven piles, Pile load test, Hyperparameter tuning.

# TABLE OF CONTENTS

<b>ABSTRACT</b> .....	i
LIST OF FIGURES .....	v
LIST OF TABLES .....	vii
LIST OF CODE SNIPPETS .....	viii
LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS.....	1
1 Chapter 1: Introduction.....	1
1.1 General .....	1
1.2 Need of research.....	3
1.3 Contribution to industry .....	3
1.4 Objective of the research work.....	3
1.5 Scheme of chapters.....	4
2 Chapter 2: Literature Review .....	6
2.1 General .....	6
2.2 Overview of Piles .....	6
2.3 Techniques for Pile Bearing Capacity Estimation.....	7
2.4 Background of Machine Learning.....	7
2.4.1 What are Machine Learning Algorithms?.....	8
2.5 Different Machine Learning Algorithms .....	10
2.5.1 Random Forest Algorithm .....	10
2.5.2 Support Vector Regression (SVR) .....	13
2.5.3 XGBoost Algorithm .....	16
2.6 Model Parameters.....	19
2.6.1 What are Hyperparameters in Machine Learning? .....	19
2.6.2 Techniques to Tune Hyperparameters:.....	20
2.6.3 Random Forest Regressor Hyperparameters: .....	20
2.6.4 Support Vector Regression (SVR) Hyperparameters: .....	20
2.6.5 XGBoost Regressor Hyperparameters:.....	21
2.6.6 Random Search for Hyperparameters:.....	21

2.6.7	Grid Search for Hyperparameters: .....	23
2.7	Model Evaluation .....	26
2.7.1	Cross-Validation:.....	26
2.7.2	Model Metrics:.....	30
2.7.3	Validation Curves:.....	31
2.7.4	Learning Curves:.....	32
2.8	Already Research .....	34
3	Chapter 3: Methodology .....	38
3.1	Data Collection.....	38
3.2	Outlier Detection and Rectification Using Gaussian Approximation: .....	40
3.3	Data Normalization: .....	41
3.4	Data Partition: .....	41
3.5	Machine Learning Models .....	42
3.6	Hyperparameter Tuning Through Random and Grid Search .....	42
3.7	Tuned Model Evaluation Through Learning Curves.....	46
3.8	Evaluation of Tuned Models Through Cross Validation Scores .....	49
3.9	Evaluation of Tuned Models on Validation Data .....	49
4	Chapter 4: Results and discussion .....	51
4.1	Preprocessed data .....	51
4.1.1	Central Tendency Measures .....	51
4.1.2	Data Distribution.....	51
4.1.3	Range .....	51
4.1.4	Variable-specific Observations .....	52
4.2	Normalized Data .....	54
4.3	Data Splitting.....	54
4.4	Results for Evaluation of Tuned Models Through Learning Curves .....	56
4.4.1	Evaluation of Learning Curves for Random Forest.....	56
4.4.2	Evaluation of Learning Curves for SVR.....	58
4.4.3	Evaluation of Learning Curves for XGBoost .....	60
4.5	Results of Evaluation for Tuned Models Through Cross Validation Scores .....	62
4.6	Evaluation of Models on Validation Data .....	65

4.7	Benefits.....	68
4.8	Limitations .....	68
5	Chapter 5: Conclusions.....	69
5.1	Development of Predictive Model .....	69
5.2	Transparent Mapping: .....	69
5.3	Establishment of Robust Models.....	69
5.4	Comparison of Machine Learning Models.....	69
5.5	Effective Tuning and Validation.....	69
5.6	Future Recommendations.....	70
5.6.1	Exploration of Other Algorithms .....	70
5.6.2	Advanced Hyperparameter Tuning.....	70
5.6.3	Feature Engineering.....	70
5.6.4	Cross-Domain Application.....	70
5.6.5	Incorporation of Domain Knowledge .....	70
5.6.6	Evaluation with More Diverse Data .....	70
	References.....	71

## LIST OF FIGURES

<b>Figure 1.1:</b> Dynamic load test and static load test at the site .....	2
<b>Figure 1.2:</b> Scheme of chapters.....	5
<b>Figure 2.1:</b> Categories of machine learning algorithms.....	8
<b>Figure 2.2:</b> Random Forest prediction model (Random Forests. Random Forests Is a Powerful Machine...   by Dr. Roi Yehoshua   Medium, n.d.).....	11
<b>Figure 2.3:</b> SVR algorithm model (Support Vector Regression (SVR)   Analytics Vidhya, n.d.)	14
<b>Figure 2.4:</b> XGBoost (Jiang et al., 2021).....	17
<b>Figure 2.5:</b> Techniques for hyperparameter tuning.....	21
<b>Figure 2.6:</b> Random Search layout (A Comparison of Grid Search and Randomized Search Using Scikit Learn   by Peter Worcester   Medium, n.d.) .....	22
<b>Figure 2.7:</b> Grid Search layout (A Comparison of Grid Search and Randomized Search Using Scikit Learn   by Peter Worcester   Medium, n.d.) .....	24
<b>Figure 2.8:</b> Grid Search CV .....	25
<b>Figure 2.9:</b> Process of Cross Validation .....	27
<b>Figure 2.10:</b> K-Fold .....	28
<b>Figure 2.11:</b> Stratified K-Fold process.....	28
<b>Figure 2.12:</b> Time-Series Split process .....	29
<b>Figure 2.13:</b> Group K-Fold process .....	30
<b>Figure 2.14:</b> Types of Validation Curves .....	32
<b>Figure 2.15:</b> Types of Learning Curves .....	33
<b>Figure 3.1:</b> Illustration of experimental layout .....	40
<b>Figure 4.1:</b> Graphical statistical summary before and after outlier removal for ‘Z <sub>p</sub> ’ .....	52
<b>Figure 4.2:</b> Graphical statistical summary before and after outlier removal for ‘Z <sub>2</sub> ’ .....	53
<b>Figure 4.3:</b> Graphical statistical summary before and after outlier removal for ‘N <sub>sh</sub> ’.....	53
<b>Figure 4.4:</b> Graphical statistical summary before and after outlier removal for ‘N <sub>t</sub> ’ .....	53
<b>Figure 4.5:</b> Learning Curves for RF during RS tuning .....	58
<b>Figure 4.6:</b> Learning Curves for RF during GS tuning.....	58
<b>Figure 4.7:</b> Learning Curves for SVR during RS tuning .....	59
<b>Figure 4.8:</b> Learning Curves for XGBoost during RS tuning.....	62
<b>Figure 4.9:</b> Learning Curves for XGBoost during GS tuning.....	62

<b>Figure 4.10:</b> Predictions vs real values for RS-RF & GS-RF .....	64
<b>Figure 4.11:</b> Predictions vs real values for RS-SVR & GS-SVR .....	64
<b>Figure 4.12:</b> Predictions vs real values for RS-XGBoost & GS-XGBoost.....	64
<b>Figure 4.13:</b> Predictions vs real values for RS-RF & GS-RF .....	67
<b>Figure 4.14:</b> Predictions vs real values for RS-SVR & GS-SVR .....	67
<b>Figure 4.15:</b> Predictions vs real values for RS-XGBoost & GS-XGBoost.....	67
<b>Figure 4.16:</b> Taylor Diagram.....	68

## LIST OF TABLES

<b>Table 3.1:</b> Introduction of input parameters .....	38
<b>Table 3.2:</b> Statistical summary of input data .....	39
<b>Table 3.3:</b> Hyperparameters for RF .....	43
<b>Table 3.4:</b> Hyperparameters for SVR .....	43
<b>Table 3.5:</b> Hyperparameters for XGBoost .....	43
<b>Table 4.1:</b> Statistical summary of data after outlier removal .....	53
<b>Table 4.2:</b> Statistical summary of normalized data .....	54
<b>Table 4.3:</b> Statistical summary of training and testing data .....	54
<b>Table 4.4:</b> Statistical summary of validation data .....	55
<b>Table 4.5:</b> Optimal hyperparameters for RF selected through GS and RS .....	55
<b>Table 4.6:</b> Optimal hyperparameters for SVR selected through GS and RS .....	55
<b>Table 4.7:</b> Optimal hyperparameters for XGBoost selected through GS and RS .....	56
<b>Table 4.8:</b> Results for Cross Validation Scores .....	64
<b>Table 4.9:</b> ML models predictions on validation data .....	66
<b>Table 4.10:</b> Proposed models comparison with literature .....	66



## LIST OF CODE SNIPPETS

<b>Code Snippet 2.1:</b> Random Forest implementation in Python .....	13
<b>Code Snippet 2.3:</b> SVR implementation in Python.....	16
<b>Code Snippet 2.4:</b> XGBoost implementation in Python.....	19
<b>Code Snippet 2.5:</b> Randomized Search CV implementation in Python .....	23
<b>Code Snippet 2.6:</b> <i>GridSearchCV()</i> implementation in Python .....	26
<b>Code Snippet 3.1:</b> Splitting of data into training and validation.....	41
<b>Code Snippet 3.2:</b> Random Search Snippet for Random Forest .....	44
<b>Code Snippet 3.3:</b> Grid Search Snippet for Random Forest .....	44
<b>Code Snippet 3.4:</b> Random Search Snippet for SVR .....	45
<b>Code Snippet 3.5:</b> Grid Search Snippet for SVR .....	45
<b>Code Snippet 3.6:</b> Random Search Snippet for XGBoost.....	46
<b>Code Snippet 3.7:</b> Grid Search Snippet for XGBoost.....	46
<b>Code Snippet 3.8:</b> Learning Curve with $R^2$ on Y-axis for Random Forest Tuned with RS.....	47
<b>Code Snippet 3.9:</b> Learning Curve with RMSE on Y-axis for Random Forest Tuned with RS..	48
<b>Code Snippet 3.10:</b> Learning Curve with $R^2$ on Y-axis for SVR Tuned with GS .....	48
<b>Code Snippet 3.11:</b> Mean, std for train & test scores of all algorithms.....	49
<b>Code Snippet 3.12:</b> Evaluation of tuned model on validation data .....	50

## LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

No.	Abbreviations	Description
1	ML	Machine Learning
2	SVR	Support Vector Regression
3	RF	Random Forest
4	PSO	Particle Swarm Optimization
5	WOA	Whale Optimization Algorithms
6	ANN	Artificial Neural Network
7	DNN	Deep Neural Network
8	GA	Genetic Algorithm
9	DLNN	Deep Learning Neural Networks
10	RMSE	Root Mean Square Error
11	$R^2$	Coefficient of Determination
12	MAE	Mean Absolute Error
13	SPT	Standard Penetration Test
14	LSSVR	Least Squares Support Vector Regression
15	OBDFP	Opposition-Based Differential Flower Pollination
16	MARS	Multivariate Adaptive Regression Splines
17	RBFNN	Radial Basis Function Neural Network

# Chapter 1: Introduction

## 1.1 General

In the world of construction and civil engineering, when we refer to the term "foundation," we are speaking about the element of a structure that connects it to the ground, ensuring stability and strength. While there are different types of foundations, piles stand out as one of the most integral components, especially for structures in challenging terrains or with specific load requirements.

Piles are long, slender, cylindrical structural elements made from materials like steel, concrete, or timber. They serve to transfer the weight of a structure deep into the ground, bypassing weak or compressible soil layers that are inadequate for supporting the structure's weight.

(Adi Pusat Pengelolaan Sumberdaya Lahan et al., 2009) explains that piles are used to transfer loads from shallow depths to deeper, more supportive layers of soil. They can be bearing piles, which penetrate a stratum of good bearing capacity, or friction piles, which rely on friction on the sides of the pile for support. By doing so, piles ensure the building or infrastructure remains stable and secure. (Verma & Gill, 2018) discusses different types of pile foundations, including bored pre-cast piles and driven steel piles. Piles are needed for high load structures, waterfront structures, expansion of existing structures and areas prone to natural calamities.

The bearing capacity of a pile is the maximum load a pile can support without risking structural failure or excessive settlement. In essence, determining the correct bearing capacity is of paramount importance. (You-xiang, 2008) emphasizes the importance of rational and economical pile foundation design, including the selection of the appropriate pile pattern and length. An underestimate might lead to structural failures, while an overestimate could lead to overspending on unnecessary materials or depth. (Chao et al., 2020) explains that pile load tests are a reliable method for determining the ultimate bearing capacity of a single pile and accurately reflecting its stress condition and deformation characteristics. Essentially, it is a procedure used to evaluate the behavior of a pile under various load conditions. By understanding this behavior, engineers and geotechnical experts can determine the ultimate load carrying capacity of the pile and its suitability for a specific project. The test also aids in ascertaining any potential settlement or recovery the pile might experience under the applied loads. (Rajapakse, 2008) discusses the general procedures and equipment required for conducting pile load tests, including driven piles, hydraulic jacks, and load

indicators. The pile load test can be categorized into two primary types based on the nature of the load: Static Load Test and Dynamic Load Test. The Static Load Test involves applying a gradual load to a pile to determine its bearing capacity, revealing the pile-soil interaction over time. It's precise but time-consuming and costly, suitable for detailed assessments of pile performance under actual loading conditions. In contrast, the Dynamic Load Test rapidly assesses a pile's capacity by analyzing its response to a high-speed impact, offering a quicker and less expensive option, though it requires calibration against more accurate tests for precise interpretation. This method is ideal for initial evaluations and large-scale projects where time and cost efficiency are paramount.



**Figure 1.1:** Dynamic load test and static load test at the site

The pile load test is indispensable for ensuring the reliability of pile foundations. It's a proactive measure, confirming whether the design assumptions align with the ground realities. By providing a clear picture of how a pile responds to loads, the test aids in averting potential structural failures, ensuring the longevity and safety of the construction. (Stirrat, 1959) emphasizes the importance of pile loading tests in determining the allowable bearing capacity of piles and achieving cost savings in construction projects.

This study focuses on exploring and enhancing the methods of pile strength estimation by utilizing machine learning algorithms, namely Random Forest (RF) (Breiman, 2001), Support Vector Regression (SVR), and Xtreme Gradient Boost (XGBoost). Each of these algorithms has its roots in prior research. We have turned to hyperparameter optimization methods to refine these algorithms, specifically Grid Search (GS) and Random Search (RS). Having been validated in prior studies, these techniques show promise in optimizing intricate models. During optimization,

hyperparameter selection is performed through detailed research from the literature. After this, range for each hyperparameter is selected through trial and error. During this process, the hyperparameters are also validated through validation curves. Learning curves are also made for optimal models against training samples to keep the process in control. This way, it can be observed that either the tuned models are overestimated or underestimated. All this process is done, and models are again validated on unseen data which was never shown to models during training.

Using a blend of machine learning techniques, our objective is to craft a model that encapsulates accuracy, resilience, and reliability. To achieve this, we delve deep into Grid Search and Random Search methods to refine the hyperparameters of diverse machine learning frameworks. Rigorous evaluations, including learning and validation curves and cross-validation metrics anchor our approach. The goal is to augment the dependability of pile load tests and address the nuances overlooked in prior research.

## **1.2 Need of research**

Machine learning methods for estimating pile strength have shown potential but also presented various challenges. Previous techniques have sometimes been limited by the specific conditions or locations for which they were developed. While a method might excel in one environment, it might not perform as well in another due to each terrain's distinct complexities. Solely depending on single machine learning tools or individual metaheuristic optimizers like GA, PSO, or WOA can lead to suboptimal results. This might result from getting stuck in local minima or the inherent complexities of tuning these optimizers. Hence, there's a need to explore diverse optimization methods and consider multiple pathways for model refinement.

## **1.3 Contribution to industry**

Recently, in a year, the artificial intelligence revolution has amazed the world. In no time, it will change every industry and not only fasten the outcomes but also change the human ideas for industry. By observing the upcoming wave, this research pushes civil engineering to the next step. It is not the one-end solution but one of the steps to a new future.

## **1.4 Objective of the research work**

This study is centered around advancing the potential of machine learning models like Random Forest (RF), Support Vector Regression (SVR), and Xtreme Gradient Boost (XG Boost), each fine-

tuned using hyperparameter optimization with Grid Search (GS) and Random Search (RS). The main highlights and directions of this research include:

- 1) Utilizing multiple modelling techniques, emphasizing the advantage of merging various models. This fusion potentially enhances predictive accuracy.
- 2) Detailing every research step, from the specifics of hyperparameter adjustments to validation stages, ensures a roadmap for reproducibility, facilitating future scientific endeavours.
- 3) An exhaustive model evaluation is conducted, rectifying limitations seen in earlier studies. This method promises accurate outcomes and the models' adaptability in diverse contexts.
- 4) Implementing cross-validation and validating against novel data showcases our dedication to creating models that generalize well—a critical aspect for applications in geotechnical engineering.

## **1.5 Scheme of chapters**

The details about the chapters are given below. The scheme of the chapters is also shown in Figure 1.2.

**Chapter. 1:** This chapter highlights the precise summary of the research work.

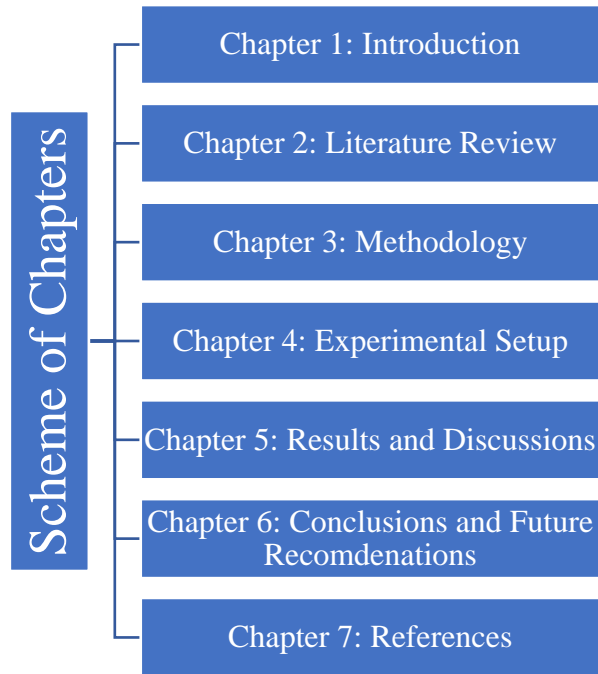
**Chapter. 2:** This chapter reports a literature review of pile-bearing capacity determination techniques, machine learning in geotechnical engineering, and previous research on pile load tests.

**Chapter. 3:** This chapter discusses research methods involved in the formulation of machine learning algorithms, and model evaluations, etc.

**Chapter. 4:** This chapter discusses the experimental setup to achieve the research objectives.

**Chapter. 4:** This chapter reports the results and discussions.

**Chapter. 5:** This chapter summarizes the conclusions and key recommendations from the study.



**Figure 1.2:** Scheme of chapters

## **Chapter 2: Literature Review**

### **2.1 General**

Pile foundations, commonly utilized to anchor structures deep within the earth, are pivotal in many construction ventures. The precise prediction of their load-bearing capacity is a pressing task faced by geotechnical engineers. Over the years, an assortment of methodologies has been introduced to gauge this capacity. The gamut of these methods includes static analyses, dynamic methodologies, empirical formulations, and in-situ tests, each with its own merits and demerits.

### **2.2 Overview of Piles**

Piles transfer structural loads to stronger soil layers or rock strata deep below the surface. These elongated columns, commonly formed from materials like steel, concrete, or wood, are critical in supporting structures where surface soil conditions are unsuitable for conventional shallow foundations. The selection and design of piles hinge significantly on the understanding of soil mechanics and site geology. Engineers consider factors such as soil type, whether it be clay, sand, silt, or gravel, and properties like cohesion, density, and shear strength. There are various types of piles, each with distinct characteristics and application scenarios. Driven piles, for instance, are hammered into the ground and are often used for their durability and strength, while bored piles, created by excavating a hole and filling it with concrete, are preferred in urban areas to reduce noise and vibration.

The installation of piles is a critical phase, involving methods like driving, drilling, or screwing, each having a unique impact on the surrounding soil and the overall stability of the foundation. The load transfer mechanism is a key aspect of pile design; end-bearing piles rely on a firm stratum at a certain depth to bear the load, whereas friction piles transfer loads through skin friction along their length. The integrity and performance of piles are ensured through rigorous testing methods, such as load tests and integrity tests, to verify that they meet the design specifications and safety standards. Additionally, environmental, and economic considerations play a vital role in shaping decision-making strategies. The impact of pile installation on groundwater, soil, and the surrounding ecosystem, along with factors like cost-effectiveness and longevity, are crucial in the selection of the appropriate pile type.



The field of pile foundation engineering continuously evolves, addressing challenges like unexpected soil conditions and integrating innovations such as the use of recycled materials or advanced sensing technologies for real-time monitoring of pile health. This dynamic nature of geotechnical engineering underscores its significance in the realm of construction and infrastructure development, ensuring structures stand on solid foundations capable of withstanding various environmental and load conditions.

### **2.3 Techniques for Pile Bearing Capacity Estimation**

Empirical Formulas are derived from a wealth of past data and observations. Liu et al. (2019) emphasized the potential pitfalls of relying on these alone, citing the myriad of variables involved – from soil mechanics to geotechnical circumstances. The diverse nature of these factors can occasionally lead to inaccuracies in empirical results. Static and dynamic field test present a more hands-on approach, assessing piles under specific conditions. While they bring forth real-time evaluations, the complexities involved can sometimes lead to discrepancies. H. Nguyen et al. (2023) and Shooshpasha et al. (2013) lauded the pile load test (method that evaluates complete pile settlements under a static load). The accuracy of this test is largely attributed to its reflection of real-world driven pile installations. However, Hoang et al. (2022) pointed out that its high cost and lengthy execution time might deter its use in smaller projects.

Since the 1970s, there's been a surge in in-situ test methods to measure soil properties. Notably, the Standard Penetration Test (SPT) stands out for its widespread utilization in determining the bearing capacities of piles. Studies for example, by Ammar et al. (2013) and Bouafia & Bouafia (2002) have reinforced the value of the SPT in this realm, emphasizing its potential as a more practical alternative to costly pile load testing.

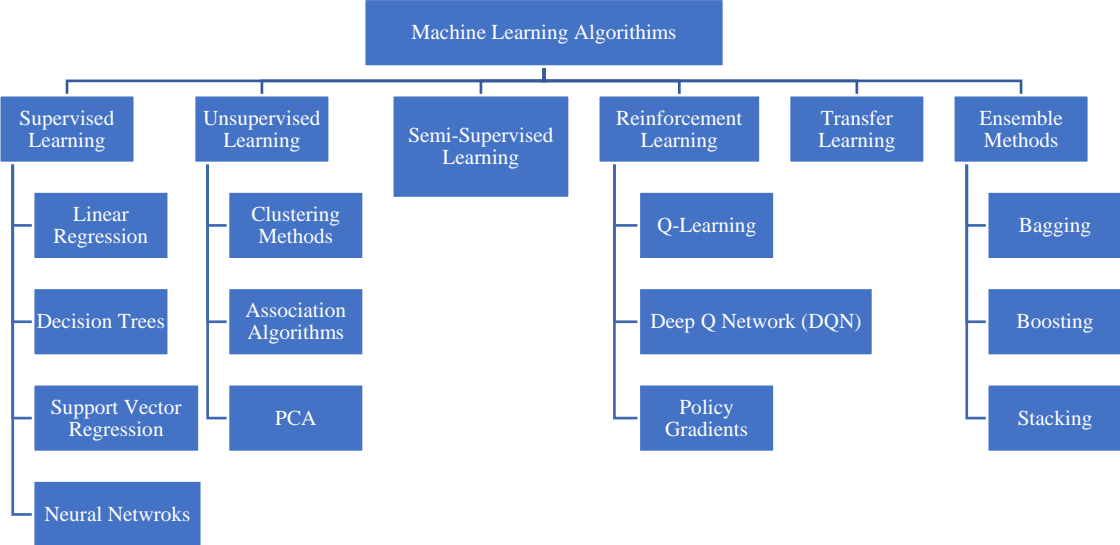
### **2.4 Background of Machine Learning**

Machine Learning, a branch of artificial intelligence, emphasizes the creation of algorithms and models enabling computers to undertake tasks autonomously, without the need for specific programming for each task. Basically, a machine learns from data to find patterns, and make predictions or decisions without any human interference. These machine learning algorithms have shown immense potential in solving complex problems across various domains. (Pulina, 2010) suggests the application of diverse machine learning algorithm portfolios for tackling intricate

tasks in robotics and automated reasoning. Unlike traditional algorithms that follow a strict set of instructions, ML algorithms adapt and learn from the data they're provided. This means they can discover hidden patterns or subtle relationships in vast amounts of data that might be challenging or time-consuming for humans to discern. Once trained, many ML models can adapt over time by learning from new data, allowing them to stay relevant in changing environments. ML algorithms are designed to take a broad view from the training data to unseen scenarios, which means they can handle a variety of situations do not present in the initial training set. Deep learning models, which fall under the umbrella of ML, have the capability to autonomously extract features from unprocessed data, eliminating the need for manual feature engineering. This attribute renders them highly effective in tasks such as image and speech recognition. ML models, especially specific types like RF or Neural Networks (NN), can handle high-dimensional data effectively, capturing interactions between various features. For tasks like data clustering or image recognition, manually defining rules or patterns would be cumbersome. ML models can automatically and efficiently handle these tasks after being trained on representative data.

**2.4.1 What are Machine Learning Algorithms?**

Machine learning algorithms consist of specific rules and procedures employed by AI systems to execute various tasks. These tasks frequently include discovering fresh insights and patterns within data, or forecasting outcomes based on a given set of input data. The general categorization of machine learning algorithms are summarized in Figure 2.1 and explained below.



**Figure 2.1:** Categories of machine learning algorithms

**1. Supervised Learning:** In supervised learning, algorithms learn from labeled data, where the desired output is known. Once trained, they aim to make predictions for new, unseen data. Examples are as follows:

- Neural Networks
- Decision Trees
- Support Vector Machines
- Linear Regression

**2. Unsupervised Learning:** Algorithms that work with unlabeled data, focusing on finding patterns or structures. Examples are as follows:

- Clusters (K-Means)
- Association Algorithms like Apriori
- Dimensionality Reduction Methods (Principal Component Analysis)

**3. Semi-Supervised Learning:** These algorithms employ a combination of labeled and unlabeled data during the training phase, typically utilizing a smaller quantity of labeled data alongside a more substantial volume of unlabeled data.

**4. Reinforcement Learning:** Algorithms designed to learn through interaction with their environment rely on feedback mechanisms, where they receive either rewards or penalties to guide their learning process. Examples are as follows:

- Q-Learning
- Deep Q Network (DQN)
- Policy Gradients

**5. Transfer Learning:** Instead of starting the learning process from scratch, these methods leverage knowledge from a previously trained model on a different but related task.

**6. Ensemble Methods:** These algorithms integrate various machine learning methods into a single predictive model, aiming to reduce variance and bias, or to enhance the accuracy of predictions. Examples are as follows:

- Bagging (Bootstrap Aggregating) like Random Forest
- Boosting like AdaBoost, Gradient Boosting Machines (GBM), and XGBoost

- Stacking

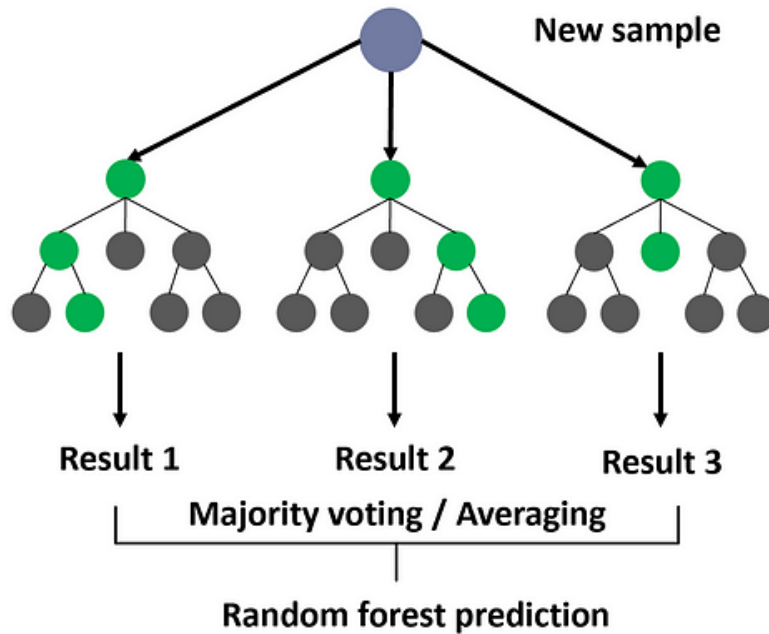
Different algorithms possess their unique advantages and limitations, making them suitable for specific kinds of tasks. The selection of an algorithm typically hinges on factors like the data size, quality, and characteristics, the nature of the task at hand, and the expected results. ML, a notable arm of artificial intelligence, has shown promise in reshaping the paradigm of geotechnical engineering. At its core, ML delves into historical data, discerning patterns and insights akin to human cognitive processes. Several studies in current years have supported the use of machine learning as a potential surrogate for traditional empirical and field test methods in estimating structural load capacities. Researchers such as (Alwanas et al., 2019; Mangalathu et al., 2022) have explored its potential advantages.

A crucial and first step in ML is data processing, that transforms raw data into a clean and structured format for model training (Y et al., 2022). Proper data preprocessing improves the accuracy and efficiency of models. Given the saying "garbage in, garbage out", without proper preprocessing, even the most sophisticated models may perform poorly or give misleading results. Data preprocessing includes techniques like data cleaning, normalization, transformation, and feature selection (Bilal et al., 2022; Lawatre, 2021; Mundargi et al., 2023; Y et al., 2022).

## **2.5 Different Machine Learning Algorithms**

### **2.5.1 Random Forest Algorithm**

It is a versatile and widely used algorithm which belongs to the ensemble learning category. It uses a collection of decision trees to perform regression and classification tasks (Liu et al., 2012) and a graphical visualization of RF model is shown in Figure 2.2. RF has been applied to many domains, including image classification, generating continuous field datasets, detection of spam mails, detection of credit card fraud, classification of genes, detection of network intrusion, email spam detection, gene classification, credit card fraud detection, and text classification (Horning, 2010; Zakariah, 2014).



**Figure 2.2:** Random Forest prediction model (Random Forests. Random Forests Is a Powerful Machine... | by Dr. Roi Yehoshua | Medium, n.d.)

Random Forest was introduced by Leo Breiman in 2001. It's an extension of his earlier work on bootstrap aggregating (or bagging) (Breiman, 1996).

### 2.5.1.1 Mathematical Formulas in the Background:

- **Entropy:**

It's a measure of disorder or impurity. For a binary classification with probabilities  $p$  and  $1 - p$ :

$$Entropy(S) = -p \log_2(p) - (1 - p) \log_2(1 - p) \quad (2.1)$$

**Gini Impurity:**

Another measure of impurity. For a binary classification:

$$Gini(S) = 1 - [p^2 + (1 - p)^2] \quad (2.2)$$

**Information Gain:**

The information that can increase the level of certainty after splitting.

$$\text{Information Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \times \text{Entropy}(S_v) \quad (2.3)$$

### 2.5.1.2 How Does It Work to Solve Problems?

- **Bootstrap Sampling:**

Randomly samples the dataset with replacement to create multiple subsets.

- **Building Trees:**

For each subset, it grows a decision tree. When splitting nodes, instead of searching for the best feature, it searches among a random subset of features for the best feature.

In summary, RF builds multiple decision trees using bootstrap samples of the training data. It uses random feature selection to determine splits in the trees, which reduces correlation between trees (Mohana et al., 2021).

- **Prediction:**

- **Classification:** Each tree casts a ‘vote’ for the class, and the class with the highest votes is the winner and tagged as ‘prediction’.
- **Regression:** The average prediction of all the trees is the forest's prediction.

### 2.5.1.3 How Does Scikit-Learn Implementation of Random Forest Work?

Random Forest implementation is designed to be user-friendly and efficient. Here's a simplified breakdown:

**1. Initialization:** When you create a ‘Random Forest Classifier’ or ‘Random Forest Regressor’, you can specify parameters like the number of trees (‘n\_estimators’), criteria for splitting (‘criterion’ can be “gini” or “entropy” for classification), maximum depth of trees, etc.

**2. Fitting:** When you call the ‘fit’ method, it begins the process of bootstrap sampling and building individual trees.

**3. Prediction:** The ‘predict’ method aggregates predictions from individual trees. For classification, it uses majority voting, and for regression, it averages the results.

**4. Feature Importance:** Scikit-learn provides an attribute `feature_importances_` which computes the importance of each feature based on the frequency and the depth it appears in all trees.

Here's a basic example in Code Snippet 2.1.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10)
rfc.fit(X_train, y_train)

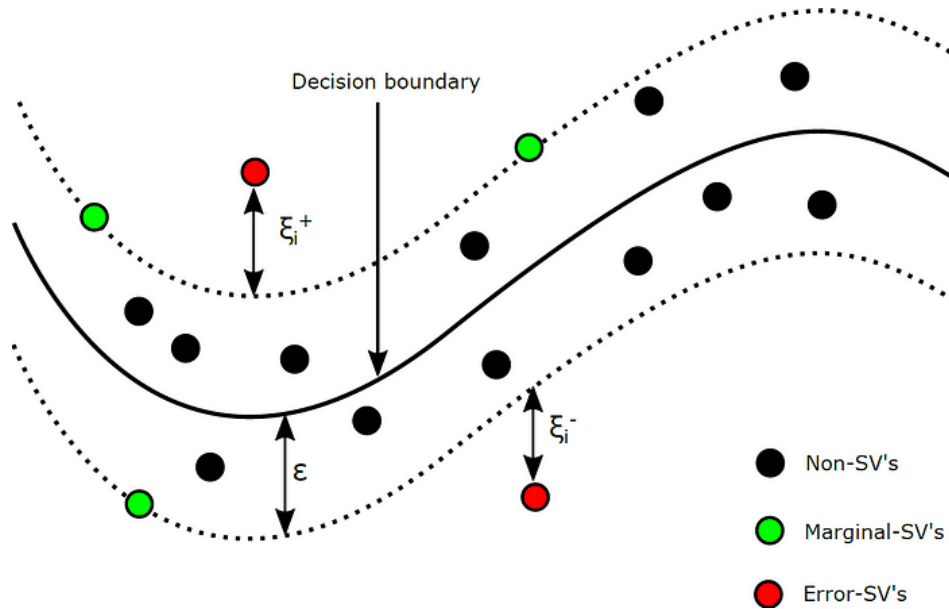
# Make predictions on test data
predictions = rfc.predict(X_test)
```

**Code Snippet 2.1:** Random Forest implementation in Python

Fundamentally, RF operates as an ensemble of decision trees, collaborating to yield more stable and precise predictive outcomes. Its ensemble nature makes it less prone to overfitting, and its versatility allows it to be used for both classification and regression tasks.

### 2.5.2 Support Vector Regression (SVR)

It's a type of Support Vector Machine which is used for regression tasks. And SVM is mainly known for classification, SVR is its adaptation for predicting continuous values. SVR, or Support Vector Regression, learns directly from the data how important different variables are in explaining the connection between inputs and outputs. This is a step away from older regression methods, which often rely on assumptions that might not match the real-world data perfectly. SVR stands out because it figures out the value of variables directly through the data it analyzes, offering a more tailored approach to understanding data relationships and shown in Figure 2.3. SVR has been applied successfully to analyze brain imaging data and reveal patterns through multiple brain regions for various disorders (Zhang & O'Donnell, 2020).



**Figure 2.3:** SVR algorithm model (Support Vector Regression (SVR) | Analytics Vidhya, n.d.)

SVM, which encompass SVR, were introduced by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in the 1960s. The concept was later refined and popularized in the 1990s, with the regression adaptation (SVR) also gaining traction.

### 2.5.2.1 Mathematical Background:

- **Linear SVR:**

The main objective is to identify a function  $f(x)$  that deviates minimally, by no more than  $\epsilon$ , from the actual training responses  $y_i$  across all the training data, while also maintaining as much simplicity or flatness as possible.

In simpler terms, SVR tries to fit the best line/hyperplane within a margin of  $\epsilon$  where the loss is equal to zero.

- **Loss Function:**

SVR uses the  $\epsilon$ -insensitive loss:

$$L_{\epsilon}(y, f(x)) = \max(0, |y - f(x)| - \epsilon) \quad (2.4)$$

- **Objective:**

- Minimize:



$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \xi_i^* \quad (2.5)$$

- Subject to:

$$y_i - f(x_i) \leq \varepsilon + \xi_i \quad (2.6)$$

$$f(x_i) - y_i \leq \varepsilon + \xi_i^* \quad (2.7)$$

$$\xi_i, \xi_i^* \geq 0 \quad (2.8)$$

Where:

$w$  is the weight vector,

$C$  is the regularization parameter,

$\xi_i$  and  $\xi_i^*$  are slack variables.

- **Kernel Trick:**

In non-linear SVR, data is mapped to a higher-dimensional space where it's linearly separable. This mapping is done using kernel functions.

### 2.5.2.2 *How Does It Work to Solve Problems?*

- **Margin and Buffer:**

SVR aims to draw a line (in a 2D space) or a hyperplane (when dealing with more dimensions) through the data so that as many data points as possible fall within a specified buffer, or margin, around this line or plane. The goal is to have this buffer—imagine it as a kind of safety zone—be just the right size: not too wide, but wide enough to include a lot of points. This method helps to ensure the model is both accurate and flexible, catching the essence of the data without being too strict or too lenient. This buffer is where the loss is considered zero.

- **Slack Variables:**

For points outside this margin, slack variables measure the magnitude of the deviation. These are penalized in the objective function to ensure minimal deviation.

### 2.5.2.3 Scikit-Learn Implementation:

SVR from sklearn is implemented in python as follows:

**Initialization:** You can initialize SVR with the desired kernel, regularization parameter  $C$ , and other parameters.

**Fitting:** The `fit` method is used to train the SVR model on the data.

**Prediction:** The `predict` method is used for making predictions on new, unseen data.

Here's a basic example using Scikit-learn's SVR shown in Code Snippet 2.3:

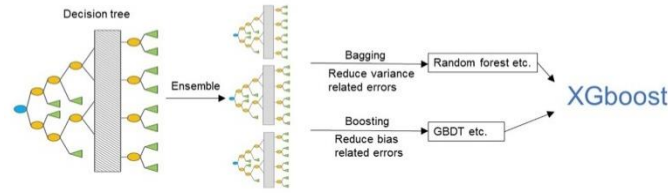
```
from sklearn.svm import SVR
# Create an SVR model with the RBF kernel
regressor = SVR(kernel='rbf', C=1.0, epsilon=0.1)
# Train the regressor on training data
regressor.fit(X_train, y_train)
# Make predictions on test data
predictions = regressor.predict(X_test)
```

**Code Snippet 2.2:** SVR implementation in Python

In summary, SVR is a powerful regression method, especially for datasets with non-linear relationships. By transforming features into higher-dimensional space (when necessary) and by defining an  $\epsilon$ -insensitive loss, SVR is designed to predict continuous values while considering both flatness and deviations.

### 2.5.3 XGBoost Algorithm

XGBoost, short for eXtreme Gradient Boosting, is a powerful tool that takes gradient boosting to the next level. It's crafted to work quickly and adaptively, making it a go-to for tackling big data challenges. Essentially, it's like having a supercharged engine for your data analysis, capable of handling tasks with speed and agility that traditional methods can't match (Chen & Guestrin, 2016). It works for both regression and classification problems. A flow chart diagram of XGBoost is shown in Figure 2.4.



**Figure 2.4:** XGBoost (Jiang et al., 2021)

While gradient boosting was a technique existed before, XGBoost brought several optimizations and became popular due to its performance and speed (Chen & Guestrin, 2016).

### 2.5.3.1 Mathematical Background

- Objective Function

At each iteration, XGBoost adds a new tree to minimize the following objective:

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \quad (2.9)$$

Where:

- $l$  = Loss function.
- $\Omega$  = Regularization term.
- $f_i$  represents each tree.
- **Regularization Term:**

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \quad (2.10)$$

Where:

- $T$  = Number of leaves in the tree.
- $w_j$  = Score on leaf  $j$ .
- $\gamma$  and  $\lambda$  are regularization parameters.

- **Gradient and Hessian:**

Instead of calculating the best split points directly, XGBoost calculates approximations using the gradient and hessian.

### *2.5.3.2 How Does It Work to Solve Problems?*

- **Boosting:**

XGBoost is a boosting algorithm. It builds trees sequentially. Each new tree tries to correct the errors made by the previous ones.

- **Regularization:**

XGBoost includes L1 (Lasso) and L2 (Ridge) regularization terms in its objective function. This prevents overfitting, making it a regularized form of boosting.

- **Handling Missing Data:**

XGBoost has an in-built routine to handle missing values. When a value is missing in a split column, it assigns a default direction (left or right) to handle the missing value.

- **Parallel Processing:**

One of the reasons for XGBoost's popularity is its capability to parallelize the construction of trees, making it faster.

- **Tree Pruning:**

Unlike other gradient boosting methods that grow trees to their maximum depth and then prune, XGBoost uses "max\_depth" parameter to grow trees to a certain depth and then starts pruning.

### *2.5.3.3 Scikit-Learn Implementation of XGBoost:*

Although XGBoost has its own Python library, it offers an API compatible with Scikit-Learn. This means you can use XGBoost models just like you would use any other model in Scikit-Learn. The basic example is shown in Code Snippet 2.4.

```
import XGBoost as xgb

# Create an XGBoost classifier
clf = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=5)

# Train the classifier on training data
clf.fit(X_train, y_train)

# Make predictions on test data
predictions = clf.predict(X_test)
```

### Code Snippet 2.3: XGBoost implementation in Python

In conclusion, XGBoost is a robust and efficient gradient boosting algorithm that's especially useful for structured/tabular data. Its performance, speed, and scalability have made it a go-to algorithm for many Kaggle competitions and real-world applications.

## 2.6 Model Parameters

### 2.6.1 What are Hyperparameters in Machine Learning?

Machine learning models have model parameters that are learned during training, such as neural network weights. In contrast, hyperparameters, which define the model's architecture, must be predefined before training (Yang & Shami, 2020). Hyperparameters are parameters that are not learned directly from the data but are set before training. They influence the behavior and performance of the learning algorithm. Examples include the learning rate, regularization coefficients, tree depth, and number of hidden layers in a neural network. Hyperparameters shape the model's ability to learn. Incorrect values can lead to underfitting (model is too simple) or overfitting (model is too complex). Using default values for model hyperparameters may lead to reduced prediction accuracy as they do not consider the dataset's dimensionality (size) or its specific features (Probst et al., 2018).

## **2.6.2 Techniques to Tune Hyperparameters:**

Researchers have proposed various search strategies to navigate this complex space, including random search, grid search, and Bayesian optimization (Claesen & Moor, 2015; Peskova & Neruda, 2019). A graphical summary of hyperparameter techniques is shown in Figure 2.5. Due to the simplicity of search-based algorithms, they are commonly used in problems with small response spaces and where evaluating hyperparameters is less resource intensive (Seifi & Niaki, 2023).

### **2.6.2.1 Grid Search:**

Systematically go through multiple combinations of hyperparameter values, train a model for each combination, and select the best combination based on performance. In this method, all parameters have an equal probability of impacting the process (Alibrahim & Ludwig, 2021).

### **2.6.2.2 Random Search:**

Randomly sample hyperparameter combinations from given ranges. Surprisingly, this can sometimes be more effective than grid search because it can explore more unique values and doesn't spend time on less influential ones (Andradóttir, 2006).

### **2.6.2.3 Bayesian Optimization:**

Bayesian Optimization (BO) is a statistical method for efficiently optimizing expensive, noisy "black-box" functions with few evaluations (Garnett, 2023). Model the objective function using a Gaussian process and choose hyperparameters to evaluate by selecting those that maximize the expected improvement.

### **2.6.2.4 Gradient-based Optimization:**

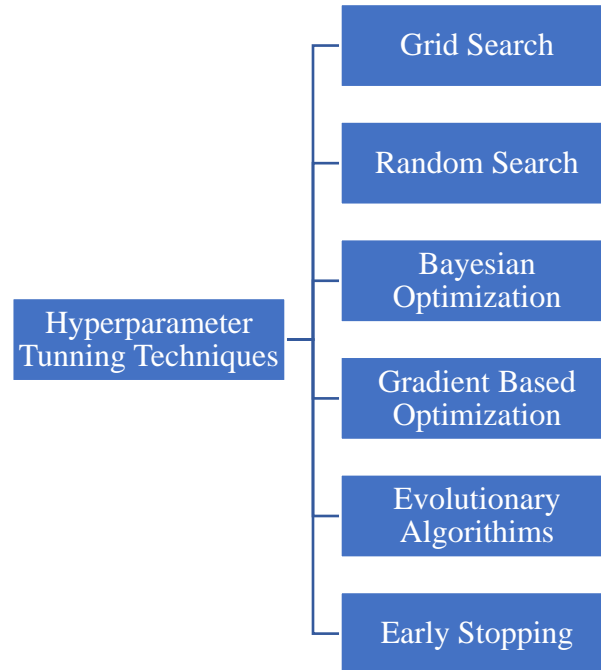
Applicable when hyperparameters are continuous. Computes gradients of the validation performance with respect to hyperparameters and adjusts them accordingly (Bengio, 2000).

### **2.6.2.5 Evolutionary Algorithms:**

An Evolutionary Algorithm (EA) falls within the realm of evolutionary computation and is classified as one of the broader categories of stochastic search algorithms (Vikhar, 2017). Treat hyperparameter optimization as a genetic algorithm problem, which maintains a population of hyperparameter sets and evolves them over time.

### 2.6.2.6 Early Stopping:

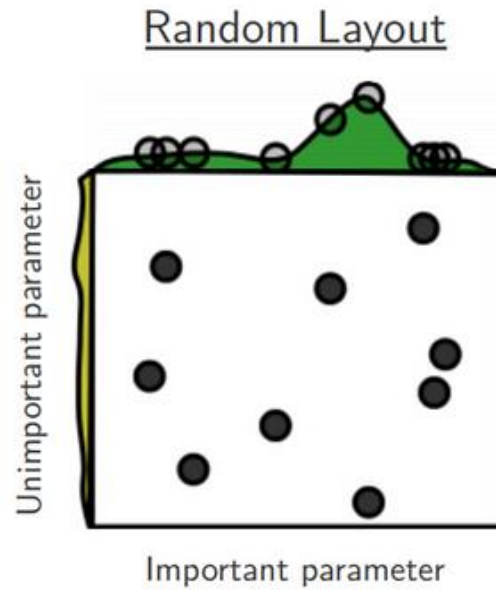
For algorithms like neural networks or gradient boosting, train for a large number of iterations but stop early if validation performance stops improving.



**Figure 2.5:** Techniques for hyperparameter tuning

### 2.6.3 Random Search for Hyperparameters:

RS is like exploring a treasure map without knowing exactly where X marks the spot. Instead of systematically checking every location, you randomly pick spots across the map, trying out different combinations of clues (or in this case, hyperparameters) for a set number of tries. With each attempt, you build and test a model based on these randomly chosen settings. It's a way of stumbling upon the best solution by chance, mixing a bit of luck with strategy to find the treasure (or the optimal model settings) more efficiently than checking every possible option and a layout is shown in Figure 2.6. As (Garnett, 2023) explains, random search sacrifices the guarantee of an optimal solution for finding a good solution quickly.



**Figure 2.6:** Random Search layout (A Comparison of Grid Search and Randomized Search Using Scikit Learn | by Peter Worcester | Medium, n.d.)

### ***2.6.3.1 How Random Search Works:***

1. Define a hyperparameter space for each hyperparameter.
2. Randomly sample from this space.
3. Train a model with the selected combination.
4. Evaluate the model using a validation set or cross-validation.
5. Repeat the process for a fixed number of iterations.
6. Select the best hyperparameters based on the performance on the validation set or cross-validation.

### ***2.6.3.2 Randomized Search Cross Validation (RSCV):***

RSCV is an implementation of random search with cross-validation. Instead of just evaluating one model for a set of hyperparameters, it evaluates the model performance across multiple folds of the data. This helps ensure that the model's performance is consistent and not the result of a specific random split of the data. In literature, there are many researchers which shows the use of RSCV as an optimization tool (Takkala et al., 2022; Vishnu et al., 2023). It works in following steps:



1. Specify a parameter grid: For each hyperparameter, define the range or distribution of values.
2. Choose the number of iterations: This determines how many different combinations to try.
3. Use cross-validation: Split the data into train/test sets multiple times and evaluate performance.
4. For each iteration, RSCV samples hyperparameters, trains the model using cross-validation, and records the performance.
5. Once all iterations are complete, RSCV provides the best set of hyperparameters and their corresponding performances. Also, here's a simple example shown in Code Snippet 2.5 using RSCV to tune hyperparameters for a RF Regressor:

```
param_dist = { 'n_estimators': np.arange(10, 200, 10), 'max_depth': [None] + list(np.arange(2, 20)),
'min_samples_split': np.arange(2, 20), 'min_samples_leaf': np.arange(1, 20), 'bootstrap': [True, False] }
rf = RandomForestRegressor()
# Set up RandomizedSearchCV
search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist, n_iter=100, cv=5,
verbose=1, n_jobs=-1)
# Execute the random search
search.fit(X, y)
# Get the best parameters and their corresponding performance
best_params = search.best_params_
best_score = search.best_score_
```

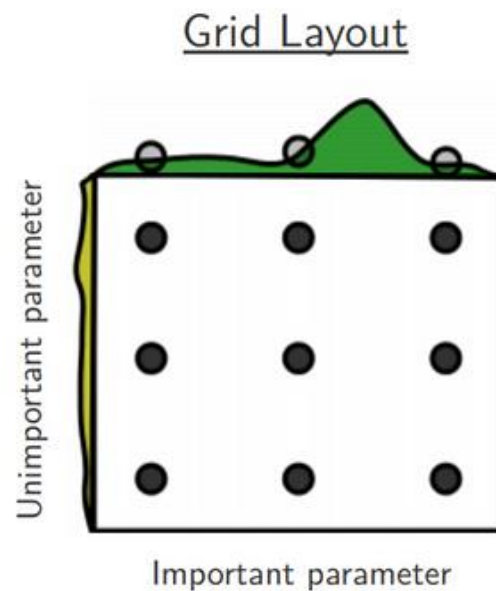
#### Code Snippet 2.4: Randomized Search CV implementation in Python

Here, 'RSCV' will perform 100 iterations, each time randomly selecting hyperparameters from 'param\_dist', training the RF Regressor, and evaluating it using 5-fold cross-validation.

#### 2.6.4 Grid Search for Hyperparameters:

Grid Search (GS) is a method for hyperparameter tuning that systematically works through multiple combinations of hyperparameter by searching over a space (Liashchynskyi & Liashchynskyi, 2019), cross-validating as it goes to determine which tune gives the best

performance. The traditional way of performing hyperparameter tuning is to use a loop nested inside another loop, which tries every single combination exhaustively as shown in Figure 2.7.



**Figure 2.7:** Grid Search layout (A Comparison of Grid Search and Randomized Search Using Scikit Learn | by Peter Worcester | Medium, n.d.)

#### 2.6.4.1 How Grid Search Works:

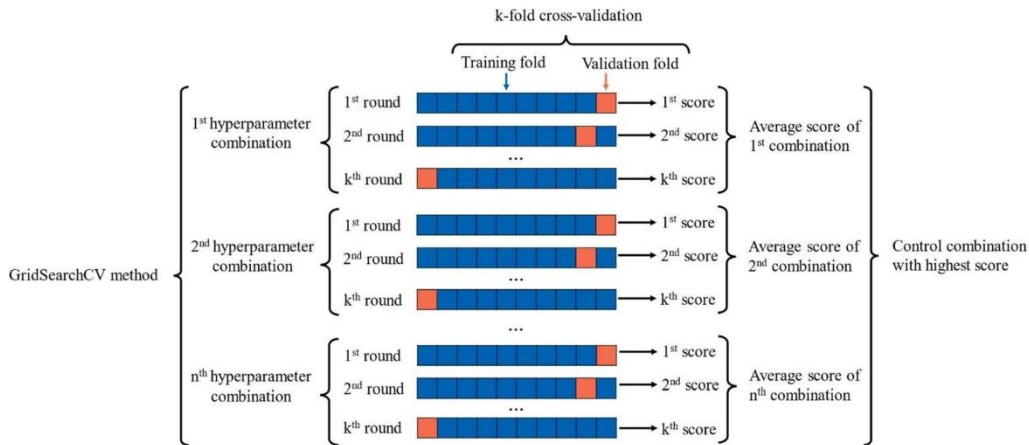
It works in following steps:

1. Define a set of possible values for each hyperparameter.
2. Create a grid of all possible hyperparameter combinations.
3. For each combination:
  - Set the hyperparameters.
  - Train the model.
  - Evaluate the model using a validation set or cross-validation.
4. Select the best hyperparameters based on model performance.

#### 2.6.4.2 Grid Search CV:

Grid Search CV (GSCV) is Scikit-Learn's implementation of grid search with cross-validation. For each combination of hyperparameters, GSCV trains the model using cross-validation and

computes a score for each fold of the data. This provides a more robust metric compared to a single train/test split and process is shown in Figure 2.8.



**Figure 2.8:** Grid Search CV

Following steps are involved in its functionality:

- Specify a parameter grid: Define a dictionary where keys are the hyperparameters, and values are lists of parameter settings to try.
- Use cross-validation: GSCV will split the data multiple times and train/test on these splits to evaluate each hyperparameter combination.
- For each combination in the grid, GSCV:
- Sets the hyperparameters to:
  - Train the model using cross-validation.
  - Compute a score for each fold.
  - Record the average score.

Once all combinations are evaluated, GSCV identifies the best hyperparameters and their corresponding performance. Also, here's an example in Code Snippet 2.6 using GSCV to tune hyperparameters for an SV Regressor:

```

from sklearn.model_selection import GridSearchCV

data = load_boston()

X = data.data, y = data.target

# Define hyperparameter grid
param_grid = { 'kernel': ['linear', 'rbf', 'poly'], 'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.01, 0.1, 1] }

svr = SVR()

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=svr, param_grid=param_grid, cv=5, verbose=1, n_jobs=-1)

# Execute the grid search
grid_search.fit(X, y)

# Get the best parameters and their corresponding performance
best_params = grid_search.best_params_

best_score = grid_search.best_score_

```

**Code Snippet 2.5:** *GridSearchCV()* implementation in Python

This example systematically evaluates each combination of the `kernel`, `C`, and `gamma` hyperparameters for an SVR, using 5-fold cross-validation, and finds the best performing combination.

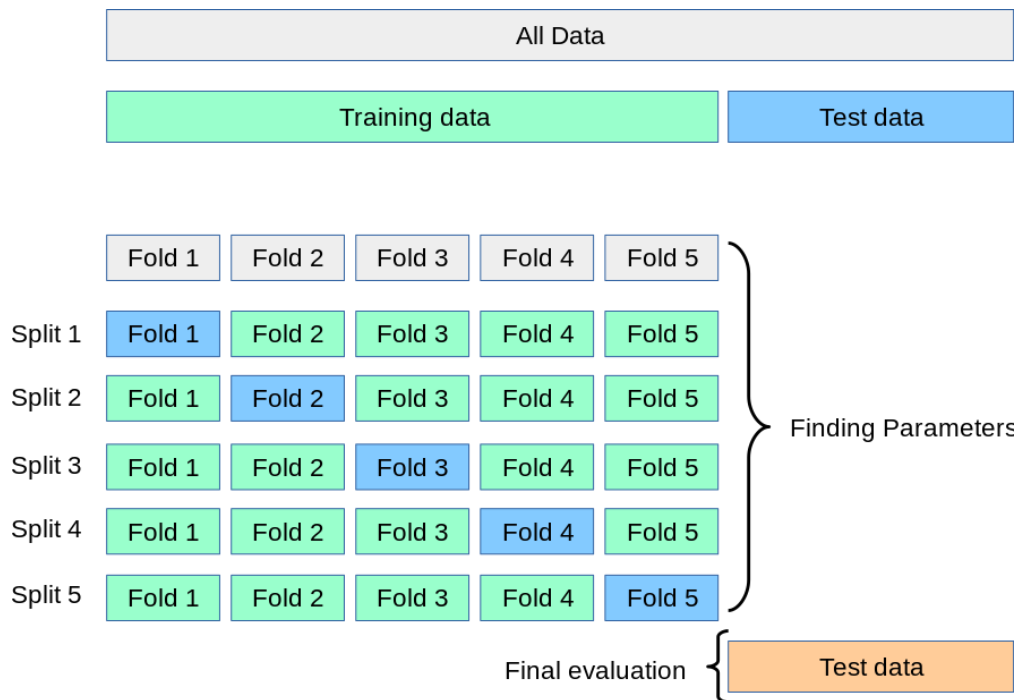
## 2.7 Model Evaluation

Model evaluation is like a check-up for your machine learning model, ensuring it's smart enough to handle new, unseen data. It's about tweaking its learning settings and making sure it's not just memorizing but truly understanding, ready to face real-world challenges.

### 2.7.1 Cross-Validation:

It involves partitioning the original training dataset into a set of subsets, holding out one as a validation set, and training the model on the rest. This process is repeated multiple times, with different subsets held out as the validation set. It is used to assess how well a model generalizes to

new data and avoids overfitting (Dinov, 2018). The procedure involved in cross validation is shown in Figure 2.9.



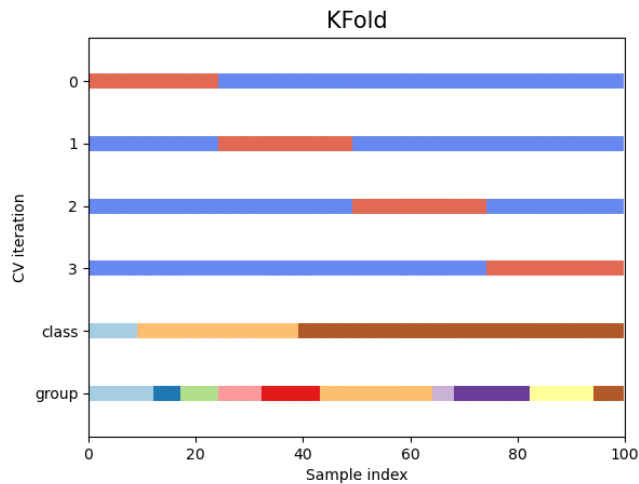
**Figure 2.9:** Process of Cross Validation

### 2.7.1.1 Different Types of Cross-Validation:

- **K-Fold Cross-Validation:**

The process involved in K-Fold can be explained in following steps:

- The data is divided into 'k' subsets.
- Each time, the k-1 subsets are used as a training set and one of the k subsets is used for the validation set.
- The final model performance is the average of the k models' validation performances. The procedure is shown in Figure 2.10.

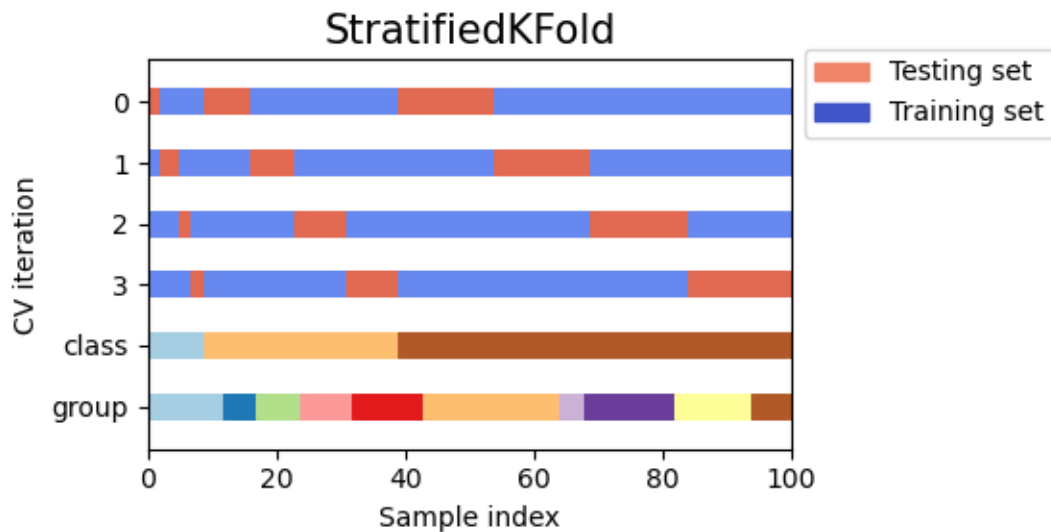


**Figure 2.10: K-Fold**

- **Stratified K-Fold Cross-Validation:**

The process involved in Stratified K-Fold Cross-Validation can be explained in following steps:

- Similar to K-Fold, but each fold is made by preserving the percentage of samples for each class.
- Stratified cross validation aims to preserve the target class distribution in each fold (Kärkkäinen, 2014). The process is shown in Figure 2.11.



**Figure 2.11: Stratified K-Fold process**

- **Leave-One-Out (LOO):**

Leave-One-Out is also one of the simple cross-validation.

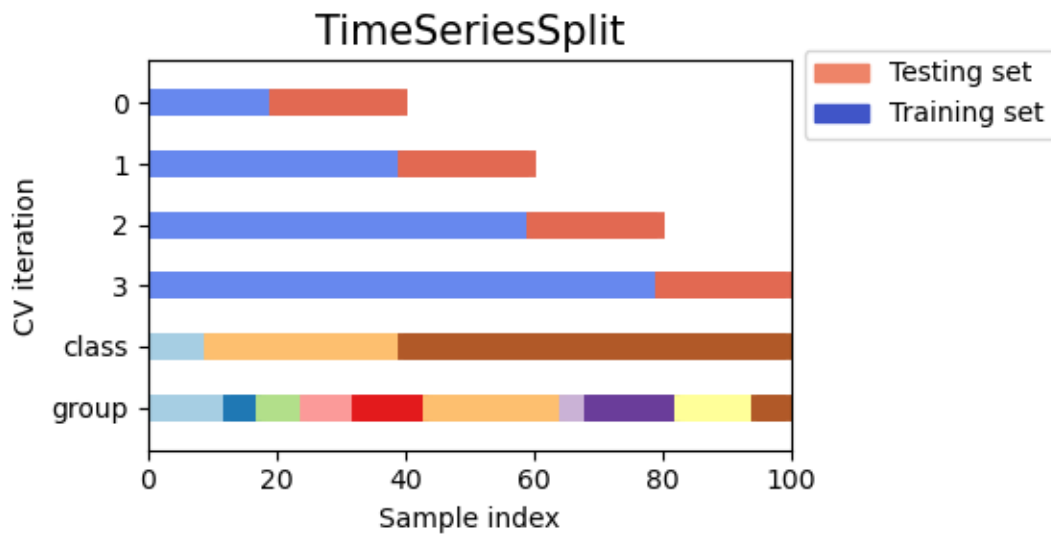
- Leave-One-Out (LOO) cross-validation is a detailed check-up for each data point in your dataset. Imagine you have N pieces of data; LOO will set up N unique learning sessions. In each session, it uses all but one data piece for training, reserving that lone piece for testing. It's like giving every single data point its moment in the spotlight to truly test the model's understanding.

- **Leave-P-Out (LPO):**

Leave-P-out is similar to Leave-one-out. This involves training on all combinations of N-P samples and validating on the remaining P samples.

- **Time-Series Cross-Validation:**

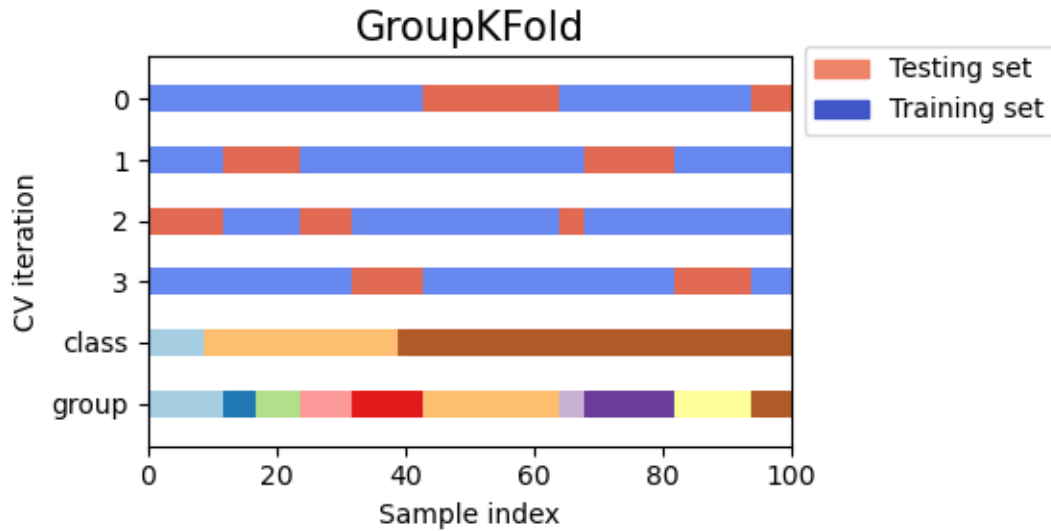
Time-Series Split (TSS) is the variation of K-Fold. This approach is specifically for cross validation of time series data. In this approach, the training set only includes observations prior to the validation set in time. The process involved in Time-Series CV technique is shown in Figure 2.12.



**Figure 2.12:** Time-Series Split process

- **Group K-Fold:**

Group K-Fold is a variation of K-Fold where the validation sets should consist of entire groups, useful when there are groups of correlated samples. It makes sure that same group data is not used for training and validation. The training and validation split procedure is shown in Figure 2.13.



**Figure 2.13:** Group K-Fold process

To utilize these methods in machine learning, libraries like Scikit-Learn provide built-in functions. For instance, `KFold()`, `StratifiedKFold()`, and `TimeSeriesSplit()` are all classes within Scikit-Learn that can be used to perform these respective types of cross-validation.

### 2.7.2 Model Metrics:

There are two types of model metrics. One is for regression and the second is for classification. The regression metrics are given below.

$$RMSE = \sqrt{\left(\frac{1}{n}\right) \times \sum_{i=1}^n [p_i - y_i]^2} \quad (2.11)$$

$$MAE = \frac{1}{n} \times \sum_{i=1}^n (|p_i - y_i|) \quad (2.12)$$



$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - p_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (2.13)$$

Where:

$n$  = sample size

$p_i$  = predicted values

$y_i$  = actual values

$\bar{y}$  = average of actual values

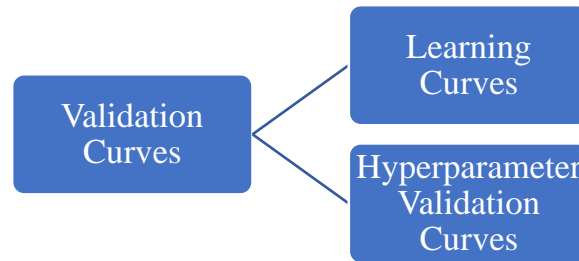
These metrics are typically specific to the context of a study and depend on the data's nature, the model's complexity, and the field of application. Generally, for  $R^2$ , a value closer to 1 indicates a better fit, while for RMSE, a lower value indicates a better fit. However, standard reference values may not be universal and can vary based on the domain or specific application.

### 2.7.3 Validation Curves:

Validation curves are graphical representations that show the performance of a machine learning model as a function of one of its hyperparameters. These curves are crucial tools for understanding the search range of hyperparameters which helps in hyperparameter tuning (Xie et al., 2018). Typically, validation curves are categorized based on the type of metric plotted and the nature of the change observed:

**Learning Curves:** These plot the model's performance (like accuracy or error) on the training set and the validation set as a function of the number of training examples (or iterations). Indicate how much the model benefits from adding more data.

**Hyperparameter Validation Curves:** These curves plot the model's performance as a function of various hyperparameter values. For instance, plotting performance with different values of `C` in a SVM or `max\_depth` in a Decision Tree.



**Figure 2.14:** Types of Validation Curves

### **2.7.3.1 Indications:**

There are few indications associated with validation curves which can be observed and are shortly summarized below.

### **2.7.3.2 Underfitting:**

- Training and validation errors converge and are high.
- The model is too simple to capture patterns in the data.

### **2.7.3.3 Good Fit:**

- Training and validation curves are close.
- The model generalizes well to unseen data.

### **2.7.3.4 Hyperparameter Optimization:**

- Helps identify values of hyperparameters where the validation performance is maximized.
- Helps in striking a balance between bias and variance.

In practice, tools like Scikit-Learn offer utilities to plot validation curves easily, aiding in making informed decisions regarding model complexity and hyperparameter values.

## **2.7.4 Learning Curves:**

Learning Curves (LC) are plots that show changes in learning performance over time in terms of experience. More specifically, in the context of machine learning, they are used to assess the performance of ML algorithm against any resource, for example, training sample etc. (Mohr & van Rijn, 2022). LC allows you to diagnose if your model is overfitting or underfitting.

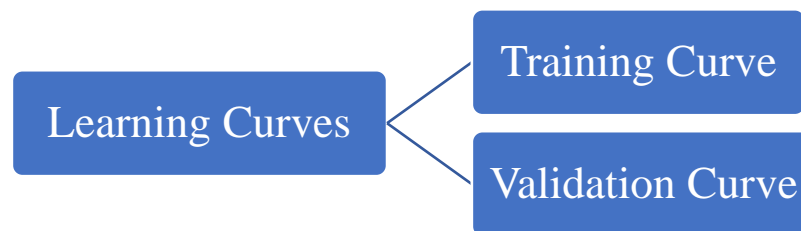
By examining learning curves, you can determine if the model would benefit from more data. If the validation performance continues to improve as more data is added, it's a sign that acquiring more data might be beneficial. They provide insight into how quickly a machine learning model can converge to a particular level of error, and can be helpful in tuning the training process, for instance, in deciding the size of mini-batches in mini-batch gradient descent.

#### 2.7.4.1 Types of Learning Curves:

Typically, LC have two types which are explained below and shown in Figure 2.15.

**Training Curve:** Plots the training error as the number of instances (or training iterations) increases.

**Validation Curve:** Plots the validation error as the number of instances (or training iterations) increases. These curves are typically plotted on the same graph for direct comparison.



**Figure 2.15:** Types of Learning Curves

#### 2.7.4.2 What They Indicate:

- **Underfitting (High Bias):**
  - Both training and validation errors are high and converge.
  - The model lacks the complexity needed to fit the data well.
- **Overfitting (High Variance):**
  - The training error is low, but there's a substantial gap between the training and validation errors.
  - The model is excessively complex and fits the training data's noise.
- **Ideal Learning Curve:**
  - The validation error decreases and converges to a point close to the training error.

- The gap between the curves is minimal, indicating that the model generalizes well.
- **Plateau in Learning:**
  - If the validation curve plateaus, no matter how much more training data you add, the model might not improve. This can be an indication that more training data might not be beneficial.

In essence, learning curves provide a visual method to gauge a model's performance relative to its amount of training. By analyzing the gap between the training and validation curves and their convergence behaviors, one can gain insights into the model's bias-variance trade-off and its data requirements.

## 2.8 Already Research

Goh's (Goh et al., 2005) pioneering efforts flashed a spotlight on the potential of Artificial Neural Networks (ANNs), constructing an intricate ANN model, which was designed for deducing the frictional dynamics of piles set within clayey terrains, using field data as the bedrock for their predictions. Later endeavors by researchers like (Shahin, 2010) extrapolated on the ANN model, utilizing a vast trove of data to prognosticate the bearing capacity of piles. Armaghani et al.'s (Armaghani et al., 2020) audacious research led to the birth of a hybridized model, intertwining the intricacies of ANN with the nuanced capabilities of particle swarm optimization (PSO) to craft predictions regarding pile settlements.

In their 2023 study, Nguyen (Nguyen et al., 2023) and colleagues introduced an innovative approach that marries the power of XGBoost, a cutting-edge machine learning technique, with the whale optimization algorithm (WOA) to predict how much weight concrete piles can support. The magic of their method lies in using XGBoost to make the final predictions based on data from experiments, while WOA hunts down the best XGBoost settings to make these predictions as accurate and reliable as possible. They put their model to the test with data from 472 real-world tests on concrete piles in Vietnam and found that their hybrid approach consistently beat both the standard XGBoost setup and traditional deep neural network models. After 20 trials, their model not only showed a remarkable improvement in accuracy but also stood out in tests against other methods, proving to be exceptionally well-suited for figuring out the capacity of concrete piles.

This breakthrough suggests that their hybrid model could be a valuable tool in civil engineering, offering a new way to ensure the safety and effectiveness of construction projects.

Tuan et al. (Pham, Tran, et al., 2020a) focused on using evolutionary algorithms like genetic algorithms (GA) to optimize deep learning neural networks (DLNN) for predicting the axial bearing capacity of driven piles. The authors collected a database of 472 driven pile static load test reports from construction sites in Vietnam. The data includes parameters like pile diameter, pile length, soil properties from SPT blow counts, etc. A GA model was used for feature selection to identify the most significant input features from the 10 original features. This reduced the features from 10 to 4 most important ones. A GA-DLNN, hybrid model, uses GA to optimize the DLNN architecture by selecting optimal parameters like number of hidden layers, neurons per layer, activation functions, training algorithms, etc. The reduced 4-input GA-DLNN model gave the best accuracy compared to models with all 10 inputs. It had  $R^2$  of 0.923 on validation set and 0.887 on test set. The GA-DLNN model performed better than just DLNN model, demonstrating the benefit of using GA to optimize DLNN parameters. The results show that evolutionary algorithms like GA can effectively optimize DLNN models for predicting pile bearing capacity based on real-world test data. The hybrid GA-DLNN approach outperformed regular DLNN and other models.

According to Pham et. al (Pham & Tran, 2022), determining the axial load capacity of piles is crucial for foundation design. Yet, field methods to determine this are often expensive and lengthy. This research aims to devise a hybrid machine-learning approach to predict this capacity. Two prominent optimization techniques, Particle Swarm Optimization (PSO) and Genetic Algorithm (GA), were applied to fine-tune the Random Forest (RF) model architecture. The study utilized a dataset with 472 pile load test outcomes from Ha Nam province in Vietnam. This dataset was split into 80% training and 20% testing segments. The model's efficacy was assessed through the Absolute Mean Error (MAE), Root Mean Square Error (RMSE), and Coefficient of Determination ( $R^2$ ). Outcomes indicated that GA outperformed PSO in optimizing the RF model. When pitted against the standard RF model, the GA-enhanced RF version showcased the most reliable results, presenting equilibrium between training and testing segments, which signifies reduced overfitting risks. These findings can guide the application of machine learning in the broader realm of engineering, especially within geotechnical sectors.

Tuan et al. (Pham, Ly, et al., 2020) developed machine learning models using ANN and RF to predict the axial bearing capacity of driven piles. The models were trained and tested on a large dataset of 2314 pile load test reports, which is claimed to be the largest dataset used for this purpose. The input variables included pile geometry factors like diameter and segment lengths, as well as soil properties like average standard penetration test (SPT) values along the pile shaft and at the pile tip. The RF model outperformed the ANN model in accuracy based on error metrics like R-squared, RMSE and MAE. RF also outperformed traditional empirical equations and multi-variable regression. Sensitivity analysis using the RF model showed that the average SPT value along the pile shaft and the pile tip elevation were the most important factors affecting the predicted bearing capacity. The RF model provides an accurate and fast numerical tool for estimating pile bearing capacity compared to traditional methods. Further improvement in accuracy using hybrid ML methods is suggested as future work.

Nhat-Duc et al. (Hoang et al., 2022) proposes a new data-driven model for predicting the axial bearing capacity of piles. The model uses a hybrid approach combining machine learning and metaheuristic optimization. The machine learning method used is Least Squares Support Vector Regression (LSSVR). LSSVR can handle multivariate data and model nonlinear relationships between inputs and outputs. The metaheuristic used is opposition-based differential flower pollination (ODFP). ODFP is used to optimize the hyperparameters of the LSSVR model. The model uses 10 input variables related to pile characteristics and soil conditions to predict pile bearing capacity. The model was trained and tested on a dataset of 472 static load tests of reinforced concrete piles. Experimental results showed the ODFP-LSSVR model achieved good accuracy in predicting pile bearing capacity, outperforming several benchmark machine learning methods. Statistical tests confirmed the superior performance of ODFP-LSSVR compared to other models. The study demonstrates the promise of using a hybrid metaheuristic-machine learning approach for estimating pile bearing capacity. The ODFP optimization helps improve the predictive accuracy of the LSSVR model.

Maaz et al. (Amjad et al., 2022) proposed machine learning models to predict the axial bearing capacity ( $P_u$ ) of driven piles. The models are based on data from 200 pile load tests conducted in Ha Nam province, Vietnam. The models use 10 input parameters related to pile geometry, pile material, and soil properties. The key parameters are pile diameter, depths of soil layers, pile elevations, and SPT blow counts along the pile shaft and at the pile tip. Five machine learning

algorithms are tested: XGBoost, Adaptive Boosting (AdaBoost), RF, Decision Tree, and SVM. The XGBoost model performs the best, with high accuracy on the test set ( $R^2 = 0.955$ , low errors). XGBoost outperforms the other methods due to its regularization and use of multiple decision trees. Sensitivity analysis shows the SPT blow count along the pile shaft has the greatest influence on predicted pile capacity. The study demonstrates machine learning can accurately estimate pile capacity using common soil properties and pile geometry data. The data driven XGBoost model outperformed traditional empirical and analytical methods. The authors conclude that the XGBoost model can be readily updated with new data and provides a practical alternative to costly pile load testing. It shows promise for estimating pile capacity for geotechnical design.

A new ensemble AI model proposed by (Cao et al., 2022), termed the Intelligence Multivariate Neural Network Inference Model (IMNNIM), is introduced in this research to provide efficient and precise pile bearing capacity predictions. The IMNNIM merges the Equilibrium Optimization Algorithm (EO) with a blend of Multivariate Adaptive Regression Splines (MARS) and Radial Basis Neural Network (RBFNN). The model refines predictions by dynamically amalgamating outputs from MARS and RBFNN, while also adjusting weights and tuning parameter values of its learning components. Using 472 static pile load test records, the IMNNIM's efficacy was assessed. Utilizing a 10-fold cross-validation approach, the model showcased superior predictive accuracy, producing top scores for MAPE (7.24%), RMSE (90.92 kN), MAE (67.98 kN), and  $R^2$  (0.930), coupled with the most consistent results. The t-test method affirmed the model's heightened capability in estimating pile bearing capacities.

## Chapter 3: Methodology

### 3.1 Data Collection

The study draws on data from 472 field tests involving pre-cast reinforced concrete piles, carried out in Ha Nam Province, Vietnam, by researchers Pham and Tran (Pham, Tran, et al., 2020b) and illustration of experimental layout is shown in Figure 3.1. The piles featured in the study were square in cross-section and designed with closed tips. They were installed into the ground using hydraulic machines that applied a continuous force, ensuring a steady installation rate. The testing protocol involved gradually increasing vertical loads to 100%, 150%, and 200% of the load design, over periods of approximately 6 hours, 12 hours, and 24 hours, respectively.

Two principles were followed for pile bearing capacity determination: either the settlement of pile top at the current load level was at least 5 times that of settlement at the previous load level, or a linear load-settlement curve was observed. The bearing capacity was defined as the load level at which the settlement exceeded 10% of the pile diameter. This comprehensive dataset of static load tests on pre-cast reinforced concrete piles is substantial and forms a robust foundation for developing and validating advanced machine learning models, which is a key aspect of this research. The introduction of parameters is given in Table 3.1 and statistical summary of input is given in Table 3.2.

**Table 3.1:** Introduction of input parameters

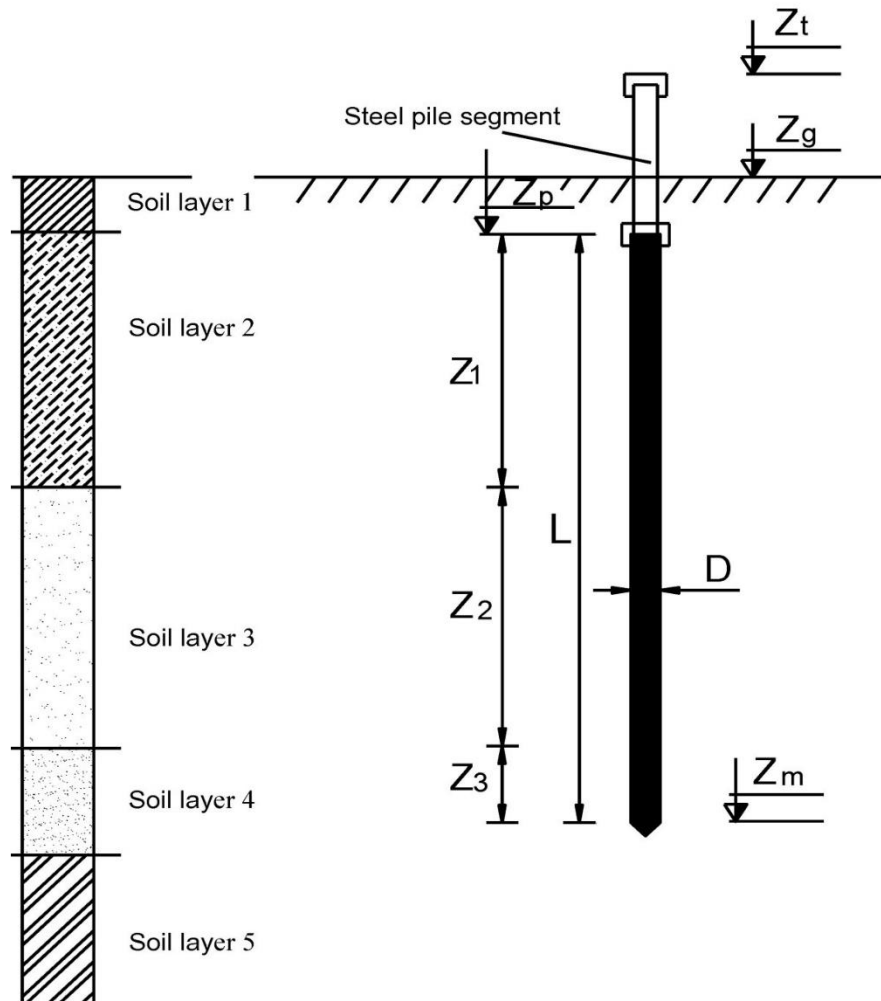
Parameters	Symbols	Count
Diameter of Pile	$D$	472
Length of soil first layer	$Z_1$	472
Length of soil second layer	$Z_2$	472
Length of soil third layer	$Z_3$	472
Elevation of pile top	$Z_p$	472
Elevation of ground level	$Z_g$	472



Elevation of pile extra segment of pile top	$Z_t$	472
Elevation of pile tip	$Z_m$	472
The avg. SPT blow count along pile length	$N_{sh}$	472
The avg. SPT blow count at the pile tip	$N_t$	472
Bearing capacity of piles	$P_u$	472

**Table 3.2:** Statistical summary of input data

	$D$	$Z_1$	$Z_2$	$Z_3$	$Z_p$	$Z_g$	$Z_t$	$Z_m$	$N_{sh}$	$N_t$	$P_u$
<b>Unit</b>	<b>mm</b>	<b>M</b>	<b>m</b>	<b>m</b>	<b>M</b>	<b>m</b>	<b>m</b>	<b>m</b>	<b>-</b>	<b>-</b>	<b>kN</b>
<b>1</b>	400	3.54	7.6	0.3	2.95	3.65	2.95	14.7	11.75	7.59	1017.9
<b>2</b>	400	4.25	8	0	2.15	3.56	2.16	15.4	13.25	7.67	1152
<b>3</b>	400	4.25	8	0	2.15	3.58	2.16	15.42	13.27	7.68	1344
<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>
<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>	<b>-</b>
<b>471</b>	300	3.4	5.26	0	3.4	3.49	3.43	12.06	8.66	6.75	508.9
<b>472</b>	400	3.85	7.6	0	2.95	3.67	3.27	14.4	11.45	7.15	1425
<b>Mean</b>	361. 745	3.80 179	6.815 817	0.346	2.815	3.495	2.936	13.78 1	10.96 5	7.17 7	1005.5
<b>Std.</b>	48.6 55	0.47 6	1.286	0.458	0.623	0.066	0.601	1.464	2.032	0.40 0	347.58
<b>Min</b>	300	3.4	5.15	0	1.95	3.34	1.96	11.95	8.55	6.71	407.2
<b>Max</b>	400	4.75	8	1.22	3.4	3.7	4.35	15.62	13.63	7.75	1551



**Figure 3.1:** Illustration of experimental layout

### 3.2 Outlier Detection and Rectification Using Gaussian Approximation:

In analyzing the current dataset, we employed the statistical methodology of Gaussian approximation. Leveraging the Python library, Feature-engine (version 1.6), this technique identifies and addresses outliers. Gaussian approximation is rooted in the properties of the Gaussian (or normal) distribution. For a standard normal distribution - the data reveals a pattern where around 68% is clustered within one standard deviation from the average, suggesting a tight grouping around the mean. About 95% of the data extends to within two standard deviations, indicating a broader but still consistent range. Impressively, nearly 99.7% of the observations fall within three standard deviations, showcasing an exceptionally high level of consistency across the dataset. Harnessing this statistical knowledge, the Gaussian approximation approach detects

outliers as observations that lie beyond three standard deviations from the mean. Any data point that falls outside this range is considered an aberration from the expected norm. The Feature-engine library automates this procedure, effectively identifying and excluding these outliers from both the upper and lower limits of the distribution. Such a systematic curation ensures that the resulting dataset is more robust and less susceptible to the distortions that outliers can introduce.

### 3.3 Data Normalization:

Initially, the raw dataset undergoes a preprocessing phase, where it is normalized using the function `MinMaxScaler`. This preprocessing technique re-scales each feature in the dataset to the range of 0 to 1, thereby ensuring that the features have the same scale. This crucial step mitigates the potential for feature dominance and facilitates a more balanced learning process for the algorithms used.

### 3.4 Data Partition:

Once the dataset is meticulously preprocessed, the next step is its splitting into distinct sets for model training and validation. The partitioning was carried out using the `'train_test_split()'` function from the Scikit-learn library. This function not only ensures a random distribution of data points into the two sets but also guarantees consistency in this random division. The result is a balanced and representative split that highlights effective model training and validation. For the current dataset, the data is split with a ratio of 90:10 and reasons are:

- 90% of the data is allocated towards the comprehensive process encompassing hyperparameter optimization, training, and internal validation of the model settings. This data is labeled as “training and testing data” and it will ensure that the model has enough information to understand and capture relationships and patterns.
- The remaining 10% is reserved for validation purposes of proposed tuned models and labeled as “validation data”.
- The code implementation is given in Code Snippet 3.1 below.

```
# Split the data into training and testing sets  
X_train, X_validation, y_train, y_validation = train_test_split(X, Y, test_size=0.1, random_state=42)
```

**Code Snippet 3.1:** Splitting of data into training and validation

### 3.5 Machine Learning Models

For the current research, machine learning models like RF, SVR and XGBoost are used. These models are implemented through Scikit-learn (Pedregosa FABIANPEDREGOSA et al., 2011) in Python version 3.11.6. It is designed for Python and is a widely used, open-source machine learning framework. It's famous for making data analysis and model building, both straightforward and effective. The library supports various learning algorithms, both supervised and unsupervised, and is developed on top of the SciPy (Scientific Python) environment, enhancing its versatility and user-friendly nature.

It is an open-source machine learning library for Python that offers simple and efficient tools for data analysis and modeling. It provides a range of supervised and unsupervised learning algorithms, is built upon the SciPy (Scientific Python) library and is known for its ease-of-use and versatility. RF Regressor, SVR, and XGBoost are carefully imported in python.

### 3.6 Hyperparameter Tuning Through Random and Grid Search

The hyperparameter tuning of machine learning algorithms is performed through GS and RS techniques. The tuning of hyperparameters through GS technique is done through `GridSearchCV()` which basically comes with integrated cross-validation capabilities. This tool establishes a rigorous framework for hyperparameter optimization, gauging model efficacy, and safeguarding the model's adaptability to data it hasn't encountered before. Within `GridSearchCV()`, there's a specific argument termed 'param\_grid'. This argument accepts a dictionary which has keys representing hyperparameters and arrays denoting corresponding values to be assessed for each hyperparameter. The choice for 'param\_grid' is informed by a comprehensive review of hyperparameters from existing literature. Initially, a broad search space is designated, which is subsequently refined, converging towards a more targeted range for hyperparameter exploration (Belete & Huchaiyah, 2021).

The tuning of hyperparameter through RS is done through `RandomizedSearchCV()` which utilizes the "param\_distributions" parameter. This allows for a specified number of parameter configurations, determined by the `n_iter` parameter, to be drawn from the provided distributions. By integrating cross-validation through the "cv" parameter, it ensures an optimal mix of computational resource management and dependable performance evaluation. RSCV has been

reported in literature (Sarajcev & Meglic, 2022). Both hyperparameters tuning techniques require a set and range of hyperparameters for tuning. So, hyperparameters are selected for each algorithm which are shown in Table 3.3, 3.4, and 3.5.

**Table 3.3:** Hyperparameters for RF

Hyperparameter	param_grid
<i>n_estimators</i>	[200, 400, 1000]
<i>max_depth</i>	[10, 20, , 50, None]
<i>min_samples_split</i>	[2, 5, 10]
<i>min_samples_leaf</i>	[1, 2, 4]
<i>max_features</i>	['auto', 'sqrt']

**Table 3.4:** Hyperparameters for SVR

Hyperparameter	param_grid
C	[1.00000000e+00, 3.72759372e+00, 1.38949549e+01, 5.17947468e+01, 1.93069773e+02, 7.19685673e+02, 2.68269580e+03, 1.00000000e+04]
Epsilon	[0.001, 0.01, 0.1, 1, 2, 4]
Kernel	['rbf', 'poly','sigmoid','linear']

**Table 3.5:** Hyperparameters for XGBoost

Hyperparameter	param_grid
<i>max_depth</i>	[5, 6, 9, 11, 12, 13, 14]
<i>n_estimators</i>	[50, 80, 100, 150, 200, 300]
<i>reg_alpha</i>	[0, 0.1, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8, 25.6]
<i>reg_lambda</i>	[0, 0.1, 0.4, 0.8, 1.6, 6.4, 12.8, 51.2, 102.4]

After selection of hyperparameters, the next step is to write code for each algorithm separately and some parts of Code Snippets for algorithms tuning through GS and RS are shown in Code Snippets 3.2, 3.3, 3.4, 3.5, 3.6, and 3.7.

```

params= { 'bootstrap': [True, False], 'max_depth': [10, 20, 30, 40, 50, None], 'max_features': ['log2', 'sqrt', 1.0],
          'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10], 'n_estimators': [200, 400, 600, 800, 1000]}

model = RandomForestRegressor(random_state=RANDOM_SEED)

# Step 2: Perform randomized search on the stacking regressor

random_search = RandomizedSearchCV(estimator=model, param_distributions= params , cv=5,n_iter=1500,
scoring=['r2',          'neg_mean_squared_error',          'neg_root_mean_squared_error',          'max_error'],
refit='neg_root_mean_squared_error',return_train_score=True)

random_search.fit(X_train, y_train)

```

**Code Snippet 3.2:** Random Search Snippet for Random Forest

```

# Step 1: Define the parameter distributions for randomized search

param_distributions = {'bootstrap': [True, False], 'max_depth': [10, 20, 30, 40, 50, None],
                       'max_features': ['log2', 'sqrt', 1.0], 'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10],
                       'n_estimators': [200, 400, 600, 800, 1000]}

model = RandomForestRegressor(random_state=RANDOM_SEED)

# Step 2: Perform grid search

grid_search = GridSearchCV(estimator=model, param_grid= param_distributions, cv=5, scoring=['r2',
'neg_mean_squared_error', 'neg_root_mean_squared_error', 'max_error'],
refit='neg_root_mean_squared_error',return_train_score=True)

grid_search.fit(X_train, y_train)

```

**Code Snippet 3.3:** Grid Search Snippet for Random Forest

```

# Step 1: Create the base models
model = SVR()

c_range = np.logspace(-0, 4, 8)

gamma_range = np.logspace(-4, 0, 8)

# Step 2: Define the parameter distributions for randomized search
params = {'C': c_range, 'degree':[1,2,3,4,5,6], 'gamma': np.logspace(-4, 0, 8), 'epsilon': [0.001, 0.01, 0.1, 1, 2, 4], 'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}

# Step 3: Perform randomized search on the stacking regressor
random_search = RandomizedSearchCV(estimator=model, param_distributions=params, cv=5, n_iter=1500,
scoring=['r2', 'neg_mean_squared_error', 'neg_root_mean_squared_error', 'max_error'],
refit='neg_root_mean_squared_error', return_train_score=True)

random_search.fit(X_train, y_train)

```

**Code Snippet 3.4:** Random Search Snippet for SVR

```

# Step 1: Create the base models
model = SVR()

c_range = np.logspace(-0, 4, 8)

gamma_range = np.logspace(-4, 0, 8)

# Step 2: Define the parameter distributions for randomized search
param_distributions = {'C': c_range, 'degree':[1,2,3,4,5,6], 'gamma': np.logspace(-4, 0, 8), 'epsilon': [0.001, 0.01, 0.1, 1, 2, 4], 'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}

# Step 3: Perform randomized search on the stacking regressor
grid_search = GridSearchCV (estimator=model, param_grid=param_distributions, cv=5, scoring= ['r2', 'neg_mean_squared_error', 'neg_root_mean_squared_error', 'max_error'], refit='neg_root_mean_squared_error',
return_train_score=True)

```

**Code Snippet 3.5:** Grid Search Snippet for SVR

```

# Step 1: Create the base models
model = XGBoost.XGBRegressor(seed=42)

# Step 2: Define the parameter distributions for randomized search
params = {'gamma': [0,0.1,0.4,0.8,1.6,3.2,6.4,12.8,51.2,102.4, 200], 'learning_rate': [0.01, 0.03, 0.06, 0.1, 0.15,
0.2, 0.25, 0.30], 'max_depth': [5,6,9,11,12,13,14], 'n_estimators': [50, 80,100,150, 200, 300],
'reg_alpha': [0,0.1,0.4,0.8,1.6,3.2,6.4,12.8,25.6], 'reg_lambda': [0,0.1,0.4,0.8,1.6,6.4,12.8,51.2,102.4]}

```

**Code Snippet 3.6:** Random Search Snippet for XGBoost

```

# Step 1: Create the base models
model = XGBoost.XGBRegressor(seed=42)

# Step 2: Define the parameter distributions for randomized search
params = {'gamma': [0,0.1,0.4,0.8,1.6,3.2,6.4,12.8,51.2,102.4, 200], 'learning_rate': [0.01, 0.03, 0.06, 0.1, 0.15,
0.2, 0.25, 0.30], 'max_depth': [5,6,9,11,12,13,14], 'n_estimators': [50, 80,100,150, 200, 300], 'reg_alpha':
[0,0.1,0.4,0.8,1.6,3.2,6.4,12.8,25.6], 'reg_lambda': [0,0.1,0.4,0.8,1.6,6.4,12.8,51.2,102.4]}

# Step 3: Perform grid search
grid_search = GridSearchCV(estimator=model, param_grid= params, cv=5, scoring=['r2',
'neg_mean_squared_error', 'neg_root_mean_squared_error', 'max_error'],
refit='neg_root_mean_squared_error', return_train_score=True)

```

**Code Snippet 3.7:** Grid Search Snippet for XGBoost

### 3.7 Tuned Model Evaluation Through Learning Curves

To make our process cleaner and more transparent, the hyperparameters are again evaluated against metrics by looking at learning curves. In our case, by using these performances in learning curves, we can get valuable insights on the performance and behavior of our optimized machine learning model. By visualizing learning curves, we can decide about hyperparameter tuning, data sufficiency and model complexity. These curves are plotted through ‘learning\_curves’, and Code Snippets 3.8, 3.9, and 3.10. and some parts of these codes are represented below.



```

from sklearn.model_selection import validation_curve
import matplotlib.pyplot as plt

# Define the range of values for the hyperparameter
param_range = [200, 400, 600, 800, 1000]

# Compute training and test scores for varying parameter values.
train_scores, test_scores = validation_curve(
    RandomForestRegressor(max_depth=random_search.best_params_['max_depth'],
max_features=random_search.best_params_['max_features'],
min_samples_leaf=random_search.best_params_['min_samples_leaf'],
min_samples_split=random_search.best_params_['min_samples_split'],
bootstrap=random_search.best_params_['bootstrap']), X_train, y_train, param_name="n_estimators",
param_range=param_range, cv=5, scoring="r2", n_jobs=-1)

train_mean = np.mean(train_scores, axis=1)

train_std = np.std(train_scores, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scores, axis=1)

test_std = np.std(test_scores, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(param_range, train_mean, label="Training score", color="black")

plt.plot(param_range, test_mean, label="Cross-validation score", color="red")

# Plot accuracy bands for training and test sets
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, color="gray")

plt.fill_between(param_range, test_mean - test_std, test_mean + test_std, color="yellow")

```

**Code Snippet 3.8:** Learning Curve with  $R^2$  on y-axis for RF tuned with RS

```

from sklearn.model_selection import learning_curve

# Compute the training scores and validation scores for increasing training set sizes
train_sizes, train_scores, test_scores = learning_curve(
    estimator=random_search.best_estimator_,
    X=X_train, y=y_train,
    train_sizes=np.linspace(0.1, 1.0, 10),
    cv=5, scoring='neg_root_mean_squared_error', n_jobs=-1)

# Calculate the mean and standard deviation for each training set size
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

```

**Code Snippet 3.9:** Learning Curve with RMSE on y-axis for RF tuned with RS

```

(max_depth=grid_search.best_params_['max_depth'],
max_features=grid_search.best_params_['max_features'],
min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
min_samples_split=grid_search.best_params_['min_samples_split'],
bootstrap=grid_search.best_params_['bootstrap']), X_train, y_train, param_name="n_estimators",
param_range=param_range, cv=5, scoring="r2", n_jobs=-1)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

```

**Code Snippet 3.10:** Learning Curve with  $R^2$  on y-axis for SVR tuned with GS

### 3.8 Evaluation of Tuned Models Through Cross Validation Scores

All the tuning process goes through cross validation which is basically integrated within libraries. Mean train score and mean test score helps in assessing the performance of different hyperparameters combination during grid and random search of hyperparameters selection. Mean train score is the average performance score of models on training data across all folds for each hyperparameter combination. Mean test score is the average performance score of models on testing data across all folds. These scores help in understanding of how well the model is learning from the training data and performing on testing data. Some parts of Code Snippet 3.11 are shown below.

```
# Extract the desired metrics for the best model
best_model_metrics = { 'mean_train_r2': results.loc[best_index, 'mean_train_r2'],
'mean_train_neg_mean_squared_error': results.loc[best_index, 'mean_train_neg_mean_squared_error'],
'mean_train_neg_root_mean_squared_error': results.loc[best_index,
'mean_train_neg_root_mean_squared_error'], 'mean_train_max_error': results.loc[best_index,
'mean_train_max_error'], 'std_train_r2': results.loc[best_index,
'std_train_r2'], 'std_train_neg_mean_squared_error': results.loc[best_index,
'std_train_neg_mean_squared_error'], 'std_train_neg_root_mean_squared_error': results.loc[best_index,
'std_train_neg_root_mean_squared_error'], 'std_train_max_error': results.loc[best_index,
'std_train_max_error'], 'mean_test_r2': results.loc[best_index, 'mean_test_r2'],
'mean_test_neg_mean_squared_error': results.loc[best_index, 'mean_test_neg_mean_squared_error'],
'mean_test_neg_root_mean_squared_error': results.loc[best_index,
'mean_test_neg_root_mean_squared_error'], 'mean_test_max_error': results.loc[best_index,
'mean_test_max_error'], 'std_test_r2': results.loc[best_index, 'std_test_r2'], 'std_test_neg_mean_squared_error':
```

**Code Snippet 3.11:** Mean, std for train & test scores of all algorithms

### 3.9 Evaluation of Tuned Models on Validation Data

After a long process, all the models along with their respective tuned hyperparameters are evaluated for real world problems through determining evaluation metrics on validation data. For this final evaluation process, Code Snippet 3.12 is as follows.

```
y_pred = grid_search.predict(X_test) #For grid search
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
error = max_error(y_test, y_pred)
```

**Code Snippet 3.12:** Evaluation of tuned model on validation data

## Chapter 4: Results and discussion

### 4.1 Preprocessed data

The process of outlier removal often yields noteworthy changes in the structure and characteristics of a dataset. When assessing the impact of outlier removal, it is vital to consider not just the data's shape, but also its statistical properties, such as the measures of central tendency and the dispersion metrics. After the removal of outliers, the dataset's shape transformed to (447,8), which suggests a significant refinement of the data points and statistical summary of data after outlier removal is given in Table 4.1. The subsequent statistical analysis provides further insights into the modifications introduced:

#### 4.1.1 Central Tendency Measures

Measures like the mean, median, and mode help to understand the central location of the data. The post-outlier removal dataset displayed slight variations in the mean. Such a trend can be attributed to the sensitivity of the mean to extreme values. Conversely, the median and mode, which are more resilient to outliers, remained largely unchanged. This consistent behavior of median and mode suggests that the core structure of the dataset has been preserved.

#### 4.1.2 Data Distribution

Skewness is an essential metric for understanding the symmetry of data distribution. A skewness of positive, negative and zero indicates right-skewed, left-skewed, and perfect symmetry respectively. Upon examination:

- For variables such as  $Z_1$ ,  $Z_3$ ,  $N_{sh}$ , and  $N_t$ , a decrease in skewness towards zero highlights a progression towards symmetrical distribution.
- For  $Z_2$  and  $Z_p$ , an increase in skewness indicates a shift in data distribution, albeit still moving closer to symmetry.
- The variable  $N_t$  stands out with its substantial change, transitioning from a heavily left-skewed distribution to near-perfect symmetry.

#### 4.1.3 Range

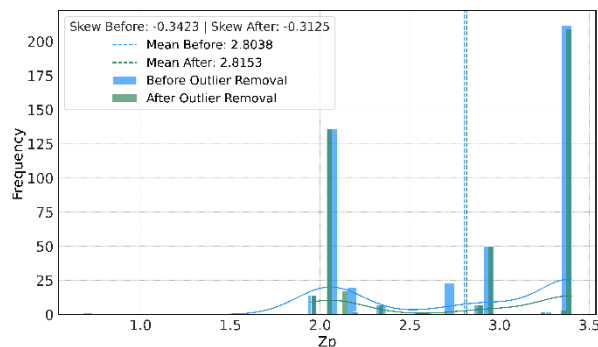
An essential metric to understand the spread of data, the range consistently shrunk for most variables. This reduction is a direct consequence of the outlier removal, indicating that extreme

values, which previously extended the data's breadth, have been effectively excluded, leading to a more concise dataset.

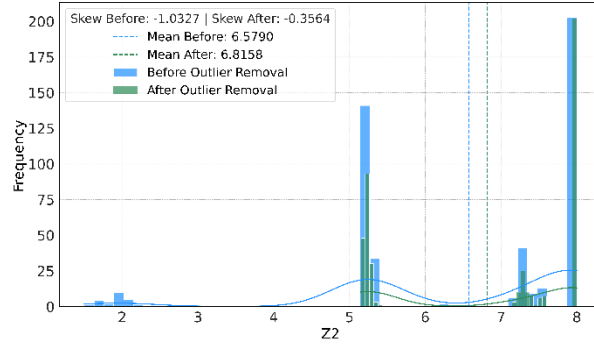
#### 4.1.4 Variable-specific Observations

- $Z_1$ : The mean dipped slightly, but there was a move towards symmetry.
- $Z_2$ : Exhibited an increased mean, with skewness moving closer to symmetry.
- $Z_3$ : Though the mean saw a slight increment, the distribution became more symmetrical.
- $Z_p$ : A negligible change in mean, accompanied by a shift towards symmetric distribution.
- $N_{sh}$ : Notable increase in mean and a stronger push towards symmetry.
- $N_t$ : Significant alteration towards symmetric distribution.
- $P_u$ : The mean inclined slightly, with a minor shift towards a negatively skewed distribution.

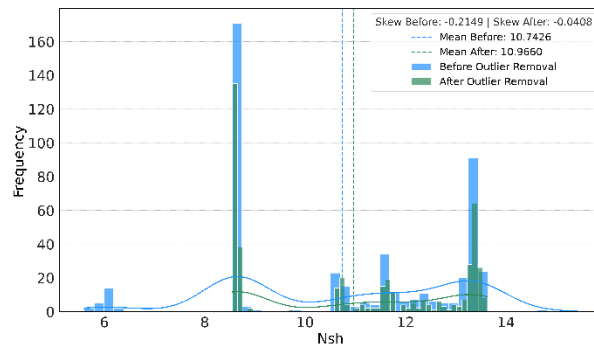
In wrapping up, the modifications introduced by outlier removal have been beneficial. The consistent reduction in range indicates the effective exclusion of extreme values, leading to a dataset that's more concentrated around its central values. Additionally, the measures of skewness, for most variables, are converging to zero, suggesting improved symmetry. Such refinements underline the importance of outlier management in preserving the inherent structure of the data while eliminating noise and extreme values. The graphical presentation of data is given in Figure 4.1, 4.2, 4.3, and 4.4.



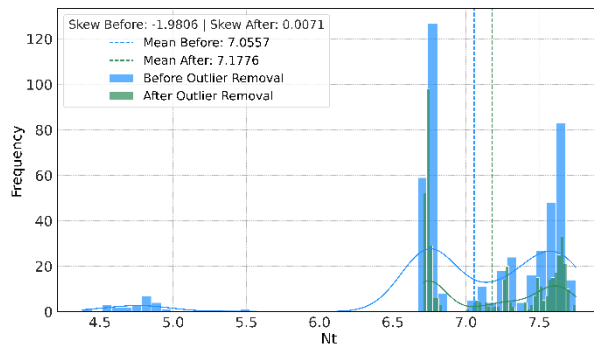
**Figure 4.1:** Graphical statistical summary before and after outlier removal for ' $Z_p$ '



**Figure 4.2:** Graphical statistical summary before and after outlier removal for ‘ $Z_2$ ’



**Figure 4.3:** Graphical statistical summary before and after outlier removal for ‘ $N_{sh}$ ’



**Figure 4.4:** Graphical statistical summary before and after outlier removal for ‘ $N_t$ ’

**Table 4.1:** Statistical summary of data after outlier removal

	<b>D</b>	<b><math>Z_1</math></b>	<b><math>Z_2</math></b>	<b><math>Z_3</math></b>	<b><math>Z_p</math></b>	<b><math>N_{sh}</math></b>	<b><math>N_t</math></b>	<b><math>P_u</math></b>
<b>Count</b>	447	447	447	447	447	447	447	447
<b>Mean</b>	361.74	3.801	6.815	0.346	2.815	10.965	7.177	1005.519
<b>Std.</b>	48.65	0.476	1.286	0.458	0.623	2.032	0.400	347.581

<b>min</b>	300	3.4	5.15	0	1.95	8.55	6.71	407.2
<b>max</b>	400	4.75	8	1.22	3.4	13.63	7.75	1551

## 4.2 Normalized Data

For each input parameter, minimum and maximum values are changed. After data normalization, the rescaled data is shown in the Table 4.2.

**Table 4.2:** Statistical summary of normalized data

	<b>D</b>	<b>Z<sub>1</sub></b>	<b>Z<sub>2</sub></b>	<b>Z<sub>3</sub></b>	<b>Z<sub>p</sub></b>	<b>N<sub>sh</sub></b>	<b>N<sub>t</sub></b>
<b>Unit</b>	mm	m	m	m	m	-	-
<b>Mean</b>	0.617	0.297	0.584	0.283	0.596	0.475	0.449
<b>Std.</b>	0.486	0.352	0.451	0.375	0.430	0.400	0.385
<b>Min</b>	0.000	0.000	0.000	0.000	0.000	0.000	0.000
<b>Max</b>	1.000	1.000	1.000	1.000	1.000	1.000	1.000

## 4.3 Data Splitting

In this research, the data is split differently than usual. Instead of the common 70:30 or 80:20 split, we're using a 90:10 ratio. We'll use 90% of the data for finding the best settings (hyperparameter search), training, and testing these settings through cross-validation. The remaining 10% is reserved for checking how well the final machine learning models work. The statistical summary for both splits of data is given in Table 4.3 and 4.4.

**Table 4.3:** Statistical summary of training and testing data

	<b>D</b>	<b>Z<sub>1</sub></b>	<b>Z<sub>2</sub></b>	<b>Z<sub>3</sub></b>	<b>Z<sub>p</sub></b>	<b>N<sub>sh</sub></b>	<b>N<sub>t</sub></b>	<b>P<sub>u</sub></b>
<b>unit</b>	<b>Mm</b>	<b>m</b>	<b>m</b>	<b>m</b>	<b>M</b>	-	-	<b>kN</b>
<b>Count</b>	402	402	402	402	402	402	402	402
<b>Mean</b>	0.616	0.302	0.585	0.289	0.591	0.478	0.451	1003.14
<b>Std.</b>	0.486	0.355	0.451	0.378	0.432	0.401	0.386	347.88
<b>min</b>	0.00	0.00	0.00	0.00	0.00	0.00	0.00	407.2
<b>max</b>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1551.0



**Table 4.4:** Statistical summary of validation data

	<b>D</b>	<b>Z<sub>1</sub></b>	<b>Z<sub>2</sub></b>	<b>Z<sub>3</sub></b>	<b>Z<sub>p</sub></b>	<b>N<sub>sh</sub></b>	<b>N<sub>t</sub></b>	<b>P<sub>u</sub></b>
<b>unit</b>	<b>Mm</b>	<b>m</b>	<b>m</b>	<b>M</b>	<b>m</b>	<b>-</b>	<b>-</b>	<b>kN</b>
<b>Count</b>	45	45	45	45	45	45	45	45
<b>Mean</b>	0.622	0.255	0.574	0.236	0.639	0.446	0.433	1026.68
<b>Std.</b>	0.490	0.333	0.456	0.347	0.412	0.387	0.375	347.98
<b>min</b>	0.00	0.00	0.00	0.00	0.00	0.00	0.00	508.90
<b>max</b>	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1473.00

Results for Hyperparameter Tunning: The machine learning models are tuned through grid and random search separately to find their respective best set of hyperparameters. Same Range of hyperparameters is given for both random and grid search for same model. After tuning, the optimal set for each model is shown in table 4.5, 4.6, and 4.7. These hyperparameters optimize the ML models to give the best predictions.

**Table 4.5:** Optimal hyperparameters for RF selected through GS and RS

<b>Random Forest</b>		
<b>Hyperparameter</b>	<b>Grid Search Optimal Value</b>	<b>Random Search Optimal Value</b>
n_estimators	600	800
min_samples_split	10	5
min_samples_leaf	1	4
max_features	log2	log2
max_depth	10	None
Bootstrap	True	True

**Table 4.6:** Optimal hyperparameters for SVR selected through GS and RS

<b>Support Vector Machine</b>		
<b>Hyperparameter</b>	<b>Grid Search Optimal Value</b>	<b>Random Search Optimal Value</b>
C	10000.00	2628.69
Degree	1	4
epsilon	4	4

gamma	1	1.0
kernel	Rbf	Rbf

**Table 4.7:** Optimal hyperparameters for XGBoost selected through GS and RS

XGBoost		
Hyperparameter	Grid Search Optimal Value	Random Search Optimal Value
Gamma	0.8	12.8
learning_rate	0.25	0.03
max_depth	5	6
n_estimators	80	300
reg_alpha	0.4	3.2
reg_lambda	102.4	51.2

## 4.4 Results for Evaluation of Tuned Models Through Learning Curves

### 4.4.1 Evaluation of Learning Curves for Random Forest

In our analysis, we examined the learning curves of a Random Forest model under two different hyperparameter tuning methods: RS and GS. The insights got from these learning curves provide a clear understanding of the model's performance and potential areas for enhancement and learning curves are shown in Figure 4.5 and 4.6.

#### 4.4.1.1 Analysis of Learning Curves with Optimal Hyperparameters via Random Search:

**Training Score Curve:** Starting at an  $R^2$  score of 0.93 and concluding around 0.96 indicates a robust fit to the training data. Such a trend reflects the model's capability to capture the underlying patterns within the training set. The marginal decrease followed by a plateau in the curve suggests that the model's performance on the training set reaches a stable point as more data is incorporated.

**Validation Score Curve:** Initiating below 0.88 and capping around 0.93 showcases the model's improving generalization capabilities with an increase in training data. However, a discernable gap between the training and validation scores indicates a mild overfitting scenario, where the model might be too attuned to the training data, affecting its performance on unseen data. Additionally, when we inspect the model's performance using the RMSE (Root Mean Square Error) on the y-

axis, similar trends are observed. RMSE effectively quantifies the error magnitude, offering a complementary perspective to the  $R^2$  score.

In summary, we can conclude that the model has a good fit to the data, as the training and validation scores are both high and stable. The model is likely generalizing well to new data, and the performance is consistent across different training sets (as indicated by the narrow confidence intervals). It also suggests that the model's capacity to learn from additional data is reaching a limit, as indicated by the plateauing of the validation score.

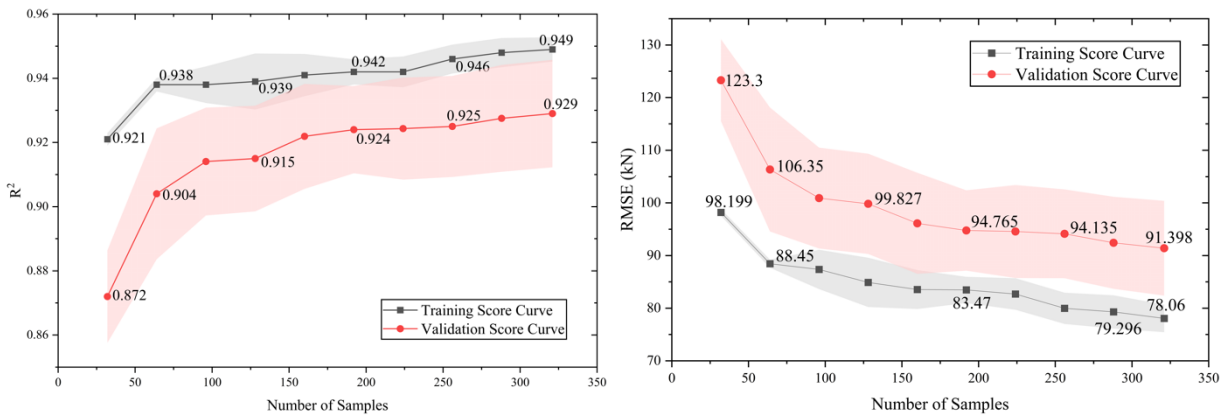
#### ***4.4.1.2 Analysis of Learning Curves with Optimal Hyperparameters via Grid Search:***

The  $R^2$  score is a metric that indicates the proportion of variance in the dependent variable that is predictable from the independent variables. In this context, it's used to evaluate the fit quality of the model. The higher the  $R^2$  score, the better the model is at making predictions. The learning curve shows that both the training score (solid line) and the validation score (dashed line) start off with a strong  $R^2$  score, even with a smaller number of training samples. The training score remains relatively stable and high as the number of training samples increases, suggesting that the model has a consistent performance and is not overfitting. Overfitting would typically be indicated by a high training score but a low validation score. The validation score, on the other hand, starts lower but increases with more data, a pattern that is generally indicative of a model that could benefit from more data points. The fact that the validation curve's trajectory is mildly upward as the number of training samples increases is encouraging. It suggests that the model is learning and generalizing well from the added data, which is the desired outcome when tuning a model. The performance of the hyperparameters identified by the RS appears to be on par with those found by the GR, which is significant because RS is often more computationally efficient than GR. Considering the slight improvement in the validation score with more data, it's plausible that the model's generalization could further improve with additional training samples. This suggests that data augmentation or collection of more data could be beneficial strategies for enhancing model performance.

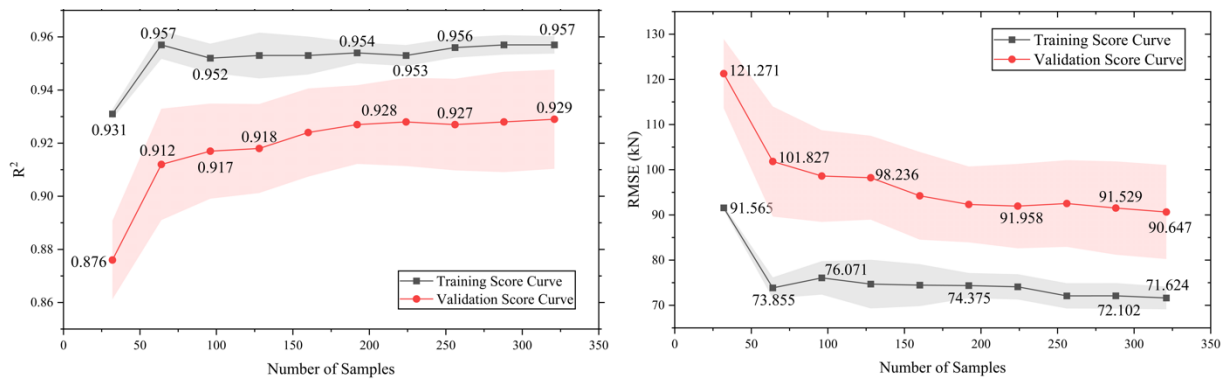
Overall, the learning curves suggest that the tuned model has a good fit to the data, as indicated by high  $R^2$  values. The GS method, despite being more exhaustive, does not appear to provide a significantly better set of hyperparameters than RS, which can be a point in favor of using RS due

to its efficiency. The model shows potential for improved generalization with more data, pointing towards data acquisition as a possible next step in model refinement.

In summary, utilizing both the  $R^2$  score and RMSE for analyzing learning curves furnishes a comprehensive view of the model's dynamics. While the  $R^2$  metric offers insights into the data fit quality, the RMSE quantifies the error's magnitude. Armed with these insights, we can make informed decisions regarding model refinements, data augmentation, or hyperparameter adjustments.



**Figure 4.5:** Learning Curves for RF during RS tuning



**Figure 4.6:** Learning Curves for RF during GS tuning

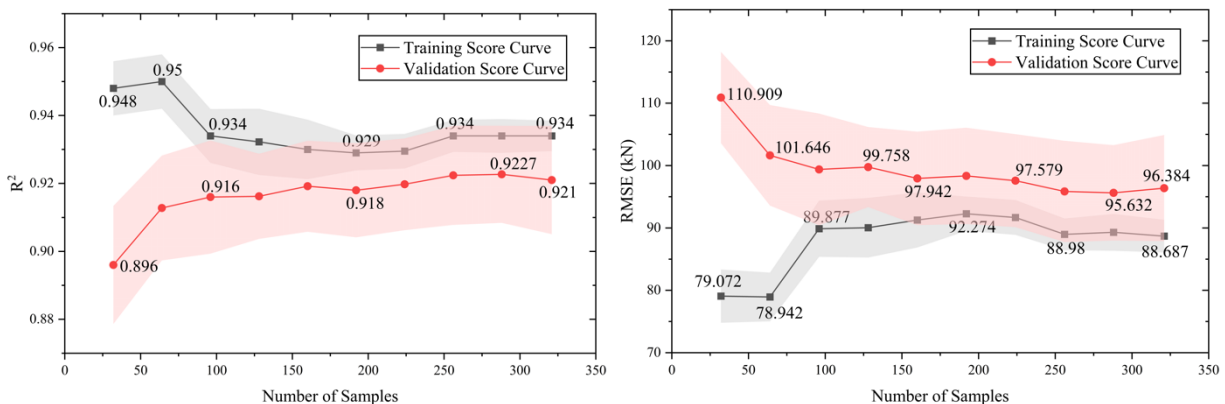
#### 4.4.2 Evaluation of Learning Curves for SVR

Support Vector Regression (SVR), part of the Support Vector Machines (SVM) framework, employs the principle of maximizing margins to generate predictive models for continuous outputs. In this evaluation, we've dissected the learning curves of an SVR model optimized via

two distinct hyperparameter tuning techniques: RS and GS as shown in Figure 4.7. The patterns depicted in these curves shed light on the model's efficacy and potential areas of refinement.

#### 4.4.2.1 Analysis of Learning Curves with Optimal Hyperparameters via Random Search:

- Characteristic dips at 190 samples:** Both the training and validation curves exhibit synchronized dips when trained on approximately 190 samples. Such synchronized perturbations suggest a challenge faced by the SVR model in fitting to or generalizing from this specific batch of data. The reason for this behavior can be multifaceted: anomalies, inherent noise, or unique attributes within this data subset could be potential culprits.
- Validation curve trend:** As the data samples increase, the validation curve presents a declining trend towards the latter part. Such a decline in performance on the validation set, in juxtaposition with the training sets consistent or improving performance, is indicative of a potential overfitting scenario. The persistent difference between the training and validation scores further accentuates this overfitting concern. The model might be capturing intricate patterns from the training data, which aren't necessarily generalizable, thereby diminishing its performance on unseen or validation data.
- The insights from the learning curves of the SVR model, optimized with both Random and Grid Search methods, underscore the importance of vigilant data inspection and potential model refinement. Recognizing challenges at specific data intervals and addressing overfitting tendencies can guide the fine-tuning of the model or instigate further investigations into the data's characteristics.



**Figure 4.7:** Learning Curves for SVR during RS tuning

### 4.4.3 Evaluation of Learning Curves for XGBoost

By delving into the learning curves of an XGBoost model, we can gauge XGBoost model performance on optimal set of hyperparameters and understand how different hyperparameter tuning strategies impact its behavior. Below, we present a meticulous analysis based on two hyperparameter tuning strategies: RS and GS as shown in Figure 4.8 and 4.9.

#### 4.4.3.1 Analysis of Learning Curve with Optimal Hyperparameters via Random Search:

**Training score curve:** Start at a score of 0.85, the training curve experiences a gradual ascent, eventually plateauing at 0.95. Starting from a lower baseline but reaching a high plateau, suggests that the model is capable of learning effectively from the training data without overfitting, as indicated by the convergence of training and validation scores.

**Validation score curve:** Starting at 0.812, the validation curve mirrors a steady climb to finally stabilize at 0.93 which indicates good generalization. However, the initial steep slope followed by a plateau suggests that additional data points beyond a certain threshold do not yield significant improvements in validation performance.

The RMSE for both training and validation decreases as more data is provided, which is desirable. However, the validation RMSE appears to plateau, indicating that while the model's error rate is decreasing, it might require further improvements to achieve lower errors on unseen data.

**Gap Analysis:** A consistent and narrow divergence between the training and validation curves throughout the learning process is evident. This modest gap is encouraging, suggesting that the model achieves a harmonious balance between bias and variance. In layman's terms, the model is well-tuned, avoiding pitfalls of both significant overfitting and underfitting. This harmonization reflects the model's robustness in generalizing to unseen data.

#### 4.4.3.2 Analysis of Learning Curve with Optimal Hyperparameters via Grid Search:

**Training curve insight:** The curve kicks off from an impressive 0.925, which could suggest that the Grid Search method has identified a set of hyperparameters that allow the model to learn effectively from the outset. It undergoes a slight trough when encountering around 100 samples, but this deviation isn't profound. It then marches forward to stabilize at a commendable 0.970.

**Validation curve insight:** The validation trajectory starts from 0.825, with a minor dip observed around the 225-sample mark could be an anomaly or reflect a particular characteristic of the data

set that is difficult for the model to learn. Despite this, it makes a comeback to stabilize around 0.93.

The RMSE curve shows a decrease in error as more data is provided, with a more gradual convergence between training and validation scores compared to RS. The smaller gap between the curves suggests that the GS model might be slightly more stable and less prone to overfitting than the RS model.

**Gap analysis:** As the model understands more data, the separation gap between the training and validation curves diminishes. However, a final split of approximately 0.04 persists. While not vast, this lingering gap suggests that the model, although fine-tuned, might be mildly overfitting. It's a subtle indication that the model could be capturing certain nuances of the training data that may not necessarily translate to optimal performance on validation or unseen datasets. The modest gaps between the training and validation curves suggest that both models are balanced in terms of bias and variance, achieving a harmonious fit to the data without overfitting or underfitting excessively.

In summary, RS-XGBoost shows good learning capabilities and generalization, with performance improvements as more data is added. The model demonstrates stability, but the plateauing validation score suggests a limit to the benefits of additional data. The GS-XGBoost, starts off with a strong fit and maintains stability throughout the learning process. The minor fluctuations indicate that it is sensitive to the training data's nuances but overall suggests a robust model with a good balance between performance on training and unseen data.

The learning curves for both models indicate that both hyperparameter tuning methods have yielded stable and well-generalized models. However, the GS approach may offer a slight edge in stability and predictive performance, as suggested by the tighter convergence of its learning curves and lower RMSE. Further experimentation with a larger dataset or more diverse hyperparameter space could potentially lead to further improvements in model performance and stability.

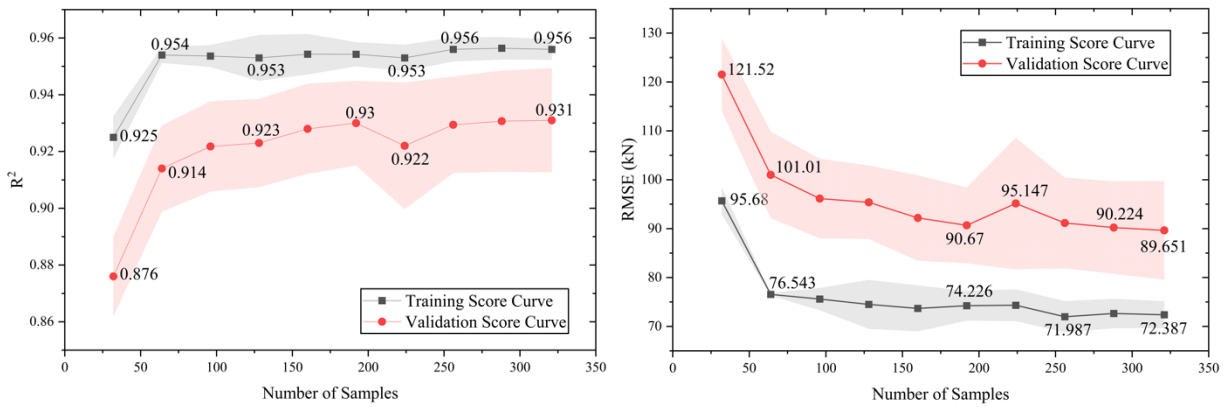


Figure 4.8: Learning Curves for XGBoost during RS tuning

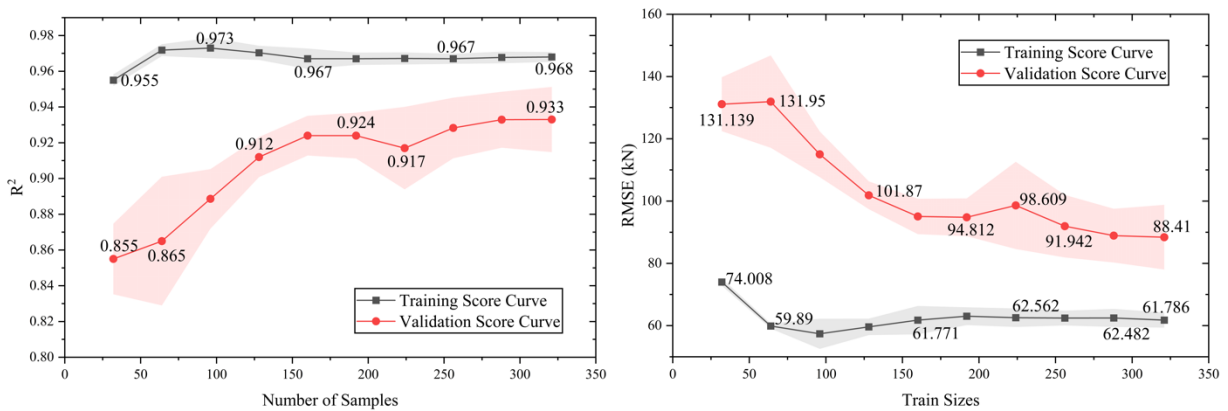


Figure 4.9: Learning Curves for XGBoost during GS tuning

#### 4.5 Results of Evaluation for Tuned Models Through Cross Validation Scores

All models are trained and tested for finding the optimum set of hyperparameters. During this process, the *GridSearchCV()* and *RandomizedSearchCV()* provides results against each set which are saved as *cv\_results*. Out of all these results, the regression metrics are exported and shown in Table 4.8. There's a noticeable, as seen in Table 4.8, though not substantial, difference between the mean training and mean test scores ( $R^2$ , RMSE, MAE) of five folds. This hints at slight overfitting, suggesting that the model might be capturing some noise present in the training data. The standard deviation between results of folds is calculated. When a model shows a lower standard deviation, it means its performance is reliably consistent, maintaining similar levels of accuracy or effectiveness across various segments of the data. The standard deviations of the metrics across the five folds, particularly for the test scores, hint at the model's stability. An  $R^2$  value close to 1 in regression analysis is associated with good performance based on the high goodness of fit provided



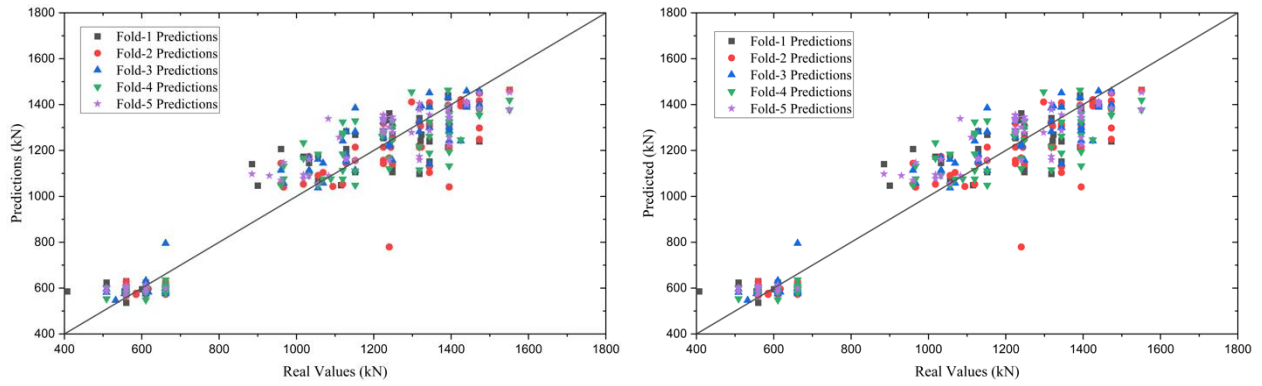
by the squared multiple correlation coefficient. Train  $R^2$  is higher than test  $R^2$  which indicates there is some indication of the model capturing noise or overfitting, albeit not severely.

The RF model shows a higher mean training  $R^2$  score and lower RMSE by using GS hyperparameters compared to RS hyperparameters, indicating better training performance with GS. However, the testing scores are quite similar for both methods, suggesting comparable generalization capabilities. Similar to RF, SVR exhibits better training performance with GS hyperparameters but shows negligible differences in testing performance between the two methods. This suggests that while GS may lead to better optimization during training, it doesn't significantly affect the generalization ability of model. In the case of XGBoost, the model demonstrates the highest training  $R^2$  scores among all, particularly with GS hyperparameters, and maintains lower RMSE scores compared to the other models. The testing scores, however, are comparable between Grid and Random Search methods.

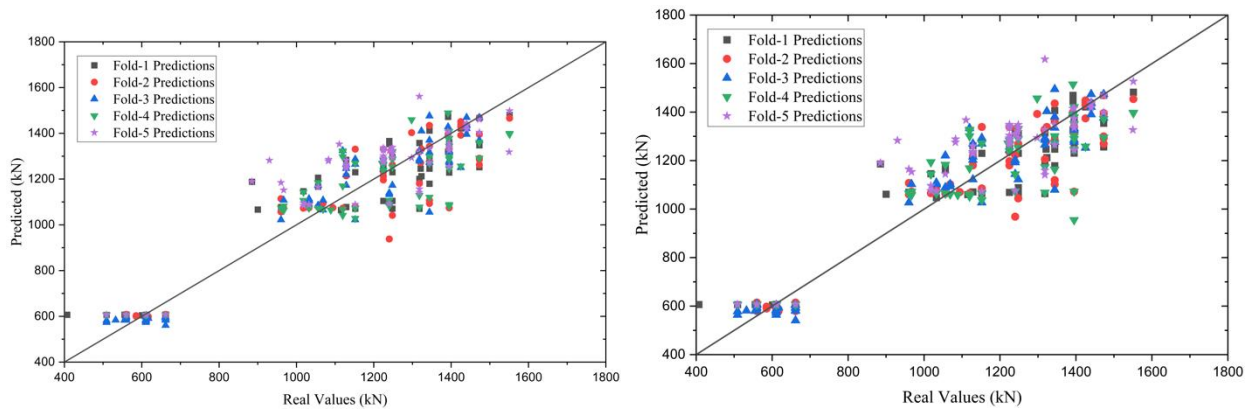
In summary, while GS generally leads to slightly better training performance, the differences in testing scores between two approaches are minimal. This indicates that both hyperparameter tuning methods are effective, with gs potentially offering more finely tuned models at the cost of higher computational resources.

The Standard Deviation (SD) for RF is relatively low for both training and testing scores, indicating consistent performance across different folds. This consistency is desirable, as it implies the model's robustness. For SVR, the standard deviation is higher, especially in testing scores, it might indicate that the model's performance is more variable and less predictable across different datasets. This could be a concern for generalization. In the end, XGBoost, the SD provides insight into model stability under different hyperparameter resulting from both searches. A lower SD in either method would suggest more consistent performance, which is advantageous.

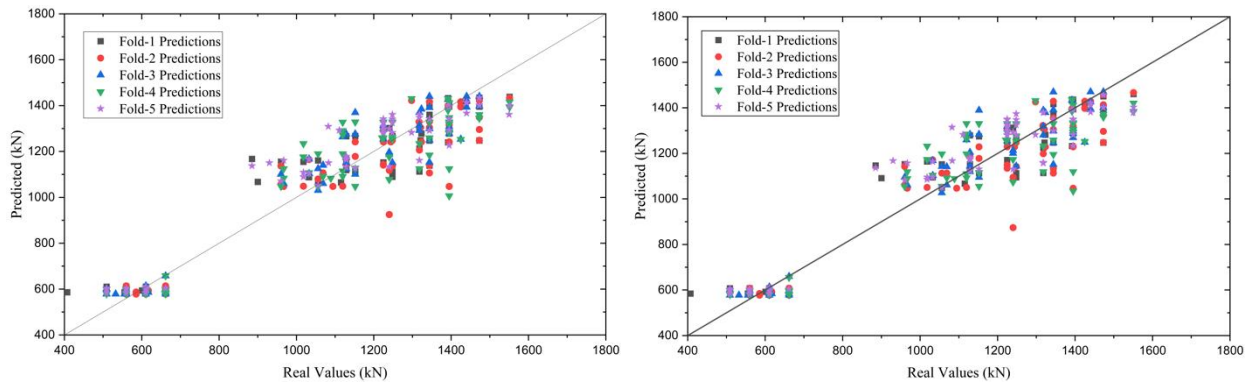
In general, models with lower SDs in testing scores are preferable, as they indicate more reliable performance when faced with new, unseen data. This aspect should be considered along with the mean scores to get a thorough understanding of each model's efficiency. The scatter plots for the pile loading capacity are shown in Figures 4.10, 4.11, and 4.12.



**Figure 4.10:** Predictions vs real values for RS-RF & GS-RF



**Figure 4.11:** Predictions vs real values for RS-SVR & GS-SVR



**Figure 4.12:** Predictions vs real values for RS-XGBoost & GS-XGBoost

**Table 4.8:** Results for Cross Validation Scores

Model	GS-Hyperparameters				RS-Hyperparameters			
	Train		Test		Train		Test	
	R <sup>2</sup>	RMSE	R <sup>2</sup>	RMSE	R <sup>2</sup>	RMSE	R <sup>2</sup>	RMSE

<b>Random Forest</b>	0.957 (+/- 0.003)	71.631 (+/- 2.287)	0.929 (+/- 0.019)	91.122 (+/- 10.760)	0.949 (+/- 0.003)	78.237 (+/- 2.533)	0.928 (+/- 0.017)	91.928 (+/- 9.679)
<b>SVR</b>	0.936 (+/- 0.003)	87.460 (+/- 2.295)	0.921 (+/- 0.020)	96.238 (+/- 11.393)	0.934 (+/- 0.004)	89.046 (+/- 2.731)	0.921 (+/- 0.016)	96.341 (+/- 8.761)
<b>XGBoost</b>	0.967 (+/- 0.002)	62.683 (+/- 2.288)	0.933 (+/- 0.0194)	88.479 (+/- 11.166)	0.953 (+/- 0.003)	74.66 (+/- 2.474)	0.931 (+/- 0.019)	89.888 (+/- 10.802)

#### 4.6 Evaluation of Models on Validation Data

In order to evaluate the effectiveness of hyperparameter tuning methods - GS and RS - across three different machine learning models: RF, SVR, and XGBoost, are trained on validation data which was separated in the start. The result for that evaluation is evaluated by regression metrics which are shown in Table 4.9.

It highlights that GS generally outperforms RS in refining model accuracy. This is evident from the higher  $R^2$  values and lower error rates (MAE, RMSE, MSE) observed with GS. XGBoost, when optimized using GS, showed particularly impressive results, achieving an  $R^2$  value of 0.952 and an RMSE of 75.26 kN, slightly better than its performance with Random Search ( $R^2 = 0.950$ , RMSE = 76.45 kN). This indicates XGBoost's superior predictive capability among the three models. The results of RS-XGBoost and GS-XGBoost are compared with proposed models in literature as shown in Table 4.10. It clearly outperforms the other models showing supremacy of RS and GS techniques.

The differences between the two tuning methods, while being modest, are still noteworthy. Despite RS undergoing more iterations, GS provided slightly better outcomes. This finding is critical for selecting the most suitable models and tuning approaches in machine learning projects, underlining the importance of strategic model choice and hyperparameter optimization.

Visual comparisons of the predicted values against the actual values were also performed shown in Figures 4.13, 4.14, and 4.15. Each algorithm, tuned with *RandomSearchCV()*, was placed side by side against its *GridSearchCV()* which is basically tuning counter of it. Additionally, a Taylor

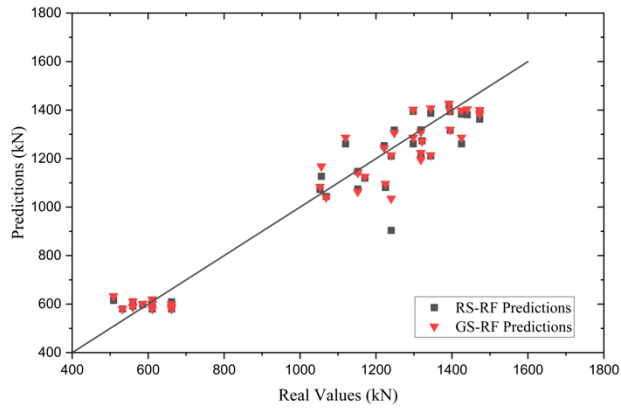
diagram was employed to analyze the predictions of each model and shown in Figure 4.16. The gap between forecasted and actual or reference points reflects the precision of predictive models (Jeon & Rahman, 2008). Here it can be observed that models tuned with Grid Search, particularly XGBoost and RF, demonstrated superior performance, showing high correlation coefficients (0.977 and 0.9769, respectively) and lower SD. This reinforces their strong linear relationship with actual values and minimal prediction errors. The GS-tuned XGBoost (GS-XGBoost) model stood out as the most effective, displaying the highest correlation and the lowest RMSE among all evaluated models. This underscores its exceptional ability to accurately capture data trends, making it a highly reliable option for similar prediction tasks. Overall, this analysis offers valuable insights into the effectiveness of different tuning methods in optimizing machine learning models.

**Table 4.9:** ML models predictions on validation data

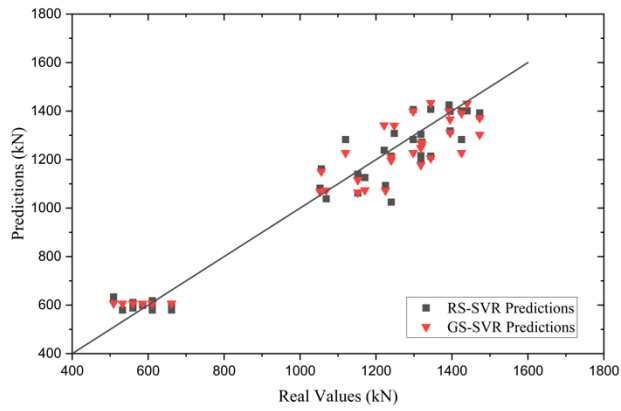
<b>Validation Data</b>						
<b>Model</b>	<b>GS-Hyperparameters</b>			<b>RS-Hyperparameters</b>		
	<b>R<sup>2</sup></b>	<b>RMSE</b>	<b>MSE</b>	<b>R<sup>2</sup></b>	<b>RMSE</b>	<b>MSE</b>
<b>Random Forest</b>	0.9499	76.982	5926.357	0.9369	86.368	7459.54
<b>SVR</b>	0.9464	79.639	6342.45	0.9424	82.54	6812.85
<b>XGBoost</b>	0.9561	72.07	0.9464	0.9522	75.1834	75.1834

**Table 4.10:** Proposed models comparison with literature

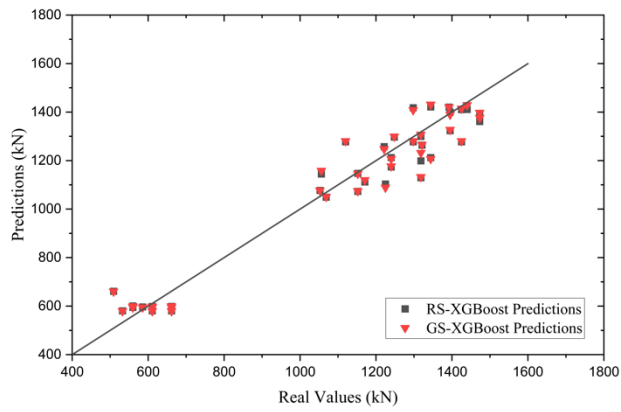
<b>Model</b>	<b>R<sup>2</sup></b>	<b>RMSE (kN)</b>
<b>WOA-XGBoost</b> (Nguyen et al., 2023)	0.92	87.72
<b>RF-PSO</b> (Pham & Tran, 2022)	0.924	93.91
<b>RF</b> (Pham, Ly, et al., 2020)	0.866	98.161
<b>ODFP-LSSVR</b> (Hoang et al., 2022)	0.93	92.19
<b>XGBoost</b> (Amjad et al., 2022)	0.955	80.653
<b>IMNNIM</b> (Cao et al., 2022)	0.933	88.5
<b>RS-XGBoost</b>	0.952	75.18
<b>GS-XGBoost</b>	0.956	72.07



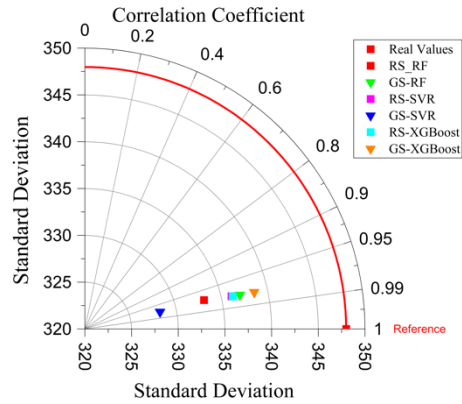
**Figure 4.13:** Predictions vs real values for RS-RF & GS-RF



**Figure 4.14:** Predictions vs real values for RS-SVR & GS-SVR



**Figure 4.15:** Predictions vs real values for RS-XGBoost & GS-XGBoost



**Figure 4.16:** Taylor Diagram

## 4.7 Benefits

The current study highlights the power of machine learning in transforming how we estimate the bearing capacity of concrete piles, making the process faster, more accurate, and cost-effective. By using advanced algorithms, engineers can predict pile behavior more reliably, reducing the need for expensive field tests. This approach not only speeds up project timelines but is also economical for small projects.

## 4.8 Limitations

The current study has one significant constraint which is the minimal consideration of material properties within the dataset. It can influence the accuracy of bearing capacity predictions in diverse geological and environmental conditions. Additionally, the focus on a singular criterion for bearing capacity—specifically, the settlement is less than 10% of the pile diameter—are considered—narrows the scope of these models. This limitation restricts the models' applicability to a wider range of practical scenarios where settlement criteria may vary, potentially affecting their utility in real-world engineering challenges.

## Chapter 5: Conclusions

### 5.1 Development of Predictive Model

The research successfully developed a predictive model to understand the intricate relationships between axial bearing capacity and its influencing factors. Various machine learning algorithms, including Random Forest (RF), Support Vector Regressor (SVR), and XGBoost, were employed and fine-tuned to enhance the model's performance.

### 5.2 Transparent Mapping:

The research transparently mapped out each phase, providing a clear and comprehensive insight into the complexities of hyperparameter tuning and validation processes. This transparency ensures the replicability and reliability of the research findings.

### 5.3 Establishment of Robust Models

The models established have demonstrated solid generalization capabilities. They have shown consistent and reliable performance, ensuring their applicability to diverse and unseen data, which is crucial for real-world applications. RS-RF ( $R^2= 0.93$  , RMSE=86.63 ), GS-RF ( $R^2= 0.949$ , RMSE=76.98), RS-SVR ( $R^2= 0.942$  , RMSE=82.54), GS-SVR ( $R^2= 0.946$  , RMSE=79.63), RS-XGBoost ( $R^2= 0.952$  , RMSE=75.18), and GS-XGBoost ( $R^2= 0.956$  , RMSE=72.07) shows satisfactory results proving generalization of tuned models.

### 5.4 Comparison of Machine Learning Models

A thorough comparison of RF, SVR, and XGBoost was conducted. The evaluation on validation data, along with the analysis of validation and learning curves, revealed that all the models performed commendably. Among them, XGBoost (RS-XGBoost and GS-XGBoost) stood out for its superior performance, affirming its effectiveness in handling the task at hand.

### 5.5 Effective Tuning and Validation

The use of *GridSearchCV()* and *RandomizedSearchCV()* for hyperparameter tuning, despite being a trial-and-error process, proved to be beneficial. The models' performance was optimized, and their robustness was validated through various curves and evaluations.

## **5.6 Future Recommendations**

### **5.6.1 Exploration of Other Algorithms**

Future research could explore other machine learning or deep learning algorithms to ascertain if better performance metrics can be achieved. Algorithms like neural networks or ensemble methods could be investigated.

### **5.6.2 Advanced Hyperparameter Tuning**

Employing more advanced hyperparameter tuning methods, such as Bayesian Optimization, could potentially enhance the model's performance further.

### **5.6.3 Feature Engineering**

Delving deeper into feature engineering could unveil more intricate relationships and improve the model's predictive power. It is recommended to explore various feature selection and extraction techniques.

### **5.6.4 Cross-Domain Application**

The models can be tested on other related domains to evaluate their adaptability and scalability, ensuring their broad applicability.

### **5.6.5 Incorporation of Domain Knowledge**

Integrating domain-specific knowledge and expertise can refine the model, making it more attuned to the nuances of axial bearing capacity and its influencing factors.

### **5.6.6 Evaluation with More Diverse Data**

It is crucial to evaluate the models with a more diverse and extensive dataset to ensure their robustness and reliability in various scenarios and conditions.

By following these recommendations, future research can build upon the current findings, enhancing the predictive models and contributing further to the understanding of axial bearing capacity and its influencing factors.



## References

- A Comparison of Grid Search and Randomized Search Using Scikit Learn | by Peter Worcester | Medium. (n.d.). Retrieved September 18, 2023, from [https://medium.com/@peterworcester\\_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85](https://medium.com/@peterworcester_29377/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85)
- Adi Pusat Pengelolaan Sumberdaya Lahan, S., dan Mitigasi Bencana Deputi TPSA, W., Teknologi Jl Thamrin No, B. M., & Pusat, J. (2009). PENGKAJIAN KAPASITAS DAYA DUKUNG TANAH GAMBUT DIDAERAH PENGEMBANGAN IRIGASI DI KALIMANTAN TENGAH. *Jurnal Air Indonesia*, 5(2). <https://doi.org/10.29122/JAI.V5I2.2438>
- Alwanas, A. A. H., Al-Musawi, A. A., Salih, S. Q., Tao, H., Ali, M., & Yaseen, Z. M. (2019). Load-carrying capacity and mode failure simulation of beam-column joint connection: Application of self-tuning machine learning model. *Engineering Structures*, 194, 220–229. <https://doi.org/10.1016/J.ENGSTRUCT.2019.05.048>
- Amjad, M., Ahmad, I., Ahmad, M., Wróblewski, P., Kamiński, P., & Amjad, U. (2022). Prediction of Pile Bearing Capacity Using XGBoost Algorithm: Modeling and Performance Evaluation. *Applied Sciences* 2022, Vol. 12, Page 2126, 12(4), 2126. <https://doi.org/10.3390/APP12042126>
- Andradóttir, S. (2006). Chapter 20 An Overview of Simulation Optimization via Random Search. *Handbooks in Operations Research and Management Science*, 13(C), 617–631. [https://doi.org/10.1016/S0927-0507\(06\)13020-0](https://doi.org/10.1016/S0927-0507(06)13020-0)
- Armaghani, D. J., Asteris, P. G., Fatemi, S. A., Hasanipanah, M., Tarinejad, R., Rashid, A. S. A., & Huynh, V. Van. (2020). On the Use of Neuro-Swarm System to Forecast the Pile Settlement. *Applied Sciences* 2020, Vol. 10, Page 1904, 10(6), 1904. <https://doi.org/10.3390/APP10061904>
- Breiman, L. (1996). Bagging Predictors. 24, 123–140.
- Belete, D. M., & Huchaiyah, M. D. (2021). Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results. <https://doi.org/10.1080/1206212X.2021.1974663>, 44(9), 875–886. <https://doi.org/10.1080/1206212X.2021.1974663>
- Bengio, Y. (2000). Gradient-Based Optimization of Hyperparameters. *Neural Computation*, 12(8), 1889–1900. <https://doi.org/10.1162/089976600300015187>
- Bilal, M., Ali, G., Iqbal, M. W., Anwar, M., Malik, M. S. A., & Kadir, R. A. (2022). Auto-Prep: Efficient and Automated Data Preprocessing Pipeline. *IEEE Access*, 10, 107764–107784. <https://doi.org/10.1109/ACCESS.2022.3198662>
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. <https://doi.org/10.1023/A:1010933404324/METRICS>

- Cao, M. T., Nguyen, N. M., & Wang, W. C. (2022). Using an evolutionary heterogeneous ensemble of artificial neural network and multivariate adaptive regression splines to predict bearing capacity in axial piles. *Engineering Structures*, 268, 114769. <https://doi.org/10.1016/J.ENGSTRUCT.2022.114769>
- Chao, Y., Yong, Z. X., & Wen, Z. S. (2020). Research on static load test of foundation piles by self-blance method. *IOP Conference Series: Materials Science and Engineering*, 794(1), 012034. <https://doi.org/10.1088/1757-899X/794/1/012034>
- Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-August-2016, 785–794. <https://doi.org/10.1145/2939672.2939785>
- Claesen, M., & Moor, B. (2015). Hyperparameter Search in Machine Learning. *ArXiv*.
- Dinov, I. D. (2018). Prediction and Internal Statistical Cross Validation. *Data Science and Predictive Analytics*, 697–734. [https://doi.org/10.1007/978-3-319-72347-1\\_21](https://doi.org/10.1007/978-3-319-72347-1_21)
- Garnett, R. (2023). Bayesian Optimization. *Bayesian Optimization*. <https://doi.org/10.1017/9781108348973>
- Goh, A. T. C., Kulhawy, F. H., & Chua, C. G. (2005). Bayesian Neural Network Analysis of Undrained Side Resistance of Drilled Shafts. *Journal of Geotechnical and Geoenvironmental Engineering*, 131(1), 84–93. [https://doi.org/10.1061/\(ASCE\)1090-0241\(2005\)131:1\(84\)](https://doi.org/10.1061/(ASCE)1090-0241(2005)131:1(84))
- Hoang, N. D., Tran, X. L., & Huynh, T. C. (2022). Prediction of Pile Bearing Capacity Using Opposition-Based Differential Flower Pollination-Optimized Least Squares Support Vector Regression (ODFP-LSSVR). *Advances in Civil Engineering*, 2022. <https://doi.org/10.1155/2022/7183700>
- Horning, N. (2010). *Random Forests : An algorithm for image classification and generation of continuous fields data sets*.
- Jeon, J., & Rahman, M. S. (2008). Fuzzy Neural Network Models for Geotechnical Problems.
- Jiang, J., Pan, H., Li, M., Qian, B., Lin, X., & Fan, S. (2021). Predictive model for the 5-year survival status of osteosarcoma patients based on the SEER database and XGBoost algorithm. *Scientific Reports*, 11(1). <https://doi.org/10.1038/S41598-021-85223-4>
- Kärkkäinen, T. (2014). On cross-validation for MLP model evaluation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8621 LNCS, 291–300. [https://doi.org/10.1007/978-3-662-44415-3\\_30/COVER](https://doi.org/10.1007/978-3-662-44415-3_30/COVER)
- Lawatré, P. (2021). An efficient data pre-processing model for machine learning.
- Liashchynskyi, P., & Liashchynskyi, P. (2019). Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. <http://arxiv.org/abs/1912.06059>

- Liu, Y., Wang, Y., & Zhang, J. (2012). New Machine Learning Algorithm: Random Forest. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7473 LNCS, 246–252. [https://doi.org/10.1007/978-3-642-34062-8\\_32](https://doi.org/10.1007/978-3-642-34062-8_32)
- Mangalathu, S., Karthikeyan, K., Feng, D. C., & Jeon, J. S. (2022). Machine-learning interpretability techniques for seismic performance assessment of infrastructure systems. *Engineering Structures*, 250, 112883. <https://doi.org/10.1016/J.ENGSTRUCT.2021.112883>
- Mohana, R. M., Reddy, C. K. K., Anisha, P. R., & Murthy, B. V. R. (2021). WITHDRAWN: Random forest algorithms for the classification of tree-based ensemble. *Materials Today: Proceedings*. <https://doi.org/10.1016/J.MATPR.2021.01.788>
- Mohr, F., & van Rijn, J. N. (2022). Learning Curves for Decision Making in Supervised Machine Learning -- A Survey. <https://arxiv.org/abs/2201.12150v1>
- Mundargi, Z., Bhatti, S., Chandra, A., Kamble, A., Jiby, B., & Arole, R. (2023). PrePy - A Customize Library for Data Preprocessing in Python. 2023 International Conference for Advancement in Technology (ICONAT). <https://doi.org/10.1109/ICONAT57137.2023.10080134>
- Nguyen, H., Cao, M. T., Tran, X. L., Tran, T. H., & Hoang, N. D. (2023). A novel whale optimization algorithm optimized XGBoost regression for estimating bearing capacity of concrete piles. *Neural Computing and Applications*, 35(5), 3825–3852. <https://doi.org/10.1007/S00521-022-07896-W/TABLES/10>
- Pedregosa FABIANPEDREGOSA, F., Michel, V., Grisel OLIVIERGRISEL, O., Blondel, M., Prettenhofer, P., Weiss, R., Vanderplas, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Gramfort, A., Thirion, B., Grisel, O., Dubourg, V., Passos, A., Brucher, M., Perrot andÉdouardand, M., Duchesnay, andÉdouard, & Duchesnay EDOUARDDUCHESNAY, Fré. (2011). Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot. *Journal of Machine Learning Research*, 12, 2825–2830. <http://scikit-learn.sourceforge.net>
- Peskova, K., & Neruda, R. (2019). Hyperparameters search methods for machine learning linear workflows. *Proceedings - 18th IEEE International Conference on Machine Learning and Applications, ICMLA 2019*, 1205–1210. <https://doi.org/10.1109/ICMLA.2019.00199>
- Pham, T. A., Ly, H. B., Tran, V. Q., Giap, L. Van, Vu, H. L. T., & Duong, H. A. T. (2020). Prediction of Pile Axial Bearing Capacity Using Artificial Neural Network and Random Forest. *Applied Sciences* 2020, Vol. 10, Page 1871, 10(5), 1871. <https://doi.org/10.3390/APP10051871>
- Pham, T. A., & Tran, V. Q. (2022). Developing random forest hybridization models for estimating the axial bearing capacity of pile. *PLOS ONE*, 17(3), e0265747. <https://doi.org/10.1371/JOURNAL.PONE.0265747>

- Pham, T. A., Tran, V. Q., Vu, H. L. T., & Ly, H. B. (2020a). Design deep neural network architecture using a genetic algorithm for estimation of pile bearing capacity. *PLOS ONE*, 15(12), e0243030. <https://doi.org/10.1371/JOURNAL.PONE.0243030>
- Pham, T. A., Tran, V. Q., Vu, H. L. T., & Ly, H. B. (2020b). Design deep neural network architecture using a genetic algorithm for estimation of pile bearing capacity. *PLOS ONE*, 15(12), e0243030. <https://doi.org/10.1371/JOURNAL.PONE.0243030>
- Probst, P., Boulesteix, A., & Bischl, B. (2018). Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *J. Mach. Learn. Res.*
- Pulina, L. (2010). Engineering portfolios of Machine Learning algorithms to solve complex tasks in Robotics and Automated Reasoning. *AI Commun.*, 23(1), 61–63. <https://doi.org/10.3233/AIC-2010-0471>
- Rajapakse, R. (2008). Pile Load Tests. *Pile Design and Construction Rules of Thumb*, 389–393. <https://doi.org/10.1016/B978-0-7506-8763-8.00026-X>
- Random Forests. Random forests is a powerful machine... | by Dr. Roi Yehoshua | Medium. (n.d.). Retrieved September 26, 2023, from <https://medium.com/@roiyehe/random-forests-98892261dc49>
- Sarajcev, P., & Meglic, A. (2022). Error analysis of multi-step day-ahead PV production forecasting with chained regressors. *Journal of Physics: Conference Series*, 2369(1), 012051. <https://doi.org/10.1088/1742-6596/2369/1/012051>
- Seifi, F., & Niaki, S. T. A. (2023). Extending the hypergradient descent technique to reduce the time of optimal solution achieved in hyperparameter optimization algorithms. *International Journal of Industrial Engineering Computations*, 14(3), 501–510. <https://doi.org/10.5267/j.ijiec.2023.4.004>
- Shahin, M. A. (2010). Intelligent computing for modeling axial capacity of pile foundations. *Canadian Geotechnical Journal*, 47(2), 230–243. <https://doi.org/10.1139/T09-094/ASSET/IMAGES/T09-094E19H.GIF>
- Sipper, M. (2022). High Per Parameter: A Large-Scale Study of Hyperparameter Tuning for Machine Learning Algorithms. *Algorithms*. <https://doi.org/10.48550/ARXIV.2207.06028>
- Stirrat, A. (1959). Test loading piles.
- Support Vector Regression (SVR) | Analytics Vidhya. (n.d.). Retrieved September 26, 2023, from <https://medium.com/analytics-vidhya/support-vector-regression-svr-model-a-regression-based-machine-learning-approach-f4641670c5bb>
- Takkala, H. R., Khanduri, V., Singh, A., Somepalli, S. N., Maddineni, R., & Patra, S. (2022). Kyphosis Disease Prediction with help of RandomizedSearchCV and AdaBoosting. 2022 13th International Conference on Computing Communication and Networking Technologies, ICCCNT 2022. <https://doi.org/10.1109/ICCCNT54827.2022.9984343>
- Tang, L. (2008). *C Cross-validation Historical Background Scientific Fundamentals*.

- verma, Mr. T., & Gill, D. (2018). TO STUDY VARIOUS TYPES OF SOIL FOR PILE FOUNDATION.
- Vikhar, P. A. (2017). Evolutionary algorithms: A critical review and its future prospects. Proceedings - International Conference on Global Trends in Signal Processing, Information Computing and Communication, ICGTSPICC 2016, 261–265. <https://doi.org/10.1109/ICGTSPICC.2016.7955308>
- Vishnu, M. K., Vishal Rupak, V. R., Vedhapriyaa, S., Sangeetha, M., Manjuladevi, R., & Sagana, C. (2023). Recurrent Gastric Cancer Prediction Using Randomized Search Cv Optimizer. 2023 International Conference on Computer Communication and Informatics, ICCCI 2023. <https://doi.org/10.1109/ICCCI56745.2023.10128409>
- Xie, Y., Zhu, C., Zhou, W., Li, Z., Liu, X., & Tu, M. (2018). Evaluation of machine learning methods for formation lithology identification: A comparison of tuning processes and model performances. Journal of Petroleum Science and Engineering, 160, 182–193. <https://doi.org/10.1016/J.PETROL.2017.10.028>
- Y, C., Kiran, P., & P B, M. (2022). The Novel Method for Data Preprocessing CLI. Advances in Intelligent Systems and Technologies, 117–120. [https://doi.org/10.53759/AIST/978-9914-9946-1-2\\_21](https://doi.org/10.53759/AIST/978-9914-9946-1-2_21)
- Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295–316. <https://doi.org/10.1016/J.NEUCOM.2020.07.061>
- You-xiang, M. (2008). Discussion on Pile Foundation.
- Zakariah, M. (2014). Classification of large datasets using Random Forest Algorithm in various applications: Survey.
- Zhang, F., & O'Donnell, L. J. (2020). Support vector regression. Machine Learning: Methods and Applications to Brain Disorders, 123–140. <https://doi.org/10.1016/B978-0-12-815739-8.00007-9>