UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

UNDERGRADUATE SCHOOL

BACHELOR THESIS

by

Vo Thuy Trang

BI11-270

Information and Communication Technology

Title:

Regression methods for spectral reflectance data

Supervisor: Dr. Tran Giang Son

Lab name: USTH ICTLab

Hanoi, September 2023

Declaration

I hereby, Vo Thuy Trang, declare that my thesis represents my own work after doing an internship at ICTLab, USTH.

I have read the University's internship guideline. In the case of plagiarism appearing in my thesis, I accept responsibility for the conduct of the procedures in accordance with the University's Committee.

Hanoi, September 2023

Signature

Vo Thuy Trang

Acknowledgement

To finish this project we cannot ignore the support of many individuals, and we would like to express our gratitude to them.

Firstly, I am grateful for having the opportunity of doing an Internship at ICTLab, University of Science and Technology of Hanoi. This is a great time for me working with outstanding people, and it is also a chance to prepare for my upcoming journey.

Especially our sincere gratitude to Dr. Tran Giang Son, out supervisor, because of giving us much support, meaningful feedback, and helpful guidance regarding every aspect of our research

Many thanks to the people who work in ICTLab for their assistance during my internship.

Finally, I would like to say thank you to my family who support and keep me in high spirit to finish this project.

Vo Thuy Trang

Hanoi, September, 2023

List of Acronyms

AdaBoost	Adaptive Boosting
CatBoost	Categorical Boosting
K	Potassium
LGBM	Light Gradient Boosting Machine
MAPE	Mean Absolute Percentage Error
MSE	Mean Square Error
P	Phosphorus
PCA	Principal Component Analysis
SVR	Support Vector Regression
XGBoost	Extreme Gradient Boosting

Table of Contents

List of Figures

List of Tables

Abstract

Over time, rice has occupied a central role in our nation's agricultural practices, holding the utmost importance in our lives. The process of growing rice is time-consuming, passing through various tasks such as the initial planting of rice seeds. Among these stages, the application of fertilizers seems to be such an important part. Fertilizers play an unchangeable role in ensuring that rice receives enough of the necessary nutrients for growing and overall development.

However, there is still a challenge for the farmers is the accuracy of nutrient levels in rice leaves before fertilization. It is tricky to know exactly how much nutrition the rice leaves actually have and how much they will need.

Noticing this ongoing problem, we have applied advanced computer techniques, particularly Machine Learning, to deal with it in this project. Our work on this project is to provide a practical solution for farmers to manage fertilizer. In the context of this project, we explore the detailed field of analyzing the nutrition in rice leaves. To do this, we have used various models in Machine Learning, which may be mentioned as Lasso Regression, Linear Regression, and Ridge Regression, which help us to understand the data better.

The impact of the models we have chosen could change the future of rice farming. It has the potential to completely transform how we traditionally accurate the nutrition in rice leaves before, during, and after applying fertilizer. This means that we could simplify one of the most complex tasks of agriculture. By providing a more efficient and accurate way to do this, our project aims to give farmers the tools they need to improve rice farming make it more sustainable, and support the ongoing success of this agricultural sector.

Introduction

1.1. Context and motivation

In farming, fertilizer management is important with growing crops. This work is to apply the right amount of nutrients like Nitrogen (N), Phosphorus (P), and potassium (K). However, in farming, especially in large rice fields, farmers have only one way to know the amount of nutrients needed for rice crops is to rely on automated machines to spread fertilizers. Moreover, they cannot tell exactly how much fertilizer each part of the field gets, and this has a huge effect on the quality of the rice produced. To get that information, the farmers could check the rice nutrient levels by hand or with a specialized machine, but it's hard when working with such a big field.

In this project, we focus on developing Regression Models, which are used in various fields to understand the relationships between different variables, with the data from a rice field in Phu Tho, provided by Dr. Tran Giang Son and his team. Our target is to predict the nutrient levels in the rice plants based on various factors. So in this project, we have tried as many techniques and models as possible to understand and give the best solution to this challenge. However, each model may tproduce slightly different results.

And to give the prediction, and to identify the most accurate model, we evaluated by using three different metrics: Mean Absolute Percentage Error, Mean Squared Error, and R-squared. These metrics help us assess how well the models predict the nutrient levels in the rice plants and allow us to choose the most effective one for this agricultural application.

1.2. Objectives

The target of this project is to develop a model that is capable of identifying the concentrations of Phosphorus (P), potassium (K), and Chlorophyll-a based on reflectance data collected from replicates and their sub-replicated. Additionally, we are engaged in an extensive examination of various regression techniques to detect P, K, and Chlorophyll-a concentration levels. In summary, our target is to comprehensively assess the accuracy and precision of our models, providing valuable insights into their performance and suitability for practical applications.

1.3. Related works

When working on this project, we also did research with other reference documents that have the same or almost similar topics.

In the research "Spectral Reflectance Characteristics and Chlorophyll Content Estimation Model of Quercus aquifolioides Leaves at Different Altitudes in Seijila Mountain"[1] Zhu, J et al, by using a dataset of 60 samples of spectral parameters explored regression models to predict Chlorophyll content. Their analysis contains 9 different spectral characteristics of plants as predictor variables, with Chlorophyll content serving as the response variable. They applied univariate regression techniques including index, linear, and quadratic polynomial models, to control accurate estimation models. As for the result, the R-squared for the RGP model seems to be the greatest one with remarkable values of 0.8523.

Another work we research is the study of "Predicting Nitrogen Content in Rice Leaves Using Multi-Spectral Images with a Hybrid Radial Basis Function Neutral Network and Partial Least-Squares Regression" [2] is the combination of Multi-Spectral Imagse, a Hybrid Radial Basis Function Neural Network, and Partial Least-Squares Regression. The evaluation metrics are MAE, MAPE, and RMSE (the same as MSE but different when RMSE is the square root of MSE). The most remarkable performance goes to the RBFNN model which is outstanding in both the growing and mature stages. During the growing stage, it achieves a MAPE of 0.5399, while with the mature stage, it records a MAPE of 0.1566. These results when compared with GRL seem to be surpassed when this model has the MAPE of 1.0545 with the growing stage and 0.7399 with the mature stage. Which also performs well is the GRM with 1.2395 of MAPE in the growing stage and 1.2272 in the mature stage.

We also made researched with some of Dr. Tran Giang Son other students' internships who worked in the same fields as rice leaves. They utilized a dataset containing numerous hyperspectral images of the rice fields captured on May 6, 2022, which included 122 channels matches with 122 wavelengths. These images served as input, focusing on the pixels within specific areas. Not only did they use Machine Learning models for training but also Deep Learning models, which could make their results better. Among these models, VGG16 stands out as one of the most robust performances. The performance of VGG16 is the most well-built one even though it does not have any impressive results but the overall results are better when compared with others. Another remarkable performance is the R^2 score achieved by DenseNet121 when working on Chlorophyll and N concentration. The difference between our work with others is that firstly, we work with spectral data containing a substantial 2150 wavelengths, which enables us to capture more information so that the result of our works' could be better than with others. Moreover, our work has a totally different way of preprocessing and using more Machine Learning Models, so our work will have more

statistics numbers for comparison and analysis.

1.4. Report Structure

Our report is structured as follows:

- Section 1: Introduction
- Section 2: Theoretical background
- Section 3: Materials and Scientific Methods
- Section 4: Result and Discussion
- Section 5: Conclusion and Future Work

Theoretical background

2.1. Spectral Reflectance

Spectral reflectance is a fundamental concept in the study of how surfaces interact with light. With varied kinds of surfaces, we receive various colors or wavelengths. Different surfaces interact with the sunlight in unique ways. This interaction can involve either reflecting the sunlight or absorbing it, and it all hinges on various factors. This behavior depends mainly on the factors' material [(5)]. Different materials have distinct properties that influence how they respond to sunlight. Additionally, the physical and chemical state of the material also takes part in affecting the reflectance - for example, a surface that is smooth and polished might reflect more light compared to a rough and textured surface.

The environment also matters. The reflectance of a surface tells us how well it bounces back the sun's energy. The angle at which the sunlight strikes the surface, the direction from which it comes, and even the polarization of the light, all contribute to how the surface interacts with the sunlight. In short, reflectance is a measure of how effective a surface is at sending back radiant energy or we can understand it as what fraction of the sunlight is redirected at the surface boundary.

Basically, reflectance is influenced by a material's electric properties and how it responds to the electromagnetic field of light. We can get a comprehensive view of how the surface reflects light by factors such as wavelengths (or frequency) of the light, the polarization, and the angle at which it hits the surface. [(6)] Reflectance across different wavelengths is the creation of a reflectance spectrum. This spectrum outlines how the surface behaves with varying colors of light. This can be particularly useful in determining the amount of P concentration, K concentration, and Chlorophyll in rice leaves

based on how they reflect light with different wavelengths for each concentration.

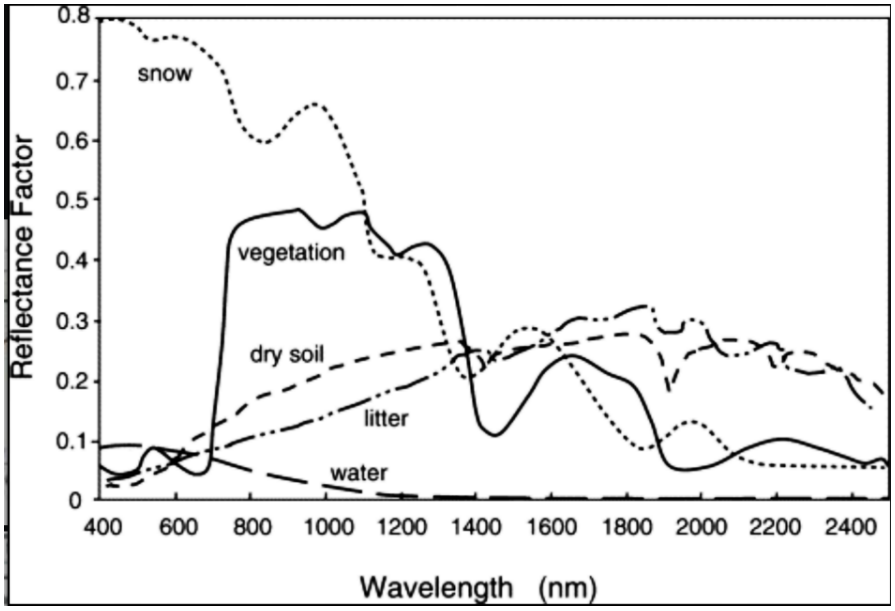

Figure 1 - Spectral reflectance signature of healthy vegetation, dry soil, gray grass litter, water, and snow [18]

Po2.2 Phosphorus concentration

Phosphorus (P) [19] is an important nutrient for plants that supports root growth, energy production, storage, and transfer, and produces flowers and fruits. The lack of P can affect negatively the growth and overall health of plants.

2.3. Potassium concentration

Potassium (K) is an essential nutrient for plant growth and development. It plays an irreplaceable role in the physiological processes within plants. It helps control tiny openings and closing pores called stomata on the surface of the leaves. It also helps move water and nutrients inside plants. [20]

2.4. Chlorophyll

Chlorophyll is a plant's special reaction that takes part in photosynthesis. It plays an important role in the process of photosynthesis which is how plants convert light energy into chemical energy. Chlorophyll's green color comes from its ability to absorb light in the blue

and red parts of the electromagnetic spectrum while reflecting green. (3)

Chlorophyll-A is a special kind of chlorophyll. It grabs most of the energy from violet-blue and orange-red light but it is not very good at catching green light. Instead of reflecting green light, it mostly uses other colors. Chlorophyll-A is the main pigment that helps plants make food from light. (4)

2.5. Regression Analysis

Regression Analysis in the field of Machine Learning is a fundamental tool that helps computers learn and predict outcomes. It helps computers understand the patterns and connections between different variables. Based on past data, it allows computers to predict the future. It predicts continuous/real values such as temperature, age, salary, price, etc. . . Regression analysis contains various algorithms, such as linear regression, polynomial regression, and more, each suited for different types of data and models. The goal is to find the best-fitting function or line that minimizes the error between the predicted values and the actual target values, allowing accurate predictions in each application.

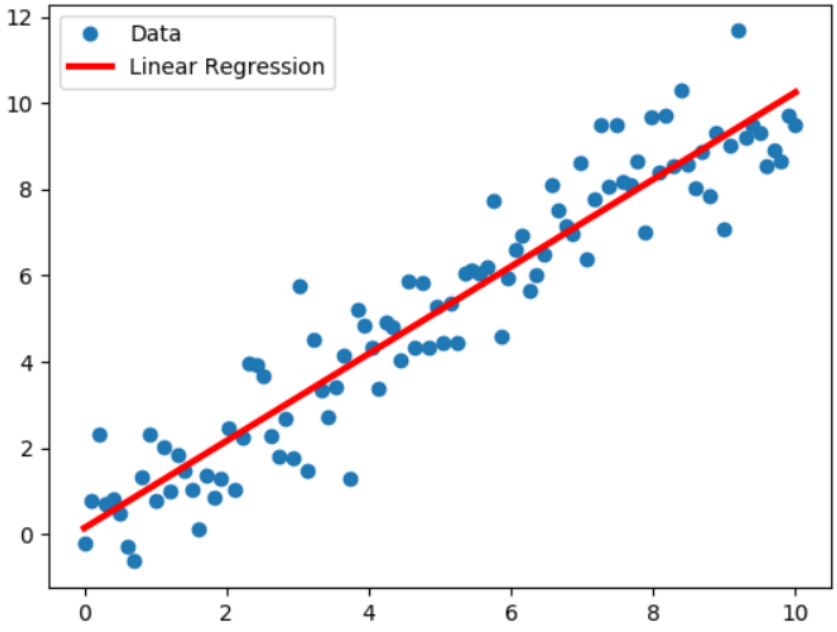

Figure 2 - The scatter plot shows the relationship between the dependent variable and independent variable (7)

2.6. Principal Component Analysis

Principal Component Analysis (PCA) is a helpful tool when dealing with big datasets with a large number of features. It helps us understand the data better by simplifying it. What PCA tries to do is take a bunch of information and figure out what parts are really important and what we can ignore. This makes data easier to understand and visualize. It's used in various fields, such as genetics and climate science, to make complex data simpler and more useful. (8)

This is how the PCA works step-by-step:

Step 1: Standardization

In this step, the goal is to standardize the variables to have the same impact on the analysis by their ranges. This is a necessary step when PCA. If some variables have much larger ranges, they can unfairly dominate the results. Standardizing involves adjusting data to a common scale by subtracting the average and dividing by the standard deviation for each value of each variable.

$z = \frac{value - mean}{standard\ deviation}$

Step 2: Covariance matrix computation

This step is to find connections between variables by comparing how they change from their average to each other. This helps spot repeated info. We use the covariance matrix for this

$cov(X,Y) = \frac{1}{n}\ (X - \underline{X})\,(Y - \underline{Y})$

Where:

- cov(X, Y) is the covariance between X & Y variables
- x & y are members of X and Y variables
- $\underline{x}$ and $\underline{y}$ are mean of X & Y variables
- n is the number of members

It's a chart that shows how variables connect based on their changes. For example, in 3D data with x, y, and z variables, the covariance matrix is a 3x3 grid.

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

The diagonal of the matrix shows each variable's variance because a variable's covariance with itself is its variance (Cov(a, a)=Var(a)). Moreover, due to the commutative property of covariance (Cov(a,b)=Cov(b, a)), the covariance matrix is symmetric which means that the upper and lower triangular sections mirror each other across the main diagonal.

Step 3: Compute the Eigenvectors and Eigenvalues of the covariance matrix to identify the principal components

An Eigenvector is a non-zero vector that changes only by a constant factor when a linear transformation is applied to it and the eigenvalues are the scaling factor. An eigenvalue always comes in pairs with an eigenvector and these numbers are equal to the number of dimensions of the data.

$Av = \lambda v$

- A is the matrix
- V is a special vector
- λ is an eigenvalue

The main key point of this step is organizing data into principal components to reduce dimensional while retaining information. It means that it will eliminate the components with low information and take the rest as the new variables. Principal components represent the directions within the data with the most variance. The eigenvectors of the covariance matrix represent the key directions along which there is the highest variance or as we can say the most information. These special directions actually are what we call Principal Components. When we arrange the eigenvectors based on the eigenvalues, from the highest to the lowest, we can get the order of principal components by their significance.

Step 4: Feature vector

By computing and sorting eigenvectors by their eigenvalues, we identify principal components in order of importance. The feature vector is a matrix formed from the chosen eigenvectors, serving as the first step in dimensionality reduction. Selecting p eigenvectors out of n reduces the dataset to p dimensions.

Step 5: Recast the data along the principal components axes

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

In this final step, the target is to use the feature vector created from the covariance matrix's eigenvectors to reshape the data from its original orientation to the principal components. This transformation is achieved through a matrix multiplication which contains the

transpose of the original dataset and the transpose of the feature vector.

2.7. Machine Learning

2.7.1. Ridge

Ridge regression is also a technique used in linear regression, the same as lasso, which is a tool that helps fix tuning models when dealing with closely related data, called a multicollinearity problem. [9] It applies L2 regularization to handle this. When data has multicollinearity, standard least-square techniques stay unbiased, but variation grows, which leads to substantial differences between predicted parameter estimation. To deal with this, ridge regression adds some bias to improve the accuracy with which parameters are estimated.

The cost function for ridge regression can be performed like this:

$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} ((h_\theta(x^{(i)} - y^{(i)})^2 + \lambda \sum_{j=1}^{m} \theta_j^2)$

where:

- $J(\theta)$: cost function,
- m: number of the training set,
- $h_\theta(x^{(i)})$: predicted output value of i^{th} training examples,
- $y^{(i)}$: real output value of i^{th} training examples,
- λ: regularization term,
- n: number of features,
- θ_j: weight of j^{th} feature.

Ridge regression works best when having more predicted variables than the observations in the data. This is especially useful in cases where the standard assumptions of linear regression might not be valid. It finds a middle ground between effectively capturing relationships within the data and preventing overfitting issues.

2.7.2. Lasso

Lasso regression is a technique used in regression analysis. Like Ridge Regression, it is a way to shrink and select coefficients in linear regression models, especially with many predictors or multicollinearity. It adds a penalty based on absolute coefficient values (L1 regularization), which can precisely zero out some coefficients. Effectively performing feature selection. This prevents overfitting, simplifies models, and is great when only a subset of predictors is vital.

$J(\theta) = \frac{1}{2m}((h_\theta(x^{(i)} - y^{(i)})^2 + \lambda\,|\theta_j|)$

where:

- $J(\theta)$: cost function,
- m: number of the training set,
- $h_\theta(x^{(i)})$: predicted output value of i^{th} training examples,
- $y^{(i)}$: real output value of i^{th} training examples,
- λ: regularization term,
- n: number of features,
- θ_j: weight of j^{th} feature.

What makes Lasso different from Ridge is that while Lasso can lead to zero coefficients, Ridge keeps predictors, although their magnitude is reduced. By ignoring some specific predictors, Lasso is best for its simplicity and clarity, whereas Ridge is well-suited for situations where numerous influential predictors are involved

2.7.3. ElasticNet

ElasticNet is a powerful and flexible regularization method which have both characteristics of Lasso and Ridge regression, it merges the capability of dealing with high-dimensionality and the need for robust regularization techniques which means that ElasticNet is the combination of L1 and L2 regularization. By mixing L1 and L2 regularization, it aims to strike a balance between simplicity and retaining relevant predictors. ElasticNet provides two parameters, alpha, and lambda, to balance these two types of regularization. The alpha parameter controls the mix of L1 and L2 regularization, allowing us to emphasize one over the other or find an equilibrium between the two. The lambda parameter controls the overall strength of regularization, controlling how much the coefficients shrink

$J(\theta) = \frac{1}{2m}((h_\theta(x^{(i)} - y^{(i)})^2 + \lambda_1\theta_j^2 + \lambda_2\theta_j^2)$

where:

- $J(\theta)$: cost function,
- m: number of the training set,
- $h_\theta(x^{(i)})$: predicted output value of i^{th} training examples,
- $y^{(i)}$: real output value of i^{th} training examples,
- λ_1, λ_2: regularization terms,
- n: number of features,

- θ_j: weight of j^{th} feature.

2.7.4. Decision Tree

Decision tree is a versatile algorithm used for classification and regression tasks.It operates by iteratively splitting data into homogenous subsets based on feature conditions, aiming for simplicity and accuracy. However, as trees grow, they risk overfitting, so pruning is included to remove less important branches. To enhance accuracy, decision trees can also form ensembles like random forests, where multiple trees work together for better predictors, especially when they are uncorrelated. This approach aligns with the principle of simplicity, adding complexity only when necessary to achieve accurate results.

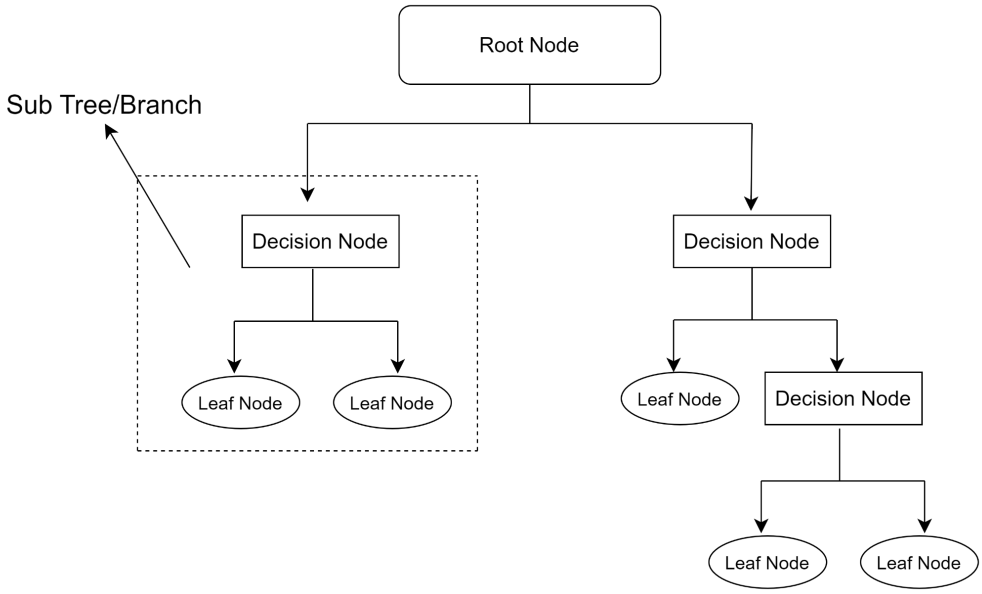

Figure 3 - Workflow diagram of Decision Tree [10]

The decision tree starts with data preparation, where a dataset with input features and a target variable is collected. It selects initial features as the root node, splits the data based on feature values, and calculates the reduction at each step. When the process chooses the best split, which will become the first split of the tree, it will repeat from the beginning to this step. To make predictions, it goes from branches based on feature values to reach leaf nodes, which contain target value predictions that have averaged. After evaluation with metrics like MSE or R squared, it may apply pruning to refine the tree's structure.

2.7.5. Random Forest

Random forest is a technique that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. The Random Forest algorithm is an improvement of the bagging method when integrating both bagging and random features or the random subspace method, to deal with uncorrelated ensemble of decision trees. This technique will make sure that each decision tree in the forest is set for a different set of features, by randomly selecting a subset of features from the dataset.

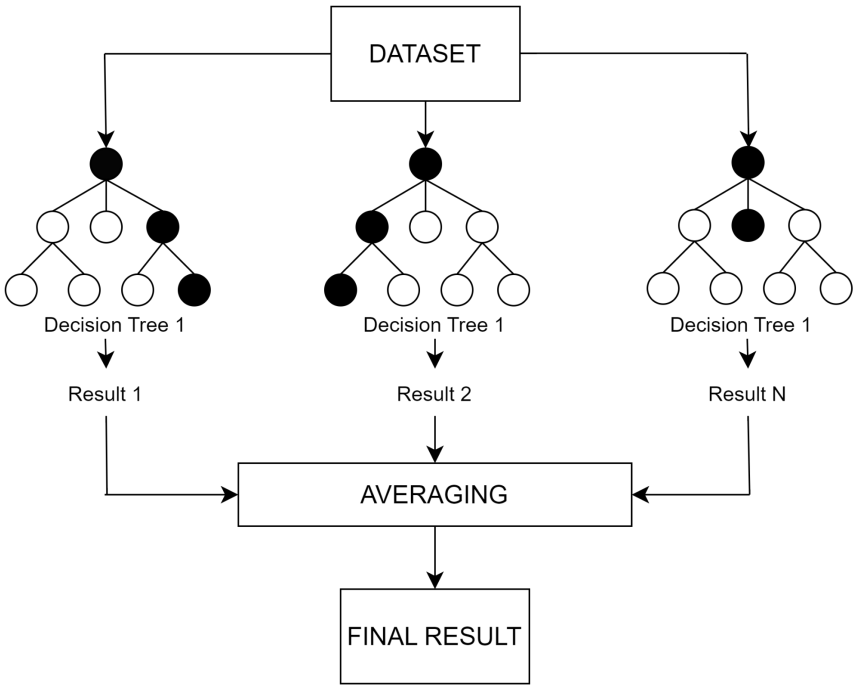

Figure 4 - Workflow diagram of Random Forest [11]

The Random Forest algorithm runs through a bunch of systematic steps to create a predictive model. It starts with data preparation and then uses bootstrapping to create diverse subsets of the data. For each subset, it constructs a decision tree that contains information about the feature randomness, which means that there is only a random subset of features at each node for splitting. Predictions from these trees are combined through averaging for regression and voting for classification to get to the final prediction. The Random Forest model can be working great through parameter adjustments and evaluated using suitable metrics, while also giving a view of feature importance.

Random Forest brings users several advantages such as their versatility and accuracy when handling diverse data types including binary, numerical, and categorical features, making them robust against outliers and nonlinear features. One of the most useful capabilities is its ability to balance errors in populations and unbalanced datasets, measuring feature importance is straightforward. However, they can be slower due to building many trees, limiting real-time use. The predictions rely on past data and may not work well with different ranges. Also, they are not easy to understand and the decisions cannot be explained easily.

2.7.6. SVR

Support Vector Regression (SVR) is a specialized type of support vector machine (SVM) used for regression tasks, targeted to predict continuous output values based on given input data. It can be used both for linear and non-linear kernels, with a simple dot product between input vectors for linear kernels while capturing more complex data patterns for non-linear kernels. Choosing between linear and non-linear is based on data characteristics and task complexity. SVR is based on SVM principles, with the same main role being error minimization while finding a hyperplane with a margin that allows for some error.

This minimization process for the parameter 'w' in the equation is akin to maximizing the margin:

$$\min ||w||^2 + C(\xi_i^+ + \ \xi_i^-)$$

In order to reduce this error, we utilize the following equation, where the summation component represents an empirical error.

To minimize the error, we use the following equation:

$$f(x) = \ (\alpha_i^* + \ \alpha_i) K \ (x, x_i) + B$$

And to calculate the kernel K we can use the following equation:

$$K\,(x, x_i) = \ \gamma\,(x * \ x_i + 1)^d$$

When comparing SVR with other algorithms such as Linear Regression, ElasticNet, it seems to be better. This one is well-suited for a large number of variables because of its capability to be attributed to its refines. It is also outstanding with its flexibility in dealing with geometric, transmission, data generalization, and kernel-related functionalities. The sensitivity to noise is intimately related to the quality of training data. SVR works best with high-dimensional regression problems even when there are more feature metrics than samples. To ensure a fair and balanced evaluation of all features, we need to implement various standardization techniques.

2.7.7. Bayesian Ridge

Bayesian linear regression predicts the mean of one variable using a weighted sum of others. Its goal is to calculate the posterior probability of regression coefficients and other distribution parameters based on observed predictors. Bayesian regression suits datasets with sparse or poorly distributed data, as it derives the posterior distribution of model parameters rather than estimating them directly.

$$p(y \mid X, w, a) \;=\; N\,(y \mid X_w, a)$$

Bayesian Ridge regression, which is the most widely used type of Bayesian regression, models regression problems by incorporating probability estimates. This helps account for uncertainty in predictions, making it useful for situations with limited data or noise. The prior for the coefficients w is given by spherical Gaussian as follows.

$$p(w \mid \lambda) \;=\; N\,(w \mid 0,\; \lambda^{-1} I_p)$$

2.7.8. Lasso Lars

Lasso Lars regression is the mixture of two techniques: Lasso and Lars. Lasso is primarily used for feature selection and addressing multicollinearity in linear regression models. On the other hand, Lars is an algorithm used for efficiently selecting variables in a high-dimensional dataset. Combining these two techniques, Lasso Lars inherits the feature selection capabilities of Lasso and the efficient variable selection process of Lars. This makes it particularly well suited for linear regression tasks involving datasets with many predictor variables or situations where multicollinearity is present.

It starts with all coefficients at zero and selects predictors based on their correlation with the target variable. As it progresses, it employs Lasso's feature selection, encouraging some coefficients to become zero, simplifying the model. This approach excels in handling high-dimensional data efficiently while building interpretable and accurate linear regression models.

2.7.9. Lars

Least Angle Regression (Lars) is used for feature selection and model building, which is specially designed for high dimensional datasets. Its main task is to select and incorporate the most correlated contributions into the model without overfitting. To fit the models, we start by normalizing all values. Then we choose the most highly correlated variable with the residual and adjust the regression line. This process continues until we have used all data or created a satisfactory model.

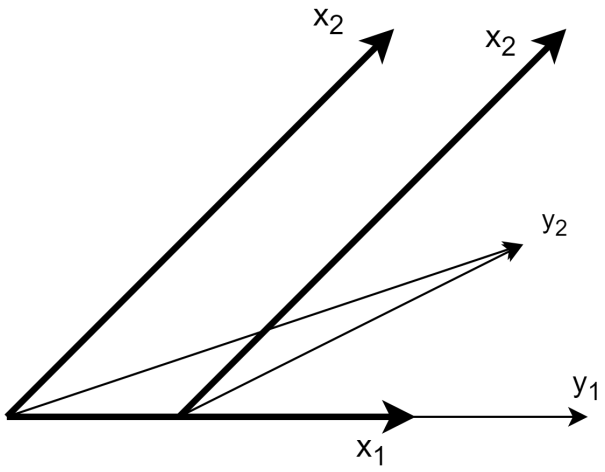

Figure 5 - Lars Skrinkage [12]

With:

- y_2 is the projection of y onto L(x_1, x_2)
- Two covariates x_1 and x_2 and the space L(x_1, x_2) that is spanned by them
- Start at $\mu_0 = 0$

Firstly, we start with all coefficients as zero and find the predictor variable xj that correlates most with the target variable y. Then we increase the coefficients Bj in the direction of this correlation until another predictor variable xk with equal or higher correlation is found. The coefficients (Bj, Bk) are adjusted so that they have the same angle with xj and xk. This process continues until all predictor variables are included in the model.

2.7.10. Boosting

Boosting is a powerful ensemble meta-algorithm in machine learning that reduces bias and variance in supervised learning. It transforms weak learners into strong ones by combining them iteratively and adjusting their weights based on accuracy. Boosting emerged from the idea of enhancing weak learners to create strong ones. Boosting methods assume training weak classifiers sequentially, making it an essential concept in machine learning and statistics. By each stage of adding, a process called 're-weighting', misclassified data points to gain higher weight, allowing weak learners to focus on them. There

are a large number of types of boosting algorithms but in this project, we are supposed to use just the most popular which are AdaBoost, XGBoost, LGBM, CatBoost, and GradientBoost.

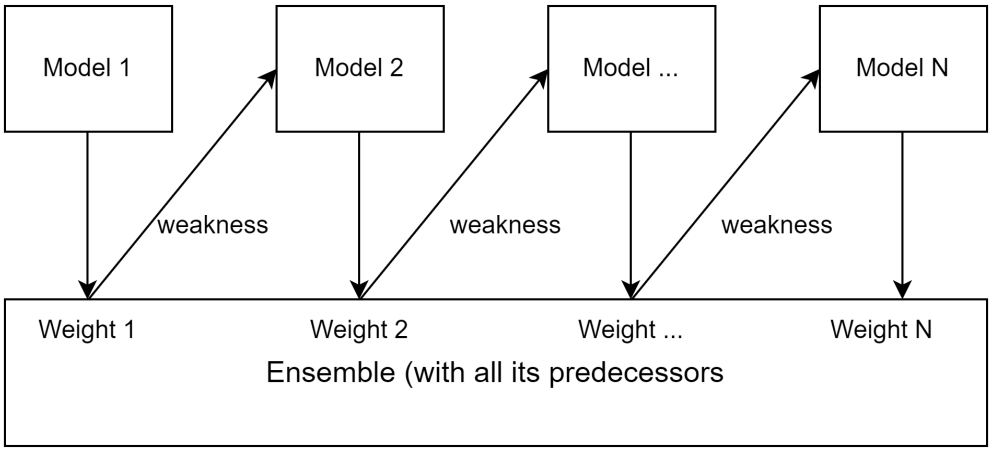

Figure 6 - Workflow of Boosting algorithm

The boosting algorithm contains several advantages, including improved accuracy achieved by combining predictions from weak models, robustness against overfitting by giving more weight to misclassified data points, and effective handling of imbalanced datasets. However, boosting algorithms also have some limitations which can be named as the sensitivity which will make them less suitable when working with real-time applications. Unlike others focused on high-quality predictions, boosting algorithms rely on weak models, each addressing the predecessor's weaknesses.

2.7.10.1. GradientBoosting

GradientBoostung is a robust boosting algorithm that combines weak learners into strong ones by training each new model to minimize the loss function of the previous model using gradient descent. In each iteration, it calculates the gradient of the loss function regarding the predictions of the current ensemble and trains a new weak model to minimize this gradient. This method often uses decision trees as weak learners to transform the data. The iterative procedure involves computing residuals, which represent the difference between predictions and actual values, and training models to map features to these residuals. These steps are to improve the overall predictive performance of the model.

We can express how GradientBoosting works through these steps:

Step 1: Assume X and Y are the input and target with N samples. The goal is to find a function f(x) that maps input features X to target variables Y. The loss function quantifies the difference between actual and predicted values

$$L(f) = \ L(y_i, \ f\left(x_i\right))$$

Step 2: Focuses on minimizing L(f) with respect to f

$$\widehat{f}_0(x) = argmin \ f \ \ L(f) = \ argmin \ f \ \ L(y_i, \ f\left(x_i\right))$$

In our gradient boosting algorithm with M stages, we improve the model f_m by introducing additional estimators denoted as h_m, where m ranges from 1 to M

$$\widehat{y_i} = \ F_{m+1}\left(x_i\right) = \ F_m\left(x_i\right) + \ h_m\left(x_i\right)$$

Step 3: Steepest Descent

$$g_{im} = \ -\lfloor \frac{\partial L(y_i, \ f(x_i))}{\partial f(x_i)} \rfloor_{f(x_i) = \ f_{m-1} \ (x_i)}$$

With the M stage of Gradient Boosting, the Steepest Descent technique is used to determine the h_m which is an important component. This is the combination of 2 elements: constant termed as step length, and the gradient of the loss function g_m. The key point of the step length is to scale the gradient of the loss function L(f).

Step 4: we update the solution iteratively using

$$f_m(x) = \ f_{m-1}(x) + (argmin \ h_{m \in H} \ \ [L\left(y_i, \ f_{m-1}\left(x_i\right) + \ h_m\left(x_i\right)\right)]\left)(x\right)$$

This process continues for M trees, refining the model at each stage to achieve a more accurate prediction. The solution can also be written as:

$$f = \ f_{m-1} - \ \rho_m g_m$$

2.7.10.2. Adaptive Boosting (AdaBoosting)

AdaBoost, short for Adaptive Boosting is a machine learning algorithm that combines the outputs of weak learners to create a strong classifier. It is known for its adaptability and the ability to handle various base learners including weak ones like decision stumps or even strong learners like deep decision trees, making it versatile. AdaBoost adapts by giving more emphasis to instances that previous learners misclassified, reducing the risk of overfitting. It assigns different weights to errors, influencing the importance of weak learners in the final model.

This is an example of how AdaBoost works through a pseudocode:

Input: Data set $D = \{(x_1,\ y_1)\,,\ (x_2,\ y_2)\,,\ \ldots\ (x_m,\ y_m)\}\,;$
Base learning algorithm L;
Output: $H(x) = sign(\alpha_t h_t(x)$
Number of learning round T.
Process:
$D_1(i) = 1/m$ % Initialize the weight distribution
for $t = 1,\ \ldots\ ,\ T$;
$h_t = L\,(D,\ D_t)\,;$ % Train a weak learner h_t from D using distribution D_t
$\epsilon_t = \Pr_{i\sim Di}[h_t(x_i\ \neq\ y_i)]$; % Measure the error of h_t
$\alpha_t = \frac{1}{2}\ln\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$; % Determine the weight of the h_t
$D_{t+1} = \frac{D_t(i)}{Z_t} \times \{\exp\ (-\alpha_t)\ \ if\ h_t(\,x_i) =$
$y_i\ exp\ (\alpha_t)\ \ if\ h_t(\,x_i)\ \neq\ y_i$
$= \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i)}{Z_t}$ % update the distribution, where Z_t is
% a normalization factor which enables D_{t+1} be a distribution
end.

AdaBoost offers several advantages, notably its ease of use with minimal parameter tuning, in contrast to more complex algorithms like SVM. However there are some disadvantages, AdaBoost's progressive learning process demands high-quality data as it's sensitive to noise and outliers.

2.7.10.3. Extreme Gradient Boosting (XGBoost)

XGBoost, short for Extreme Gradient Boosting, is a versatile and powerful machine-learning library designed for solving regression, classifications, and ranking problems. XGBoost relies on Decision Trees as its core component and employs ensemble learning methods such as gradient boosting. Decision trees, which use a series of feature-based questions to predict outcomes, can be employed for regression on predicting numeric values.

Gradient Boosting Decision Tree (GBDT) is an ensemble method that is similar to random forest but differs in how it constructs and combines trees. How GBDT works is to construct a sequence of shallow trees. Each new tree focuses on correcting the errors of the previous ones.

Below is the pseudo-code of how XGBoost works:

Input:
1. Given training data from the instance space
$S = \{(x_1,\ y_1)\ ,\ (x_2,\ y_2)\ ,\ \ldots\ (x_m,\ y_m)\}$ where $x_i\ \epsilon\ X$ and $y_i\ \epsilon\ y = \{\text{-1, +1}\}$
2. Initialize the distribution $D_1(i) = \ \frac{1}{m}$
Process:
for t = 1, . . . , T: do
Train a weak learner $h_t\ : X\ \rightarrow R$ *using distribution* D_t
Determine weight α_t of h_t
Update the distribution over the training set
$D_{i+1}(i) = \ \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i)}{Z_t}$
where Z_t is a normalization factor chosen so that D_{i+1} will be a distribution
end.
Output: $f(x) = \ H(x) = sign(\alpha_t h_t(x)$

XGBoost advantages include its strong track record, scalability, customizability, built-in support for missing values, and feature interpretability. However, it also comes with computational complexity, potential overfitting issues, hyperparameter tuning challenges, and memory requirements, making it difficult to fine-tune its parameters for optimal performance.

2.7.10.4. Categorical Boosting (CatBoost)

CatBoost is an open-source boosting library designed for tackling regression and classification problems that involve a vast number of independent features. What makes CatBoost extended with others Boosting is its ability to work directly with categorical data and its improved implementation of gradient boosting, enhancing speed and performance. CatBoost can deal with a mix of categorical and non-categorical features, Moreover, CatBoost includes symmetric trees, when all decision nodes at the same depth level have the same split criteria, improving its efficiency. This advance is built based on decision tree and gradient boosting principles. The idea is to sequentially combine weak models to construct a predictive model. By fitting gradient boosting over the decision trees sequentially, it has learned from the errors of former trees and reduced prediction errors. This process continues until the chosen loss function can not be reduced anymore.

Instead of following traditional gradient boosting models, CatBoost uses another strategy for constructing decision trees. It uses the oblivious trees, where nodes at the same level evaluate the same predictor with the same condition, allowing the calculation of leaf indices using bitwise operations. This will not only simplify the fitting process and improve CPU efficiency but also can pretend to

be a regularization method to achieve optimal solutions and avoid overfitting.

What makes CatBoost a utility is that it provides a wide range of features that contain native support for categorical characteristics, easy management of missing data, automated feature scaling, and an integrated cross-validation technique for tuning hyperparameters. It also provides the flexibility of using both L1 and L2 regularization methods to deal with the overfitting issue.

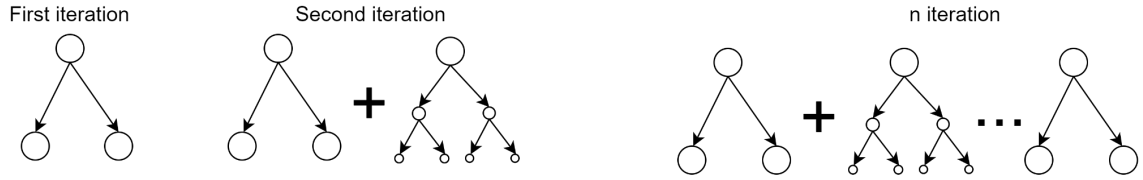

Figure 7 - Illustration of how decision tree works in CatBoost [13]

CatBoost could be the best choice when dealing with small datasets and it has great capability when working with categorical features, as it eliminates the need for extensive feature conversions.

2.7.10.5 LGBM

Light Gradient Boosting Machine, which can be shorted as LGBM, is an open-source gradient boosting framework developed by Microsoft. Its main idea is to address some of the limitations of traditional gradient-boosting methods.

What makes LGBM different from other boosting is that it grows trees by a leaf-wise growth strategy while other algorithms expand trees in a broad way. In the level-wise approach, all the leaf nodes at a particular depth are expanded simultaneously in a breadth-first manner. LGBM selects the leaf node with the maximum delta loss to grow, focusing on nodes that can most efficiently reduce the loss. This leaf-wise strategy often results in fewer levels and, therefore, a shallower tree, which can lead to faster training times and reduced memory usage.

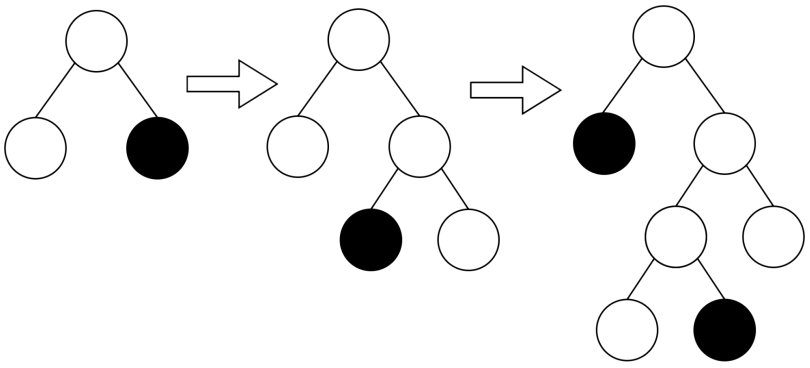

Figure 8 - Illustration of Leaf Wise Tree Grow Architecture of LGBM (14)

LGBM offers faster training speed, lower memory usage, and improved accuracy, making it suitable for large datasets and complex models. LGBM supports parallel, distributed, and GPU learning, enhancing its performance, efficient memory handling, and scalability making it become the best

3. Materials and Scientific Methods

3.1. Data Description

Working on this project, we are under the guidance of Dr. Tran Giang Son and his team. He and his team had divided a field into 3 different sections, each growing three different types of rice seeds and each section received varying nutrient amounts. These nutrients, represented as P, K, and Chlorophyll were applied in different percentages. To capture important information, they used a specialized divide to photograph three specific points in each area to measure spectral reflectance.

Now, as part of our project, we assigned a folder containing two types of files: csv files and a set of sed files. With the csv file, the data is formatted by rows with each row corresponding to data in the sed file. Meanwhile, the sed files contain a large scale of information in machine details, date-time records, and most important thing, the data that aligns with the data rows in the csv file. Our job is to merge the data from these two files together to create a comprehensive and informative dataset.

3.1.1. Structure of .sed files in folder Spectral reflectance measurement

- The number of .sed files: 260 files.

- Name file:
- Types of name:
- Content in a .sed file:
- Version: 2.3 [1.2.6250C]
- File Name
- Instrument: PSR-2500_SN1726293 [2]
- Detectors: 512,0,256
- Measurement: REFLECTANCE
- Date
- Time
- Temperature (c)
- Battery Voltage
- Averages: 10,10
- Integration: 10,30,20,30
- Dark Mode: AUTO, AUTO
- Foreotopic: LENS$ {RADIANCE}, LENS4 {RADIANCE}
- Radiometric Calibration: RADIANCE
- Units: W/m2 /sr/nm
- Wavelength Range: 350,2500
- Latitude
- Longtitude
- Altitude
- GPS Time
- Satellites
- Calibrated Reference Correction File: none
- Channels: 2151
- Columns [4]
- Number of data in a .sed file: 2150 data, corresponding to wavelength from 350-2500, each including:
- Wavelength (Wvl)
- Reference Radian (Rad. (Ref.))

- Target Radian (Rad. (Target.))
- Reflect (Reflect.%)

3.1.2. Structure of DATA_Mua1_2022.csv file

- There are 61 replicates, 55 replicates with 3 sub-replicates
- The number of data rows: 171
- Including:
- Latitude
- Longitude
- Replicate, Sub-replicate
- Concentration of P, K (mg/kg)
- Chlorophyll-a

This is the table showing the nutrients statistics information:

Figure 9 - Nutrients' statistics description

Nutrients	Sample	Mean	Min	Max	St.Deviation
Chlorophyll	171	41.84	33.0	48.6	3.10
P concentration	171	4317.11	1124.0	7740.0	805.68
K concentration	171	35975.34	12620.0	59730.0	10578.94

3.2. Scientific Methods

3.2.1. Overall Framework

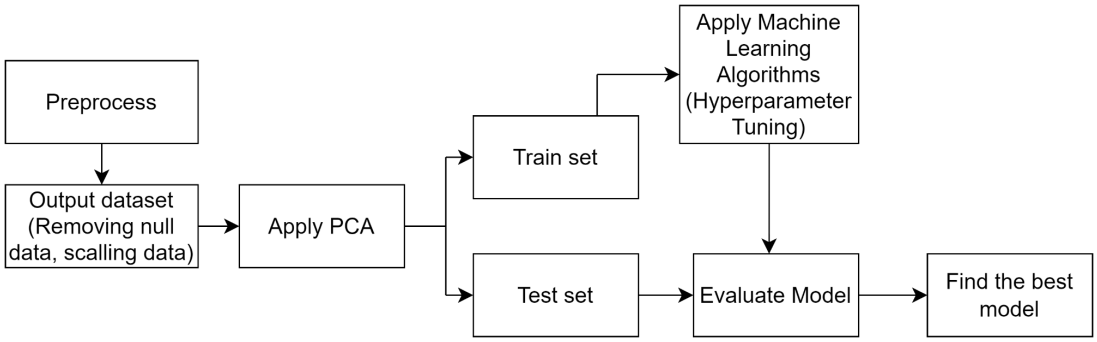

Figure 10 - The workflow for developing the model

Firstly, we work with the Spectral reflectance measurement folder which contains 260 .sed files and each .sed file includes 2150 rows of information. In the preprocessing part, we match the data from .sed files to the data of Replicate and Sub-replicate in DATA_MUA1_2022.csv. After the preprocessing part, we can get an output .csv file of 168 rows and 2154 columns which has removed null data. The next step, the main work on this step is to find and study the dataset when splitting the dataset into the Train set and Test set. While working with models, after standardizing the dataset, we apply machine learning algorithms with hyperparameter tuning techniques to find the most suitable hyperparameter for each model. Because of the high dimension of the dataset, we also apply PCA for it and also try hyperparameter tuning with the after PCA dataset. This hyperparameter tuning is based on the target of satisfying the statistical metrics. We could see how well the model is by evaluating the models. Until we can find the best model based on the results through these metrics, the process will continue iterated. The result of all projects finally is the prediction data and from that, we can conclude which one will be the best model to deal with our challenge.

3.2.2. Preprocess Data

When working on this project, we chose the reflectance data as the input for our regression model. Our objective is to match the reflectance feature from each .sed file, which is integrated with the Replicate and Sub-Replicate specified in the DATA_MUA1_2022.csv file. Before any preprocessing, each .sed file contains a remarkable number of 2150 rows of Reflectance data. What we have to do is transfer these data rows from each sed file into separate feature columns. Each of these feature columns represents data in a different Wvl, Rad. (ref), and Rad. (Target). The actual targets we aim to predict, are named P conc. (mg/kg), K conc. (mg/kg), and Chlorophyll-a are extracted from the DATA_MUA1_2022.csv file, it adds three different actual outputs so that we end up with an impressive total number of 2150 feature columns. After the preprocessing part, our dataset in total has 2153 columns.

3.3. Tools & Library

In this project, I was using Google Colab to run most of the parts and to apply models in machine learning. Moreover, I also used other different libraries for this work:

- Scikit-learn: Scikitlearn is an open-source machine learning library for Python. It offers several tools for various machine

learning tasks including classification, regression, clustering, dimensional reduction, model selection, and data preprocessing.

- Optuna: This is an open-source hyperparameter optimization for machine learning models, which helps in finding the best hyperparameters, making it easier to improve performance and accuracy. Optuna contains various optimization algorithms to search for the optimal hyperparameter in a defined space.
- XGBoost, CatBoost, LGBM: different gradient boosting algorithms to deal with categorical features. Using these boosting algorithms increases performance, speed, and efficiency, reducing overfitting with large-scale datasets.
- Numpy, Matplotlib: used for scientific computing and visualization. Numpy is used for numerical computing and Matplotlib is a tool for creating data visualization.

3.4. Model Configuration and Training

3.4.1. Machine Learning

PCA

Our dataset in this project is pretty large and contains a high number of dimensions, so we apply PCA to reduce the dimension and increase the model's efficiency. To find the best number of components that will be well-suited to the data, we use the result of the explained variance and the cumulative variance [(15)] The figure below shows the information on the explained variance of the dataset.

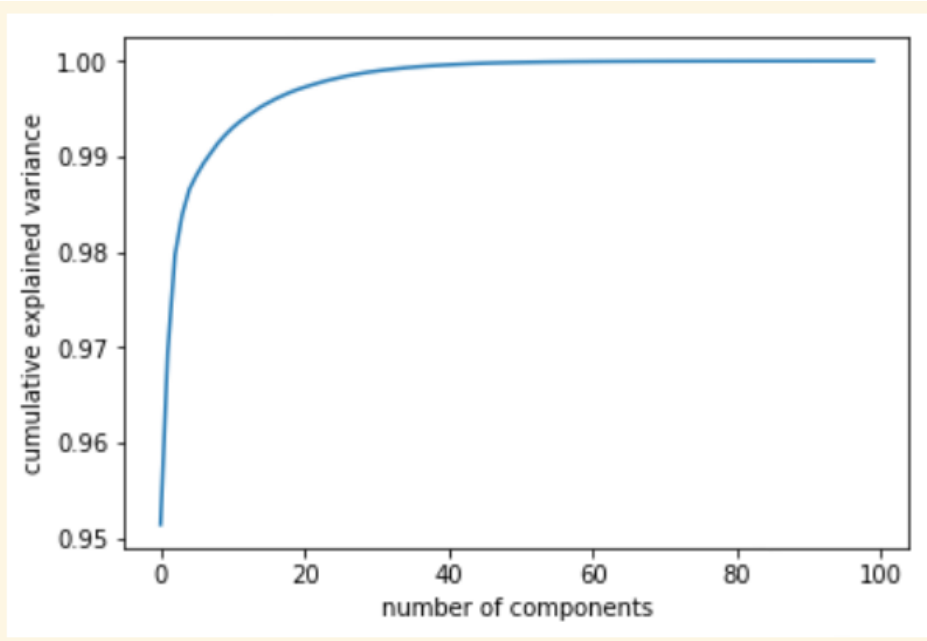

Figure 11 - The number of components needed to explain variance

As we can see in the figure, to get more than 99% of the variance explained, 1 component is enough. So our choice is 1 component of PCA. However in this project, to have more information and more statistical numbers for comparison, we also apply PCA with 3, 5, and 7 components for machine learning models.

Hyperparameter Optimization

Hyperparameter Optimization[17] in Machine learning is the tool to select the best parameters for the learning algorithm. These hyperparameters find the values that can improve the efficiency of model performance. The main idea is to tune parameters to ensure that the model can handle data patterns and minimize a predefined loss function. Cross-validation is commonly used to estimate performance and choose the hyperparameter values that maximize it. There are several ways to optimize hyperparameters such as Grid Search, Random Search, etc. but in this project, we mainly focus on the traditional one, Grid Search. Grid Search is working when specifying a range of parameter values and testing them to see which will work best. This process requires performance metrics like cross-validation to guide the choice. However, when dealing with parameters that can take on real or unbounded values, we may need to set boundaries and discrete values before conducting the grid search. Grid Search is not such a great choice when working with high dimensional parameters

space. So to work with XGBoost, AdaBoost, CatBoost, and LightGBM, we choose Optuna, which is an automatic hyperparameter optimization framework, as a backup plan. Optuna[16] runs based on the define-by-run approach, which will bring flexibility when working with high-dimensional spaces for hyperparameters.

Architecture of Machine Learning Model

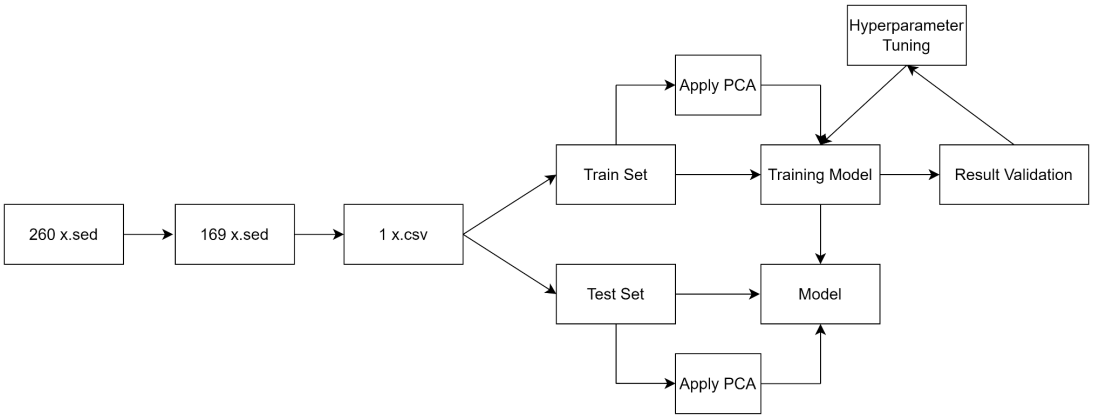

Figure 12 - Illustration of our machine learning workflow

The dataset from the beginning have 260 .sed file in a folder called Spectral reflectance measurement then after preprocessing, eliminating all the null data, and standardizing, we will have the input dataset of the machine learning models. There are 80% of training and the rest 20% is for testing. While working with hyperparameter tuning, we chose a cross-validation of 5 to find the hyperparameter that was well-suited.

3.5 Model Evaluation

3.5.1. Mean Squared Error (MSE)

MSE quantifies the average squared difference between estimated values and actual values. In machine learning, it is one of the popular metrics when evaluating the quality of predictors or estimators. MSE is a non-negative value increasing as errors grow in the model. It considers both variance (how wide estimates vary across data samples) and bias (how far the average between estimates and true value). For an unbiased estimator, MSE equals the variance of the estimator. MSE can be given as the following equation:

$MSE = \frac{1}{n}(Y_i - \widehat{Y}_i)^{\mathbf{2}}$

where:

- Y_i: the i_th observed value,
- $\widehat{Y}_i$: the corresponding predicted value,
- n = the number of observations.

3.5.2. R square

R squared or R^2 (coefficient of determination) is one of the metrics to understand how the output values (variance of a dependent variable) are explained by an independent variable. The value ranges from 0 to 1 and can be negative if the model performs worse than the average fit. An R^2 above 0.7 can be signified as a strong correlation, while below 0.4 signified a weaker one.

$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}}$

where:

- $SS_{Regression}$: the sum of squares due to regression,
- SS_{Total} : the total sum of squares.

3.5.3. MAPE

Mean Absolute Percentage Error (MAPE) is a metric for evaluating the accuracy of predicting methods, especially when we are working with large and nonzero dataset values. It calculates the percentage error between predicted and actual values. A MAPE below 5% is a highly accurate prediction, while a MAPE between 10% and 25% is acceptable. But when it exceeds 25%, it means that the results come out have very low accuracy, equivalent to an unacceptable prediction.

$MAPE = \frac{1}{n} \left| \frac{A_t - F_t}{A_t} \right|$

where:

- n : sample size,
- A_t: the actual data value,
- F_t: the forecasted data value.

4. Result and Discussion

4.1. Chlorophyll Model Prediction and Comparision

Table 1 - Comparison of Learning Models Performance in Chlorophyll Prediction

Models	Without PCA			PCA (1 component)		
	MSE	R^2	MAPE	MSE	R^2	MAPE
Linear Regression	10.16	-0.00	6.38%	10.84	-0.02	6.63%
Lasso Regression	10.64	-0.01	6.57%	10.83	-0.02	6.63%
Decision Tree	11.53	-0.09	6.69%	13.12	-0.24	7.48%
Random Forest	9.41	0.07	6.27%	11.10	-0.05	6.66%
Ridge	10.60	0.00	6.65%	10.83	-0.02	6.63%
ElasticNet	10.41	0.02	6.57%	10.72	-0.01	6.58%
SVR	10.23	-0.01	6.42%	19.29	-0.82	8.49%
Bayesian Ridge	10.17	0.00	6.38%	10.71	-0.01	6.57%
Gradient Boosting	9.97	0.02	6.42%	12.76	-0.21	7.30%
Lasso Lars	10.17	0.00	6.38%	10.83	-0.02	6.63%
Lars	10.64	-0.01	6.57%	10.83	-0.02	6.63%
AdaBoost	11.10	-0.05	6.52%	12.19	-0.15	7.19%
XGBoost	**8.62**	**0.15**	**5.87%**	**8.76**	**0.14**	**6.10%**
CatBoost	9.34	0.08	6.23%	9.73	0.04	6.42%
LGBM	10.17	0.00	6.28%	10.07	0.01	6.42%

Table 2 - Comparison of Learning Models Performance in Chlorophyll Prediction with 3, 5, 7 PCA Components

Models	PCA (3 components)			PCA (5 components)			PCA (7 component)		
	MSE	R^2	MAPE	MSE	R^2	MAPE	MSE	R^2	MAPE
Linear Regression	9.93	0.02	6.34%	9.71	0.04	6.30%	**9.85**	**0.03**	**6.29%**
Lasso Regression	10.17	0.00	6.38%	10.17	0.00	6.38%	10.17	0.00	6.48%
Decision Tree	10.45	-0.03	6.57%	10.45	-0.03	6.57%	10.45	-0.03	6.57%
Random Forest	10.29	0.02	6.53%	9.67	0.05	6.32%	10.38	-0.02	6.41%
Ridge	10.14	0.00	6.49%	9.95	0.02	6.48%	10.87	0.01	6.44%
ElasticNet	10.17	0.00	6.38%	10.17	0.00	6.38%	10.17	0.00	6.38%
SVR	10.24	0.01	6.42%	10.24	0.01	6.42%	10.24	-0.01	6.42%
Bayesian Ridge	10.20	0.01	6.44%	10.20	0.01	6.44%	10.20	-0.01	6.43%
Gradient Boosting	10.78	0.06	6.66%	10.84	0.07	6.74%	10.62	-0.05	6.64%
Lasso Lars	10.18	0.00	6.41%	10.18	0.00	6.41%	10.18	0.00	6.41%
Lars	10.14	0.00	6.49%	9.95	0.02	6.48%	10.09	0.01	6.44%
AdaBoost	10.33	0.02	6.63%	**9.55**	**0.06**	**6.28%**	10.04	0.01	6.46%
XGBoost	**9.71**	**0.04**	**6.35%**	9.78	0.04	6.35%	10.11	0.00	6.41%
CatBoost	9.99	0.01	6.43%	10.05	0.01	6.43%	9.93	0.02	6.37%
LGBM	10.09	0.01	6.36%	10.13	0.00	6.46%	10.09	0.01	6.36%

The Chlorophyll performance with the dataset basically is strong with a quite low result in MAPE of less than 7% for most of the models, some have higher MAPE result, especially after applying 1 PCA component. Among the models, the XGBoost model without applying PCA seems to be the top performer (with an impressive score of 8.62 of MSE, the highest R^2 score, and a MAPE of 5.87% which is also the best one). Even though 1, 3, and 5 components of PCA, XGBoost still performs the best result CatBoost without applying PCA also presents a good result (with 0.08 in R^2 score). With this result, we can conclude that the boosting algorithms are the most well-suited to the dataset. For the machine learning algorithm, there is nothing better than the result of Random Forest without PCA when it has 6.27% in MAPE, 9.41 in MSE and the R^2 score is 0.07 which is also a good score. When taking a look at the higher number of PCA components, we can see that the result seems to be better in some models, especially with AdaBoosting when the R^2 score increases impressively (from -0.15 to 0.01 in R^2 score) through each higher stage of PCA. The results in MAPE and MSE are more stable than with the lower number of components which is around 10 in MSE and around 6.5 in MSE. If choosing one best model for each 3, 5, and 7 PCA components, the greatest in order are XGBoost, AdaBoost, and Linear Regression, among them, Linear Regression

seems to have the overall results pretty well. The worst performance is SVR after applying 1 PCA component's results (with -0.82 in R^2 score, the highest MSE 19.29, and the highest MAPE 8.49%). After applying PCA the results are much better but still not so great. The other bad performance is the Decision Tree after applying 1 component of PCA.

Overall Chlorophyll seems to have the best performance for machine learning models because of the best result when compared with P and K concentrations.

4.2. P Model Prediction and Comparison

Table 3 - Comparison of Learning Models Performance in P Concentration Prediction

Models	Without PCA			PCA (1 component)		
	MSE	R^2	MAPE	MSE	R^2	MAPE
Linear Regression	600400.27	0.01	17.85%	590452.22	0.03	17.61%
Lasso Regression	809576.62	0.00	21.79%	595473.73	0.02	17.63%
Decision Tree	798576.62	0.01	21.57%	582468.28	0.04	17.58%
Random Forest	673434.06	-0.11	18,71%	652626.33	-0.07	18.39%
Ridge	779860.59	0.03	21.29%	590451.83	0.03	17.61%
ElasticNet	805652.41	0.00	21.52%	590618.05	0.03	17.60%
SVR	661599.95	-0.09	18.24%	661215.46	-0.09	18.62%
Bayesian Ridge	591150.94	0.03	17.56%	591921.09	0.03	17.58%
Gradient Boosting	615169.00	0.02	17.94%	580239.58	0.05	17.52%
Lasso Lars	627916.94	-0.33	18.21%	590653.17	0.03	17.59%
Lars	831396.92	-0.03	22.05%	590452.22	0.03	17.61%
AdaBoost	793464.37	0.02	23.31%	563522.39	0.07	17.32%
XGBoost	562302.08	0.07	17.10%	**523516.92**	**0.14**	**16.83%**
CatBoost	563728.23	0.07	17.44%	578611.94	0.05	17.67%
LGBM	627958.64	-0.03	18.21%	627958.64	-0.03	18.21%

Table 4 - Comparison of Learning Models Performance in P Concentration Prediction with 3, 5, 7 PCA Components

Models	PCA (3 components)			PCA (5 components)			PCA (7 component)		
	MSE	R^2	MAPE	MSE	R^2	MAPE	MSE	R^2	MAPE
Linear Regression	590118.51	-0.03	17.64%	602786.91	-0.01	17.58%	605368.62	-0.00	17.66%
Lasso Regression	607261.99	-0.00	17.75%	606748.22	-0.00	17.61%	574846.81	-0.05	**17.05%**
Decision Tree	**582468.28**	-0.04	**17.58%**	642453.51	-0.06	18.60%	642453.51	-0.06	18.60%
Random Forest	661956.18	-0.09	18.49%	685801.38	-0.13	18.53%	601722.18	-0.01	17.72%
Ridge	607261.99	-0.00	17.75%	622814.65	-0.03	17.81%	585243.96	-0.04	17.13%
ElasticNet	607261.98	-0.00	17.75%	614590.82	-0.01	17.72%	581882.20	-0.04	17.17%
SVR	652209.79	-0.07	18.58%	636115.22	-0.05	17.94%	607906.34	-0.00	17.58%
Bayesian Ridge	592322.54	-0.03	17.58%	**591923.27**	**-0.03**	**17.57%**	591301.76	-0.03	17.56%
Gradient Boosting	630209.33	-0.04	18.32%	644217.55	-0.06	18.41%	**560875.99**	**0.08**	17.39%
Lasso Lars	607261.99	-0.00	17.76%	622582.18	-0.02	17.81%	585026.27	-0.04	17.13%
Lars	607261.99	-0.00	17.75%	622791.61	-0.03	17.81%	585244.72	-0.04	17.13%
AdaBoost	653114.10	**-0.07**	18.58%	647898.54	-0.07	18.49%	606994.96	-0.00	18,10%
XGBoost	593746.98	-0.02	17.69%	607394.79	-0.00	17.84%	599482.31	-0.01	17.76%
CatBoost	632922.32	-0.04	18.32%	613780.26	-0.01	18.04%	577259.24	-0.05	17.47%
LGBM	627958.64	-0.03	18.21%	627958.64	-0.03	18.21%	627958.64	-0.03	18.21%

Now moving to the performance of P concentration. Generally, MAPE is quite good (~20%), it seems worse than Chlorophyll's performance but still acceptable. The MSE and the R^2 score are unfortunately high, some of the R^2 scores are very low which cannot beated. Take a look at the models, the results between models are not so different, the changes can be clearly seen before and after applying PCA. We can clearly see that XGBoost with 1 PCA component has the best performance with an impressive R^2 score of 0.14 and the lowest MSE and MAPE. The MSE and MAPE in 3, 5, and 7 PCA components do not change much (around 600000 for MSE and around 18% for MAPE) which can be concluded that they have a stable outcome and don't differ much from the lower component. In order of 3, 5, and 7 PCA components, we can pick out some great results which are the MSE and MAPE of 3 components and an impressive R^2 score belonging to AdaBoost, the results of Bayesian Ridge, and with 7 components we have great values of MSE and R^2 from Gradient Boosting and the MAPE result of Lasso Regression. The performance of Gradient Boosting

with 7 PCA components surprisingly have low results in MSE and high result in R^2 , and it even has better result when compared with 1 PCA component. Even though the overall performance is great there are still some not-great results with a little high MAPE (>20%) without applying PCA, such as Lasso, Ridge, AdaBoost, ElasticNet, and Lars, but Lars seems to have the worst performance when compared in total (with highest MSE and a not to beat R^2 score, and the MAPE is also high but not as high as AdaBoost's MAPE).

4.3. K Model Prediction and Comparison

Table 5 - Comparison of Learning Models Performance in K Concentration Prediction

Models	Without PCA			PCA (1 component)		
	MSE	R^2	MAPE	MSE	R^2	MAPE
Linear Regression	107236125.48	0.01	27.88%	103348457.47	0.02	27.16%
Lasso Regression	127745308.26	0.12	36.57%	103388230.49	0.02	27.22%
Decision Tree	134777212.87	0.18	38.00%	98501105.28	0.07	26.34%
Random Forest	95390644.85	0.10	25.69%	107740656.17	0.02	28.27%
Ridge	127064172.96	0.11	35.70%	103348496.81	0.02	27.16%
ElasticNet	119845469.73	0.05	35.21%	103356522.65	0.02	27.19%
SVR	106589283.16	0.01	27.52%	102270252.63	0.03	27.37%
Bayesian Ridge	107221952.93	0.01	27.88%	103666880.00	0.02	27.35%
Gradient Boosting	96605006.46	0.09	26.19%	104890891.46	0.01	26.73%
Lasso Lars	107233510.81	0.02	27.88%	103348515.03	0.02	27.16%
Lars	126196026.94	0.11	36.11%	103348457.47	0.02	27.16%
AdaBoost	148137256.36	0.30	38.73%	99389152.40	0.06	26.78%
XGBoost	**88801471.26**	**0.16**	**23.80%**	**96620729.06**	**0.09**	**26.80%**
CatBoost	92969278.99	0.12	25.14%	100335635.61	0.05	26.78%
LGBM	104259020.48	0.01	26.89%	105869897.76	0.00	27.70%

Table 6 - Comparison of Learning Models Performance in K Concentration Prediction with 3, 5, 7 PCA Components

Models	PCA (3 components)			PCA (5 components)			PCA (7 component)		
	MSE	R^2	MAPE	MSE	R^2	MAPE	MSE	R^2	MAPE
Linear Regression	106906837.92	-7.21	27.90%	107373204.57	-0.52	27.97%	106619178.77	-8.71	27.84%
Lasso Regression	103388230.49	-0.42	27.22%	103388230.49	-0.42	27.22%	103388230.49	-0.42	27.22%
Decision Tree	98501105.81	4.07	26.34%	98501105.81	4.07	26.34%	111934470.56	-9.56	28.53%
Random Forest	92032913.25	13.13	26.23%	107056102.26	-1.26	27.54%	105495312.60	0.60	26.98%
Ridge	105305752.04	-2.04	27.74%	106784938.11	-1.11	27.62%	101153905.64	5.64	26.97%
ElasticNet	103684573.28	-0.82	27.26%	103690201.92	-0.92	27.24%	103522484.07	-0.73	27.22%
SVR	106627352.45	-1.45	27.55%	106614398.08	-1.08	27.54%	106608902.06	-1.06	27.54%
Bayesian Ridge	107221960.42	-2.42	27.88%	107221946.62	-2.62	27.88%	107221946.62	-2.62	27.88%
Gradient Boosting	109582953.87	-4.87	28.18%	99717999.86	6.86	26.52%	101163563.04	4.04	27.34%
Lasso Lars	103452724.71	-0.72	27.18%	103450924.98	-0.98	27.17%	103408383.92	-0.92	27.17%
Lars	105308653.30	-3.30	27.74%	106815493.01	-3.01	27.63%	101035100.45	5.45	26.96%
AdaBoost	106373421.03	-1.03	27.56%	98205464.30	7.07	26.76%	111498145.16	-9.16	29.01%
XGBoost	**83106778.25**	**16.75**	**23.37%**	**82847194.33**	**17.33**	**23.17%**	**82840259.29**	**17.29**	**23.72%**
CatBoost	100473439.56	0.56	27.00%	85922695.49	14.49	23.73%	83251747.21	16.21	23.66%
LGBM	107236125.03	-1.03	25.35%	91714224.13	10.13	25.69%	86069526.29	13.29	25.19%

Moving on with the K prediction, in general, the MAPE is not so good, especially when compared with Chlorophyll and P. The amount in MAPE is quite high but still not too high (<40%) though MSE is not as good as we expect. The R^2 also has the same problem with Chlorophyll and P is that some of them cannot beat the R^2 score. This is maybe because the quality of our dataset doesn't have a good measure. Overall, to give a comment on these results, we could say that almost every R^2 score when applying 1 PCA component, beats the R^2, and most of the models' results seem to be the best when compared with the same one in other PCA components or without applying PCA. We can see that XGBoost is the best model even without applying PCA, applying PCA with 1 component, and also with higher components. Among them, the XGBoost with 5 PCA components has the best amount (0.22 in R^2 score, 23.17% in MAPE, and 82847194.33 in MSE). Another model that has very good performance is CatBoost with 7 components. On the other hand, AdaBoost without PCA became the worst model when having such high results in MSE and MAPE, and the lowest R^2 (148137256.36 in MSE, -0.30 in R^2 score and surprisingly high

MAPE 38.73%). We can conclude that with K concentration, applying PCA improves the overall performance of all the models, making the outcomes more stable and better.

5. Conclusion and Future Work

5.1. Conclusion

In this project, we work with the prediction of nutrient regression, with such different kinds of concentration, chlorophyll content, P concentration, and K concentration. From the above comparison, we can conclude that the best result is the prediction of the chlorophyll content. This is because the dataset of Chlorophyll is pretty well which can be concluded that the wavelength when collecting data is good. The P and K concentration on the other hand is not as great as with Chlorophyll, especially the worst one among the fields is the K concentration because of its poor performance in MAPE and MSE. Luckily the R^2 score was not the greatest but it is still not so bad when compared to P concentration and Chlorophyll content. The possible causes for this bad performance in P and K may be because of the imbalance in measures P and K concentrations, or an insufficient amount of training data available for learning models. When comparing between applying or without applying PCA, and comparing between applying 1 component of PCA or higher components, we can barely conclude that PCA with 1 component can be a good choice to have better results. Applying higher components like 3, 5, or 7 is also good, but some of them show a decrease in prediction quality so choosing 1 component is better if considered in full view. The only thing that makes the higher components great is their stability through each stage of PCA, which is not so much different from the result with only one component. To choose which is the greatest model that can work best with most of the concentrations is XGBoost when it is outstanding in all of the fields (P, K, and Chlorophyll all have XGBoost as the best performance).

In the modern age, the integration of advanced technologies into agriculture has gained a great amount of popularity. Plant health is important in agriculture, and these technological advancements provide farmers with huge support. Farmers may optimize their operations by using the potential of these technologies, resulting in savings in both time and effort.

5.2. Future Work

In future work, we would like to improve the quality of the dataset and increase the sample in the dataset. Because the dataset is in one season, just having 3 kinds of concentrations is not enough for us to

train a good model. Besides, studying and applying more models and different kinds of learning algorithms such as the models of Deep Learning may work and get better results than what we got here. What we want to work with more is to work with other kinds of objects such as vegetable leaves and also find the best model among a large number of concentrations. We also want to work with not just the PCA but other techniques to deal with the high dimensional data issues

Reference

(1) Zhu, Jiyou and He, Weijun and Yao, Jiangming and Yu, Qiang and Xu, Chengyang and Huang, Huaguo and Jandug, Catherine Mhae. 2020. "(PDF) Spectral Reflectance Characteristics and Chlorophyll Content Estimation Model of Quercus aquifolioides Leaves at Different Altitudes in Sejila Mountain."

(2) Yu, Peigen & Low, Mei & Zhou, Weibiao. 2017. "Development of a partial least squares-artificial neural network (PLS-ANN) hybrid model for the prediction of consumer liking scores of ready-to-drink green tea beverages." 103. 10.1016/j.foodres.2017.10.015.

(3) "Chlorophyll | Definition, Function, & Facts." 2023. Britannica.

(4) "Difference Between Chlorophyll A and Chlorophyl B - BYJU'S." 2021. BYJU'S.

(5) "Spectral Reflectance – The Project Definition." 2015. The Project Definition.

(6) "Spectral Reflectance Curve." 2020. Energy Interactions (continued...).

(7) "What is Regression Analysis?" n.d. TIBCO Software.

(8) Jaadi, Zakaria. n.d. "Principal Component Analysis (PCA) Explained." Built In.

(9) Ashok, Prashanth. n.d. "Ridge Regression Definition & Examples | What is Ridge Regression?" Great Learning. Accessed September 19, 2023. https://www.mygreatlearning.com/blog/what-is-ridge-regression/.

(10) "Decision Tree Algorithm in Machine Learning." n.d. Javatpoint. Accessed September 19, 2023.

(11) "What is a Random Forest?" n.d. TIBCO Software. Accessed September 19, 2023.

(12) Panthagani, Phanindra. n.d. "The Gifted Regressor: Lasso Lars. When it comes to regression, everyone... | by Phanindra

Panthagani." Medium. Accessed September 19, 2023.

(13) Jain, Sandeep. 2023. "CatBoost in Machine Learning." GeeksforGeeks.

(14) Mandot, Pushkar. 2017. "What is LightGBM, How to implement it? How to fine tune the parameters?" Medium.

(15) Kumar, Ajitesh. 2023. "PCA Explained Variance Concepts with Python Example." Analytics Yogi.

(16) Chakrabarti, Sion. 2021. "Optimize your optimizations using Optuna." Analytics Vidhya.https://www.analyticsvidhya.com/blog/2021/09/optimize-your-optimizations-using-optuna/.

(17) "A Comprehensive Guide on Hyperparameter Tuning and its Techniques." 2022. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/.

(18) Joseph, Ajith. n.d. "Fig. 4 Spectral Reflectance Curves for vegetation, soil and water..."

(19) "Why phosphorus is important." NSW Department of Primary Industries.

(20). 2014. "Fertilizer 101: The Big 3 - Nitrogen, Phosphorus and Potassium." The Fertilizer Institute.

Appendix A

Tables of Hyperparameter for each Machine learning models

Table A.1 - Table of good performance' hyperparameters for each models of Chlorophyll

Models	Default	Hyperparmeters
Linear Regression (PCA 7 components)	fit_intercept: True, n_jobs: None, positive: False	fit_intercept: True, n_jobs: 5, positive: True
Lasso Regression (PCA 3 components)	alpha: 1.0, fit_intercept: True, max_iter: 1000	alpha: 34.3, fit_intercept: True, max_iter: 40
Decision Tree (PCA 3 components)	max_leaf_nodes: None, min_samples_split: 2 max_depth: None, max_features: 1.0, min_samples_leaf: 1, min_sample_split: 2,	max_leaf_nodes: 2, min_samples_split: 6 max_depth: 10, max_features: 'auto', min_samples_leaf: 20, min_sample_split: 8,
Random Forest	n_estimators: 100	n_estimators: 10
Ridge (PCA 5 components)	alpha: 1, fit_intercept: True, solver: 'auto'	alpha: 1, fit_intercept: True, solver: 'saga'
ElasticNet (PCA 3 components)	alpha: 1.0, l1_ratio: 0.5, max_iter: 1000	alpha: 100, l1_ratio: 0.4, max_iter: 1
SVR (PCA 3 components)	C: 1,0, coef0: 0.0, epsilon: 0.1, gamma: scale, kernel: 'rbf'	C: 10, coef0: 5.0, epsilon: 0.01, vn gamma: 'scale', kernel: 'sigmoid'
Bayesian Ridge	Alpha_1: 1e-06, alpha_2: 1e-06, lambda_1: 1e-06, lambda_2: 1e-06, n_inter: int	alpha_1: 1e-06, alpha_2: 0.0001, lambda_1: 0.1, lambda_2: 1e-09, n_inter: 100
Gradient Boosting	learning_rate: 0.1, max_depth: 3, n_estimators: 100	learning_rate: 0.01, max_depth: 3, n_estimators: 100
Lasso Lars	alpha: 1.0, eps: np.finfo(float).eps, fit_intercept: True, max_iter: 500, verbose: False	alpha: 0.1, eps: 200.5, fit_intercept: True, max_iter: 10, verbose: True
Lars (PCA 5 components)	eps:, fit_intercept: True, n_nonzero_coefs:	eps: 0.1, fit_intercept: True, n_nonzero_coefs: 10
AdaBoost (PCA 5 components)	learning_rate: 1.0, n_estimators: 50	learning_rate: 0.1, n_estimators: 30
XGBoost	max_depth: 6, learning_rate:, n_estimators: , min_child_weight: 1, gamma: 0, subsample: 1, colsample_bytree: 0.5, reg_alpha: 0, reg_lambda: 1	max_depth: 4, learning_rate: 0.03, n_estimators: 436, min_child_weight: 9, gamma: 0.08, subsample: 0.43, colsample_bytree: 0.06, reg_alpha: 0.0009, reg_lambda: 0.05
CatBoost	Iterations: 500, depth: , learning_rate: reg_alpha: 0.0, reg_lambda: 0.0,44 colsample_bytree: 1.0, subsample: 1.0, learning_rate: 0.1,	iterations: 107, depth: 10, learning_rate: 0.09 reg_alpha: 0.006, reg_lambda: 0.04, colsample_bytree: 0.5, subsample: 0.4, learning_rate: 0.02,
LGBM (PCA 1 component)	min_child_samples: 20, min_data_per_groups: 20	min_child_samples: 78, min_data_per_groups: 41

Models	Default	Hyperparmeters

Models	Default	Hyperparmeters
Linear Regression (PCA 7 components)	fit_intercept: True, n_jobs: None, positive: False	fit_intercept: True, n_jobs: 5, positive: True
Lasso Regression (PCA 3 components)	alpha: 1.0, fit_intercept: True, max_iter: 1000	alpha: 34.3, fit_intercept: True, max_iter: 40
Decision Tree (PCA 3 components)	max_leaf_nodes: None, min_samples_split: 2 max_depth: None, max_features: 1.0, min_samples_leaf: 1, min_sample_split: 2,	max_leaf_nodes: 2, min_samples_split: 6 max_depth: 10, max_features: 'auto', min_samples_leaf: 20, min_sample_split: 8,
Random Forest	n_estimators: 100	n_estimators: 10
Ridge (PCA 5 components)	alpha: 1, fit_intercept: True, solver: 'auto'	alpha: 1, fit_intercept: True, solver: 'saga'
ElasticNet (PCA 3 components)	alpha: 1.0, l1_ratio: 0.5, max_iter: 1000	alpha: 100, l1_ratio: 0.4, max_iter: 1
SVR (PCA 3 components)	C: 1,0, coef0: 0.0, epsilon: 0.1, gamma: scale, kernel: 'rbf'	C: 10, coef0: 5.0, epsilon: 0.01, vn gamma: 'scale', kernel: 'sigmoid'
Bayesian Ridge	Alpha_1: 1e-06, alpha_2: 1e-06, lambda_1: 1e-06, lambda_2: 1e-06, n_inter: int	alpha_1: 1e-06, alpha_2: 0.0001, lambda_1: 0.1, lambda_2: 1e-09, n_inter: 100
Gradient Boosting	learning_rate: 0.1, max_depth: 3, n_estimators: 100	learning_rate: 0.01, max_depth: 3, n_estimators: 100
Lasso Lars	alpha: 1.0, eps: np.finfo(float).eps, fit_intercept: True, max_iter: 500, verbose: False	alpha: 0.1, eps: 200.5, fit_intercept: True, max_iter: 10, verbose: True
Lars (PCA 5 components)	eps:, fit_intercept: True, n_nonzero_coefs:	eps: 0.1, fit_intercept: True, n_nonzero_coefs: 10
AdaBoost (PCA 5 components)	learning_rate: 1.0, n_estimators: 50	learning_rate: 0.1, n_estimators: 30
XGBoost	max_depth: 6, learning_rate:, n_estimators: , min_child_weight: 1, gamma: 0, subsample: 1, colsample_bytree: 0.5, reg_alpha: 0, reg_lambda: 1	max_depth: 4, learning_rate: 0.03, n_estimators: 436, min_child_weight: 9, gamma: 0.08, subsample: 0.43, colsample_bytree: 0.06, reg_alpha: 0.0009, reg_lambda: 0.05
CatBoost	Iterations: 500, depth: , learning_rate: reg_alpha: 0.0, reg_lambda: 0.0,45 colsample_bytree: 1.0, subsample: 1.0, learning_rate: 0.1, min_child_samples: 20,	iterations: 107, depth: 10, learning_rate: 0.09 reg_alpha: 0.006, reg_lambda: 0.04, colsample_bytree: 0.5, subsample: 0.4, learning_rate: 0.02,
LGBM (PCA 1 component)	min_data_per_groups: 20	min_child_samples: 78, min_data_per_groups: 41

Table A.2 - Table of good performance' hyperparameters for each models of P

Models	Default	Hyperparmeters
Linear Regression (PCA 3 components)	fit_intercept: True, n_jobs: None, positive: False	fit_intercept: True, n_jobs: 5, positive: True
Lasso Regression (PCA 7 components)	alpha: 1.0, fit_intercept: True, max_iter: 1000	alpha: 104.76, fit_intercept: True, max_iter: 40
Decision Tree (PCA 3 components)	max_leaf_nodes: None, min_samples_split: 2	max_leaf_nodes: 2, min_samples_split: 6
Random Forest (PCA 7 components	max_depth: None, max_features: 1.0, min_samples_leaf: 1, min_sample_split: 2, n_estimators: 100	max_depth: 10, max_features: 'auto', min_samples_leaf: 20, min_sample_split: 8, n_estimators: 10
Ridge (PCA 7 components)	alpha: 1, fit_intercept: True, solver: 'auto'	alpha: 1, fit_intercept: True, solver: 'saga'
ElasticNet (PCA 3 components)	alpha: 1.0, l1_ratio: 0.5, max_iter: 1000	alpha: 100, l1_ratio: 0.4, max_iter: 1
SVR (PCA 3 components)	C: 1,0, coef0: 0.0, epsilon: 0.1, gamma: scale, kernel: 'rbf'	C: 10, coef0: 5.0, epsilon: 0.01, gamma: 'scale', kernel: 'sigmoid'
Bayesian Ridge	alpha_1: 1e-06, alpha_2: 1e-06, lambda_1: 1e-06, lambda_2: 1e-06, n_inter: int	alpha_1: 1e-06, alpha_2: 0.0001, lambda_1: 0.1, lambda_2: 1e-09, n_inter: 100
Gradient Boosting (PCA 7 components	learning_rate: 0.1, max_depth: 3, n_estimators: 100	learning_rate: 0.01, max_depth: 3, n_estimators: 100
Lasso Lars (PCA 7 components)	alpha: 1.0, eps: np.finfo(float).eps, fit_intercept: True, max_iter: 500, verbose: False	alpha: 0.1, eps: 200.5, fit_intercept: True, max_iter: 10, verbose: True
Lars (PCA 5 components)	eps:, fit_intercept: True, n_nonzero_coefs:	eps: 0.1, fit_intercept: True, n_nonzero_coefs: 10
AdaBoost (PCA 5 components)	learning_rate: 1.0, n_estimators: 50	learning_rate: 0.1, n_estimators: 30
XGBoost (PCA 3 components)	max_depth: 6, min_child_weight: 1, gamma: 0, subsample: 1, colsample_bytree: 0.5, reg_alpha: 0, reg_lambda: 1	max_depth: 6, learning_rate:0.14, n_estimators: 110, min_child_weight: 4, gamma: 3.04e-05, subsample: 0.43, colsample_bytree: 0.20, reg_alpha: 0.03, reg_lambda: 0.002
CatBoost	Iterations: 500, learning_rate: 0.1	iterations: 107, depth: 10, learning_rate: 0.092
LGBM	reg_alpha: 0.0, reg_lambda: 0.0,48 colsample_bytree: 1.0, subsample: 1.0, learning_rate: 0.1, min_child_samples: 20, min_data_per_groups: 20	reg_alpha: 7.85, reg_lambda: 0.001, colsample_bytree: 0.8, subsample: 0.6, learning_rate: 0.006, min_child_samples: 74, min_data_per_groups: 62

Models	Default	Hyperparmeters

Models	Default	Hyperparmeters
Linear Regression (PCA 3 components)	fit_intercept: True, n_jobs: None, positive: False	fit_intercept: True, n_jobs: 5, positive: True
Lasso Regression (PCA 7 components)	alpha: 1.0, fit_intercept: True, max_iter: 1000	alpha: 104.76, fit_intercept: True, max_iter: 40
Decision Tree (PCA 3 components)	max_leaf_nodes: None, min_samples_split: 2	max_leaf_nodes: 2, min_samples_split: 6
Random Forest (PCA 7 components	max_depth: None, max_features: 1.0, min_samples_leaf: 1, min_sample_split: 2, n_estimators: 100	max_depth: 10, max_features: 'auto', min_samples_leaf: 20, min_sample_split: 8, n_estimators: 10
Ridge (PCA 7 components)	alpha: 1, fit_intercept: True, solver: 'auto'	alpha: 1, fit_intercept: True, solver: 'saga'
ElasticNet (PCA 3 components)	alpha: 1.0, l1_ratio: 0.5, max_iter: 1000	alpha: 100, l1_ratio: 0.4, max_iter: 1
SVR (PCA 3 components)	C: 1,0, coef0: 0.0, epsilon: 0.1, gamma: scale, kernel: 'rbf'	C: 10, coef0: 5.0, epsilon: 0.01, gamma: 'scale', kernel: 'sigmoid'
Bayesian Ridge	alpha_1: 1e-06, alpha_2: 1e-06, lambda_1: 1e-06, lambda_2: 1e-06, n_inter: int	alpha_1: 1e-06, alpha_2: 0.0001, lambda_1: 0.1, lambda_2: 1e-09, n_inter: 100
Gradient Boosting (PCA 7 components	learning_rate: 0.1, max_depth: 3, n_estimators: 100	learning_rate: 0.01, max_depth: 3, n_estimators: 100
Lasso Lars (PCA 7 components)	alpha: 1.0, eps: np.finfo(float).eps, fit_intercept: True, max_iter: 500, verbose: False	alpha: 0.1, eps: 200.5, fit_intercept: True, max_iter: 10, verbose: True
Lars (PCA 5 components)	eps:, fit_intercept: True, n_nonzero_coefs:	eps: 0.1, fit_intercept: True, n_nonzero_coefs: 10
AdaBoost (PCA 5 components)	learning_rate: 1.0, n_estimators: 50	learning_rate: 0.1, n_estimators: 30
XGBoost (PCA 3 components)	max_depth: 6, min_child_weight: 1, gamma: 0, subsample: 1, colsample_bytree: 0.5, reg_alpha: 0, reg_lambda: 1	max_depth: 6, learning_rate:0.14, n_estimators: 110, min_child_weight: 4, gamma: 3.04e-05, subsample: 0.43, colsample_bytree: 0.20, reg_alpha: 0.03, reg_lambda: 0.002
CatBoost	Iterations: 500, learning_rate: 0.1 reg_alpha: 0.0, reg_lambda: 0.0,49	iterations: 107, depth: 10, learning_rate: 0.092
LGBM	colsample_bytree: 1.0, subsample: 1.0, learning_rate: 0.1, min_child_samples: 20, min_data_per_groups: 20	reg_alpha: 7.85, reg_lambda: 0.001, colsample_bytree: 0.8, subsample: 0.6, learning_rate: 0.006, min_child_samples: 74, min_data_per_groups: 62

Table A.3 - Table of good performance' hyperparameters for each models of K

Models	Default	Hyperparmeters
Linear Regression (PCA 1 components)	fit_intercept: True, n_jobs: None, positive: False	fit_intercept: True, n_jobs: 5, positive: True
Lasso Regression (PCA 1 components)	alpha: 1.0, fit_intercept: True, max_iter: 1000	alpha: 9111.62, fit_intercept: True, max_iter: 40
Decision Tree (PCA 1 components)	max_leaf_nodes: None, min_samples_split: 2	max_leaf_nodes: 2, min_samples_split: 2
Random Forest (PCA 3 components)	max_depth: None, max_features: 1.0, min_samples_leaf: 1, min_sample_split: 2, n_estimators: 100	max_depth: 9, max_features: 'sqrt', min_samples_leaf: 10, min_sample_split: 4, n_estimators: 20
Ridge (PCA 7 components)	alpha: 1, fit_intercept: True, solver: 'auto'	alpha: 1, fit_intercept: True, solver: 'saga'
ElasticNet (PCA 7 components)	alpha: 1.0, l1_ratio: 0.5, max_iter: 1000	alpha: 100, l1_ratio: 0.0, max_iter: 1
SVR (PCA 1 components)	C: 1,0, coef0: 0.0, epsilon: 0.1, gamma: scale, kernel: 'rbf'	C: 1, coef0: 1.5, epsilon: 0.1, gamma: 'scale', kernel: 'poly'
Bayesian Ridge (PCA 1 components)	alpha_1: 1e-06, alpha_2: 1e-06, lambda_1: 1e-06, lambda_2: 1e-06, n_inter: int	alpha_1: 0.001, alpha_2: 1e-06, lambda_1: 0.1, lambda_2: 0.1, n_inter: 100
Gradient Boosting	learning_rate: 0.1, max_depth: 3, n_estimators: 100	learning_rate: 0.01, max_depth: 3, n_estimators: 100
Lasso Lars (PCA 1 components)	alpha: 1.0, eps: np.finfo(float).eps, fit_intercept: True, max_iter: 500, verbose: False	alpha: 0.1, eps: 200.5, fit_intercept: True, max_iter: 10, verbose: True
Lars (PCA 7 components)	eps:, fit_intercept: True, n_nonzero_coefs:	eps: 0.1, fit_intercept: True, n_nonzero_coefs: 10
AdaBoost (PCA 5 component)	learning_rate: 1.0, n_estimators: 50	learning_rate: 0.1, n_estimators: 30
XGBoost (PCA 5 components)	max_depth: 6, learning_rate:, n_estimators: , min_child_weight: 1, gamma: 0, subsample: 1, colsample_bytree: 0.5, reg_alpha: 0, reg_lambda: 1	max_depth: 4, learning_rate: 0.032, n_estimators: 436, min_child_weight: 9, gamma: 0.079, subsample: 0.43, colsample_bytree: 0.06, reg_alpha: 0.0009, reg_lambda: 0.053
CatBoost (PCA 7 components)	iterations: 500, learning_rate: reg_alpha: 0.0, reg_lambda: 0.0,52 colsample_bytree: 1.0, subsample: 1.0, learning_rate: 0.1, min_child_samples: 20,	iterations: 404, depth: 5, learning_rate: 0.057 reg_alpha: 0.003, reg_lambda: 0.77, colsample_bytree: 0.4, subsample: 0.7, learning_rate: 0.006,
LGBM (PCA 7 components)	min_data_per_groups: 20	min_child_samples: 3, min_data_per_groups: 100

Models	Default	Hyperparmeters

Models	Default	Hyperparmeters
Linear Regression (PCA 1 components)	fit_intercept: True, n_jobs: None, positive: False	fit_intercept: True, n_jobs: 5, positive: True
Lasso Regression (PCA 1 components)	alpha: 1.0, fit_intercept: True, max_iter: 1000	alpha: 9111.62, fit_intercept: True, max_iter: 40
Decision Tree (PCA 1 components)	max_leaf_nodes: None, min_samples_split: 2	max_leaf_nodes: 2, min_samples_split: 2
Random Forest (PCA 3 components)	max_depth: None, max_features: 1.0, min_samples_leaf: 1, min_sample_split: 2, n_estimators: 100	max_depth: 9, max_features: 'sqrt', min_samples_leaf: 10, min_sample_split: 4, n_estimators: 20
Ridge (PCA 7 components)	alpha: 1, fit_intercept: True, solver: 'auto'	alpha: 1, fit_intercept: True, solver: 'saga'
ElasticNet (PCA 7 components)	alpha: 1.0, l1_ratio: 0.5, max_iter: 1000	alpha: 100, l1_ratio: 0.0, max_iter: 1
SVR (PCA 1 components)	C: 1,0, coef0: 0.0, epsilon: 0.1, gamma: scale, kernel: 'rbf'	C: 1, coef0: 1.5, epsilon: 0.1, gamma: 'scale', kernel: 'poly'
Bayesian Ridge (PCA 1 components)	alpha_1: 1e-06, alpha_2: 1e-06, lambda_1: 1e-06, lambda_2: 1e-06, n_inter: int	alpha_1: 0.001, alpha_2: 1e-06, lambda_1: 0.1, lambda_2: 0.1, n_inter: 100
Gradient Boosting	learning_rate: 0.1, max_depth: 3, n_estimators: 100	learning_rate: 0.01, max_depth: 3, n_estimators: 100
Lasso Lars (PCA 1 components)	alpha: 1.0, eps: np.finfo(float).eps, fit_intercept: True, max_iter: 500, verbose: False	alpha: 0.1, eps: 200.5, fit_intercept: True, max_iter: 10, verbose: True
Lars (PCA 7 components)	eps:, fit_intercept: True, n_nonzero_coefs:	eps: 0.1, fit_intercept: True, n_nonzero_coefs: 10
AdaBoost (PCA 5 component)	learning_rate: 1.0, n_estimators: 50	learning_rate: 0.1, n_estimators: 30
XGBoost (PCA 5 components)	max_depth: 6, learning_rate:, n_estimators: , min_child_weight: 1, gamma: 0, subsample: 1, colsample_bytree: 0.5, reg_alpha: 0, reg_lambda: 1	max_depth: 4, learning_rate: 0.032, n_estimators: 436, min_child_weight: 9, gamma: 0.079, subsample: 0.43, colsample_bytree: 0.06, reg_alpha: 0.0009, reg_lambda: 0.053
CatBoost (PCA 7 components)	iterations: 500, learning_rate: reg_alpha: 0.0, reg_lambda: 0.0,53 colsample_bytree: 1.0, subsample: 1.0, learning_rate: 0.1, min_child_samples: 20,	iterations: 404, depth: 5, learning_rate: 0.057 reg_alpha: 0.003, reg_lambda: 0.77, colsample_bytree: 0.4, subsample: 0.7, learning_rate: 0.006,
LGBM (PCA 7 components)	min_data_per_groups: 20	min_child_samples: 3, min_data_per_groups: 100