Master's Thesis

Deggendorf Institute of Technology

Faculty of Computer Science

Master Life Science Informatics

Development of an R Toolbox for Near-Infrared Spectroscopy Data Processing and Analysis of Plant Metabolic Phenotypes

Master's thesis to obtain the academic degree:
Master of Science (M.Sc.)

Submitted by:
Methun George

Matriculation Number: 00805682

Supervised by:
PD Dr. habil. rer. nat. Steffen Neumann
Prof. Dr. Melanie Kappelmann-Fenzl

Place, Date:
Deggendorf, February 17, 2025

Acknowledgements

This Master Thesis has been an incredibly enjoyable experience. In pursuit of my passion for bioinformatics and molecular biology, this master's thesis project has been a profoundly enriching journey. It offered me an incredible platform to implement the knowledge I gained in various programming languages and bioinformatics tools during my master's studies along with the knowledge of botany I gained from my bachelor degree. This thesis not only deepened my technical expertise but also provided an invaluable opportunity to explore the practical applications of these skills in real-world research.

I am deeply indebted to my supervisor, Dr. Steffen Neumann, at the Leibniz Institute of Plant Biochemistry (IPB Halle), for his unwavering support, insightful guidance, and expert advice. The mentorship of Dr. Neumann played a great role in navigating the complexities of this project and sharpening my understanding of computational approaches in plant biology. Thank you for making this possible and for the always open door.

I express my heartfelt gratitude to my professor, Dr. Melanie Kappelmann-Fenzl, at the Technische Hochschule Deggendorf (THD), for her academic guidance and constant encouragement throughout my master's degree. Her feedback and insights were instrumental in shaping the direction and quality of my research.

I am profoundly grateful to the Computational Plant Biochemistry (CPB) and MetaCom team at IPB Halle, especially Dr. Henriette Uthe, Dr. Rene Meier, Dr. Michael Wenk, Oliver Duchrow, Dr. Khabat Vahabi, and Norman Storz, whose expertise and assistance greatly enriched my learning experience. Their collaborative spirit and support made this research endeavor not only productive but also highly inspiring.

My deepest gratitude goes to my family and friends, whose unending love, unwavering support, and endless encouragement have been my foundation throughout this journey. Their belief in me has been the source of my strength and motivation.

Lastly, I would like to express my sincere appreciation to all the faculty members of the Life Science Informatics program at the Technische Hochschule Deggendorf and all the members in Leibniz-Institute for Plant Biochemistry. Their guidance and encouragement at various stages of my degree have been invaluable.

This thesis is a culmination of the collective support and encouragement that I have received from all these individuals and institutions, and I am ever grateful for their contributions to my academic and professional growth.

Abstract

The abstract of this thesis is.....

Contents

Abbreviations

Abbreviation	Full Form
NIRS	Near Infrared Spectroscopy
LC-MS	Liquid Chromatography Mass Spectroscopy
CNN	Convolutional Neural Network
DL	Deep Learning
PLSR	Partial Least Squares Regression
PCA	Principal Component Analysis
ML	Machine Learning
NIRS	Near-Infrared Spectroscopy
SLA	Specific Leaf Area
RF	Random Forest
SLA	Specific Leaf Area
LDMC	Lead Dry Matter Content
R^2	Coefficient of Determination
ADAM	Adaptive Moment Estimation
SGD	Stochastic-Gradient Descent
AdaGrad	Adaptive Gradient Algorithm
RMSProp	Root Mean Square Propagation
MSC	Multiplicative Scatter Correction
SNV	Standard Normal Variate
GMO	Genetically Modified Organisms
HCA	Hierarchical Cluster Analysis
PLSDA	Partial Least Squares Discriminant Analysis
ANA	Artificial Neural Network
LDA	Linear Discriminant Analysis
RNN	Recurrent Neural Network
CRAN	Comprehensive R Archive Network
LC	Liquid Chromatography
RT	Retention Time
MAE	Mean Absolute Error
API	Application Programming Interface

Chapter 1

Introduction

The understanding of interplay between plant physiology and its hidden biochemical process is crucial for the improvement of basic plant science and addressing global challenges such as food security, crop resilience and combating climate change [1]. In recent years, advanced High-throughput analytical techniques such as Near-Infrared Spectroscopy (NIRS) and Liquid Chromatography-Mass Spectrometry (LC-MS) has instigated a paradigm shift in plant biology [2,3]. These High-throughput techniques are mostly used in areas like genomics, imaging and spectroscopy and are known for their ability to collect and analyse the data faster than traditional techniques[3]. High-throughput techniques are widely used since they enable the efficient collection of vast amount of data at various scales, from molecular to field level over significant time periods[4]. The big data generated by these high throughput procedures present both opportunities and challenges at the same time. It requires efficient processing to extract the maximum useful results and this is where Machine Learning (ML) or Deep Learning (DL) becomes indispensable [4,5]. ML as part of Artificial Intelligence (AI) refers to the ability of computers to find patterns and learn from existing data, which can be employed in processing high-dimensional data [6,4]. The ML algorithms are powerful enough to analyse complex, high dimensional datasets, enabling accurate predictions of plant traits or other features based on the input data. Additionally, integrating these big data with ML could help the researchers to optimize data processing pipelines, enhance predictive accuracy and thereby enter into a new era of data-driven decision-making [4,5]. This project employs linear model, non-linear model and neural networks to predict various plant features and compare their predictive accuracy, error rate and training time.

A significant shift in the realm of the biomedical community has brought new guidelines to ensure readability, modularity, transparency and extensibility of computational toolboxes. A toolbox, which stores multiple functions, parameters and results in a central location should be maintainable and uncomplicated for the developers and members of the open-source community [7]. R is a powerful and widely used programming language in the analysis and processing of high throughput data. Additionally, R contains a multitude of statistical and high quality visualization packages such as ggplot2 which are capable of processing and integrating big data to different ML methods [8]. Bioconductor is an open source software for bioinformatics, which contains more than 3691 packages (according to the last update in 2024) for statistical computing. This offers an object oriented framework for the high dimensional data, cutting edge visualization capabilities and interoperability [9]. Existing tools in Near-Infrared Spectroscopy (NIRS) data processing lack functionalities that could simplify and standardize data workflows when integrated with the SummarizedExperiment framework from the Bioconductor package. To address these gaps, the R toolbox, "nearspectRa" was developed for processing NIRS data. This package has a modular structure which creates a SummarizedExperiment object from

NIRS data.

Metabolomics, the study of small molecular compounds in biological systems, is a rapidly advancing field of science with applications in biotechnology, medicine, synthetic biology and environmental science [6]. Metabolomics has emerged as a transformative tool in plant biology, enabling cost-efficient and high throughput molecular characterization. The integration of metabolomics with different omics approaches has proven invaluable for functional genes identification and developing trait specific markers [10]. Metabolomics, which is built on the advancement of phenomics and genomics, provides high throughput and precise profiling of metabolites, revealing the physiological state of cells [6,10]. Metabolites play a crucial role in plant metabolism, influencing its biomass and architecture therefore study of these small molecules will aid in uncovering plant regulatory mechanisms and pathway interactions [10]. The coupling of liquid or gas chromatography with mass spectrometry or nuclear magnetic resonance spectroscopy (NMR) facilitates measurement of thousands of metabolites, thereby providing a comprehensive view of biochemical and biological mechanisms [11]. Therefore, Mass spectrometry(MS) remains the most widely used analytical approach among others due to its versatility and sensitivity [6]. Mass spectrometry based metabolomics generate data of high sensitivity and throughput requiring advanced computational methods. Machine learning not only offers a powerful solution to analyse such data, but also helps in resolving the challenges like noise, batch effects and missing values [12]. Integrating ML with Liquid Chromatography-Mass Spectroscopy (LC-MS) data helps us to analyse this complex heterogeneous data rapidly, enabling deeper insights.

Near-Infrared Spectroscopy (NIRS) is an advanced high throughput and non-destructive analytical technique that uses light in the near-infrared region (780-2500 nm) to assess the chemical composition of samples [13]. The light is either absorbed or reflected by the sample at different wavelengths and thereby creating a spectrum[13]. The NIRS is widely used in plant research due to its ability in predicting sample structure and traits by analysing the spectral patterns. NIRS can also be used in the quantitative analysis of key plant features such as protein and carbohydrate content, secondary metabolites and physiological traits such as Specific Leaf Area (SLA) by developing calibration models between spectra and reflectance trait data [14,13]. NIRS is not only used in plant biology but also in various fields such as food science, agriculture and pharmaceuticals. When compared to other analytical techniques, NIRS is rapid, requires minimal sample preparation and less expensive, which makes it more attractive and interesting to the scientific communities [13]. However, on the flip side it requires complex statistical methods to extract different complex features due to the highly-correlated nature of NIRS data [13]. To tackle this problem, the conventional methods such as Partial Least Square Regression (PLSR) and Principal Component Analysis (PCA) imply dimension reduction which result in loss of information and often struggles to extract important features from the spectral data [13]. To address the challenge of data complexity and generalizability, different ML methods can be used to predict the traits from the NIRS data [13,15]. In this project different ML and Deep Learning (DL) has been employed to predict different plant leaf traits with use of NIRS data.

In recent years, studies using NIRS data coupled with PLSR have been used as an alternative for traditional methods such as high-performance liquid chromatography (HPLC) and mass spectrometry which are both labor-intensive and expensive to predict different plant traits. A notable example is the prediction of glucobrassicin (GBS) concentrations from NIRS data. This has shown that GBS concentrations could be reliably predicted from NIRS data [16]. Another prominent example is the tree and mycorrhizal fungal diversity experiment and trait variation

in temperate forests conducted by Pablo Castro Sanchez-Bermejo, where he combined Deep Learning (DL) approaches with leaf-level spectral data to predict 5 different leaf traits [15]. Another good example is a project involving the development of white-box workflow for regression tasks [17]. The project marks the potential of Regression (Sensitive) Neural Gas (RSNG) for generating interpretable results while maintaining high accuracy [17]. From the above studies, it is evident that NIRS data has a wide range of applications in plant research. This can also be expanded further to predict complex metabolites which are usually assessed via techniques like LC-MS. Moreover, integrating NIRS with advanced ML could further enhance the prediction accuracy and unlock new possibilities in plant science.

The past decade has witnessed the increasing popularity of Artificial Intelligence (AI) in different fields. However, this idea of AI has been under development since 1956, starting from the concept of "programming computers to think and reason" [18]. In other words, AI can be described as "automating intellectual tasks normally done by humans" [18]. Machine learning (ML) and Deep learning (DL) are the methods that fall under the realm of AI [18,19]. Nowadays, there are different ML algorithms in use, in which the most popular ones include Partial Least Square Regression (PLSR), Random Forest (RF) and Convolutional Neural Network (CNN) [20,18]. PLSR is a linear and one of the most simple ML approach. It uses the advantages of PCA and linear regression to solve the regression problem in the high dimensional data [18]. On the other hand, Random forest is a non-linear approach in ML that is primarily used for classification. It can also be used for regression tasks and can be represented as a decision tree with a series of nodes starting from a root node. The terminal node will predict the class of data in classification tasks [18]. For regression tasks, the random forest works differently compared to classification but the core principles remains the same. The terminal node of RF in regression takes the avarage of predictions from the individual trees [44]. The Convolutional neural networks are a specialized type of neural network which are mainly used in the field of image processing [21]. A recent study of mycorrhizal fungal diversity experiment and trait variation in temperate forests conducted by Pablo Castro Sanchez-Bermejo, demonstrated the application of CNN in predicting the leaf trait values from NIRS data, achieving superior results [15]. This outcome strongly suggests the potential of CNN not only for classification tasks such as image processing but also for regression tasks. In another project, CNN was used to predict gene expression status on the basis of sequence of gene transcription start regions. The CNN model had achieved roughly 80% accuracy [22]. These studies highlight the growing versatility of CNN models.

Omics is a term associated with the field of large scale biological data, including genomics, epigenomics, proteomics, transcriptomics and metabolomics [23]. Combination of data from these techniques along with advanced microscopy techniques helps in the study of biomolecules in cellular and subcellular levels [23]. However, the high throughput data from these omics instruments poses challenges in processing and analysing it without the loss of information [23,10]. The complexity and scale of this data make ML essential for effective integration and analysis, raising the critical question: which ML model is best suited to handle this data? How much programming expertise is necessary to implement these models? And, which models are most suitable for regression tasks?. Each ML model handles the data differently. For instance, PLSR uses latent variables to capture the covariance between predictor and response variables. Moreover, PLSR uses the combination of Principal Component Analysis (PCA) and linear regression [20]. In the case of RF, it follows the concept of "a forest made of many trees" which uses the combination of predictions from many trees [18]. Among these ML techniques, CNN is gaining attention on its ability in handling high throughput data and predicting with remarkable accuracy [15,22]. These ML models also require different levels of programming

proficiency and computational resources, depending on the scale of data.

In the light of the findings, it is clear that ML can significantly improve the analysis and processing of high throughput data from analytical techniques such as LC-MS and NIRS [15,22,17]. Among these, NIRS stands out as a non-destructive, cost-effective and rapid method, offering valuable insights into the chemical composition of the biological samples [12,13,14]. These qualities make NIRS a promising technique to optimize and integrate with ML and DL models for predictive accuracy.

Given the popularity of R programming within the ecological and bioinformatic community, it was chosen as the foundation for this project [7,9]. Recognizing the need for specialized tools to process the NIRS data, an R package, nearspectRa, was developed to handle data from two widely used NIRs instruments namely "ASD Fieldspec 4" and Spectra Vista Corporation (SVC) HR-1024i. Leveraging supporting packages like "R-FieldSpectra", the high dimensional data was structured into a "SummarizedExperiment" object, aligning with Bioconductor standards for interoperability and integration.

Apart from developing a Good Scientific Practice (GSP) compliant package, this project involved two key analyses: first, predicting plant leaf traits from NIRS data using three popular ML methods, PLSR, RF and CNN and second predicting LC-MS features from NIRS data using the same models. To evaluate these approaches, performance was compared using metrics such as the coefficient of determination (R^2), Root Mean Squared Error (RMSE) and training time of each model. Additionally, extrapolation studies were conducted on PLSR and RF to assess the robustness and performance of those beyond training data. This project not only exemplifies good scientific practice in developing an R toolbox but also provides a comprehensive comparison of linear, non-linear and neural network based NL approaches in predicting plant traits and LC-MS features from NIRS data. By achieving this, the project makes a significant milestone, paving the way for a new era of cost-effective, rapid biochemical analysis in metabolomics.

Chapter 2

Background

2.1 Near Infrared Spectroscopy (NIRS)

Near infrared spectroscopy (NIRS) is a non-invasive measurement technique that uses light in the near infrared region to analyse a sample [35,13]. Infrared (IR) is a form of electromagnetic radiation and it interacts with samples through absorption or reflection. The NIRS is widely used in plant research due to its ability in predicting sample structure and traits by analysing the spectral patterns. Based on the absorbance or reflectance of light at different wavelengths, Infrared radiation is classified into three categories:(1) Near Infrared (NIR), ranging from 780 to 2500 nanometers (nm), (2) Mid Infrared (MIR),ranging from 2500 to 25,000 nm, and (3) Far Infrared (FIR), ranging from 25,000 to 1,000,000nm [35]. When a substance such as plant leaf is subjected to NIR light, the molecular bonds in the infrared range will interact with the light thereby causing absorption or reflection of light by the sample. The light transmitted or reflected is then measured to produce the NIR spectrum. This spectrum provides a detailed picture of molecular composition of the substance and the peaks in the spectrum corresponding to different vibrational modes of different chemical bonds. This occurs due to the change in the vibrational or rotational state of molecules or transition between their energy levels [35,13]. The group which is the most dominant in absorbing the NIR are hydrogen-containing groups such as C-H, O-H and N-H while other groups like C=C and C=O absorbs light in weaker intensities. These groups are key components in organic substances and their absorption wavelengths and intensities differ depending on the chemical composition of the substance [35].

Figure 2.1 shows an example of the relationship between reflectance and wavelength for near-infrared (NIR) spectra where the variations in reflectance give valuable information about the chemical and physical properties of the sample. Compared to other analytical methods, NIRS is faster, requires little sample preparation, and is more cost-effective, making it a highly inviting choice for researchers and scientists [13]. NIR spectrometers typically consist of a light source, a beam splitter, optical detectors and optionally a processing system or a monitor. The components differ depending on the purpose of the instrument to make sure accuracy and consistency. These systems can operate in different modes such as transmission, reflection, diffuse reflectance or transmittance depending on the type of analysis [36]. The collected spectra will be later put through chemometric analysis to develop a calibration model using key NIR bands. Nevertheless, NIRS require reference data from traditional chemical analysis for accurate quantitative analysis [36]. The spectra are usually exposed to preprocessing to remove irrelevant information to improve the accuracy of the analysis. The most common preprocessing techniques include, baseline correction, scatter correction, noise removal and wavelength selection. The scatter correction method and spectral derivatives are the most used preprocessing approaches in NIRS. The scatter correction method includes, Multiplicative scatter correction

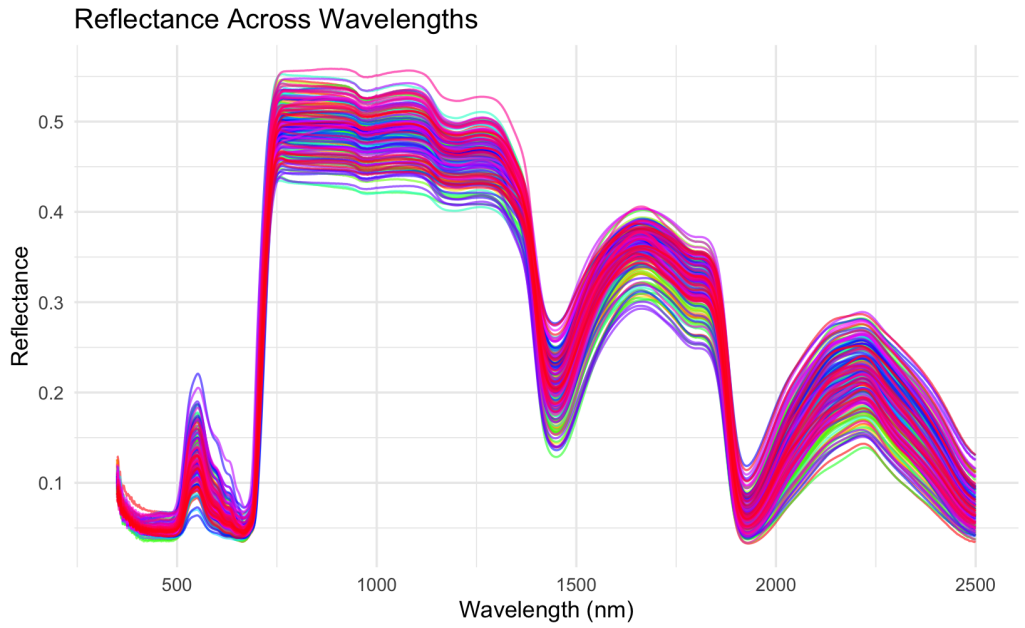

Figure 2.1: Example near-infrared spectra illustrating reflectance across wavelengths. Each line represents a unique sample, showing how NIRS captures molecular absorption/reflectance patterns. The data is from the project [15]

(MSC) and Standard normal variate (SNV). Additionally a wide range of normalization methods such as mean centering, auto scaling, vector normalization and area normalization are also used in preprocessing of NIRS data. The preprocessing methods are aimed to highlight important spectral features but must be carefully selected to avoid losing critical information [36].

NIRS consist of overlapping weak bands that require multivariate calibration for quantitative analysis. These methods analyze the data in a way that can identify patterns or make predictions [36]. Techniques like Principal component analysis (PCA), Partial least squares discriminant analysis (PLSDA), Hierarchical cluster analysis (HCA) and Linear discriminant analysis (LDA) are used for classification, while others like Artificial neural networks (ANNs) handle more complex relationships. Deep learning (DL) is a subset of ML that has a wide range of applications including image and audio recognition [36]. ML can also be used in the handling of "Big Data" emerging from the growing spectral libraries. DL models, unlike the traditional neural networks, consist of multiple layers to extract different features from the data resulting in better predictions [36]. In recent years techniques like Convolutional neural network (CNN), recurrent neural network (RNN) and DeepSpectra model have gained popularity. Combining DL with spectroscopy shows a great promise in applications like food quality assessment and genetically modified organisms (GMO) detection [36].

NIRS has diverse biological applications, particularly in agricultural and soil sciences. In recent years, it has been employed to analyse crops, food properties and plant traits, including water content, pH, oil, protein, and fatty acids. Another example would be the use of NIRS to predict anthocyanin levels in grapes and black rice seeds. Researchers have also used NIRS to identify pest attacks and pesticide residues in crops making it invaluable for quality assessment in agriculture [36]. Even though much of the NIRS applications focused on agriculture and food quality, its applications in food safety are also growing. For instance,NIRS can effectively evaluate the lamb meat tenderness, pH, fat and water content. The portable NIRS devices allow real time industrial monitoring making it suitable for large scale monitoring of meat,

fruits, vegetables and beverages for quality and safety [36]. Genetically modified organisms (GMO) pose a risk of gene transfer, which could threaten environmental biodiversity. NIRS has shown potential for rapid and cost effective GMO detection in labs and fields, making it a favored tool among researchers for monitoring GMOs [36].

NIRS Instruments The ASD FieldSpec4 and SpectraVista Corporation (SVC) HR-1024i are two widely used NIRS instruments in plant biology are highly relevant to this project.

ASD FieldSpec4

ASD FieldSpec4 is a portable NIRS instrument produced by Malvern Panalytical and comes in different variants. It has a wide range of applications including, plant physiology, remote sensing and geology, atmospheric remote sensing research and biomass analysis among others. It provides data across the wavelength of 350 to 2500 nm. There are multiple optional accessories, contact probe (contact measurements like solid raw materials, grains, other granular materials), reflectance panels, pistol grips and leaf clip version 2 are few out of many [40].

Figure 2.2: ASD FieldSpec 4 instrument for measuring NIRS data.
https://www.malvernpanalytical.com

SVC HR-1024i

This is a high resolution portable spectroradiometer which provides data across a full spectral region from 350 to 2500 nm. Compared to ASD FieldSpec4, SVC HR-1024i comes with a built-in touch screen to set parameters and review data in real time, which makes it more advanced. It is also light weight and has an internal GPS and a camera which adds critical information to the spectral signature. The output from this instrument is human readable and easy to interpret without the use of any package [41].

2.2 Liquid Chromatography Mass Spectrometry

Liquid Chromatography-Mass Spectrometry(LCMS) is the combination of two powerful analytical techniques for the separation, identification and quantification of biological compounds from complex mixtures. This technique has a wide range of applications in Metabolomics, Pharmaceuticals, Environmental science and Food chemistry [43]. Liquid Chromatography (LC) is a technique used to separate compounds in a mixture based on the rates at which they move through a stationary phase under the influence of mobile phase gradient. The varying affinities of the compound to the stationary and mobile phases leads to their separation. Some

Figure 2.3: SVC HR-1024i instrument for measuring NIRS data.https://spectravista.com

compounds attached to the mobile phase and eluting faster, while others attach to the stationary phase and thereby elute more slowly, leading to different retention times (RT) [43]. Mass spectrometry (MS) is one of the several detectors which can be coupled with LC to identify compounds. MS detects the mass to charge ratio (m/z) and the abundance of ions generated during ionization of the sample extracts [43]. The MS is made of an ionization source, a mass analyzer and a detector, all these should be kept under vacuum to optimize the ion transmission. A mass spectrum is then generated with the help of a computer [43].

	mass_to_charge	retention_time	pos_122_2018_E_PHLPRA_A002_a_1.A.7_01_13468	pos_014_2018_E_PHLPRA_A002_b_1.A.7_01_13082
209	163.0392	374.98360	0.00	0.000
210	163.0393	393.11750	0.00	0.000
211	163.0392	144.55073	312299.86	434848.335
212	163.0394	206.38270	1604733.65	2193657.016
213	163.0393	246.33116	335844.14	547290.203
214	163.0393	219.15808	305827.01	431516.720
215	163.0393	500.32537	55333.82	54814.704
216	163.0392	451.27309	100301.04	135660.000
217	163.0395	182.52874	261521.97	390857.688
218	163.0392	418.45835	0.00	0.000
219	163.0396	326.28534	92144.63	161584.752
220	163.0404	635.68351	0.00	0.000

Figure 2.4: LCMS dataset after initial processing using the `metabolighteR` package, revealing the selected columns relevant for downstream analysis [14].

The diversity of the metabolic profile analyzed in Liquid Chromatography-Mass Spectrometry (LC-MS) makes it one of the essential tools in plant metabolomics. LC-MS has a great number of uses within the field of plant biology for plant metabolite identification and analysis. Unlike other techniques such as gas chromatography coupled to mass spectrometry, the steps involving derivatization are avoided in LC-MS [43]. This makes LC-MS particularly fit for the detection of different metabolites, including semi-polar compounds and secondary metabolites common in plants [43]. The flexibility of LC-MS in gradient elution further allows effective separation of complex mixtures. Using different polarities of solvents during the elution process allows for the separation of polar and nonpolar compounds. Reverse-phase LC-MS, a widely used approach, often employs a gradient starting with an aqueous solvent and progressively increasing the proportion of an organic solvent, ensuring the efficient separation of a broad spectrum of metabolites [43].

2.3 R Programming

R is an open-source and free software developed by professors Robert Gentleman from the University of Waterloo and Ross Ihaka from University of Auckland as a programming language to teach statistics [37]. This programming language is widely used for statistical analysis, data visualization and graphical output [38]. Quoting the words of Ross Ihaka, "R is a computer language and run-time environment which can be used to carry out statistical (or other quantitative) computations" [37]. The name "R" itself comes from the shared first letters of both authors and being the successor of "S" language [37]. The strength of R is its extensive add-on packages which help to analyse and visualize complex data. Even though R primarily has a command line interface, multiple third party graphical user interfaces such as Rstudio, Jupyter, Shiny are available for user friendly experience [37,38]. R is a versatile software suited for data manipulation, computational and graphical representation [39]. The GNU General Public License of R offers users significant freedom, including the ability to use, modify and distribute the software making it a favourite programming language for researchers across all the fields [37,39].

R is often associated with statistics creating a versatile environment that incorporates many traditional and modern statistical techniques [39]. Some of these techniques are part of base R setup while others are available through additional packages. R offers a greater flexibility and control over analytical processes by step-by-step analysis with intermediate results saved as objects. This allows users to interact with and refine results using additional R functions [39]. R is also case-sensitive, meaning variables like "A" and "a" are distinct and names must start with a letter or a period not followed by a digit [39]. The commands in R can either be expressions or assignments. Expressions are evaluated, displayed and then discarded. Assignments, on the other hand, evaluate expressions and assign their results to variables without displaying it, allowing more flexible programming [39].

Initially, R gained popularity for its syntax, closely resembling the S programming language, which made it easy for the S-PLUS users to transition to R. The versatility of R made it popular since it could run on various platforms, such as Linux, Unix, tablets and smartphones due to its open-source nature [8]. The frequent updates, including annual major release and timely bug fixes, underline the active development in R. Its advanced graphical capabilities is another very good quality among many, supporting detailed and publication-quality visuals with help of packages such as "ggplot2" surpass many competitors [8]. These visualization packages allows high-dimensional visualizations and are more user friendly [8]. R remains true to its original philosophy, offering an interactive and programmable environment for creating tools. Its strong community contributes a wide range of packages, share knowledge and support on the forums like Stack Overflow, allowing innovation and collaboration among users worldwide [8]. The main R system can be downloaded from the Comprehensive R Archive Network (CRAN), which also hosts many additional packages that extend R's capabilities [8]. The R system is divided into two parts

- The core R system, which can be downloaded from CRAN for different platforms like Linux, Windows and MacOS.
- Additional functionality through different packages.

R is a powerful tool for statistical computing, this along with high quality visualization capabilities provides a robust environment for machine learning (ML) and deep learning (DL) [42,8]. The availability of a wide range of supporting packages and libraries makes R a powerful

tool for both traditional ML methods and modern DL techniques [42]. There are various packages available to perform ML and DL in R, some of them include caret, randomForest, keras and tensorflow. These packages contain the algorithm for different ML and DL models. This project involves the implementation of various ML and DL models using the R programming language.

2.4 Machine Learning

Machine learning (ML) as the name indicates is the field of computer science which applies mathematical models and algorithms to enable the system to learn and make predictions without being programmed explicitly [18,19]. In contrast to classical programming where someone explicitly program an algorithm to execute predefined tasks, ML uses a subset of (training) data to learn patterns and relationships within the data to create a function which can generalize to unseen data[18]. This versatility and flexibility enables ML methods to improve performance over time, leading to advanced data driven decision making [23,18]. As a subgroup of Artificial intelligence (AI) is often simply represented as a 3 layer model which includes an inputlayer that receives the data, a hidden second layer which processes the data according to the mathematical backend of the model and finally the third output layer that outputs the prediction [21]. The hidden layer which does the linear regression or classification differs according to the ML model in use and these are often compared to a single human neuron, where dendrite represents input layer, cell body corresponds to hidden layer, and axon functions as output layer [21]. ML employs four primary learning methods namely, supervised, unsupervised, semi-supervised and reinforcement learning [18].

Supervised Learning Supervised learning is a ML approach in which model is learned from labeled data. It excels in tasks where the patterns in data can augment human decision making [18]. For instance, handwriting recognition or object classification (example, distinguishing an elephant and tiger). These tasks are easily done by humans and supervised learning strives to replicate or enhance this performance [18]. Another example would be in medicine where supervised ML identifies patterns in the electrocardiogram (ECG) which is an easy job for a trained cardiologist. These classification tasks from supervised learning are achieved through training an algorithm on labeled datasets containing ECG features (heart rate, rhythm and waveform shape) and their corresponding diagnosis. By mapping these features (X) to diagnostic outcomes (Y), the algorithm learns the function $f(X)$ to accurately predict for new unseen ECG data [18,24]. Another common application of supervised learning is in regression and classification tasks [18]. Regression focuses on predicting continues numerical values such as LC-MS values from NIRS data and test scores. In contrast, classification predicts which category does the given instance belong such as elephant or tiger as seen in the previous example [23,24].

Unsupervised Learning Unsupervised learning focuses on discovering patterns or groupings within data without predefined targets[18]. Common unsupervised learning tasks include clustering, association and anomaly detection, where the algorithm independently identifies underlying structures in the data [23,18]. For instance, clustering data points into separate groups based on the shared features (2.5).

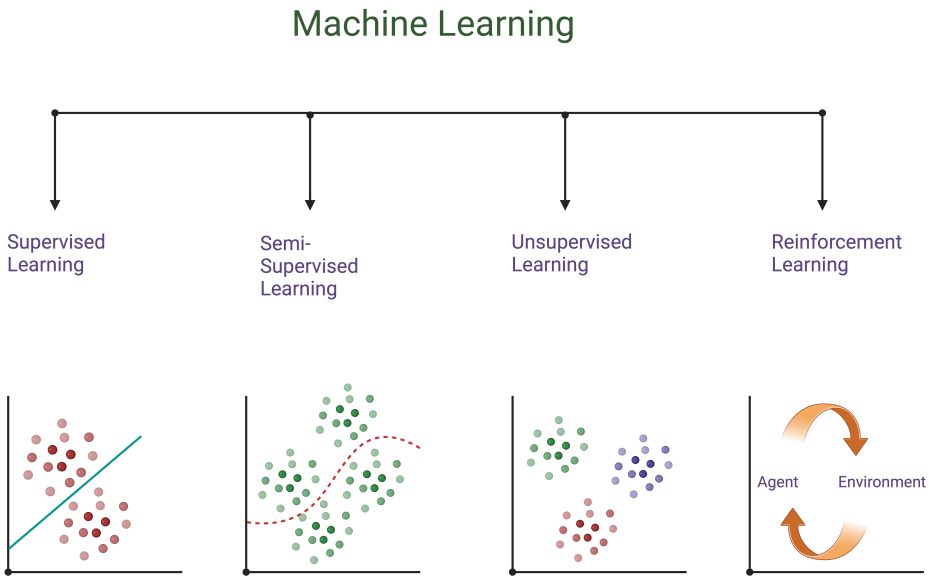

Figure 2.5: Different machine learning models illustrating the four primary learning methods: supervised, unsupervised, semi-supervised, and reinforcement learning.

Semi-supervised and Reinforcement Learning Semi-supervised learning bridges the gap between supervised and unsupervised learning by utilizing datasets that have both labelled and unlabelled data. On the other hand, reinforcement learning mimics the human learning process, where it relies on trial and error then solely on data [24].

In the era of modern molecular plant breeding, integration of ML with the large, noisy and heterogeneous data is important to uncover complex patterns and enable accurate predictions of plant features [23]. The "big data" resulting from high throughput techniques in plant sciences can be leveraged to drive discoveries, enhanced precision and accelerate advancement in plant research [23]. A plant genetic makeup (genotype) has a significant influence in its growth, development and biochemical composition. This results in the expression of plant traits such as yield, stress tolerance and pest resistance. Understanding how genotype and environment influences on phenotypes is crucial for insights into regulatory mechanisms, and development of plants [23]. This knowledge enables the prediction of yield and other plant traits based on the genotypes under different environmental conditions, which in turn paves the way for modern molecular plant breeding [23]. Different ML approaches such as Partial Least Square Regression (PLSR), Random Forest (RF) and Convolutional Neural Networks (CNN) can be employed to make predictions by leveraging patterns in the data which are explained in the following sections.

2.4.1 Partial Least Square Regression (PLSR)

In machine learning, Partial Least Square Regression (PLSR) is a statistical method which combines the benefits of Principal Component Analysis (PCA) and linear regression to predict the outcomes [26]. It is a linear regression model which is arguably one of the simplest machine learning algorithm that uses a line to solve a regression problem [18]. PLSR uses the advan-

tage of PCA for dimensionality reduction and the regression for prediction [27]. This fitting of linear regression between two data matrices has a wide range of application in plant biology, especially in crop breeding, ecosystem monitoring and predicting plant traits from its spectral data [25]. In PLSR, the predictor variable (often denoted as X) refers to a set of independent variables or features that are used to predict response variable (y). The predictor variables are typically high dimensional and often include multiple correlated features. The response variable (y) represents the outcome or dependent variable. PLSR works by identifying the latent variables, which summarizes the covariance between predictor and response variables. This latent variable captures the most relevant information from the predictors (X) in relation to response (y) variables, allowing the model to predict y more effectively [25,27,28].

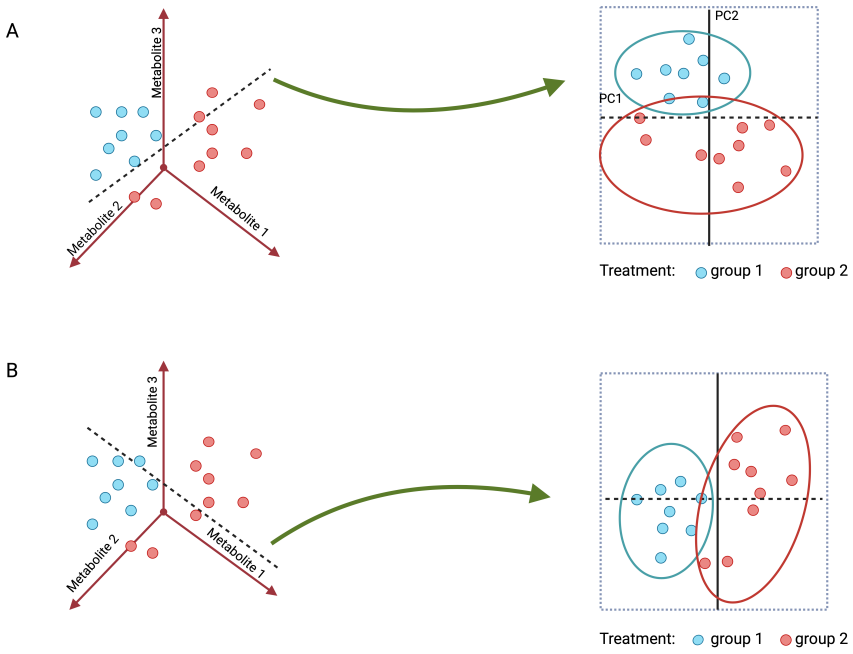

Figure 2.6: A comparison of PCA (A) and PLS (B). In the PCA plot, the PC1(x-axis) represents a combination of variables (e.g., three metabolites) that captures the greatest variation in the dataset, independent of group classification. In contrast, PLS focuses on explaining the relationship with an explanatory variable, such as "Treatment" in this example

Principal Component Analysis (PCA) uses the principal components as explanatory variables to capture most of the variance in predictor variables (X), ensuring dimensionality reduction while retaining most of the data's variability. Partial Least Squares (PLS) prioritize relevance to both X (predictor) and y (response) variables, rather than maximising the variance in X alone (2.6) [27]. By combining the PCA with linear regression, PLSR constructs the latent variable that summarizes predictor variables and maximizes their relevance to the response variables. This makes PLSR particularly suitable for high dimensional datasets such as spectral data from NIRS instruments.

PLSR is employed when the predictor variables are highly collinear and it works by creating latent components that maximizes covariance between predictors and response.

The predictor matrix X and response matrix Y can be represented as:

$$X \in \mathbb{R}^{n \times p} \tag{2.1}$$

$$Y \in \mathbb{R}^{n \times q} \tag{2.2}$$

where:

- n is the number of samples
- p is the number of predictors (spectral data)
- q is the number of response variables
- $\mathbb{R}$ refers to the real numbers

PLSR decomposes predictor(X) and response(Y) matrix into l number of latent components

$$X = TP^T + E \tag{2.3}$$

$$Y = UQ^T + F \tag{2.4}$$

Here;

- T is an $n * l$ matrix which contains the projections of X onto the latent components
- U is an $n * l$ matrix which contains the projections of Y onto the latent components
- P^T shows how much of the original features in X contribute to each latent component
- Q^T shows how much of the original response variables in Y contribute to each latent component
- E and F are the left over after the decomposition

2.4.2 Random Forest (RF)

Random Forest (RF) is a non-linear ensemble technique used in machine learning which is also depicted as a forest of decision trees [18,29]. Random forest is not only used for classification tasks but also for regression such as predicting the leaf trait values from the NIRS data [18]. It is a supervised learning method consisting of decision trees and the root node serves as the initial point for dividing the dataset. Recursive partition in which the data is separated into two binary part begins with the root node [18]. RF combines multiple trees to improve accuracy, robustness and reduce overfitting [18,29]. In the classic tree based model, the dataset is divided into two groups based on the certain criteria until it encounters a predetermined stopping condition. The endpoints of a decision tree, known as leaf nodes, represents the final data division. A random forest model consisting of an ensemble of decision trees, can be utilized for both regression and classification tasks, depending on the partitioning strategy and stopping criteria [30]. RF has proven its importance in many scientific domains and one of which was in environmental science, in a study employed learning algorithms such as Least Absolute Shrinkage and Selection Operator (LASSO) Regression, RF and neural network to predict ragweed pollen concentrations, with RF delivering the most accurate predictions [30,31]. A graphical representation of the RF model with decision trees can be found in the figure (2.7) .

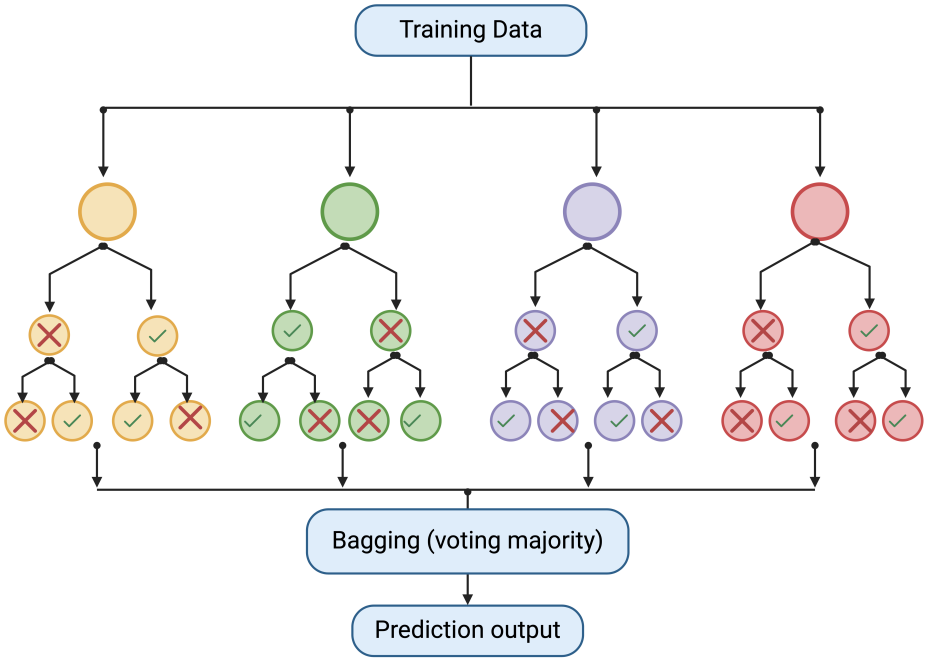

Figure 2.7: Schematic representation of a Random Forest model (classification).

Random Forest often outperforms linear regression models since, linear regression assumes a linear relationship between the variables. Even though this makes linear regression easier to interpret, it limits their flexibility in capturing complex patterns in the dataset. On the flip side, RF can easily adapt to non linear relationships, making them more flexible and suited for such tasks [30]. The RF algorithm estimates the error rate by out-of-bag (OOB) during training time. Each tree in the model is built using a subset of data, called a bootstrap sample and about one third of the data is left out during this process. These excluded data points are the OOB. Hence, minimizing the OOB is crucial for better model performance and robustness [30].

For regression tasks, the random forest works differently compared to classification but the core principles remain the same. The terminal node of RF in regression takes the average of predictions from the individual trees [44]

- **Bootstrapping and Tree Construction**
  Each tree in a RF model is trained using a bootstrap sample, which is a subset of the sample from the training data of the original dataset. Approximately one third of the samples (data points) are left out during bootstrap sampling for each tree and these left out samples are referred to as Out-Of-Bag (OOB) samples. Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where:
  - x_i represents the feature (e.g., spectral data),
  - y_i represents the response variable (e.g., SLA, LDMC, or other traits),

  For each tree (t), a bootstrap sample D_t is taken out of dataset D, and the OOB for that tree is calculated as:

$$D_t^{\text{OOB}} = D - D_t \tag{2.5}$$

- **Model training and Prediction**
  Each tree in a RF model is trained with a separate bootstrap sample (D_t) and later exposed to test data for predictions to be made. For the test data X_{test}, the prediction from each tree (t) is denoted as $\hat{y}_t(x_{\text{test}})$. The final prediction from the RF is obtained by taking the average value of predictions from all the individual trees:

$$\hat{y}_{\text{RF}}(x_{\text{test}}) = \frac{1}{T}\sum_{t=1}^{T}\hat{y}_t(x_{\text{test}}) \tag{2.6}$$

  where:

  - $\hat{y}_t(x_{\text{test}})$ is the prediction from the t-th tree,
  - T is the total number of trees in the Random Forest.

In this project, we will be using RF models to do regression tasks on NIRS data and compare the coefficient of determination R2, RMSE and training time with that of linear model and neural network, we will also discuss developing the RF model and associated functionalities.

2.4.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a type of deep learning (DL) model inspired by the way neurons in the brain process visual information [32,33]. It is primarily composed of three core components: A convolutional layer that extracts features, a pooling layer to reduce the dimensionality of data and a fully connected layer that produces the final output [32]. Out of these three main layers, the convolutional layer is considered as a fundamental component of a CNN, consisting of a series of mathematical operations, including convolution, which is a distinct form of linear regression [32]. It involves the application of kernels, which is a small array of numbers across the input (tensor) to compute elementwise products. These results are summed to create an output called feature map. Each kernel extracts a different feature of the input data and thereby different feature maps with different characteristics of the input data.The number of kernels determines the depth of the data and is also selected based on the scale of input data. A stride is known as the step size for moving the kernel across the tensor, commonly a stride of 1 is used. To capture the outermost element in the tensor, zero padding technique, which involves adding rows and columns of zero on the sides of input tensor is used to prevent downsizing in the convolutional layer [32]. After the convolutional layer the feature map is then processed through a nonlinear activation function and then to the pooling layer for downsampling [32]. In CNN the widely used pooling method is max pooling, which divides the feature map into small patches and then keeps only the highest values from each patch and ignores the rest. The output from the last pooling layer is flattened to 1 dimensional array and passed to fully connected layers, where each input is connected to every output with a learnable weight. These layers map the extracted features to the final output [32].

A loss function evaluates how well the predicted values match the actual ones. For classification cross-entropy loss is commonly used whereas mean squared error (MSE) is preferred for regression tasks. Choosing the right loss function is essential and is determined according to the given task [32]. In deep neural network training, the weights of each neuron is estimated to establish an accurate relationship between inputs and outputs with a desired level of precision [34]. It is categorized into supervised learning, for classification and regression tasks,

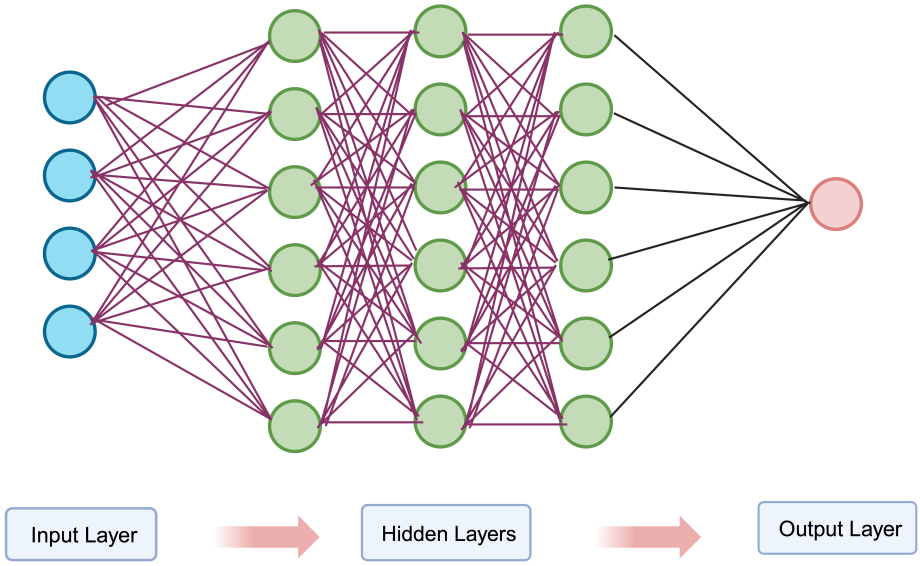

Figure 2.8: Schematic representation of Convolutional Neural Network

and unsupervised learning, for clustering with input data only. Several methods are employed for training deep neural networks, including Backpropagation, Gradient Descent, Stochastic Gradient Descent (SGD), and others. Among these, SGD stands out as one of the simplest and most widely used optimization algorithms in machine learning [34]. Adaptive Moment Estimation (ADAM) is an advanced optimization algorithm which combines the benefits of two SGD variants: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) [34]. It requires minimal memory and dynamically adjusts the learning rate for each parameter, making it more computationally efficient [34]. In this project CNN is employed to predict the output values from NIRS data.

Root mean square error (RMSE)

Root mean square error (RMSE) is a used as another metric for evaluating the performance of regression models in machine learning. The RMSE measures the average magnitude of prediction error [53]. Two important error metrics used in ML are:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)

These metrics measure how well the model performs by calculating the error rate,

Mean Squared Error (MSE)

The Mean Squared Error (MSE) quantifies the average squared difference between observed and predicted SLA values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_i)^2 \tag{2.7}$$

where:

- n is the number of samples,
- y_i is the observed value ,
- f_i is the predicted value ,
- $(y_i - f_i)$ is the residual (error) for each sample.

Best case: $MSE = 0$ (perfect predictions).

Worst case: $MSE \to +\infty$ (large errors in predictions).

Since MSE uses squared differences, it magnifies the impact of large errors and makes it to detect the outliers.

Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) is the square root of MSE:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f_i)^2} \tag{2.8}$$

Similar to MSE, a lower RMSE indicates better model performance.

Best case: $RMSE = 0$.
Worst case: $RMSE \to +\infty$

The main difference between MSE and RMSE is that MSE squares the error, giving more weight to larger errors and making it more sensitive to outliers. The units used to represent MSE values are squared. RMSE is the square root of MSE, converting the units back to the original scale. This makes it easier to interpret since it has the same unit as the target variable. RMSE is also used as a metric to evaluate the model's performance in this project.

Coefficient of Determination (R^2)

It is a statistical measure to indicate a regression model's good fit. The R^2 provides information on how well the observed outcomes are replicated by the statistical model. R^2 value ranges from 0 to 1, in which 1 indicates a perfect fit. In machine learning, the R^2 value of 1 indicates that the predicted value by the ML model and the actual or measured value are the same [52,53]. It is calculated by:

The calculation of R^2 in the context of this study can be explained as follows:

Let $y_1, y_2, \ldots, y_n$ be the observed SLA values for n plant samples. These can be represented as a vector:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

The predicted SLA values can be represented as:

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

where f_i represents the estimated SLA for the i-th plant sample based on its spectral features.

The difference between the observed and predicted SLA values is referred to as the **residual**:

$$e_i = y_i - f_i \tag{2.9}$$

These residuals measure the prediction error for each sample. The overall model performance is calculated using the residual sum of squares (SS_{res}) and the total sum of squares (SS_{tot}):

$$SS_{\text{res}} = \sum_{i=1}^{n} (y_i - f_i)^2 \tag{2.10}$$

$$SS_{\text{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{2.11}$$

where $\bar{y}$ is the mean observed SLA:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{2.12}$$

The coefficient of determination (R^2) is then calculated as:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{2.13}$$

In this project, the R^2 is calculated for each machine learning and deep learning model and later used for comparison to understand model performance.

Chapter 3

Implementation

This chapter provides a detailed account of the materials and methods used in developing R package and workflow for two analysis using three machine learning models. The first analysis predicting five different leaf traits from its NIRS data uses three machine learning models and evaluate its predictive accuracy and in the second analysis called Jena experiment, evaluate the accuracy for predicting LCMS features from its NIRS data. A comprehensive explanation of the three machine learning models used, including their workflows is presented. For enhanced visualization and further details, links to the vignettes are also provided.

3.1 Overview

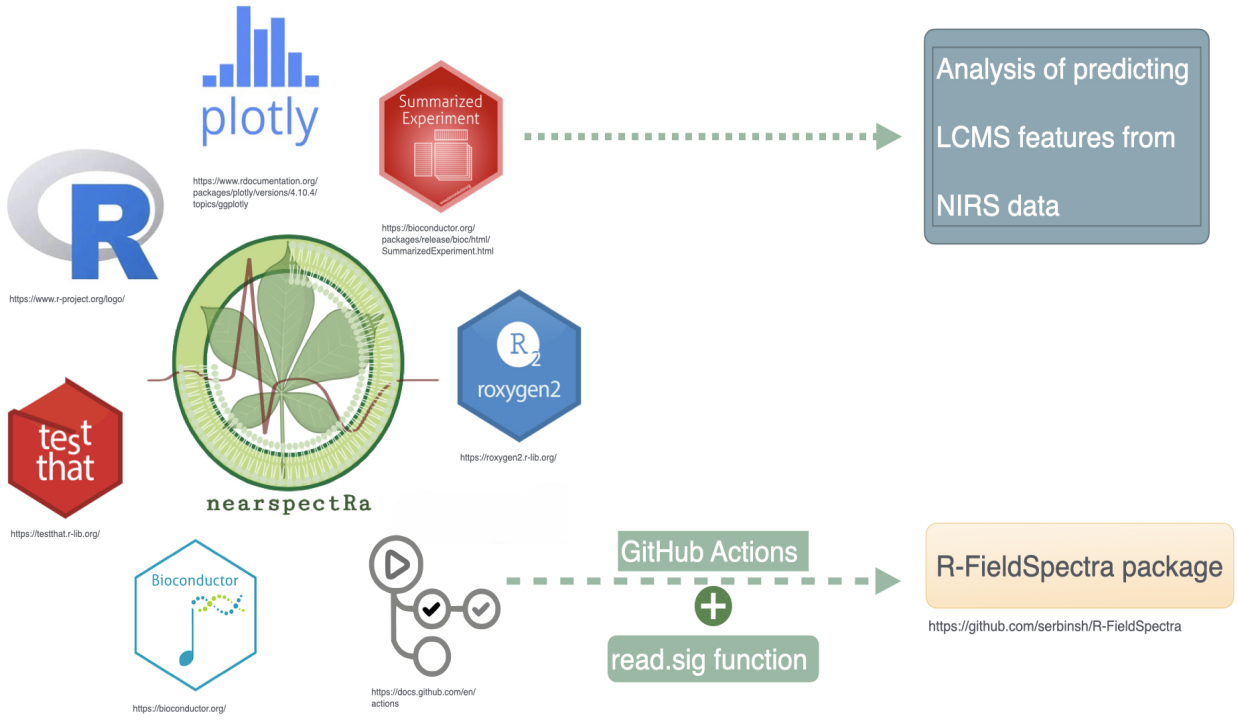

Figure 3.1: Architecture of the R package, illustrating the packages and functions used. The SummarizedExperiment function (with modifications) is later used for LCMS feature prediction. This figure also highlights contributions to the R-FieldSpectra package.

The overview of this project includes;

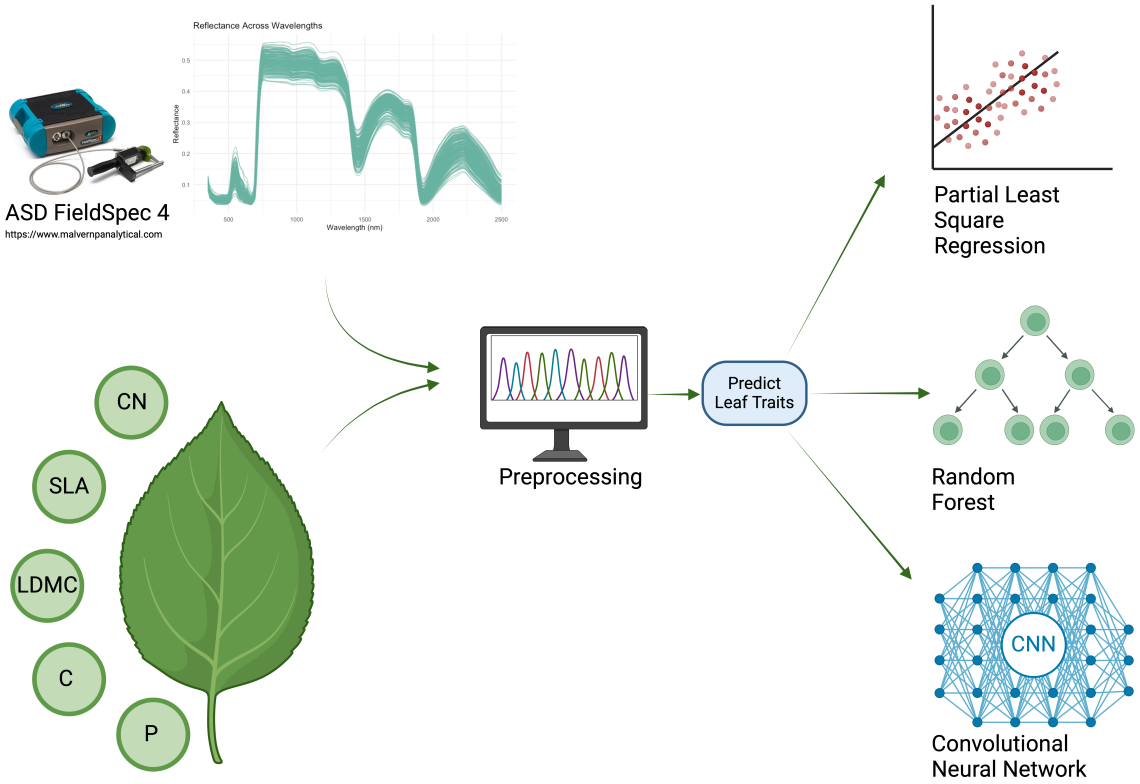

Figure 3.2: A simplified workflow displaying the process of trait prediction from NIRS data. Each of this trait data (response data) is predicted using the NIRS data (predictor data) measured using ASD FieldSpec 4 instrument. This is then preprocessed and later analyzed using three machine learning models.

- Development of R toolbox
- Contributions to R-FieldSpectra package
- Prediction of leaf traits using PLSR, RF and CNN
- Prediction of LCMS features using PLSR, RF and CNN

Figure 3.1 illustrate the packages that are used to create "nearspectRa" package for data analysis. The function available in this package is later used for analysis of predicting LCMS features from NIRS data. Apart from that, contributions were made to "R-FieldSpectra" package to enhance its usability in the scientific community.

The first analysis in this project is focused on predicting leaf traits from NIRS data. Figure 3.2 presents the outline of the first analysis. A linear , non-linear and convolutional neural networks were used to predict the leaf traits. It is then evaluated using three different metrics such as R^2, RMSE and training time for each model. We also investigated variable importance, the extrapolation capabilities of each machine learning model, and the impact of dimensionality reduction on model performance.

Second analysis includes the prediction of LCMS features from NIRS data using the same three ML models from the first analysis. Figure 3.25 illustrate the summarized workflow of the workflow. A high-performance computing (HPC) queuing system facilitated the analysis of this large dataset. The predictive ability and accuracy is then compared for all the ML models.

Complete analyses, including code and figures, are documented in vignettes, the links to which are provided in the Appendix.

Libraries and Dependencies

Base R Functions: These functions come pre-installed and do not require additional libraries or packages. For instance, readLines, list.files, files.info

`FieldSpectra`: is Used for reading the ".asd" files and to extract the metadata. The version used is FieldSpectra 0.9.7

`SummarizedExperiment`: is central to this package and it enable structured storage of spectral data and integrate to the bioconductor world. The version used is 1.34.0

`roxygen2`: is used for generating documentation directly from inline comments. The version used is 7.3.2

`readxl`: For importing the spectral and trait data from excel sheet. The version used is 1.4.3

`plantspec`: For managing the spectral data. The version used is 1.0

`ggplot2`: For graphical representation of the data. The version used is 3.5.1

`plotly`: For interactive graphical representation of the data in the vignettes. The version used is 4.10.4

`keras3`: Keras served as the foundation for the deep learning implementation. Its flexibility and modularity allow to efficiently design and train the CNN. The version used is 1.1.0

`metabolighteR`: It is used to read the LCMS data. The version used is 0.1.4

`tfruns`: For training convolutional neural networks. The version used is 1.5.3

`caret`: For model training and validation. The use of this package facilitated a consistent approach to machine learning model development. A version of 7.0.1 is used for this project. The "caret" package which stands for "classification and regression training" contains numerous functions from data preprocessing and splitting to training of a ML model. This package simplifies model training by providing a standardized architecture, making it easier for users. The caret package leverages functions from more than 25 different packages and is also available at the Comprehensive R Archive Network (`http://CRAN.R-project.org/package=caret`) [54].

3.2 R Toolbox Development

The idea of developing of a new toolbox arise from the realization that existing R packages were lacking functions to compartmentalize spectral data in a way that could be easily integrated into the Bioconductor ecosystem. Bioconductor offers remarkable advantages in terms of integration and interoperability. These are highly valuable for advanced data analysis. Existing packages commonly used in ecological studies, such as **FieldSpectra**, **plantspec**, and **spectacles**, were reviewed. However, none provided functionality to convert high-dimensional spectral data into

a SummarizedExperiment object, a data structure widely used in Bioconductor for organizing and analyzing complex datasets.

After recognizing the potential applications of the SummarizedExperiment framework in plant biology, along with its ability to simplify workflows and facilitate future analyses, a decision was made to develop a toolbox to address this gap. The primary objective during development was to make sure vendor independence, making the toolbox broadly applicable. Vendors such as Malvern Panalytical offer software for processing data from instruments such as the ASD FieldSpec 4, while the SVC HR-1024i from Spectra Vista Corporation requires additional computational effort to structure the data for analysis. In addition, the ASD FieldSpec 4 generates binary output files. To process these files, functions from the `FieldSpectra` package were utilized as a foundation, with additional custom functions developed to convert the data into SummarizedExperiment objects. The entire workflow, including data reading, processing, and conversion, is detailed in this chapter.

3.2.1 Introduction to SummarizedExperiment package

The two functions in "nearspectRa" package utilize the SummarizedExperiment from Bioconductor ecosystem to enable organized storage of experiment data. The SummarizedExperiment package provides well structured frame work for organizing and managing experimental data not just in genomics and transcriptomics but also in spectral analysis (figure 3.3). It is designed to handle high dimensional datasets and finds great use in the NIRS analysis. It is part of Bioconductor ecosystem, which ensures its interoperability with other Bioconductor packages and simplifies its use. A summarized experiment object includes the following features:

- Assay data: It is the main experimental measurements and is stored in matrix. The rows of it represent features(wavelength in this study) and the columns represents samples.
- Row Data (rowData): Metadata for each feature
- Column Data (colData): Metadata for each sample
- Metadata: Additional information related to experiment

3.2.2 Toolbox: nearspectRa

The toolbox, named **nearspectRa**, is specifically designed to handle Near-Infrared Spectroscopy (NIRS) data. This name was chosen to align with its functionality and field of usage, focusing on plant metabolic phenotype analysis. The toolbox is publicly available on GitHub and can be accessed at the following link: https://github.com/georgejr45/nearspectRa

3.2.3 Functions of nearspectRa

Function 1: `read_summarizedexperiment_asd`

Key Features:

- Input: A path to either a single .asd file or a directory containing multiple .asd files.
- Output: A SummarizedExperiment object with:

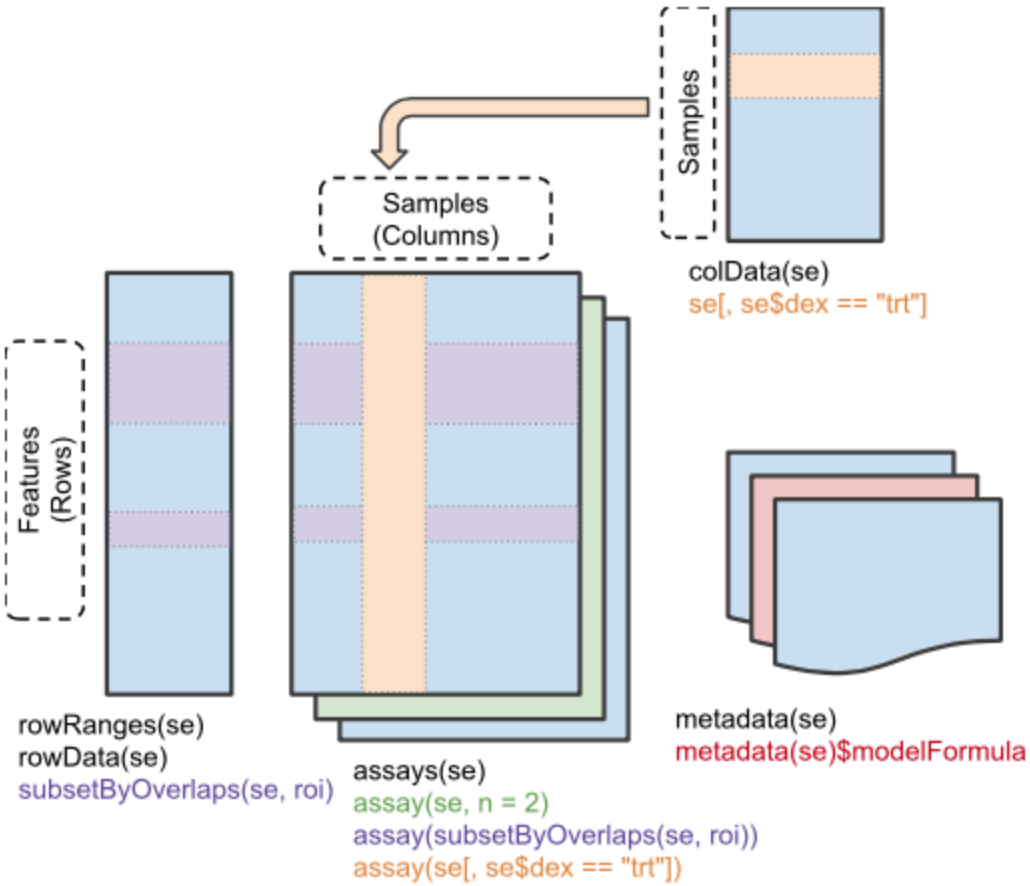

Figure 3.3: Architecture of SummarizedExperiemnt[55]

  - Assays: Reflectance data as a matrix where rows are wavelength and columns are samples.
  - colData: Sample metadata.
  - rowData: Wavelength metadata.

- Dependencies: Utilizes `read.asd` and `extract.metadata` from the FieldSpectra package for reading and extracting data, and the `SummarizedExperiment` library from Bioconductor.

The `read_summarizedexperiment_asd` function imports spectral data and metadata from ASD files, either as a single file or multiple files from a directory and assembles them into SummarizedExperiment object. The function first checks for a single file or directory and then proceeds to extract the spectral data using `read.asd` function from `R-FieldSpectra` package. The corresponding metadata is also extracted using `extract.metadata` function. These data is then combined into matrix and the SummarizedExperiment object is created. A vignette containing detailed version of all the process involved can be accessed using the following link https://github.com/georgejr45/nearspectRa. An example output of this function can be seen in figure 3.5.

Example Usage:

```
path <- "path/to/your/asd/files"
se <- read_summarizedexperiment_asd(path)
print(se)
```

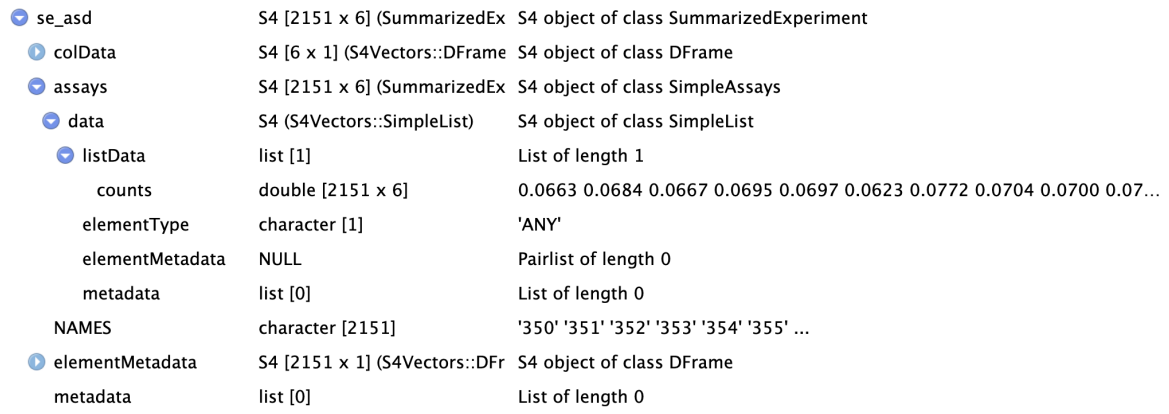

Figure 3.4: Example output of the `read_summarizedexperiment_asd function` showing the assay matrix which contain the spectral data, information on row data and column data of the experimental data.

Function 2: `read_summarizedexperiment_sig`

Key Features:

- Input: A path to a single `.sig` file or a directory containing multiple `.sig` files.
- Output: A `SummarizedExperiment` object with:
  - Assays: Reflectance data as a matrix where rows are wavelengths and columns are samples.
  - colData: Sample metadata.
  - rowData: Wavelength metadata.
- Dependencies: Utilizes `readLines` for file reading, `grep` for pattern matching, and the `SummarizedExperiment` library from Bioconductor.

The `read_summarizedexperiment_sig` function imports ".sig" files from the instruments such as SVC HR-1024i and structures this into a SummarizedExperiment object similar to the previous function. For each files, this function reads the file content using `readLines` then extracts the sample name, prioritizing the name found in a line starting with "name=" and falling back on the filename if this is absent. The spectral data and reflectance is also extracted in the similar way and all extracted data is combined by wavelengths, creating a SummarizedExperiment object. A detailed version of the codes and explanations can be found in the vignette in the package (https://github.com/georgejr45/nearspectRa).

Example Usage:

```
path <- "/path/to/sig/files"
se <- read_summarizedexperiment_sig(path)
print(se)
```

3.2.4 Package Development Overview

Purpose and Objectives

This package is aim to ease the NIRS data processing by reading the raw data and outputs a `SummarizEdexperiment` object. The programming language and tools used for this develop-

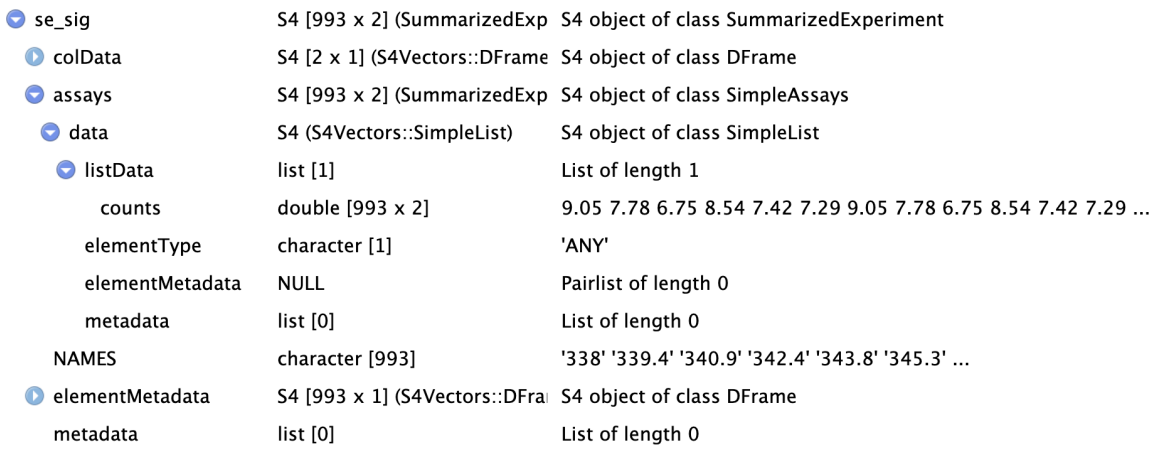

Figure 3.5: Example output of the `read_summarizedexperiment_sig function`

ment process is mentioned below.

Language

R: is the core programming language used for the development. It is choosen because of its powerful data analytical capabilities and wide usage in bioinformatics. The R version 4.4.0 (released on April 24, 2024) has been used for this project.

Version Control

One of the main challenges during software development is tracking and documenting the code during the development process. These challenges include, managing multiple code versions or integrating collaborative edits. Manuel merging and making multiple copies of each version is time consuming and error-prone [46].To overcome this challenge, Git is used for version control and collaborative development. A git version of 2.32.0 is used.

Examples of few git commands used in this project are given below:

```
# Navigate to the project directory
cd/path/to/the/project

# Initialize a Git repository
git init

# Add a remote repository
git remote add origin https://github.com/username/project.git

# Check the status of the repository
git status

# Stage all files for commit
git add .

# Commit the changes with a message
git commit -m "First commit"
```

```
# Push the changes to the remote repository
git branch -M main  # Rename branch to main
git push -u origin main
```

Testing and Validation

Unit tests are implemented under "test" directory to ensure a robust function performance across different datasets. `testthat` package is used for the unit test development. It is crucial for good software development practice since it make sure the function behaves as expected and detects the potential errors early which helps in debugging errors and most importantly ensures stability and reliability of the software overtime. For instance, the unit test for read_summarizedexperiment_sig function in `nearspectRa` package verifies if the function is converting spectral data from ".sig" files to a SummarizedExperiment object. It generates a dummy dataset, checks if the function return correct object type and make sure the presence of all the SummarizedExperiment object features. Figure 3.6 illustrates the output of the unit test for the read_summarizedexperiment_sig function. The code can be accessed through https://github.com/georgejr45/nearspectRa/tree/main/tests/testthat.

```
• extract.metadata
• read.sig
[ FAIL 0 | WARN 0 | SKIP 0 | PASS 10 ]
Test complete
```

Figure 3.6: Example output of the unit test for `read_summarizedexperiment_sig function. The "pass 10" indicates that the function has successfully passed 10 individual tests.`

Documentation

Documentation is a fundamental aspect of software development, regardless of the programming language or platform. It serves as a vital tool for communication between developers and end users, providing the necessary information for effectively utilizing the software to its full potential [45]. In developing `nearspectRa`, we prioritized clear and comprehensive documentation to ensure the package is user-friendly, while adhering to FAIR principles—Findable, Accessible, Interoperable, and Reusable. This documentation was created with the help of the `roxygen2` package version 7.3.2.

3.2.5 Challenges and Future Work

This package can be further developed to accommodate more functions and fine tune to make it suit for other analytical data as well. Future work can also include the integration of Artificial Intelligence to address the growing challenges in data analysis.

3.3 Contributions to other packages

3.3.1 GitHub Actions to R-FieldSpectra

GitHub Actions is a platform offering continuous integration and continuous delivery (CI/CD) which helps to automate the software workflows and directly integrated into `GitHub`. It not

only allows developers to automate workflows including building, testing and deploying but also other processes within their repositories [47]. During the development of `nearspectRa` package for NIRS data analysis, we identified an opportunity to improve and contribute to `R-FieldSpectra` package, which is widely used in plant biology. Even though this package packed with multiple higher level functions, the lack of GitHub Actions caught our eyes. To address this, we submitted a pull request introducing GitHub Actions to the `R-FieldSpectra` package.

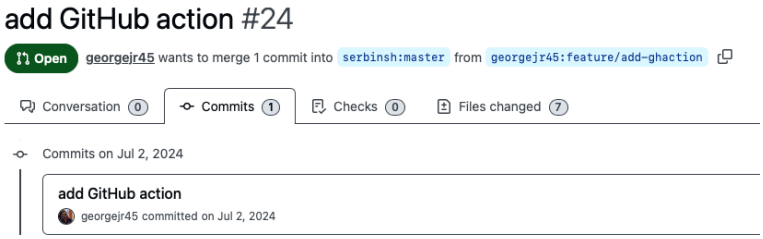

Figure 3.7: Contribution of GitHub Actions to the R-FieldSpectra package. https://github.com/serbinsh/R-FieldSpectra/pull/24/commits

The GitHub actions work flow is derived from https://github.com/r-lib/actions/tree/v2/examples and is configured to run the automated checks whenever changes are pushed to the main or master brach of `R-FieldSpectra` package. The current script runs tests on Ubuntu using the latest R version and additional configurations (currently disabled) can test macOS, Windows, R devel or older R versions. The significance of the contribution includes:

- Automated Testing: Verifies the package is functional and error-free after each update
- Cross-Platform Development: Allows easy way to test on different operating systems and R versions
- Easy Collaboration: Automated quality checks helps contributors by saving time and manuel efforts

3.3.2 Contribution of the read.sig Function to R-FieldSpectra

Spectra Vista Corporation offers advanced analytical instruments such as SVC HR-1024i for precise measurement of NIRS data. Discussions with researchers in this field revealed that SVC instruments are gaining popularity due to their advanced technical and analytical features. This led us to a closer look into the available functionalities in `R-FieldSpectra` to read the `.sig` files generated by these instruments. The lack of such function to read `.sig` files directly from the instrument and to better structure it for downstream analysis was lacking, and there we identified an opportunity to contribute to the scientific community by creating such one. We created `read.sig` function and opened a pull request to `R-FieldSpectra` (https://github.com/serbinsh/R-FieldSpectra/pull/25/commits) The `read.sig` is a simple function which read the spectral data, extract wavelengths and reflectance values and make it into a data frame along with file names and a couple of optional columns.

Functionality
Inputs:

- Path: Specifies the location of `.sig` files (directory or single files)

- Default: A boolean parameter determining all columns or only selected columns are retained in the output.

Outputs: A data frame containing,

- Wavelengths (nm): Wavelengths extracted from the data
- Reflectance: Percentage of reflectance
- File name: Source file information
- Optional columns: Reference and target values

The function starts by validating the file or directory path followed by reading and extracting relevant data from each file. This processed data is then merged into a data frame.

	Wavelengths(nm)	Reference Values	Target Values	Reflectance(%)	file_name
1	338.0	874.34	79.17	9.05	copy LILY LEAF i5679.sig
2	339.4	911.80	70.91	7.78	copy LILY LEAF i5679.sig
3	340.9	935.30	63.17	6.75	copy LILY LEAF i5679.sig
4	342.4	979.83	83.64	8.54	copy LILY LEAF i5679.sig
5	343.8	1010.03	74.98	7.42	copy LILY LEAF i5679.sig
6	345.3	1070.98	78.08	7.29	copy LILY LEAF i5679.sig
7	346.8	1134.62	70.59	6.22	copy LILY LEAF i5679.sig

Figure 3.8: Example output of read.sig function

```
# Specify the path to the directory containing .sig files
path_to_sig_files <- "/path/to/sig/files/"

# Use the read.sig function to read and extract spectral data
# Retain all columns by setting `default = TRUE`
spectral_data <- read.sig(path_to_sig_files, default = TRUE)

# Display the first few rows of the combined data
print(head(spectral_data))
```

This function also provides detailed documentation created using `roxygen2` and unit tests created using `testthat` functions. The documentation make sure that all users can easily understand the usage, purpose and arguments of the function. Unit tests verify the accuracy and reliability of the function in various scenarios, ensuring robust performance and minimizing the likelihood of errors during use. Below is an example of unit test used available in the package testing for summarizedexperiment object output.

```
# Unit test checking for the summarizedexperiment object output

test_that("Function returns a SummarizedExperiment object", {
  test_dir <- setup_test_sig_data()
  result <- read_summarizedexperiment_sig(test_dir)
  expect_s4_class(result, "SummarizedExperiment")
})
```

3.4 Prediction of Leaf Traits From NIRS Data

Background of the study

The data for the first analysis is collected from the study conducted by Pablo Castro Sanchez-Bermejo in the experiment named *Tree and mycorrhizal fungal diversity drive intraspecific and intraindividual trait variation in temperate forests: Evidence from a tree diversity experiment* which aimed to study the role of tree species richness and mycorrhizal fungal diversity in driving intraspecific and intraindividual trait variation in temperate forests [15]. The study is conducted at Bad Lauchstädt Experimental Research Station in Saxony-Anhalt, Germany. The experimental setup included 80 plots , planted with 10 native deciduous tree species, each associated with arbuscular mycorrhizal (AM) or ectomycorrhizal (EM) fungi. The dataset contains measurements from 3200 leaves, sampled from 640 trees spanning across 10 tree species. Leaf spectral data were collected using a FieldSpec4 Wide-Res Spectroradiometer, covering the 350–2500 nm range.

In addition to this, the data excel sheet also contains spectral and trait data from a different study. It is from a study named Kreinitz experiment in which they studied *Leaf trait variation within individuals mediates the relationship between tree species richness and productivity* [49,15]. The data from this experiment is used in combination with the data from leaf trait study [15] only in analysis of phosphorus content. The laboratory analysis of leaf phosphorus content faced technical problems during laboratory analysis in the first experiment [15]. Therefore, to compensate the data loss, additional but comparable samples from the Kreinitz experiment were used.

Morphological and chemical traits were determined, including:

- Specific Leaf Area (SLA) [mm^2/mg]
- Leaf Dry Matter Content (LDMC) [mg/g]
- Carbon-to-Nitrogen ratio (C:N) [g/g]
- Carbon content (C) [%]
- Phosphorus content (P) [μmol/g]

Apart from these five traits, nitrogen content(N), calcium(Ca), magnesium(Mg) and potassium(k) were also measured in this study [15]. However, the analysis focused on these (figure:3.9) five traits due to time constraints and their relatively lower proportion of missing values compared to other measured traits. The figure 3.9 illustrate the ecological relevance of these traits.

3.4.1 Trait Prediction Using PLSR

Libraries

The following libraries were imported for the successful execution of Partial Least Square Regression (PLSR):

- readxl: For importing the spectral and trait data from excel sheet.
- dplyr and tidyr: For data preparation

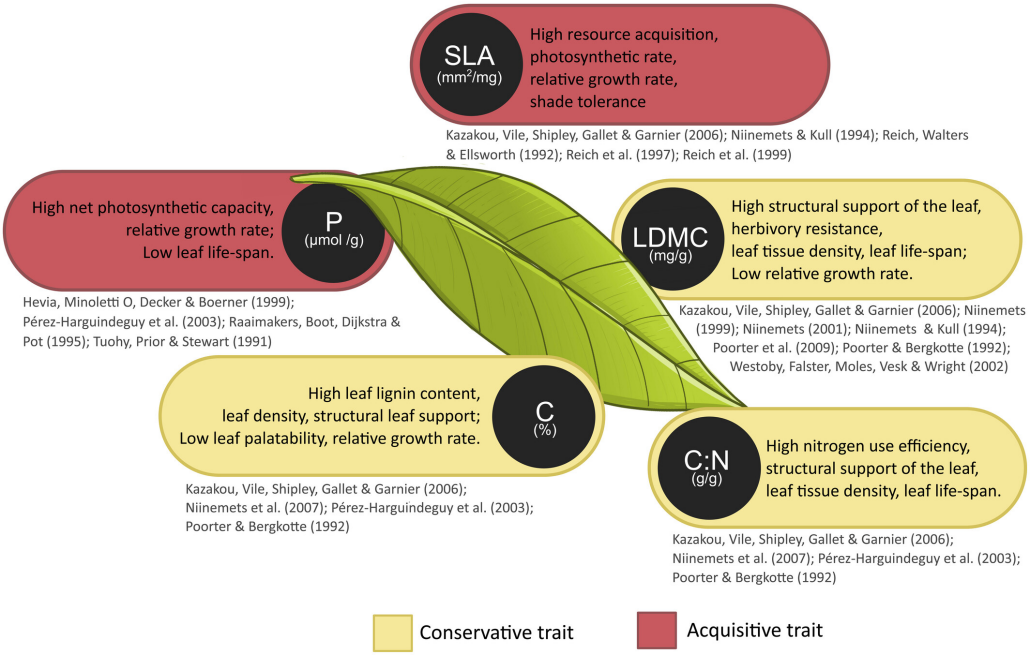

Figure 3.9: Leaf traits and their ecological function along with list of literature describing them. The traits are colored differently according to the leaf economics spectrum (LES) [15]

- caret: For model training and validation
- plantspec: For managing the spectral data
- ggplot2 and plotly: For graphical representation of the data

Data Loading and Preprocessing of SLA

The spectral data and trait data, which was stored in Excel sheets were imported using the `read_excel` function from the `readxl` package. Each dataset was read from its respective sheet in the excel file and then preprocessed to ensure the compatibility with PLSR analysis.

```
# Reading the excel sheet
plantspec.spectra <- read_excel("/path/to/dataset.xlsx", sheet = "
   Calibration set spectra")
plantspec.data <- read_excel("/path/to/dataset.xlsx", sheet = "
   Calibration set traits")
plantspec.data$SLA <- as.numeric(plantspec.data$SLA)

# Removing the outliers
plantspec.data <- plantspec.data[!is.na(plantspec.data$SLA), ]
plantspec.data <- subset(plantspec.data, Tree != "70_Ti3")
```

Data Splitting

The `subdivideDataset` function from `plantspec` package is used for splitting the spectral and trait data. The spectral data is passed on to the function as a matrix and the trait data is passed to the function as a vector. The dimensions of the data were checked after the preprocessing step to avoid any errors during data splitting. The spectral data was a matrix of 196 data points and 2151 features and the response variable SLA had 196 data points. The seed is

set to 0 for reproducibility.

```
training_set <- !(subdivideDataset(
  spectra = input,
  component = component_SLA,
  method = "PCAKS",
  p = 0.3,
  type = "validation" ))
```

- spectra: The predictor data of class list or matrix and in this case the spectral data matrix.
- component: Response variable for the analysis. In this study a vector variable of 196 data points are used for SLA, Carbon Nitrogen ratio, Carbon content analysis and 194 samples for LDMC. In the analysis of Phosphorus content, due to the use of additional data set 225 number of samples were used.
- method: Specifies the desired method. In this project "PCAKS" is used. It is a hybrid method combining Principal Component Analysis and Kennard-Stone Algorithm to split the data.
- p: The propotion of the dataset to select the "calibration" or "validation" group
- type: Depending on the type of subset, "calibration" or " validation" is selected.

This function returns a logical index in which `TRUE` meaning the sample is part of training set and `FALSE` meaning sample is part of validation set. The negation (!) operator negates the logical result generated from this function reversing the rows belongs to the validation test to training set. Since the parameter `p` is set to 0.3, that is 70% is allocated to the training set. The "PCAKS" is selected to improve model performance and its better representation of data especially when handling the high dimensional spectral data. Principal Component Analysis (PCA) simplifies the data by reducing its dimensionality while preserving its essential structure. Kennard-Stone Algorithm (KS) uses the Euclidean distance and selects representative samples from a dataset based on their spatial distribution in the feature space.

Data Scaling

The data scaling is performed using the SNV function, which stands for Standard Normal Variate. It is a normalization technique to remove the scatter effects and baseline shifts by taking each spectrum and doing the centering and scaling individually. SNV ensures that subsequent analysis will remain unbiased from variations in sample properties [48]. The SNV transformation was implemented using a custom R function.

```
# Define SNV function
SNV <- function(spectra) {
  spectra <- as.matrix(spectra)
  spectrat <- t(spectra)
  spectrat_snv <- scale(spectrat, center = TRUE, scale = TRUE)
  spectra_snv <- t(spectrat_snv)
  return(spectra_snv)
}
```

```
# Apply SNV function to the spectral data
input <- data.frame(SNV(input))
```

The above script first converts the spectral data into a matrix format to enable matrix operations then transpose matrix to ensure scaling along each row (spectrum). Centering and scaling is then implemented in each spectrum with the use of `scale` function. Finally the transformed matrix is transposed back to its original orientation.

The trait values were also scaled to make sure they have a mean of zero and a standard deviation of one.

```
component_SLA_scaled <- scale(component_SLA)
```

	component_SLA
1	13.742429
2	8.254233
3	17.742290
4	10.720819
5	11.328094
6	14.664960
7	17.995866

Figure 3.10: Component SLA before scaling

	V1
1	-0.59794219
2	-1.44772775
3	0.02139133
4	-1.06580458
5	-0.97177498
6	-0.45509869
7	0.06065477

Figure 3.11: Component SLA after scaling

Figure3.10 shows the first trait in the analysis in its original values. This vector values are then transformed into a matrix which is shown in figure3.11. The loss of column name is due to the `scale` function used for here and this function does not carry column names by default.

Split data into train and test

The data is then split into training and test sets based on the `training_set` index. Both the predictors and response data sets are split into train and test. The train data will be used for creating the PLSR model and then the test data will be used to predict how well does the model perform on unseen data.

```
X_train <- input[training_set, ]
Y_train <- component_SLA_scaled[training_set]
X_test <- input[!training_set, ]
Y_test <- component_SLA_scaled[!training_set]
```

Following the data split, a data frame is created using the SLA values comes under train (`Y_train`) and the spectral values for train (`X_train`). The first column will have the SLA values and spectral data in the following columns. Training control object is also created to use with `train` function from `caret` package in R. The `trainControl` function defines parameters for resampling for model training.

```
# Create a data frame for the train function
train_data <- data.frame(SLA = Y_train, X_train)

# Create train_control object
train_control <- trainControl(method = "cv", number = 20)
```

- method: specifies the preferred resampling method during model training. "cv" stands for cross-validation, which is a technique used to test the model performance by splitting the data into multiple subsets. The model will be trained on some subsets and validated on the rest. This process is then repeated multiple times to evaluate the generalizability of the model.
- number: this specifies the number of subsets to be used for cross-validation. In this project 20 subsets were used. The data is split into 20 subsets and the model will be trained for 20 times, each time using 19 subsets for training and remaining one for testing.

Model Training

In this step a PLSR model is trained to predict the SLA values using the data stored in `train_data` data frame. The training time was also recorded to compare with the rest of the models.

```
start_time_pls <- Sys.time()

pls_model <- train(SLA ~ ., data = train_data, method = "pls",
   tuneLength = 20, trControl = train_control)

end_time_pls <- Sys.time()
```

The PLSR model was trained using the `train` function from `caret` package. The parameters used here are:

- $SLA \sim .$: This specifies to take the column with name SLA as response variable from the dataset and the rest of the columns as predictors for creating the PLSR model.
- data: this state, which dataset to be used for creating the model. In this model, a data frame named train_data is passed to the model with both the predictor and the response variable that contains the training data.
- method: this specifies which method to be used for training.
- tuneLength: It is an integer evaluating specified number of values for an optimal number of components to use in PLSR model. In this model `tuneLength` is set to 20 and it evaluate 20 different values and gets the best value in which the model performs well.
- trControl: this specifies the number of subsets for cross-validation

Figure 3.12 illustrate the summanry of the PLSR model created for predicting the SLA values. This summary give information on the dimension of the data as well as the number of components considered while creating the model. The values under each components shows the percentage of variance explained by the model with increasing number of components. the `X` represents the predictor variable and `.outcome` points to response variables. For instance, when

```
Data:   X dimension: 137 2151
        Y dimension: 137 1
Fit method: oscorespls
Number of components considered: 11
TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X           83.22    94.43    96.24     97.9    98.21    98.52    99.13    99.30
.outcome    75.86    81.15    88.09     89.2    91.36    92.66    93.08    93.83
          9 comps  10 comps  11 comps
X           99.38     99.49     99.54
.outcome    94.70     95.07     95.53
```

Figure 3.12: Summary of PLSR model for SLA trait, explaining the data dimension used for creating the model and the number of components considered. This shows how much variance is explained in X (predictors) and Y (outcome SLA) as the number of components increases. During PLSR model training only 11 components were selected based on the model performance metrics(R^2, RMSE, MSE) since performance diminished beyond this point despite considering 20 components.

considering only one component or one latent variable, 83.22% of variance in `X` is explained and for response (`Y`) variable it was limited to 75.86%. The fit method used for creating this model is `oscorespls`. It is an algorithm for fitting PLSR models.

Figure 3.13 illustrate the relationship between RMSE values and number of components. Out of 20 components considered, 11 was chosen due to the lowest RMSE, which will facilitate the predictive accuracy of the model. Apart from this, the PLSR model also give additional performance metrics, including R^2 and Mean Absolute Error(MAE) for each number of components.

```r
# Find the index of the minimum RMSE
rmse_values <- pls_model$results$RMSE

optimal_components <- pls_model$results$ncomp[which.min(rmse_values)]
```

The optimal number of components were selected by extracting the RMSE values from the model and choosing the one which has minimum value. In this case, 11 was chosen and passed as parameter for predicting in unseen data.

Model Evaluation

The final model's performance was evaluated using the test dataset. For predicting `predict` function and `postResample` function is used to evaluate the performance of PLSR model on test data, both are from the package called `caret`.

```r
# Predictions
predictions_pls <- predict(pls_model, ncomp = optimal_components,
   newdata = X_test)

# Evaluation
results_pls <- postResample(predictions_pls, Y_test)
```

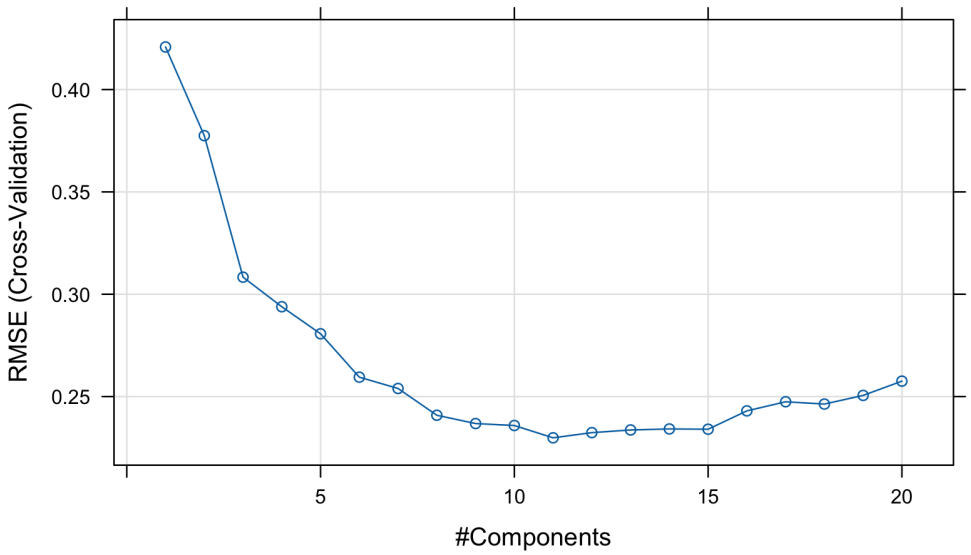

Figure 3.13: Plot of PLSR model for SLA trait prediction illustrating how the number of components were selected. A total of 20 components were considered but only 11 are selected considering different metrics, one being RMSE. This plot shows component 11 has the lowest RMSE value.

In R, the `predict` function is a generic function and it automatically allocates the appropriate method based on the object class passed to it. This is the reason why the `predict.train` is not explicitly given here. To predict the SLA values, the trained model (`pls_model`) is passed as the first parameter followed by the optimal number of components (`optimal_components`) and the test predictor data (`X_test`).

The performance of the model is evaluated using the `postResample` function from `caret` package. It takes two parameters, one is the predictions the previous function and the next is the test values of the response variable (`Y_test`). This function outputs the values for RMSE, R^2 and MAE, which is then used to compare with the rest of the models in this project.

Visualization

For the easy understanding and comparison, it was necessary to visualize the result. The plots were created with the help of `ggplot2` and `plotly` packages. The `ggplot2` in this case creates a scatter plot with actual SLA values on X and predicted SLA values on Y axis. The `ggplotly` function is then applied on to the resulting plot to make it interactive and suited well for the vignette.

Helper functions

Two custom helper functions were developed to ease the subsequent analysis and improve code readability by reducing the complexity in the vignette. First function is called `perform_PLSR`, which accepts the preprocessed spectral data along with the trait data and outputs the trained model along with training time and test data set for model evaluation. It is highly useful since the analysis follows a standard workflow in which the functions and packages used are the same across all the five traits. The only change is in the preprocessing of spectral data, which is done separately for each trait.

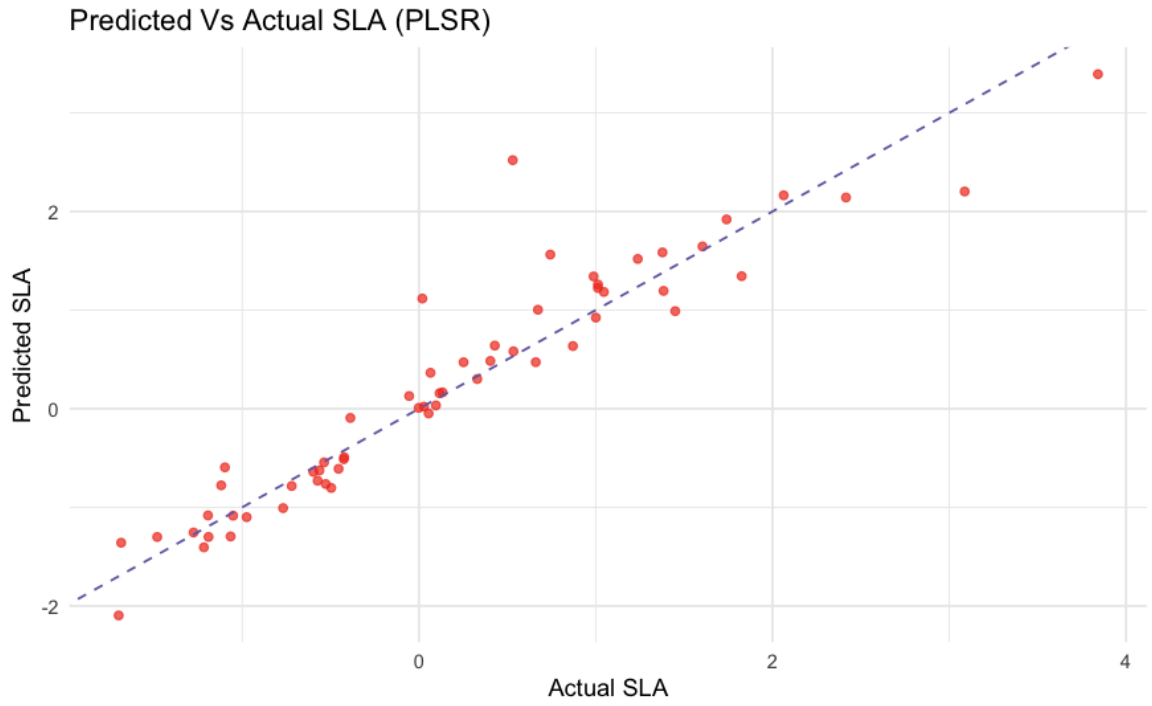

Figure 3.14: The scatter plot illustrating the predicted and actual SLA values.

The second function is `plot_gg`, which accepts multiple arguments including, predicted values, actual values, title of choice, x and Y labels and color of choice for the plot. This function creates a ggplot object from the data provided. Both of this functions have been used in the analysis of the next four traits. The R scripts of both functions are provided as additional files.

```
# Source the perform_PLSR function
source("/path/to/the/function/perform_plsr.R")

# Source plot_gg function
source("/path/to/the/function/plot.R")
```

Prediction of LDMC using PLSR

The prediction of LDMC using PLSR follows similar steps as of SLA prediction except the preprocessing. In LDMC, data corresponding to this specific trait is first extracted from the excel sheet and outlayers were removed.

```
# Removing the outlayers
plantspec.data[which(is.na(plantspec.data$LDMC)),]
input <- as.matrix(plantspec.spectra[, -1])
input <- input[-which(is.na(plantspec.data$LDMC)),]
plantspec.data.ldmc <- plantspec.data[!is.na(plantspec.data$LDMC),]
input <- input[-which(plantspec.data$Tree == "70_Ti3"),]
plantspec.data.ldmc <- subset(plantspec.data.ldmc, Tree != "70_Ti3")

# Extract the LDMC values
component_LDMC <- plantspec.data.ldmc$LDMC
```

After the necessary preprocessing steps, the custom helper function is applied to create the PLSR model for LDMC.

```
# Train the PLSR-LDMC model
model_ldmc <- perform_PLSR(input = input, trait = component_LDMC)

# Extracting the model from the output
results_ldmc <- model_ldmc$plsr_model
```

The `perform_PLSR` function takes both predictor and response variables as inputs and do the splitting, scaling and training of the model for the specific trait. The output of the function is shown in the figure 3.15. From this, the model is filtered out for passing on to predict function for the downstream analysis.

Name	Type	Value
model_ldmc	list [4]	List of length 4
plsr_model	list [24] (S3: train, train.formu	List of length 24
training_time	double (S3: difftime)	1.100238 mins
Y_test	double [58]	0.7488 0.1645 0.6610 0.4613 -0.0633 0.4534 ...
X_test	double [58 x 2151]	-0.920701 -0.961543 -0.795333 -0.893112 -0.849143 -1....

Figure 3.15: Illustrate output of the `perform_PLSR` function

The `predict` function from caret package is used to make predictions on the model using the test data. It is similar to the prediction of SLA. A detailed version with the codes are provided in the vignette.

Visualization using the `plot_gg` function

The visualization is implemented with the help of custom `plot_gg` function. It takes seven parameters such as prediction data, test data for response variable, title, labels for x and y axis , point color and line color. Close attention was paid to each trait, with a particular focus on visualizing the results to make sure clarity and improve meaningful comparisons.

```
plot_ldmc <- plot_gg(predictions_pls_ldmc, Y_test_ldmc,
                     title = "Predicted Vs Actual LDMC (PLSR)",
                     x_label = "Actual LDMC",
                     y_label = "Predicted LDMC",
                     point_color = "#31a354",
                     line_color = "#e34a33")

# For interactive plots
ggplotly(plot_ldmc)
```

Prediction of C content and C to N ratio (C:N)

The prediction of Carbon content and Carbon to Nitrogen ratio follows similar steps as the LDMC prediction. The only difference is in the preprocessing steps, in which each trait filter out columns relevant to that specific analysis. It uses the advantage of the helper function to

execute the long scripts. The results from each of these analysis were carefully studied and saved. The plots were created and is available in the supplementary vignette as interactive plots. The results are provided in the final chapter of this thesis.

Prediction of Phosphorus content using PLSR

The analysis of phosphorus content involved additional steps compared to the analysis of other traits. It used additional data from Kreinitz experiment [49] along with data from leaf trait study [15]. This is due to the loss of data from a technical error occurred during the laboratory analysis of Phosphorus content. The data is chosen considering the comparability of the samples, instruments and method used [15].

```
# Removing the outlayers

plantspec.spectra2 <- read_excel("/path/to/the/data/dataset_20231010.
   xlsx", sheet = "Kreinitz set spectra") # Spectral Kreinitz set
plantspec.data2 <- read_excel("/path/to/the/data/dataset_20231010.xlsx
   ", sheet = "Kreinitz set traits") # Traits Kreinitz set

cbind(plantspec.spectra2$Spectral.file.name, plantspec.data2$Level_ID)
    # Checking correlation between both sets
```

The preprocessing is done to both data sets. The dimension of the data from Kreinitz experiment[49] was 101 samples and 2152 features. The dimension of the leaf trait data was 198 samples and 2152 features. Both this data set is subjected to preprocessing and the dimension of the combined data for the prediction of Phosphorus content had 225 samples and 2151 features.

```
# Removing outlayer from the samples
plantspec.data[which(is.na(plantspec.data$P)),]
input <- as.matrix(plantspec.spectra[, -1])
input <- input[-which(is.na(plantspec.data$P)),]
plantspec.data.p <- plantspec.data[!is.na(plantspec.data$P),]
input <- input[-which(plantspec.data.p$Tree == "75_So2"),]
plantspec.data.p <- subset(plantspec.data.p, Tree != "75_So2")
input <- input[-which(plantspec.data.p$Tree == "16_Be3"),]
plantspec.data.p <- subset(plantspec.data.p, Tree != "16_Be3")
input <- input[-which(plantspec.data.p$Tree == "71_Fa3"),]
plantspec.data.p <- subset(plantspec.data.p, Tree != "71_Fa3")

plantspec.data[which(is.na(plantspec.data2$P)),]

# Preprocessing of second dataset
input2 <- as.matrix(plantspec.spectra2[, -1])
input2 <- input2[-which(plantspec.data2$Level_ID == "B16_Li17_3"),]
plantspec.data2 <- subset(plantspec.data2, Level_ID != "B16_Li17_3")
input2 <- input2[-which(plantspec.data2$Level_ID == "B41_Es11_3"),]
plantspec.data2 <- subset(plantspec.data2, Level_ID != "B41_Es11_3")

# Studying the distribution of the values for P
component_P <- c(plantspec.data.p$P, plantspec.data2$P)
```

After preprocessing, the analysis followed steps similar to the LDMC, C and C:N. It used the helper functions to create the model and visualize the data.

3.4.2 Extrapolation study of PLSR

The extrapolation study is considered to evaluate the performance of the Partial Least Squares Regression (PLSR) under extreme conditions. This extrapolation studies are applied to both high and low data ranges by a specific data splitting strategy. In the highest range extrapolation study, the top 30% of values reserved for testing while the rest 70% of the data is used for training. This means, the model will be trained using the values of less variance and then subjected to test and predict on the unseen data of high variance (data outside of the training domain).

A similar approach was applied to the lowest data range as well, where the lowest or the bottom 30% of values were selected for testing while the rest of the data is used for training the model. This methodology allows for assessing the model's ability to predict beyond the range of trained or observed values and provide insights into its extrapolation capabilities.

The extrapolation study is limited to a single trait, with SLA chosen as the response variable. PLSR produced excellent results in predicting SLA values, which is why SLA was selected for the extrapolation study.

Extrapolation at upper data range

Extrapolation at upper data range include, the upper 30% for testing and the rest for training. This follows similar steps compared to the previous PLSR analysis, except the data splitting strategy.

```
# Removing outlayer from the samples
plantspec.data[which(is.na(plantspec.data$SLA)),]
input <- as.matrix(plantspec.spectra[, -1])
input <- input[-which(is.na(plantspec.data$SLA)),]
plantspec.data.sla <- plantspec.data[!is.na(plantspec.data$SLA),]
input <- input[-which(plantspec.data$Tree == "70_Ti3"),]
plantspec.data.sla <- subset(plantspec.data.sla, Tree != "70_Ti3")

# Studying the distribution of the values for SLA
component_SLA <- plantspec.data.sla$SLA

#we manually select the extreme values from both sides
X <- input
Y <- plantspec.data.sla$SLA

# Scale the input (spectral data) and SLA values for better model
X_scaled <- scale(X)
Y_scaled <- scale(Y)
```

The preprocessing is done by removing the outlayers and extracting the values necessary for the analysis. Both X and Y variables are then scaled and prepared for splitting the data.

```
# Highest 30% of Values
cutoff <- quantile(component_SLA, 0.7, na.rm = TRUE)

# Index for test set
test_set_index_h <- which(component_SLA >= cutoff)
```

```
# Index for trainig set
training_set_index_h <- which(component_SLA < cutoff)

# Split the scaled data accordingly
X_train_e_h <- X_scaled[training_set_index_h, ]
Y_train_e_h <- Y_scaled[training_set_index_h]
X_test_e_h <- X_scaled[test_set_index_h, ]
Y_test_e_h <- Y_scaled[test_set_index_h]

train_control <- trainControl(method = "cv", number = 20)

# Create data frame
train_data_e_h <- data.frame(SLA = Y_train_e_h, X_train_e_h)
```

Here instead of using the `subdivideDataset` from `plantspec` package, the data is split using the `quantile` function from base R. This function calculates specific percentiles of a numeric vector thereby allowing the data selection based on a resulting threshold. In this case, the cutoff value is determined using the 70th percentile of component_SLA. The parameter "na.rm", upon setting to TRUE ensures that the NA values are ignored in the calculation.

The `quantile` function has given the output of 70% of the values in component_SLA are below 20.266(original SLA values before scaling), meaning the highest 30% fo the values are above this threshold. The cutoff of 20.266 was determined from unscaled SLA values, and the same indices are applied to Y_scaled to ensure correct selection. Values above or equal to this threshold are assigned to the test set and those below are assigned to the training set.

Using the index for test and train the spectral data(X) and the trait data(Y) are split. Which is then used to create the train data frame to pass on to the `train` function. The rest of the process follows similar steps compared to the previous PLSR analysis. The `train` function is employed to train and create the model and `predict` function is used to evaluate the model followed by `postResample` function to get the R^2 and $RMSE$ values which is then saved to the results table along with training time of the model. The plots are created using `ggplot2` for better visualization.

Extrapolation at lower data range

Similar to extrapolation using the highest values, it was also to evaluate the PLSR model performance at the lowest values. In this case lower 30% is used for testing and the rest for training the model. This uses similar functionalities and trait data compared to the former and only differs in how the parameters are used in `quantile` function.

```
# Lower 30% of values
cutoff_low <- quantile(component_SLA, 0.3, na.rm = TRUE)

# Index for test set
test_set_index_l <- which(component_SLA <= cutoff_low)

# Index for trainig set
training_set_index_l <- which(component_SLA > cutoff_low)
```

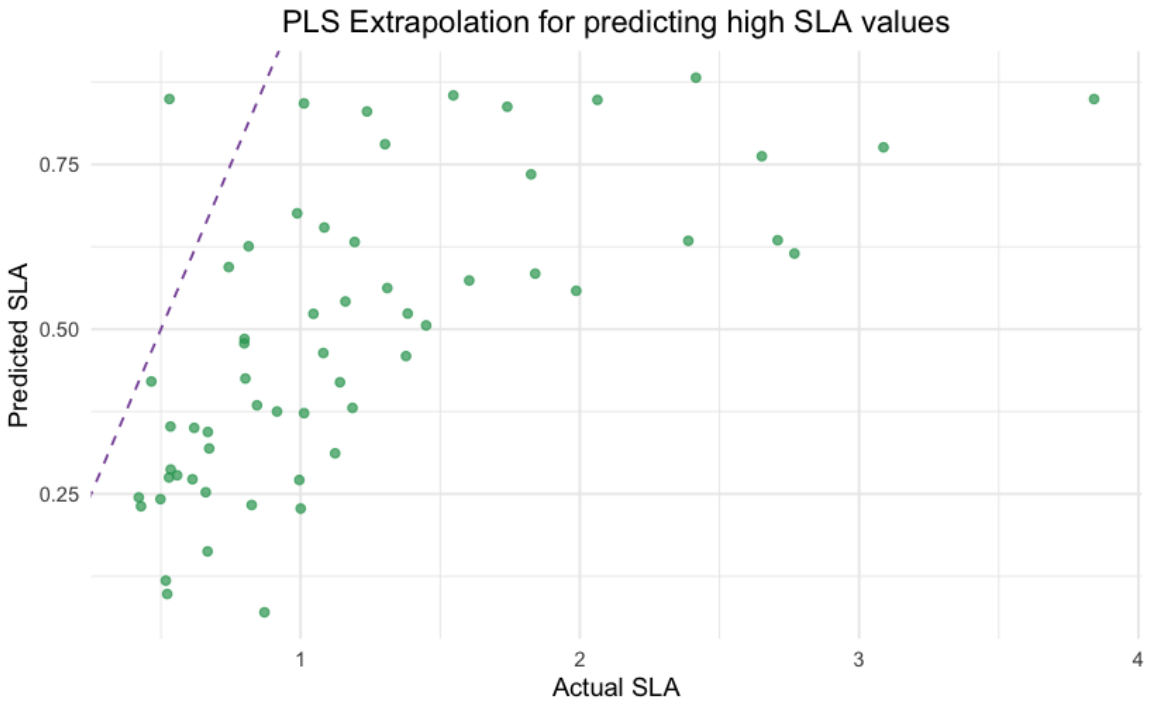

Figure 3.16: The scatter plot illustrates the predicted versus actual SLA values for the extrapolation of the PLSR model on the upper data range.

This function outputs, 30% of the values in component_SLA are below 13.60, meaning the lowest 30% of the values are below this threshold. The test and training set indices are created by selecting the indices of component_SLA such that the values less than or equal to the threshold (cutoff_low) are assigned to the test set, and the remaining values are assigned to the training set. This is then used to split the data accordingly and then proceed to the downstream analysis in the similar was to extrapolation study on upper data range.

3.4.3 Variable importance in PLSR

Variable importance is a technique used to identify the most influential features for the model's predictions. In this analysis, variable importance is found out using the `varImp` function from `caret` package. This function measures the variable importance for PLSR based on the weighed sums of the absolute regression coefficients. The weights are determined by how much they reduce the sum of squared errors across the PLS components. They are calculated individually for each outcome [50].

For this analysis, the variable importance of SLA and carbon content was evaluated using the `varImp` function from `caret` package. This function calculates the importance of each features based on the level of its contribution to the overall prediction of the traits.

Variable importance of SLA

The following R script is used to analyze the important variable during SLA prediction of PLSR model.

```
# Variable importance of SLA in PLSR
vi_sla <- varImp(pls_model, scale = F)
```

```
# Extract the importance scores and make it into a data frame
vi_sla_df <- as.data.frame(vi_sla$importance)

# Define wavelength range
wavelengths <- seq(350, 2500)

# Create a data frame for plotting
vi_sla_df <- data.frame(Wavelength = wavelengths, Overall = vi_sla_df)
```

The `varImp` function is used and the PLSR model for SLA from the previous analysis is passed on to the function as the first parameter. It also accepts optional parameters and can be given according to the data type and the research goal. This function will output data with feature name and its importance score. The scores are given to all the features in the study in this case the wavelength from 350nm to 2500nm. The output of this function is an object of class `varImp.train`. To visualize the output, the importance scores are extracted from the object and converted into a data frame. This data frame is then used in the ggplot2 function to create a graph of variable importance. Figure 3.17 shows the variable importance scores on the Y axis and features(wavelengths) in the X axis.

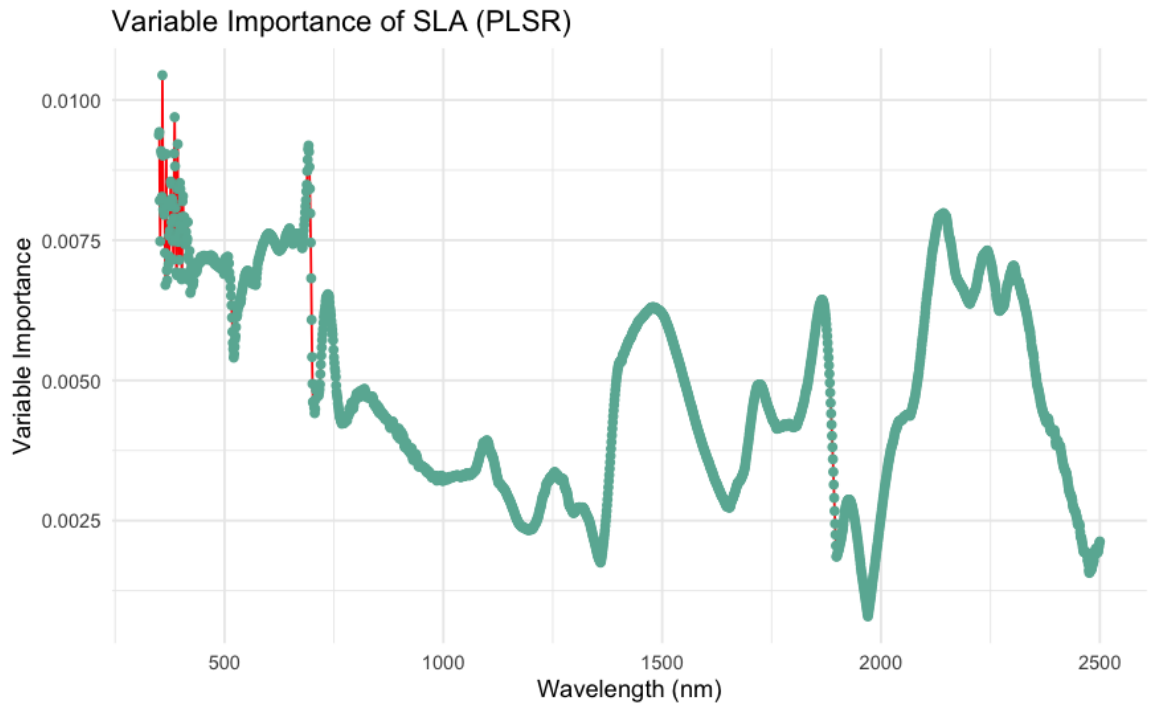

Figure 3.17: The variable importance plot of PLSR model, illustrating the contribution deapth of different features(wavelength) during SLA trait prediction. The importance scores are given in the Y axis and higher the score more significant the corresponding wavelength is in predicting SLA trait of a plant.

Variable importance of Carbon content

The variable importance in carbon content prediction using PLSR is also measured with the help of `varImp` function. The model corresponds to the carbon content is passed on to the function and it created a train object, from which the importance scores are extracted. This is

followed by the visualization by `ggplot2`.

3.4.4 Trait Prediction Using Random Forest

Random Forest(RF), a nonlinear approach, was also chosen to evaluate predictive ability and model performance. Instead of classification, regression was performed using Random Forest. The data used is the same for the previous PLSR analysis. The aim is to evaluate and compare the results of RF with PLSR and CNN. The analysis does not require any additional libraries, since it follows a uniform approach and only differs in the method used for training. It uses the `plantspec` package for data splitting and `caret` package for training model and predicting the results. The visualization is done with the use of `ggplot2` and `plotly` packages. All five traits were predicted and the results were recorded for comparison.

Data Loading, Preprocessing, Scaling, and Splitting

The steps for data loading, preprocessing, scaling and splitting follows the same approach as described for SLA prediction using PLSR. The main differences begin in the next step with model training for Random Forest

Model Training

The `train` function from `caret` package is employed to create the RF model for predicting SLA trait. Compared to the PLSR model, the method used here is "rf" (Random Forest) instead of "pls" (Partial Least Squares Regression). Additionally, the RF model excludes certain parameters that were present in the PLSR model.

```
# Training the model
start_time_rf <- Sys.time()
rf_model_sla <- train(SLA ~ ., data = train_data, method = "rf", ntree
    = 500)
end_time_rf <- Sys.time()
```

- method: "rf" specifies the model being trained here is Random Forest
- ntree: It is set to 500 to improve accuracy and is set constant for every trait being analyzed by RF model.

The time for training the model was recorded for comparing the training time of each model from different ML approaches. In this analysis the `ntree` is set to 500, it is the number of decision trees which will be used to built the RF model. The number of trees can be tuned for each data but we have used 500 since, It is considered as a default value in many analysis.

Model Evaluation

The performance of the model was evaluated using the test dataset. For predicting the SLA values, `predict` function from the `caret` package was used. In addition to that, `postResample` function from `caret` package is used to get the R^2 and RMSE values for comparison.

```
# Predictions
predictions_rf <- predict(rf_model_sla, newdata = X_test)

# Evaluation metrics
results_rf <- postResample(predictions_rf, Y_test)
R_squared_rf <- results_rf[["Rsquared"]]
rmse_values_rf <- results_rf[["RMSE"]]
```

Visualization

The `ggplot2` and `plotly` packages are employed for better visualization of the data. A dataframe with the test trait data (X_test) and predicted trait values were created and passed on to the `ggplot` function to create the graph (Figure 3.18).

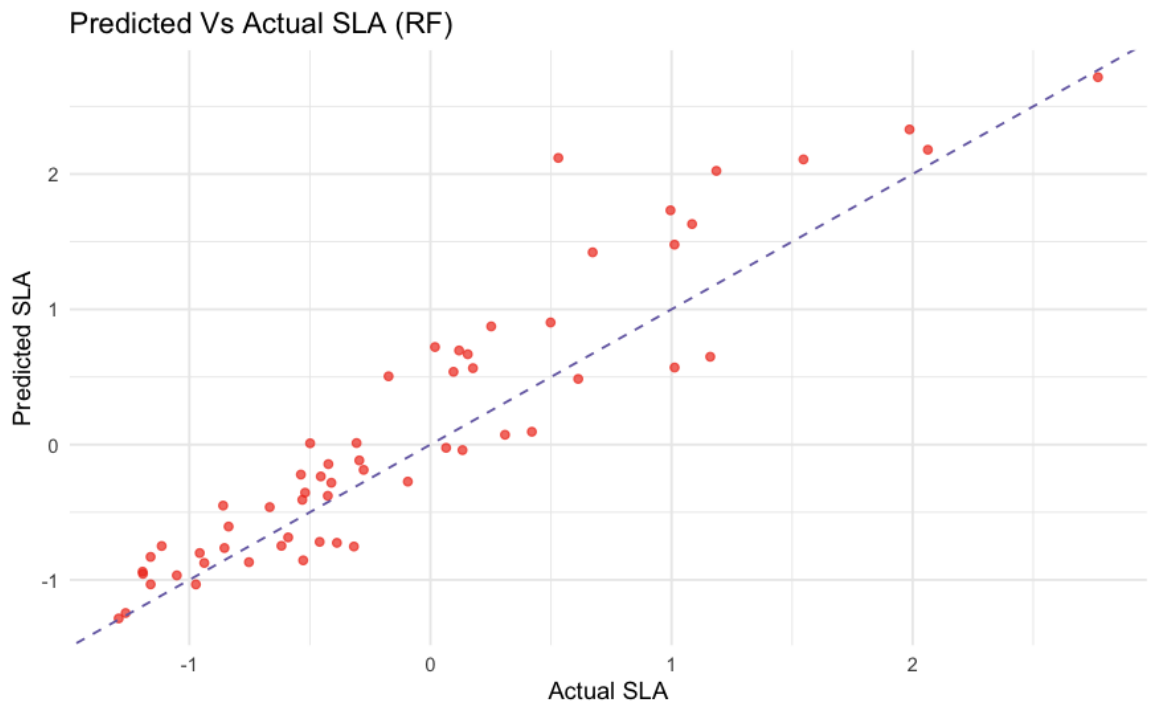

Figure 3.18: The scatter plot created using ggplot2 package, illustrating the predicted and actual SLA values of random forest model

Prediction of LDMC, C:N ratio, C and P content

The analysis of the rest of the traits follows the same preprocessing steps of the same in PLSR. The scaling and data partition uses the similar approaches to that of SLA prediction using RF. The only difference is that instead of using the trait name "SLA", the respective trait name for the analysis is used. The preprocessing steps vary across different traits but remain consistent with those of the PLSR model, as the same data is used. A uniform approach is followed to ensure comparability between models. The analysis of phosphorus content involved additional steps compared to the analysis of other traits. It used additional data from Kreinitz experiment [49] along with data from leaf trait study [15]. A detailed analysis is provided as a vignette, including the complete script and interactive plots.

3.4.5 Extrapolation study of RF

Extrapolation at upper data rage

Extrapolation at upper data range in random forest include, the upper 30% for testing and the rest for training. This follows similar steps compared to the previous RF analysis, except the data splitting strategy. Trait SLA has been chosen for the extrapolation study. The data is split using the `quantile` function from base R. The rest of the analysis is similar to the prediction of SLA using RF.

```
# Highest 30%
cutoff <- quantile(component_SLA, 0.7, na.rm = TRUE)

#index for test set
test_set_index_h <- which(component_SLA >= cutoff)

#index for trainig set
training_set_index_h <- which(component_SLA < cutoff)
```

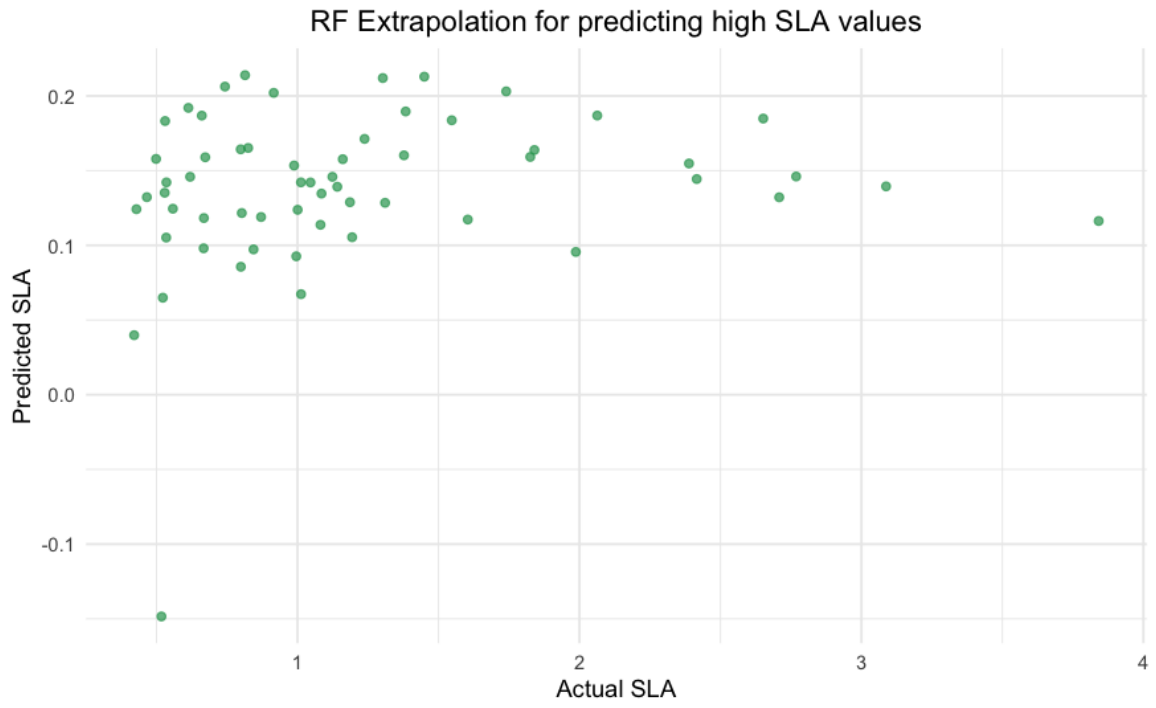

Figure 3.19: The scatter plot illustrates the predicted versus actual SLA values for the extrapolation of the random forest model on the upper data range

Extrapolation at lower data rage

Extrapolation at lower data range is similar to that of upper data range, it is to evaluate the RF model performance at the lowest values. In this case, lower 30% is used for testing and the rest for training the model. This uses similar functionalities and trait data compared to the former and only differs in how the parameters are used in `quantile` function.

```
# Lowest 30%
```

```
cutoff_low <- quantile(component_SLA, 0.3, na.rm = TRUE)

# Index for test set
test_set_index_l <- which(component_SLA <= cutoff_low)

# Index for trainig set
training_set_index_l <- which(component_SLA > cutoff_low)
```

3.4.6 Variable importance in RF

Similar to the PLSR, variable importance of SLA and Carbon content was assessed using the `varImp` function from `caret` package. This function calculates the importance of each feature based on the level of its contribution to the overall prediction of the traits.

Variable importance of SLA

The following R script is used to analyze the important variable during SLA prediction of RF model. Figure 3.20 illustrates the variable importance of the SLA trait in random forest model.

```
#Variable importance of SLA in RF
vi_sla_rf <- varImp(rf_model_sla, scale = F)

vi_sla_rf_df <- as.data.frame(vi_sla_rf$importance)
```

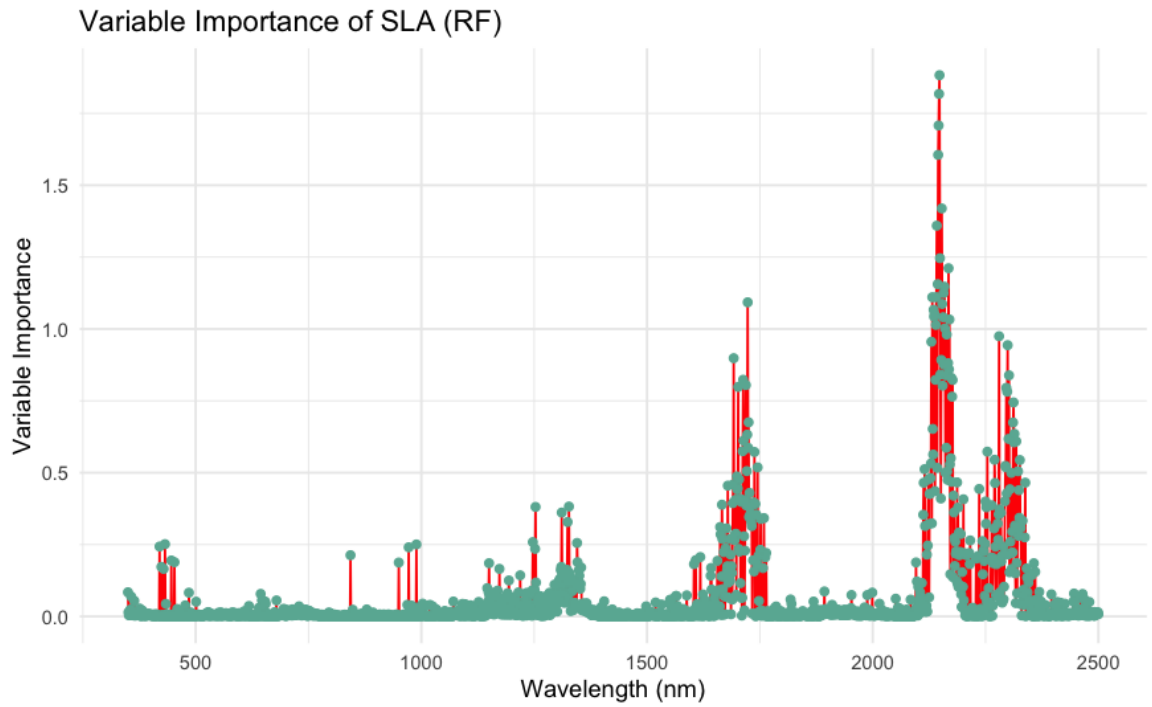

Figure 3.20: Variable importance graph of SLA trait in random forest model

Variable importance of Carbon content

The variable importance in carbon content prediction using random forest is also measured with the help of `varImp` function. The model corresponds to the carbon content is passed on to the function and it created a train object, from which the importance scores are extracted. This is followed by the visualization by `ggplot2`.

3.4.7 Trait Prediction Using Convolutional Neural Network

Unlike the previous ML models such as PLSR and RF, CNN is automatically learns features from the spectral data. It is a deep learning approach to predict the five leaf traits and the goal is to evaluate whether CNN can outperform the conventional models by comparing the R^2, RMSE and training time across all these models. This analysis follows similar preprocessing steps as the previous two ML approaches, as we are dealing with the same data and following a uniform protocol for better comparison. The data is from the leaf trait study [15] and all five traits were predicted using CNN. The CNN model, adapted from the leaf trait study [15], was reimplemented with two key modifications: the addition of an early stopping callback and padding within the CNN architecture. Only the essential scripts for this analysis were used, omitting some components from the original study.

Prediction of SLA using CNN

The SLA is the first trait in this analysis. The data is read from the excel sheet and the necessary preprocessing is done as same as in the previous analysis of SLA. The spectral data is saved into variable named "input" and the SLA values were extracted and saved into the the variable named "component_SLA". The SLA trait data is then normalized using the `scale` function from R base package. A histogram and a QQ plot is created to study the distribution of SLA values. This part of the analysis includes recreating the results from leaf trait study[15] and hyper parameter tuning for better comparison with other ML models.

```
# Studying the distribution of the values for SLA
component_SLA <- plantspec.data.sla$SLA

component_SLA <- scale(component_SLA)

par(mfcol = c(1, 2))
qqnorm((component_SLA), pch = 1, frame = FALSE) # Checking normality
   of the variable
qqline((component_SLA), col = "red", lwd = 2)

hist((component_SLA)) # Checking distribution of the variable
```

Similar to the previous SLA analysis, `subdivideDataset` function from `plantspec` package has been employed for defining the index for test and train split. It is followed by a custom function to scale the input (spectral) data. This function performs centering and scaling of the spectral data to remove the scattering and baseline shift. The data is then split into test and training set to train the CNN model and predict the response variables. About 70% of the data is used for training the model while the rest is saved for testing. A detailed script along with plots are provided in the vignette.

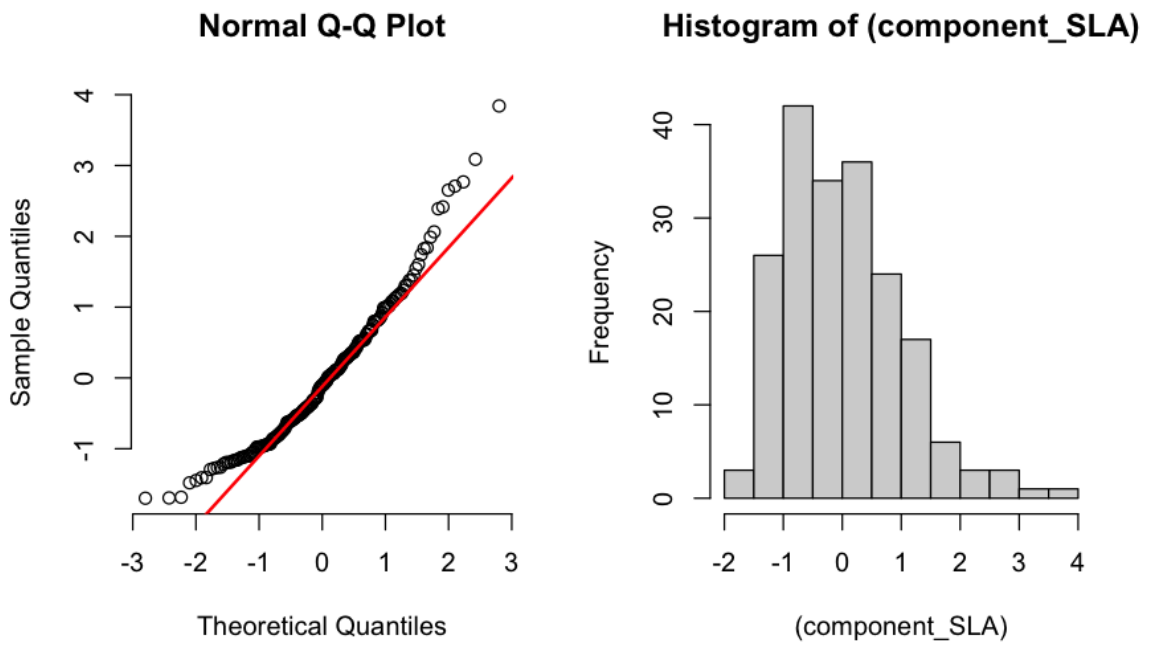

Figure 3.21: Q-Q plot (left) illustrating normality of SLA and histogram (right) showing the distribution of SLA [15]

CNN Architecture

The CNN model was implemented with the use of `Keras3` (version 1.1.0) library which works on the `TensorFlow` background, which is an open source ML frame work developed by google. `TensorFlow` provides an ecosystem to develop, train and deploy the ML and DL models and `Keras` is a high level DL application programming interface (API) which is built in top of `TensorFlow`.

```
#### Convolutional neural network (CNN) Architecture

model.sla <- keras_model_sequential(input_shape = c(NULL, 2151, 1))
   %>%

  layer_conv_1d(filter = 2, kernel_size = 50, padding="same", kernel_
     initializer=initializer_glorot_uniform(seed=0)) %>%

  layer_batch_normalization() %>%

  layer_max_pooling_1d(pool_size = 2) %>%

  layer_flatten() %>%

  layer_dense(128, kernel_initializer=initializer_glorot_uniform(seed
     =0)) %>%
  layer_dense(32, kernel_initializer=initializer_glorot_uniform(seed
     =0)) %>%
  layer_dense(8, kernel_initializer=initializer_glorot_uniform(seed=0)
     ) %>%
  layer_dense(1, kernel_initializer=initializer_glorot_uniform(seed=0)
```

```
    )
```

This CNN architecture consists of many layers for feature extraction and regression. The model is initialized with the help of `keras_model_sequential` function from `keras` and input shape is passed on to it as the argument. The input shape include, batch size, number of features(change with dimension of dataset) and number of channels. The output of this function is then passed as the first argument to the next layers through pipe operator(% > %). The first layer following the model initialization is the convolutional layer. This layer extracts spectral patters and in this project with the help of 2 filters of size 50 (each filter spans 50 consecutive wavelengths). The `kernel_initializer` argument specifies how the initial weights of the layer should be and is set to `initializer_glorot_uniform`, which helps to maintain gradient stability during model training. The `seed` is set to 0 for reproducibility. Padding is set to "same" to escape the default "valid" padding from `keras`. In default settings, no padding is added at the edge of input data and this results in reduction of the input length.

Followed by the first convolutional layer is the batch normalization layer. This layer is responsible for improving the model stability and training speed by computing the mean and variance and rescaling them to get consistent distribution.

Normalized data from this layer is then passed on to a max-pooling layer for dimensionality reduction. The pool size is set to 2, which reduces the spectral data (2151 features) to half (1075 features) of its original size.

The next layer is called flatten layer, which is responsible for converting the outputs of previous layers into a one dimensional vector for passing to the dense layers of CNN model. This layer bridges convolutional layers to dense layers by applying the `layer_flatten` function to the outputs of max-pooling layer.

Fully connected layer or in other words dense layers is created using four layers with different numer of neurons in each layer. Each layer in the dense layer takes the number of neurons as the first argument followed by `kernel_initializer`, specifying how the weights of the layer should be initialized. In this project we have used "Glorot Uniform initialization" with a fixed seed. Each layer has different purposes, the first dense layer with $128l$ neurons helps to learn the high level spectral patters and captures the abstract features. This is then passed on to the next layer with 32 neurons and it reduce the complexity by retaining only the important features. Third layer with 8 neurons will further refines the already extracted features. The final layer with 1 neuron will outputs a single continuous value. In this case each spectrum will output one SLA value.

Next the CNN model is passed to `compile` function from `keras`, specifying how the model should be optimized and what loss function to use.

```
model.sla %>% compile(
  optimizer = "adam",
  loss = "mean_squared_error")
```

The "adam" optimizer is used to optimize the model. Adam stands for Adaptive Moment Estimation and is an advanced gradient decent optimization technique used in DL. The optimizer controls the magnitude and speed of each parameter update by adjusting the learning rate, which guides the weights of the network based on the loss function gradient [51]. In this

Model: "sequential_7"

Layer (type)	Output Shape	Param #	Trainab…
conv1d_6 (Conv1D)	(None, 2151, 2)	102	Y
batch_normalization_5 (BatchNormalization)	(None, 2151, 2)	8	Y
max_pooling1d_3 (MaxPooling1D)	(None, 1075, 2)	0	-
flatten_2 (Flatten)	(None, 2150)	0	-
dense_5 (Dense)	(None, 128)	275,328	Y
dense_6 (Dense)	(None, 32)	4,128	Y
dense_7 (Dense)	(None, 8)	264	Y
dense_8 (Dense)	(None, 1)	9	Y

Total params: 279,839 (1.07 MB)
Trainable params: 279,835 (1.07 MB)
Non-trainable params: 4 (16.00 B)

Figure 3.22: CNN model architecture and parameters summary. This table summarizes the structure of the Convolutional Neural Network (CNN) used for SLA prediction. It includes details on each layer, output shape, and the number of trainable parameters.

model, the loss function is defined as the "mean squared error".

Define Early Stopping Callback

This deep learning model is trained over multiple "epoches" to minimize the loss function. Training for a longer period of time can lead to overfitting, in which the model performs well during training and fails to the test data and a larger computational load. The `callback_early_stopping` function from `keras` is used to regulate the training process by stopping the training if no improvement in loss function is shown after a certain number of epoches. Each epoch is one complete pass of the entire dataset through the neural network.

```
# Define Early Stopping Callback

early_stopping <- callback_early_stopping(
  monitor = "val_loss",  # Monitor validation loss
  patience = 500,        # Stop if no improvement for 500 epochs
  restore_best_weights = TRUE  # Restore weights from the best epoch
    )
```

Here, the model stops the training if there is no improvement in loss function after 500 epoches. This helps to reduce the training time to a huge extent and prevents overfitting. The training time for the CNN model predicting SLA was 28.11 minutes before implementing the early stopping callback. After incorporating early stopping, the training time was reduced to 7.30 minutes while maintaining the same R^2 value. This demonstrate the improved efficiency of the model without compromising in model performance.

Training CNN model for SLA

The training of CNN for predicting SLA from NIRS data is implemented using the `fit` function from `keras` package. The time for training the model was recorded.

```
start_time_sla_cnn <- Sys.time()

history <- model.sla %>% fit(x=as.matrix(x.train), y=as.matrix(y.train
   ),
                             epochs = 5000,
                             verbose = 1,
                             validation_split = 0.2,
                             shuffle = FALSE,
                             callbacks = list(early_stopping)
                             )

end_time_sla_cnn <- Sys.time()
```

The first two arguments to the `fit` function is the spectral training data and the trait data. Both of them are converted to matrix while passing to the function to make it compatable with `TensorFlow`. It is then followed by defining the epoches number. Number of epoches varies depending on the trait, in the analysis of SLA trait, 5000 epoches were given, meaning the dataset passes through the neural network 5000 times if the early callback function not satisfied. In each epoch the model update its weight to minimize the loss function. The verbose is set to 1 to monitor real-time training progress. The validation split is set to 0.2, which means 20% of the training data will be allocated and used for validation of the model. During training, the model will be trained on 80% of the training data and evalutes its performance on the remaining 20% of data. The "shuffle" is set to "FALSE" to prevent the shuffling of data before training. Finally, the call back function is implemented to allow the model to stop training if validation loss does not improve for 500 epoches (Figure 3.23).

Model evaluation

The performance of the model was evaluated using the test dataset. For predicting the SLA values, `predict` function from the `caret` package was used. Once the model is trained it is then used to predict on unseen dataset (X.test).

```
prediction.test <- predict(model.sla, as.matrix(x.test))
```

The R^2 and RMSE values for the CNN model is calculated and is saved to the results for comparing with the other previous ML approaches.

```
# R^2 Value
R2_test.sla <- cor(prediction.test, as.matrix(y.test))^2; R2_test.sla

# Calculating the RMSE
prediction.test <- as.numeric(prediction.test)
y.test <- as.numeric(y.test[[1]])
rmse_sla <- sqrt(mean((prediction.test - y.test)^2))
```

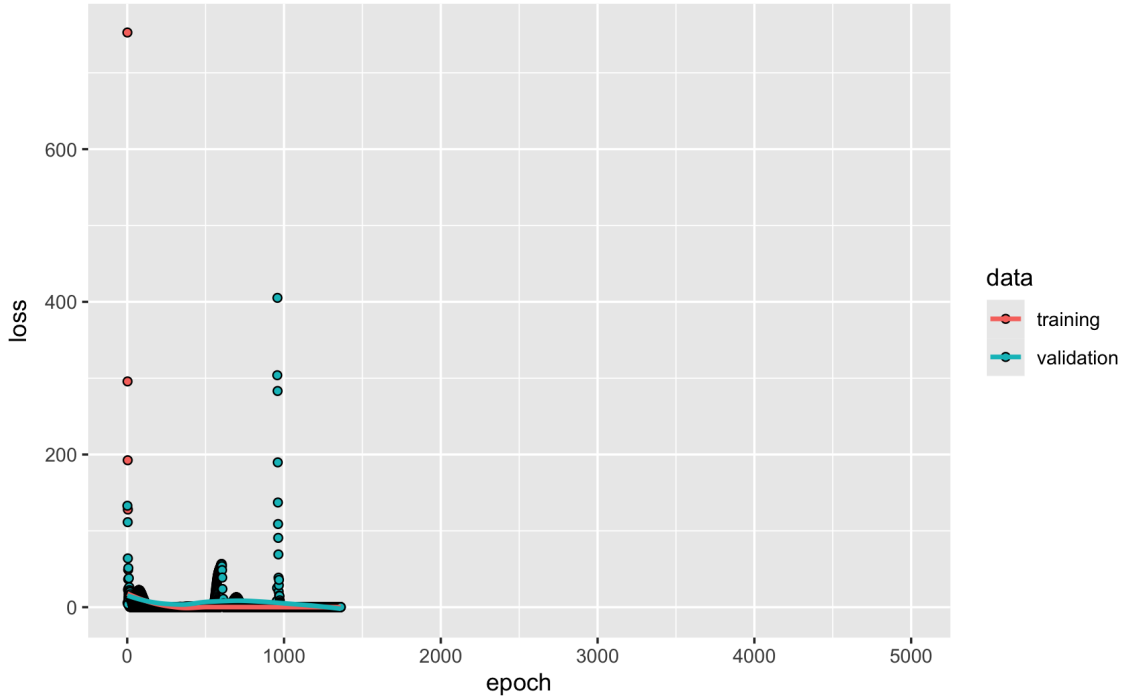

Figure 3.23: The CNN training model for SLA trait, illustrating the training and validation loss during training the model. The model's training was set to 5000 epoches and it stopped before reaching 1500 epoches.

Visualization

The visualization is done with the help of `ggplot2` and `plotly` packages. Figure 3.24 illustrates scatter plot created using the predicted from the CNN model and actual SLA values from this analysis.

Prediction of LDMC, C:N ratio, C and P content

Prediction of the rest of the leaf traits were also performed using the CNN model. The LDMC, Carbon content, Phosphorus content and Carbon to Nitrogen ratio is predicted using CNN. Each of these trait data were preprocessed similar to its analysis in the previous PLSR and RF model. It is then used to train the CNN model which uses the same CNN architecture for all traits. Since this is a recreation of the original script from the leaf trait study with few modification to save training time and computational cost, each trait has different values for parameters. For instance, the kernal size of LDMC is different from that of Phosphorus content and the number of neurons are also different in each analysis. The only change we did compared to the original version is to add an early call back function to reduce computational load during training and zero padding to make all data points available during training. A detailed version of the analysis with interactive plots were presented as vignette.

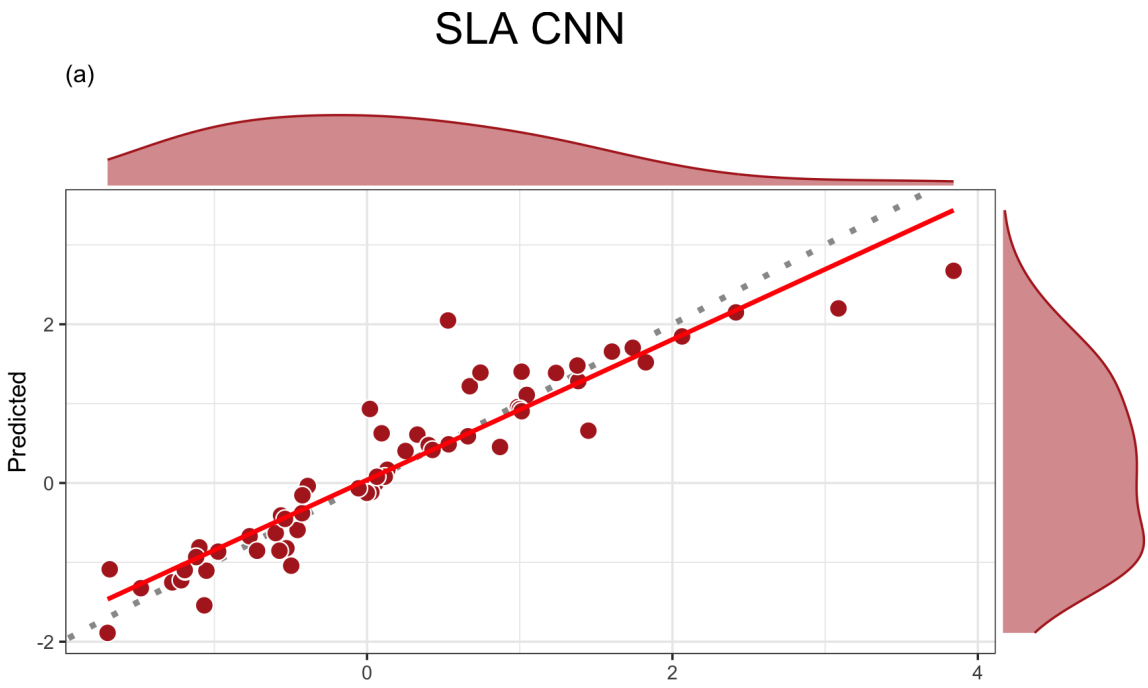

Figure 3.24: This plot illustrates the predictive ability of CNN for SLA trait

3.5 LCMS Feature Prediction From NIRS Data

Background of the study

The dataset utilized for the prediction of LCMS features from NIRS data originates from a large scale field experiment known as the Jena experiment. This biodiversity experiment named *LC-MS based plant metabolic profiles of thirteen grassland species grown in diverse neighbourhoods* has conducted in the city of Jena, Germany [14]. This study focuses on the influence of different ecological settings on secondary metabolites in plants.

Figure 3.25 illustrates the above ground plants from thirteen grassland species (seven grass and six herbs) which collected at four time points throughout the growing season, May, July, August and October in the year 2017. These plants were grown in plots with different diversity levels, ranging from one species per plot to eight species per plot. The LCMS metabolic profiling was performed on shoot extracts using an Ultra-Performance Liquid Chromatography coupled with an Electrospray Ionization Quadrupole Time-of-Flight Mass Spectrometer (UPLC-ESI-Qq-TOF-MS). This acquired raw data is then preprocessed by applying missing data imputation, validity check on the features and batch correction. On the other hand, NIRS data for the same samples were collected separately using the ASD Fieldspec 4 instrument.

3.5.1 LCMS feature prediction using PLSR

Helper functions

Two helper functions were created to make a SummarizedExperiment object from the NIRS and LCMS data. The function to make SummarizedExperiment object from NIRS data uses the source code of `read_summarizedexperiment_asd` function from "nearspectRa" package, which we created in the beginning of this project. The function was not used directly, as mod-

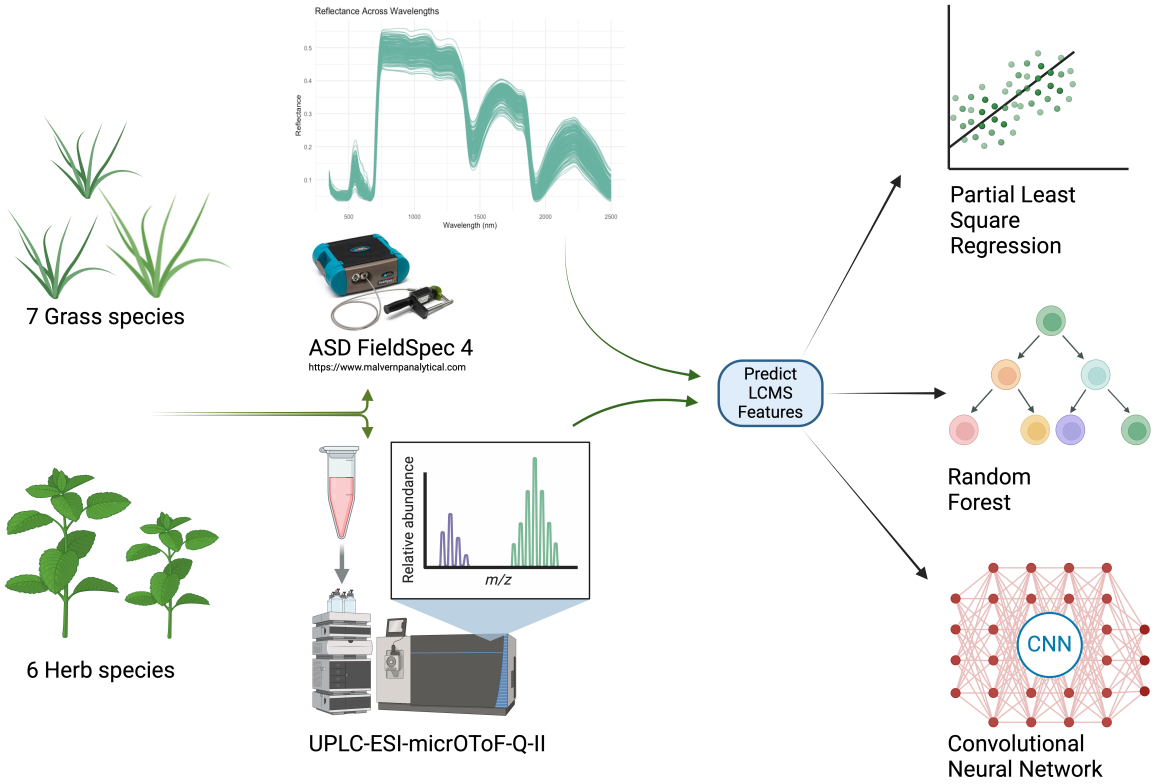

Figure 3.25: Schematic representation of the Jena experiment

ifications were required to extract additional metadata embedded in each sample name. Each sample had information about the season, replicate, species and campaign which were unique to the Jena experiment. The `read_summarizedexperiment_asd` function which takes the file path as an input and outputs a SummarizedExperiment object is called from another R script named "NIRS_summarizedExperiment_sue.R".

```
# source the SE function
source("/Users/methungeorge/Desktop/IPB/all_scripts/NIRS_
   summarizedExperiment_sue.R")
```

The sample names of NIRS data had a pattern of "101_2018_E_FESRUB_A064_b.asd" and this contained valuable metadata which is extracted using the following function after confirm it in https://www.ebi.ac.uk/metabolights/editor/MTBLS679/samples. The full version of the R script is provided as a supplement to this project.

```
# Function to extract the metadata from filename
    #. "101_2018_E_FESRUB_A064_b.asd."

    extract_metadata_from_filename <- function(file_base) {
      parts <- strsplit(file_base, "_")[[1]]
      return(data.frame(
        species = parts[4],
        replicate = parts[6],
        campaign = parts[2],
        season = parts[3]
      ))
    }
```

The LCMS data is converted into a SummarizedExperiment object and integrated into the R script for training the PLSR model. It uses `read.MAF` function from `metabolighteR` package to read the LCMS data stored in a ".tsv" file. From this file, the columns containing mass-to-charge ratio, retention time, and sample data were extracted to create the SummarizedExperiment, where each metabolite name is a combination of its mass-to-charge ratio and retention time. Similar to the NIRS data, the metadata is extracted from the sample names.

```
# source the SE function
source("/Users/methungeorge/Desktop/IPB/all_scripts/LCMS_
   summarizedExperiment.R")
```

Predicting LCMS features from NIRS data using PLSR

Both LCMS and NIRS data used in this analysis is received from the Jena experiment [14]. The LCMS and NIRS received from the study is taken from the same samples(510). However, due to variations in the number of replicates across different seasons, the total number of samples, including replicates, for NIRS data amounted to 1245. The assay data from the SummarizedExperiment objects, created using LC-MS and NIRS data through helper functions, is saved as response_data and predictor_data, respectively.

```
# Extract assay data
predictor_data <- assays(NIRS_se)$counts # NIRS is the predictor
response_data <- assays(LCMS_se)$intensities # LCMS is the response
```

The "response_data" (LCMS data) is a matrix of 5889 metabolites (row) and 510 samples (column). On the other hand, "predictor_data" (NIRS data) is a matrix of 2151 (row) features (wavelength) and 1245 samples (including the replicates). To standardize sample names across both matrices and retain only the samples present in both datasets and thereby filtering out replicates, both matrices were transposed and processed using a custom function to ensure consistent sample naming. The common samples were then selected from both matrices as part of the data preprocessing step and this reduce the size of both LCMS and NIRS data to 490 samples for each. Since this analysis focused on the accuracy of prediction in each ML model, it was necessary to filter out the metabolites in LCMS data containing fewer number of NA or zero values. As part of this process, a threshold of a maximum of 10 zeros was set, and metabolites containing more zeros than this threshold were eliminated. This process left us with a total number of 53 metabolites in this analysis. Figure 3.26 illustrates the dimensions of both the NIRS and LCMS data in this study."

Both predictor(NIRS) and response(LCMS) data are scaled and a data frame named "results" is initialized with columns to add the metabolite name, R^2, RMSE and training time from the following iterations. A for loop iterates over the 53 metabolite columns in the scaled response data matrix (490 samples $\times$ 53 metabolites). In each iteration, a single metabolite column is selected as the response variable (Y), while the predictor matrix (490 samples $\times$ 2151 features) remains unchanged. During each iteration, the data is split into training and test sets, and a PLSR model is trained for that specific metabolite and later the predictions are made using that PLSR model. The evaluation metrics (R^2, RMSE and training time) are calculated and appended to the "results" data frame, which is then saved as a CSV file for further analysis (figure 3.27). The model architecture of PLSR in predicting LCMS features from NIRS data

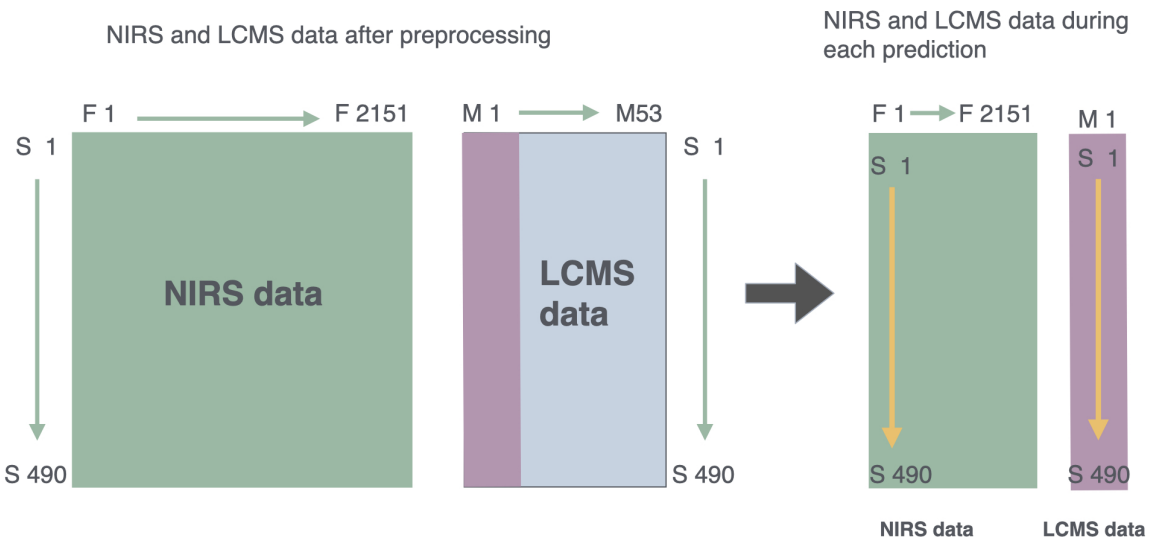

Figure 3.26: Schematic representation of the prediction of LCMS features from NIRS data with "F" representing features, "S" stands for samples and "M" for metabolites. During each iterations, a single metabolite column is selected (vector) and is then predicted using NIRS data matrix with each sample row of NIRS data(sample spectra) predicting the LCMS data corresponding to that sample.

is same as that of predicting leaf traits from NIRS data. Different kinds of plots are generated using `ggplot2` package to visualize the results.

3.5.2 LCMS feature prediction using RF

3.6 HPC runs

	Metabolite	R2	RMSE	TrainingTime
1	m/z_119.0858658_RT_845.0360081	0.12144953	0.9501881	20.48157
2	m/z_181.1222311_RT_854.3462146	0.08590772	1.0256871	18.04829
3	m/z_188.0706505_RT_145.5962756	0.41320463	0.7822594	17.29591
4	m/z_205.0793257_RT_351.1913522	0.03194651	0.9349084	17.21084
5	m/z_205.0970447_RT_145.2870487	0.40299602	0.7778582	19.96537
6	m/z_207.1377636_RT_256.1395654	0.45461987	0.7773245	17.73780

Figure 3.27: A example CSV output of PLSR model for predicting LCMS features from NIRS data. The first column contains the details of each metabolite as a combination of its retention time and mass to charge ratio. The rest of the columns contains the R^2, RMSE values and training time (in seconds) for each metabolite. A total of 53 metabolites were analysed and predicted in this manner using PLSR machine learning approach.

Chapter 4

Results and Discussion

This section of the thesis presents and discuss the results obtained from the analysis of PLSR, RF and CNN machine learning models. The performance of each of the model is compared and evaluated based on the coefficient of determination (R^2), Root Mean Squared Error (RMSE) and training time. The R^2 explains how well the model "fits" the data or in other words, the predictive accuracy of the model to unseen data. It ranges from 0 to 1 and higher value indicate a better fit for the model. The second parameter used is RMSE values and it measures the average difference between the predicted and actual values. This gives information on the average error rate of the ML model. A lower RMSE indicates less error for prediction and a value of 0 means the model perfectly predicts all values. The third parameter used in this analysis is the training time of the model. It is calculated both in minutes and seconds, depending on the model. Longer training time translates to higher computational cost and are therefore used as a metric to evaluate the model performance.

4.0.1 Comparison of results from leaf trait analysis

Table 4.1 present the results of predicting leaf traits from the NIRS data. Three machine learning models (PLSR, RF, CNN) were evaluated and compared using three metrics: the coefficient of determination (R^2), root mean squared error (RMSE) and models training time. These metrics give an overall assessment of the model performance.

Comparing predictive accuracy using R^2 value

The predictive accuracy of the model is evaluated by comparing the R^2 and RMSE values of the ML models. Table 4.1 illustrate the predictive accuracy of all three ML models on the five leaf traits. The boxplot (Figure4.1) exemplifies the distribution of R^2 values across the three ML models for predicting the five leaf traits. PLSR shows a higher median R^2 value (0.72) compared to the other two ML models, which are 0.54 and 0.59 for RF and CNN respectively. Even though CNN performs comparably to PLSR, especially for the SLA(0.88) and LDMC(0.85) traits, it shows slightly higher variability which is explained by the interquartile range (IQR), in other words the box of the plot. Random forest model on the other hand shows the lowest median R^2 values and the widest spread of the three ML models. This trend suggest a lower predictive ability of RF model compared to the other two in this group.

Figure 4.2 is a heatmap highlighting the accuracy of each ML model for predicting leaf traits from the spectral data. This shows that, certain traits such as SLA and LDMC have consistently high R^2 values across all models. This results also indicates that, the PLSR outperforms

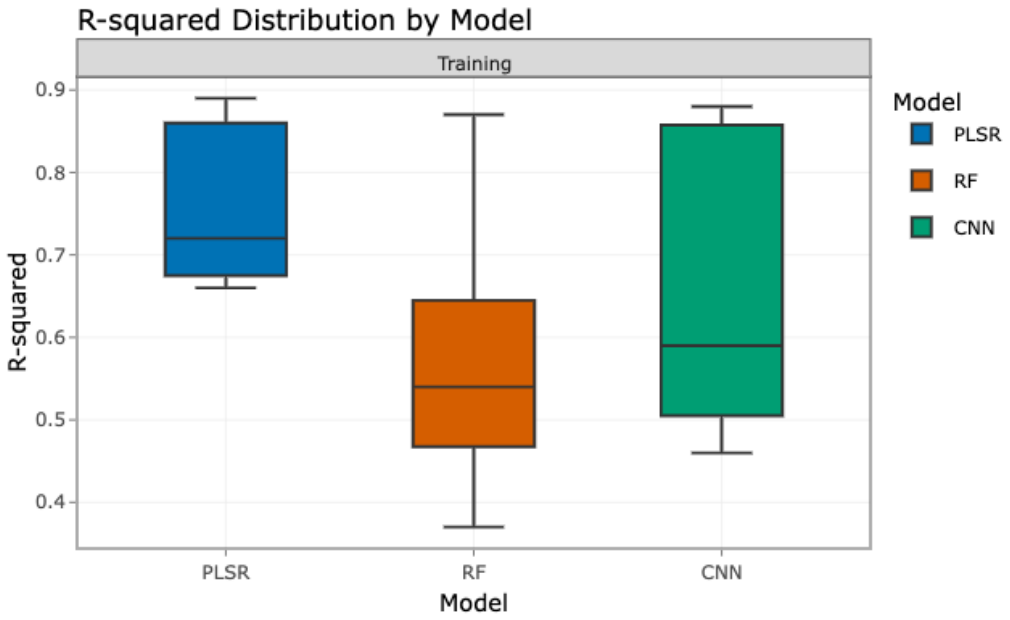

Figure 4.1: Distribution of R^2 values across all ML models in leaf trait analysis.

RF and CNN for most of the traits except for SLA in which all three models have a very similar R^2 values ranging from 0.88 to 0.89. The SLA is a trait which is closely linked to the leaf structure, thickness and water content [15]. This is well captured in reflectance data therefore SLA has a strong relation to spectral data when compared to other leaf traits such as carbon and phosphorus which has a lower spectral signals resulting a higher predictive accuracy in all three ML models. Specific leaf area is a morphological trait which makes it strongly correlated to spectral data there by making it easier to predict for all models. Another factor to consider is the number of missing values in the data, with phosphorus having 69 missing values whereas the rest of the traits has less than 4. In summary, when comparing the R^2 of the ML models across five leaf traits, it is evident that PLSR is the most accurate and reliable model for trait prediction. While CNN shows a competitive performance, it shows reduced predictive ability for few traits. Random forest demonstrated inconsistent results across all five traits with a higher R^2 for SLA(0.87) prediction and the lowest for the prediction of carbon content(0.37). This inconsistency observed in the Random Forest model suggests that it may not be the most suitable choice for this analysis.

Comparing predictive accuracy using RMSE value

Root Mean Squared Error (RMSE) is the second metric used in this study to evaluate the model's performance. It is the average magnitude of prediction errors between predicted and actual values. The boxplot (figure4.3) illustrates the comparison of RMSE values across the three ML models. Lower RMSE values points to a better model performance and it ranges from zero to infinity. In the case of PLSR, the median RMSE value is 0.50, which is smaller compared to that of RF(0.66) and CNN(0.60). The interquartile range (IQR) of PLSR is also small compared to that of CNN. This indicates the PLSR prediction are more consistent for all traits. RMSE values for RF model ranges from a minimum of 0.42 for SLA to maximum of 0.82 for carbon content with a median of 0.66. The CNN model has a median of 0.60 which is in between that of RF and PLSR. CNN has the lowest RMSE value for LDMC prediction but compared to PLSR the CNN is less consistent across all five traits.

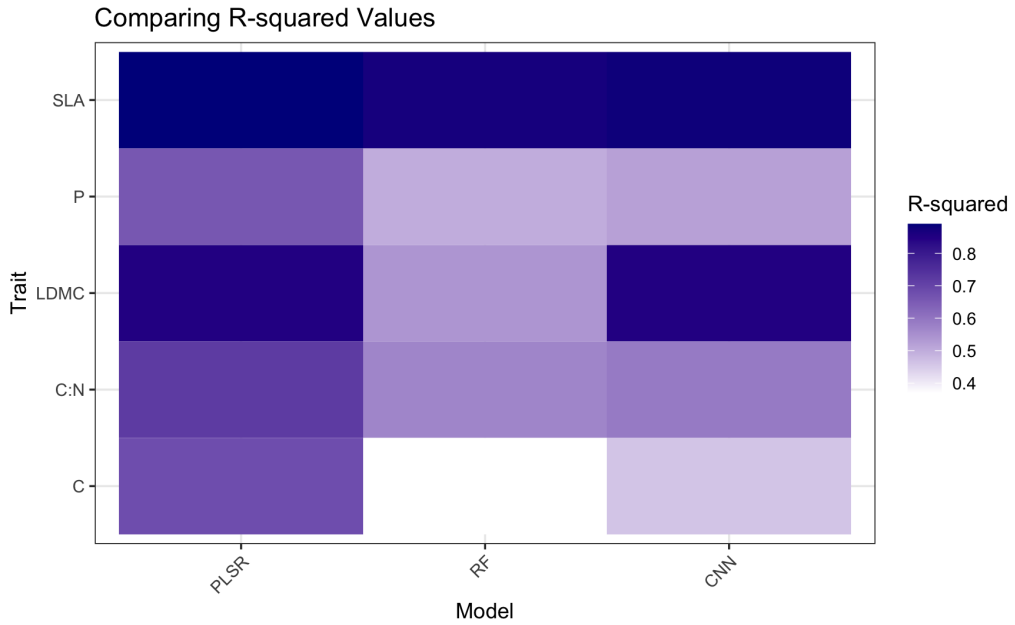

Figure 4.2: This heatmap comparing the R^2 values of five leaf traits across three machine learning models. The color gradient represents R^2 values, with darker shades indicating higher predictive accuracy.

The scatter plot (figure 4.4) explains the relationship between the R^2 and the RMSE for three ML models. It clearly explains the predictive accuracy of the models in this analysis by taking R^2 and RMSE into account. Even though CNN has the least RMSE with high R^2 for two traits, it failed to keep the trend going for the rest of the traits. PLSR at the same time showed a better R^2 values for almost all traits with lower RMSE. The RF performed the worst out of these three models with lower predictive accuracy and higher error rates. This leads to the conclusion that PLSR is the best performing ML model for predicting the leaf traits compared to CNN and RF.

Comparing Training time

The evaluation of a machine learning model is carried out by considering multiple parameters and is not limited to R^2 and RMSE, though they are important factors to consider. While these two factors helps to asses the predictive accuracy of a model, the practical applicability is largely dependent on its computational efficiency. In this study, we measured training time for each ML model across all five traits including extrapolation studies to analyze and compare the predictive accuracy of a model to its computational efficiency. A model with exceptional predictive ability is of limited practical use if it require excessive computational resources and longer training time.

As shown in the figure4.5, the heatmap provides a visual representation of the training time from the table4.1 (excluding extrapolation). It is evident that PLSR has the lowest training time for analyzing all five leaf traits. The average training time for PLSR is 13.94 seconds which is significantly faster compared to the average training times of 12.80 minutes for RF and 4.91 minutes for CNN. Random forest is the slowest model and it takes significantly longer time than PLSR and CNN. The fastest training time recorded for RF is 12.11 minutes for the LDMC trait which showed a R^2 value of 0.54 with a large error rate(RMSE=0.72). Both PLSR and CNN showed faster training times for predicting phosphorus content, with training times ranging

Table 4.1: Comparison of 3 ML models across data

Partial Least Square Regression (PLSR)			
Trait	R^2	**RMSE**	**Training time**
SLA	0.89	0.40	13.76 sec
LDMC	0.85	0.41	11.40 sec
C:N	0.72	0.50	15.71 sec
C	0.68	0.55	13.63 sec
P	0.66	0.64	15.19 sec
PLSR Extrapolation			
SLA (H 30%)	0.26	1.02	14.18 sec
SLA (L 30%)	0.27	0.41	12.91 sec
Random Forest (RF)			
Trait	R^2	**RMSE**	**Training time**
SLA	0.87	0.42	12.85 min
LDMC	0.54	0.72	12.11 min
C:N	0.57	0.63	12.60 min
C	0.37	0.82	13.17 min
P	0.50	0.66	15.25 min
RF Extrapolation			
SLA (H 30%)	0.05	1.29	11.78 min
SLA (L 30%)	0.08	0.74	11.98 min
Convolutional Neural Network (CNN)			
Trait	R^2	**RMSE**	**Training time**
SLA	0.88	0.39	8.84 min
LDMC	0.85	0.35	6.01 min
C:N	0.59	0.60	4.02 min
C	0.46	0.70	5.07 min
P	0.52	0.70	30.78 sec
CNN Extrapolation			
SLA (H 30%)	0.22	0.66	6.46 min
SLA (L 30%)	0.31	0.65	3.20 min

from 15 to 31 seconds. Surprisingly, this is a lower training time for CNN, considering that the training times for other traits range from 4 to 9 minutes for each trait. When comparing the predictive ability of CNN across all five traits, phosphorus content had the second lowest R^2 value (0.52) and the highest error rate (0.70). CNN provides a balance between RF and PLSR models with an average training time of 4.91 minutes for predicting the leaf traits from NIRS data. It is much faster than RF, which has an average training time of 12.80 minutes and slower when compared to PLSR. Even the slowest training time recorded for CNN(8.84 minutes) is significantly lower than the fastest(12.11 for LDMC) RF model. From this comparison it is obvious that, PLSR is the best choice for rapid analysis and RF is computationally expensive.

In summary, PLSR demonstrated the best performance in all three traits considered in this

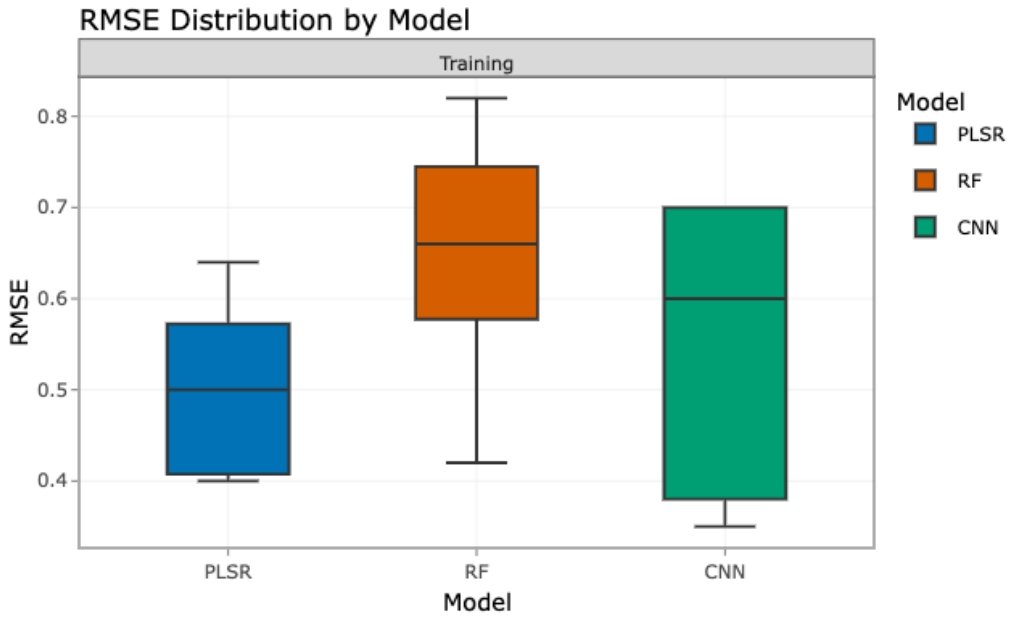

Figure 4.3: Distribution of RMSE values across all ML models in leaf trait analysis.

study. It has the highest predictive accuracy among these three ML models and least RMSE values which translate to minimum error rate for each prediction. The range of error rate is also significantly lower than PLSR and RF, which further highlight the predictive accuracy of the PLSR model. It recorded the least training time for all five traits in this study, with the fastest is 11.40 seconds for LDMC and slowest is 15.09 seconds for phosphorus content. The outstanding performance of PLSR for all three factors- R^2, RMSE and training time proves its efficiency and accuracy in predicting leaf traits from NIRS data. CNN is the second best performing ML model in this analysis. It even performed better than PLSR in the analysis of trait such as LDMC, where the CNN had an error rate of 0.35 and PLSR had 0.41 with same R^2 value(0.85). A noticeable difference is observed in the training time, where PLSR completes the training in seconds, while CNN takes several minutes to train the model. Random Forest is the least favored in this analysis due to its lower predictive ability and longer training time. Even though it had a R^2 values similar to PLSR and CNN, the error rate was significantly larger of all these ML models. This, combined with its longer training time, makes Random Forest the least favored model for predicting leaf traits from NIRS data.

Optimization of CNN and its influence in predictive accuracy and training time

The base CNN architecture we received from the leaf trait study[15] is perfectly recreated and later used to optimize for better results that suits the objective of this analysis. The change is most noticeable in the training time. While the original CNN-based script had a training time of over 30 minutes for predicting SLA, optimizing the model by adding an early callback function from the Keras package helped reduce the training time to one third (8.84 minutes) of its original duration for the same trait.

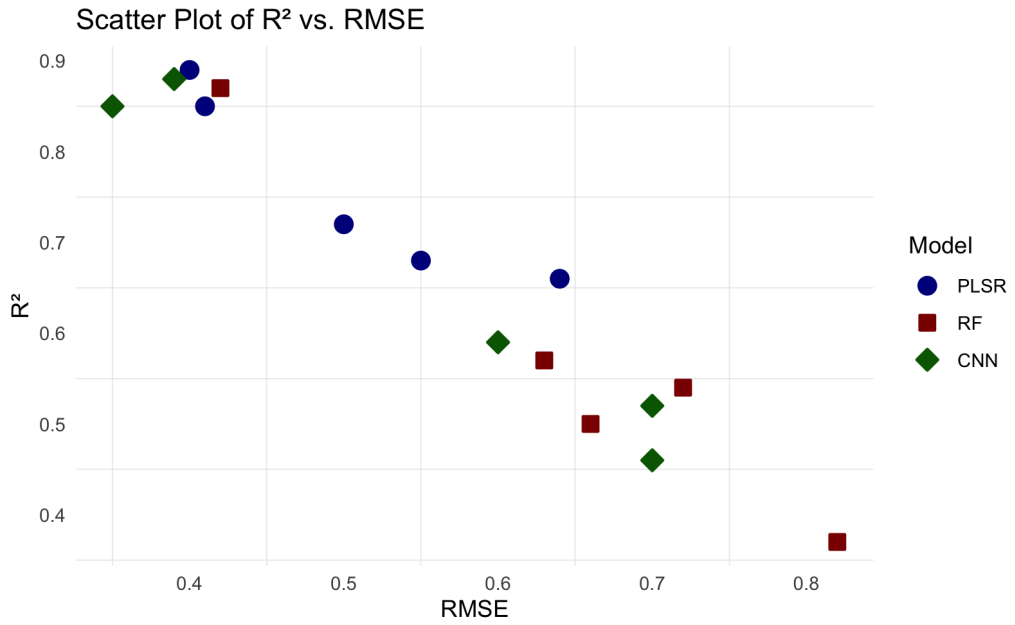

Figure 4.4: Scatter plot comparing the R^2 and RMSE of all three ML models

Extrapolation study on ML models

4.1 Data charecterestics

histogram, spectra

4.2 Baseline Machine Learning Models Pablo

PLS, RF, CNN

4.2.1 Variable importance

4.3 Variations in Baseline systems

4.3.1 modifying the Test and Training split

4.3.2 input data length

4.4 Sues

4.5 Appendix

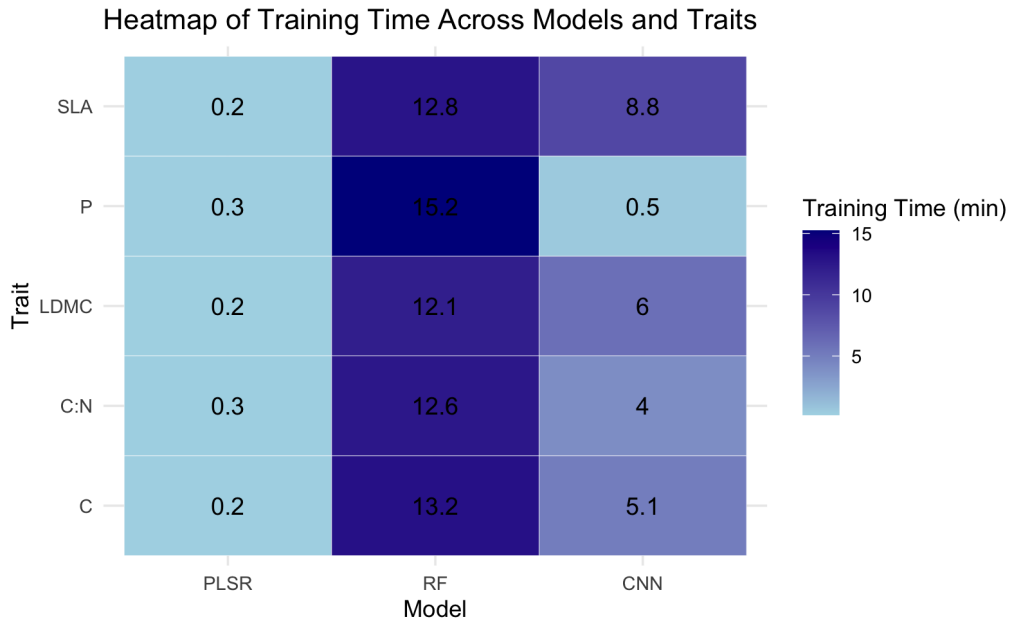

Figure 4.5: This heatmap visualizes the training time (in minutes) required for different machine learning models (PLSR, RF, CNN) across various plant traits. Darker shades indicate longer training durations.

Chapter 5

Reference

1. Pieruschka R, Schurr U. Plant Phenotyping: Past, Present, and Future. Plant Phenomics. 2019 Mar 26;2019:7507131. doi: 10.34133/2019/7507131. PMID: 33313536; PMCID: PMC7718630.

2. Pulok K. Mukherjee, Quality control and evaluation of herbal drugs, Evaluating natural products and traditional medicine. 2019, doi:10.1016/C2016-0-042328, ISBN:978-0-12-813374-3.

3. Nizamani, M. M., Zhang, Q., Muhae-Ud-Din, G., & Wang, Y. (2023). High-throughput sequencing in plant disease management: A comprehensive review of benefits, challenges, and future perspectives. Phytopathology Research, 5(44). https://doi.org/10.1186/s42483-023-00215-7.

4. Lane, H. M., & Murray, S. C. (2021). High throughput can produce better decisions than high accuracy when phenotyping plant populations. Crop Science, 61(3), 1473–1484. https://doi.org/10.1002/csc2.20514.

5. Zhang, N., Zhou, X., Kang, M., Hu, B.-G., Heuvelink, E., & Marcelis, L. F. M. (2023). Machine learning versus crop growth models: An ally, not a rival. Journal of Experimental Botany, 74(4), 1259–1276. https://doi.org/10.1093/jxb/erac517.

6. Zhu H. Big Data and Artificial Intelligence Modeling for Drug Discovery. Annu Rev Pharmacol Toxicol. 2020 Jan 6;60:573-589. doi: 10.1146/annurev-pharmtox-010919-023324. Epub 2019 Sep 13. PMID: 31518513; PMCID: PMC7010403.

7. Kelsey Chetnik, Elisa Benedetti, Daniel P Gomari, Annalise Schweickart, Richa Batra, Mustafa Buyukozkan, Zeyu Wang, Matthias Arnold, Jonas Zierer, Karsten Suhre, Jan Krumsiek, maplet: an extensible R toolbox for modular and reproducible metabolomics pipelines, Bioinformatics, Volume 38, Issue 4, February 2022, Pages 1168–1170, https://doi.org/10.1093/bioinformatic

8. Peng, Roger D. R programming for data science. Victoria, BC, Canada: Leanpub, 2016.

9. www.bioconductor.org

10. Kumar, R., Bohra, A., Pandey, A. K., Pandey, M. K., & Kumar, A. (2017). Metabolomics for plant improvement: Status and prospects. Frontiers in Plant Science, 8, 1302. https://doi.org/10.3389/f

11. González-Domínguez, R., García-Barrera, T., & Gómez-Ariza, J. L. (2014). Metabolite profiling for the identification of altered metabolic pathways in Alzheimer's disease. Journal of Pharmaceutical and Biomedical Analysis, 107, 75–81. https://doi.org/10.1016/j.jpba.2014.10.010.

12. Liebal, U. W., Phan, A. N., Sudhakar, M., Raman, K., & Blank, L. M. (2020). Machine learning applications for mass spectrometry-based metabolomics. Metabolites, 10(6), 243. https://doi.org/10.3390/metabo10060243.

13. Vaillant, A., Beurier, G., Cornet, D., Rouan, L., Vile, D., Violle, C., & Vasseur, F. (2024). NIRSpredict: a platform for predicting plant traits from near infra-red spectroscopy. BMC Plant Biology, 24(1), 1100. https://link.springer.com/article/10.1186/s12870-024-05776-0.

14. Marr S, Hageman JA, Wehrens R, van Dam NM, Bruelheide H, Neumann S. LC-MS based plant metabolic profiles of thirteen grassland species grown in diverse neighbourhoods. Sci Data. 2021 Feb 9;8(1):52. doi: 10.1038/s41597-021-00836-8. PMID: 33563993; PMCID: PMC7873126.

15. Sánchez-Bermejo, P. C., Monjau, T., Goldmann, K., Ferlian, O., Eisenhauer, N., Bruelheide, H., Ma, Z., & Haider, S. (2024). Tree and mycorrhizal fungal diversity drive intraspecific and intraindividual trait variation in temperate forests: Evidence from a tree diversity experiment. Functional Ecology. https://doi.org/10.1111/1365-2435.14549.

16. Renner, I.E., Fritz, V.A. Using Near-infrared reflectance spectroscopy (NIRS) to predict glucobrassicin concentrations in cabbage and brussels sprout leaf tissue. Plant Methods16, 136 (2020). https://doi.org/10.1186/s13007-020-00681-7.

17. R. Schubert et al., "A White-Box Workflow for the Prediction of Food Content From Near-Infrared Data Based on Fourier-Transformation," 2023 13th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS), Athens, Greece, 2023, pp. 1-5, doi: 10.1109/WHISPERS61460.2023.10430694.

18. Choi RY, Coyner AS, Kalpathy-Cramer J, Chiang MF, Campbell JP. Introduction to Machine Learning, Neural Networks, and Deep Learning. Transl Vis Sci Technol. 2020 Feb 27;9(2):14. doi: 10.1167/tvst.9.2.14. PMID: 32704420; PMCID: PMC7347027.

19. Pichler, M., & Hartig, F. (2022). Machine learning and deep learning—A review for ecologists. Methods in Ecology and Evolution, 13(9), 1984–2000. https://doi.org/10.1111/2041-210X.14061.

20. Ji, F., Li, F., Hao, D., Shiklomanov, A. N., Yang, X., Townsend, P. A., Dashti, H., Nakaji, T., Kovach, K. R., Liu, H., Luo, M., & Chen, M. (2024). Unveiling the transferability of PLSR models for leaf trait estimation: Lessons from a comprehensive analysis with a novel global dataset. New Phytologist. https://doi.org/10.1111/nph.19807.

21. Georgios Kourounis, Ali Ahmed Elmahmudi, Brian Thomson, James Hunter, Hassan Ugail, Colin Wilson, Computer image analysis with artificial intelligence: a practical introduction to convolutional neural networks for medical professionals, Postgraduate Medical Journal, Volume 99, Issue 1178, December 2023, Pages 1287–1294, https://doi.org/10.1093/postmj/qgad095.

22. Liu S, Cheng H, Ashraf J, Zhang Y, Wang Q, Lv L, He M, Song G, Zuo D. Interpretation

of convolutional neural networks reveals crucial sequence features involving in transcription during fiber development. BMC Bioinformatics. 2022 Mar 15;23(1):91. doi: 10.1186/s12859-022-04619-9. PMID: 35291940; PMCID: PMC8922751.

23. van Dijk ADJ, Kootstra G, Kruijer W, de Ridder D. Machine learning in plant science and plant breeding. iScience. 2020 Dec 5;24(1):101890. doi: 10.1016/j.isci.2020.101890. PMID: 33364579; PMCID: PMC7750553.

24. Deo, R. C. (2015). Machine Learning in Medicine. Circulation, 132(20), 1920–1930. https://doi.org/10.1161/CIRCULATIONAHA.115.001593.

25. Angela C Burnett, Jeremiah Anderson, Kenneth J Davidson, Kim S Ely, Julien Lamour, Qianyu Li, Bailey D Morrison, Dedi Yang, Alistair Rogers, Shawn P Serbin, A best-practice guide to predicting plant traits from leaf-level hyperspectral data using partial least squares regression, Journal of Experimental Botany, Volume 72, Issue 18, 30 September 2021, Pages 6175–6189, https://doi.org/10.1093/jxb/erab295.

26. Xin, Y., Sheng, J., Miao, M., Wang, L., Yang, Z., & Huang, H. (2022). A review of imaging genetics in Alzheimer's disease. Journal of Clinical Neuroscience, 93, 1-9. https://doi.org/10.1016/j.jocn.20

27. Dormann, C. F., Elith, J., Bacher, S., & Buchmann, C. M. (2013). Collinearity: A review of methods to deal with it and a simulation study evaluating their performance. Ecography, 36(1), 27-46. https://doi.org/10.1111/j.1600-0587.2012.07348.x.

28. Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. Chemometrics and Intelligent Laboratory Systems, 2(1-3), 37-52. https://doi.org/10.1016/0169-7439(87)80084-9.

29. Couronné, R., Probst, P. & Boulesteix, AL. Random forest versus logistic regression: a large-scale benchmark experiment. BMC Bioinformatics 19, 270 (2018). https://doi.org/10.1186/s12859-018-2264-5.

30. Schonlau, M., & Zou, R. Y. (2020). The random forest algorithm for statistical learning. The Stata Journal, 20(1), 3-29. DOI: 10.1177/1536867X20909688.

31. Liu X, Wu D, Zewdie GK, et al. Using machine learning to estimate atmospheric Ambrosia pollen concentrations in Tulsa, OK. Environmental Health Insights. 2017;11. doi:10.1177/117863021769939

32. Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9.

33. Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021). https://doi.org/10.1186/s40 021-00444-8.

34. Reyad, M., Sarhan, A. & Arafa, M. A modified Adam algorithm for deep neural network optimization. Neural Comput & Applic 35, 17095–17112 (2023). https://doi.org/10.1007/s00521-023-08568-z.

35. Zhang W, Kasun LC, Wang QJ, Zheng Y, Lin Z. A Review of Machine Learning for Near-Infrared Spectroscopy. Sensors. 2022; 22(24):9764. https://doi.org/10.3390/s22249764.

36. Sohn SI, Pandian S, Oh YJ, Zaukuu JZ, Kang HJ, Ryu TH, Cho WS, Cho YS, Shin EK, Cho BK. An Overview of Near Infrared Spectroscopy and Its Applications in the Detection of Genetically Modified Organisms. Int J Mol Sci. 2021 Sep 14;22(18):9940. doi: 10.3390/ijms22189940. PMID: 34576101; PMCID: PMC8469702.

37. Ihaka, R. (2017). The r project: A brief history and thoughts about the future. Univ. Auckl, 4, 22. https://www.stat.auckland.ac.nz/ ihaka/downloads/Otago.pdf.

38. Ndaba, S. (2023). A Review of the Use of R Programming for Data Science Research in Botswana. International Journal of Database Management Systems, 15(1), 16. https://doi.org/10.5121/ijd

39. Venables, W. N., & Smith, D. M. (2003). An introduction to R: notes on R: a programming environment for data analysis and graphics, version 1.9. 1. CRID: 1130282269082172544, ISBN: 0954161742, https://cir.nii.ac.jp/crid/1130282269082172544.

40. www.malvernpanalytical.com

41. www.spectravista.com

42. Lantz, B. (2019). Machine learning with R: expert techniques for predictive modeling. Packt publishing ltd.

43. Allwood, J. W., & Goodacre, R. (2010). An introduction to liquid chromatography–mass spectrometry instrumentation applied in plant metabolomic analyses. Phytochemical Analysis: An International Journal of Plant Chemical and Biochemical Techniques, 21(1), 33-47. https://doi.org/10.1002/pca.1187.

44. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010

45. Chomal, V. S., & Saini, J. R. (2015). Software project documentation-an essence of software development. International Journal of Advanced Networking and Applications, 6(6), 2563.

46. Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A quick introduction to version control with Git and GitHub. PLoS computational biology, 12(1), https://doi.org/10.1371/journal.pcbi.100466

47. https://github.com/features/actions

48. Chi, K., Lin, J., Chen, M., Chen, J., Chen, Y., & Pan, T. (2023). Changeable moving window-standard normal variable transformation for visible-NIR spectroscopic analyses. Spectrochimica Acta Part A: Molecular and Biomolecular Spectroscopy, 123726. https://doi.org/10.1016/j.saa.2

49. Proß, T., Haider, S., Auge, H., & Bruelheide, H. (2023). Leaf trait variation within individuals mediates the relationship between tree species richness and productivity. Oikos, 2024, e10255. https://doi.org/10.1111/oik.10255.

50. Kuhn Max (2019). The caret package. https://topepo.github.io/caret/

51. Wang, Y., Xiao, Z., & Cao, G. (2022). A convolutional neural network method based on Adam optimizer with power-exponential learning rate for bearing fault diagnosis. Journal of Vibroengineering, 24(4). https://doi.org/10.21595/jve.2022.22271

52. https://en.wikipedia.org/wiki/Coefficient_of_determination

53. Chicco, D., Warrens, M. J., & Jurman, G. (2021). The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. Peerj computer science, 7, e623.

54. Kuhn, M. (2008). Building Predictive Models in R Using the caret Package. Journal of Statistical Software, 28(5). doi:10.18637/jss.v028.i05.

55. Morgan, M., Obenchain, V., Hester, J., & Pagès, H. (2023). SummarizedExperiment: Coordinating Experimental Assays, Samples, and Regions of Interest. https://www.bioconductor.org/packages/devel/bioc/vignettes/SummarizedExperiment/inst/doc/SummarizedExperiment.html