Research Project:
Retrieval of plant biophysical and biochemical variables from remote sensing data using a hybrid machine learning method

Trier University

Zavud Baghirov

Environmental Sciences

University of Trier

Summer Semester 2021

Abstract

This will be the abstract at the end [TO BE UPDATED]

Contents

List of Figures

List of Tables

List of Abbreviations

3D . . . . . Three-dimensional

INFORM . . . Invertable Forest Reflectance Model

RTM . . . . . Radiative Transfer Model

SAIL . . . . . Scattering by Arbitrary Inclined Leaves

PROSAIL . . The combination of PROSPECT and SAIL models

FLIM . . . . . Forest Light Interaction Model

LAI . . . . . Leaf Area Index

MLRA . . . . Machine Learning Regression Algorithms

ML . . . . . Machine Learning

DT . . . . . Decision Trees

ANN . . . . . Artificial Neural Networks

KBMLRM . . Kernel-Based Machine Learning Regression Methods

RF . . . . . Random Forest

RFR . . . . . Random Forest Regression

LUT . . . . . Look-Up-Table

NN . . . . . Neural Networks

SVR . . . . . Support Vector Regression

SVM . . . . . Support Vector Machines

GPR . . . . . Gaussian Process Regression

GP . . . . . Gaussian Process

VI . . . . . Vegetation Index

DR . . . . . Dimensionality Reduction

WT . . . . . Wavelet Tranform

PCA . . . . . Principal Component Analysis

AL . . . . . Active Learning

NIR . . . . . Near Infrared

SWIR . . . . . Short Wave Infrared

PC . . . . . Principal Component

DNN . . . . . Deep Neural Networks

MSE . . . . . Mean Squared Error

SGD . . . . . Stochastic Gradient Descent

MAE . . . . . Mean Absolute Error

ReLU . . . . . Rectified Linear Activation Unit

1 Methods

This section explains the methods used in this research.

1.1 Local sensitivity analysis

Local sensitivity analysis was performed to assess the effect of each of the main 6 plant biochemical and biophysical variables on the PRISMA image bands. In the local sensitivity analysis simulation is performed by keeping all the variables constant at their determined fixed or default values except the parameter of interest. This way the effect of a specific parameter on the simulated spectra can be assessed. In this research the plant parameters C_{ab}, C_w, C_m, LAI_s, CD and SD were varied each 15 times (Table (1.1)), while keeping the rest of the variables at their default values (Table (1.2)). The default and varied values were chosen based on the literature (e.g. Darvishzadeh et al. (2019); Laurent et al. (2011); Schlerf and Atzberger (2012)) where similar RTM method used to simulate reflectance for Spruce trees.

Table 1.1 shows the 6 parameters that were varied, their units, minimum and maximum values. Each parameter was varied 15 times, meaning 15 different spectra were simulated for each variable.

Table 1.1: INFORM Parameters varied in local sensitivity analysis (each parameter were varied 15 times)

Parameter	Abbrev.	Unit	Min	Max
Chlorophyll content	C_{ab}	$\frac{\mu g}{cm^2}$	20	60
Equivalent water thickness	C_w	$\frac{g}{cm^2}$	0.0035	0.035
Leaf dry matter content	C_m	$\frac{g}{cm^2}$	0.008	0.03
Leaf area index (single)	LAI_s	$\frac{m^2}{m^2}$	0	7
Stem density	SD	ha^{-1}	200	5000
Crown diameter	CD	m	1.5	8.5

Table 1.2 shows the determined default values for each INFORM parameter that were kept during the sensitivity simulation while one of the parameter was varied (Table 1.1).

Table 1.2: INFORM Parameters that were kept constant while one parameter was varied at a time

Parameter	Abbr	Unit	Value
Leaf structure parameter	N	$-$	3
Chlorophyll content	C_{ab}	$\frac{\mu g}{cm^2}$	40
Leaf cartenoid content	C_{ar}	$\frac{\mu g}{cm^2}$	8
Brown Pigment Content	C_{brown}	$-$	0.001
Equivalent water thickness	C_w	$\frac{g}{cm^2}$	0.0117
Leaf dry matter content	C_m	$\frac{g}{cm^2}$	0.03
Average leaf inclination angle	$ALIA$	$^\circ$	65
Leaf area index (single)	LAI_s	$\frac{m^2}{m^2}$	6
Leaf area index (understorey)	LAI_u	$\frac{m^2}{m^2}$	0.5
Hot spot parameter	Hot	$\frac{m}{m}$	0.02
Solar zenith angle	tts	$^\circ$	45.43
Observer zenith angle	tto	$^\circ$	0
Sun-sensor azimuth angle	psi	$^\circ$	181.41
Soil brightness	α_{soil}	$-$	0.5
Stem density	SD	ha^{-1}	700
Crown diameter	CD	m	5
Mean Height	H	m	20
Fraction of diffuse incoming	$skyl$	$-$	0.1
Soil reflectance spectrum	B_g	$-$	default

Solar zenith angle and *Sun-sensor azimuth angle* were calculated based on the PRISMA image acquisition parameters (date, lat/long etc.) using the *solar position calculator* at `https://www.esrl.noaa.gov/gmd/grad/solcalc/azel.html`.

RTM models PROSPECT5, 4SAIL and FLIM were coupled (INFORM) in order to simulate canopy reflectance. Simulations were carried out using the *ccrtm* package (Visser, 2021) in *R* (R Core Team, 2021). The default soil spectra provided by the the *ccrtm* package (Visser, 2021) was used for the simulations. Spectral resampling was performed in order to resample the INFORM output spectra (1nm

resolution) into PRISMA image bands. For spectral resampling the *R* package *hsdar* (Lehnert et al., 2019) was utilized.

1.2 RTM simulation (INFORM)

PROSPECT5, 4SAIL and FLIM RTM models were coupled (INFORM) to simulate forest canopy reflectance based on different values of plant biophysical and biochemical parameters. The 6 parameters that were mentioned in the previous chapter were varied and spectra was simulated based on each combination of these variables. The number of combinations increase exponentially, which in turn requires increased computational power. Therefore, the trade-off must be taken into account between computational power or time and accurate simulation.

Different authors suggest different number of LUT size for RTM simulation. For example, Danner et al. (2021) mention that LUT size of minimum 50,000 is recommended. Ali et al. (2020) and Darvishzadeh et al. (2019) created a LUT size of 100,000 and 500,000 respectively.

In this research, LUT size of 316,800 was created based on each combination of different plant biophysical and biochemical parameters. The range of the varied parameters and parameters that were kept constant were determined based on the suggestions of the studies that were mentioned in the previous chapter. These studies used similar methods to simulate canopy reflectance for mainly Spruce forests/trees.

Table 1.3 shows the variables that were used to simulate forest canopy parameters. Table 1.3 also contains information about the range of the values and how many times each parameter was varied.

Table 1.3: Range of full input parameters that were used to create a LUT size of 316800

Parameter	Abbr	Unit	Min	Max	Steps
Leaf structure parameter	N	–	3	3	–
Chlorophyll content	C_{ab}	$\frac{\mu g}{cm^2}$	20	60	15
Leaf cartenoid content	C_{ar}	$\frac{\mu g}{cm^2}$	8	8	–
Brown Pigment Content	C_{brown}	–	0.001	0.001	–
Equivalent water thickness	C_w	$\frac{g}{cm^2}$	0.0035	0.035	10
Leaf dry matter content	C_m	$\frac{g}{cm^2}$	0.008	0.03	11
Average leaf inclination angle	$ALIA$	$^\circ$	65	65	–
Leaf area index (single)	LAI_s	$\frac{m^2}{m^2}$	0	6.5	16
Leaf area index (understorey)	LAI_u	$\frac{m^2}{m^2}$	0.5	0.5	–
Hot spot parameter	Hot	$\frac{m}{m}$	0.02	0.02	–
Solar zenith angle	tts	$^\circ$	45.43	45.43	–
Observer zenith angle	tto	$^\circ$	0	0	–
Sun-sensor azimuth angle	psi	$^\circ$	181.41	181.41	–
Soil brightness	α_{soil}	–	0.5	0.5	–
Stem density	SD	ha^{-1}	200	5000	4
Crown diameter	CD	m	1.5	8.5	3
Mean Height	H	m	20	20	–
Fraction of diffuse radiation	$skyl$	–	0.1	0.1	–
Soil reflectance spectrum	B_g	–	default	default	–

All simulations were performed using the library *ccrtm* (Visser, 2021) in *R* programming language (R Core Team, 2021) using the most recent version 4.1.0. Generating a LUT size of 316,800 is an expensive process from a computational standpoint (depending on how much computer resources and time are available this might change). Also, all simulations are independent of each other, meaning simulation of one spectra has no effect on the other, as every simulated spectra is simulated based on a different combination of parameters. These two factors make the generation of such a large LUT good candidate for parallel computation. Therefore, the software packages *doParallel* (Corporation and Weston, 2020) and *foreach* (Microsoft and Weston, 2020) were utilized for parallel computation (using all the available cores) in *R* programming language (R Core Team, 2021). This significantly reduced the computational time. All of the simulations were computed on a Lenovo Thinkpad E480 running under Windows 10 operating system with a

processor Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 2001 Mhz, 4 Core(s), 8 logical processor(s).

1.3 Spectral resampling

The output of INFORM simulations have 1nm spectral resolution within the range of 400nm-2500nm and needs to be spectrally resampled to PRISMA image bands. In this research, the spectral response function of the PRISMA image was used. Band center wavelengths and full width half maximum values were extracted from the PRISMA image metadata and used for spectral resampling. For spectral resampling, the *R* package *hsdar* (Lehnert et al., 2019) was utilized.

1.4 Statistics of simulated data and PRISMA image

Statistical information such as standard deviation and mean were calculated for the simulated (and resampled to PRISMA bands) data and all the pixels of the PRISMA image within the study area. Pixels that are out of the study area boundary were masked out. Then, average spectra in the LUT (synthetic database) and PRISMA image (only study area) were compared to each other. LUT contains 316,800 simulated spectra, the number of pixels within the study area in the PRISMA image is only 95517. Statistical information were extracted using the libraries in the *tidyverse* package (Wickham et al., 2019) and the plots for visualization were produced using *ggplot2* (Wickham, 2016).

1.5 Gaussian noise

Simulated reflectance data usually do not contain any noise. This is, however, not the case with remote sensing data as they are commonly found to contain various types of noise (Rivera-Caicedo et al., 2017). In this study, 3% Gaussian noise was added to each simulated spectra in the LUT in order to make the simulated data

more similar to the real remote sensing data. In order to assess the effect of adding 3% Gaussian noise to the simulated data, one spectra from the LUT and one pixel from the PRISMA image were randomly picked and plotted.

1.6 Defining training, validation and testing sets

The data in the LUT was divided into training, validation and testing sets. Model building and training will be done using only the training set. Validation set will be used to validate the model (e.g. assessing the impact of different hyper-parameters) and the performance of the final model will be tested using the testing set. This step is important because it will allow us to monitor whether the model can generalize to the data (e.g. testing set) it was not trained on.

First, the full data set was shuffled and about 20% of the data was randomly sampled and assigned to validation and testing sets (10% validation, 10% testing sets). Random sampling ensures that there is no any pattern contained in any of the divided data sets.

1.7 Data processing

First, the simulated canopy reflectance in the training data set was normalized and standardized using the Equation (1.1):

$$Band_{n_{scaled}} = \frac{Band_n - \mu_{Band_n}}{\sigma_{Band_n}} \tag{1.1}$$

Here $Band_n$ refers to the reflectance values in the nth simulated band and μ_{Band_n} and σ_{Band_n} are mean and standard deviation of the reflectance values in the nth simulated band. $Band_{n_{scaled}}$ is a transformed version of $Band_n$ that has a mean of 0 and standard deviation of 1. This step ensures that all simulated bands have the same mean and standard deviation.

Data normalization and standardization were only performed using training data set. Mean and standard deviation of the training set were then used to transform the validation and testing data sets.

1.8 Principal Component Analysis (PCA)

Hyperspectral remote sensing data can contain many highly correlated bands. Dimensionality reduction techniques can be efficiently used to reduce the dimensions of hyperspectral remote sensing data. Benefits of reducing the dimensions of simulated data in plant biophysical variable retrieval studies have been demonstrated (Danner et al., 2021; Rivera-Caicedo et al., 2017). In this study, one of the most commonly used DR technique Principal Component Analysis (PCA) was performed. In general, PCA tries to capture as much variation as possible with smaller number variables compared to the original data. PCA produces new variables called Principal Components and each Principal Component (PC) contains certain amount of variation available in the original data. Typically first PC contains the most variation, the second PC contains the second most variation and so on (Bro and Smilde, 2014).

Like in the processing step, PCA was only applied to the training data and the PCA result in the training data was used to transform the validation and testing sets. Cumulative sum of the variations the PCs contain was calculated in order to assess the proportion of the variation that can be explained with fewer variables than the original data (LUT). PCA and data processing performed using the package *recipes* (Kuhn and Wickham, 2021) in *tidymodels* (Kuhn and Wickham, 2020).

1.9 Artificial Neural Networks (ANN)

Considering the fact that we have relatively large number of training examples, non-linear relationship between the input variables (image bands or PCs) and target variables (biophysical and biochemical variables), many different modern and optimization algorithms developed that can learn various types of non-linear relationships, and finally availability of the optimized open source software packages and support, Artificial Neural Network was chosen as a training model in this study.

Deep Neural Networks (DNN) is a specialized name for Artificial Neural Networks. DNN are also called Deep Feedforward Networks. Feedfoward network algorithm refers to an algorithm that tries to use the input example data x to learn a function

$f()$ that approximates the output variable y by adjusting the weights variable θ (Goodfellow et al., 2016):

$$y = f(x;\ \theta) \tag{1.2}$$

These algorithms are called networks due to the fact that they usually consists of multiple functions connected to each other with networks. Neural Networks are typically composed of multiple layers, such as input, hidden and output layers. Input layer is typically the training examples and output layer is the target variable (e.g. variable to predict). Hidden layers are in between input and output layers and different functions can be applied to different hidden layers (Goodfellow et al., 2016). For example, the Equation (1.3) shows a function $f(x)$ that is formed by three hidden layers:

$$f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right) \tag{1.3}$$

In the Equation (1.3), $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ are the first, second and third layers, respectively. During the learning process, the neural network model is shown the output variables y of corresponding input variables x and the job of the hidden layers is to figure out how to match the target variable y as closely as possible (Goodfellow et al., 2016).

In this research, two neural network models were built. The first model was trained using only 5 PCs as input variables, and the second neural network was trained using simulated 231 PRISMA bands. The aim of building two neural network models is to compare the performance of a neural network using only 5 PCs to a neural network that uses the original 231 simulated PRISMA bands to predict the output variables.

1.9.1 Cost function

Choosing an appropriate cost function is an important part of building neural networks model. Essentially, cost function is what the neural network tries to

minimize after each iteration. The cost function mean squared error (MSE) is the most widely used cost function for neural network models when the target variable is continuous (Allaire, 2018).

In this study, MSE was chosen to be the cost function of the neural network models. The Equation (1.4) shows how the cost function MSE is defined:

$$MSE \;=\; \frac{1}{N}\sum_{i\,=\,1}^{n}\left(y_i \;-\; f\left(x_i;\;\theta\right)\right)^2 \tag{1.4}$$

In the Equation (1.4), N is the number of examples, y_i is the ith true value and $f(x_i;\theta)$ is the predicted ith value. x_i refers to the input parameter of ith example, and θ typically refers to weight term w and a bias term b.

Apart from MSE, mean absolute error (MAE) was calculated as a metric and monitored during the training. MAE is defined as shown in the Equation (1.5):

$$MAE \;=\; \frac{1}{N}\sum_{i\,=\,1}^{n}\left|y_i \;-\; f\left(x_i;\;\theta\right)\right| \tag{1.5}$$

1.9.2 Optimizer algorithm

In this research, Adam optimizer (Kingma and Ba, 2014) was applied to the neural network model. Before explaining what Adam does, it is important to visit the Stochastic gradient descent (SGD) algorithm.

Stochastic gradient descent is a very important algorithm used to build neural network models. SGD is used to update the learned weight and bias parameters θ. Let's assume that we have a model that tries to minimize the cost function $J(\theta)$

$$J\left(\theta\right) \;=\; \frac{1}{m}\sum_{i\,=\,1}^{m} L\left(x^i,\; y^i,\; \theta\right) \tag{1.6}$$

where L is the amount of loss for the ith example. Then, we compute the gradient as shown in the Equation (1.7):

$$\nabla_\theta J\left(\theta\right) \;=\; \frac{1}{m}\sum_{i\,=\,1}^{m} \nabla_\theta L\left(x^i,\; y^i,\; \theta\right) \tag{1.7}$$

Finally, we can update the weight and bias parameters within the term θ:

$$\theta \leftarrow \theta - \epsilon \nabla_{\theta} J(\theta) \tag{1.8}$$

In the Equation (1.8) ϵ is called learning rate and needs to be tuned. The learning rate parameter ϵ is a very important hyperparameter for neural network and it is considered to be the most difficult hyperparameter to tune (Goodfellow et al., 2016). Batch gradient descent uses the same idea, but unlike SGD, batch gradient descent makes an update for the whole training set after each iteration (Ruder, 2016).

Usually, some of the directions in the parameter space can have a significant effect on the cost and some of them may not have any effect at all. Adaptive learning algorithms can efficiently solve this problem. One of the most commonly used adaptive learning algorithm is Adam (Kingma and Ba, 2014). Essentially, Adam can be thought as a combination of RMSProp (Hinton, 2012) and Momentum (Polyak, 1964) algorithms with minor differences. In the RMSProp algorithm, exponentially decaying averages are used to mitigate the problem of extreme updates. This helps the model converge more rapidly and less sensitive to extreme cases. The Momentum algorithm uses additional parameter called velocity v apart from learning rate ϵ. And during the gradient descent update, the velocity term v is additionally taken into consideration. The Adam algorithm implements first and second order moment terms. Unlike RMSProp, Adam also implements correction for the bias of first and second order moments (Goodfellow et al., 2016). Goodfellow et al. (2016) can be referred to for more detailed explanation of how the algorithms RMSProp, Momentum and Adam are implemented.

The main drawback of Adam is that now there are more hyperparameters to tune compared to simpler algorithms such as SGD. The Adam optimizer requires the hyperparameters step size ϵ, exponential decay rates $p1$ and $p2$ and a constant δ. In this study, the suggested default value of 0.001 was used for the step size term ϵ. $p1$ and $p2$ were set to 0.9 and 0.999 respectively (suggested values). The constant term δ is 10^{-8}. Different most commonly used learning rate values (lr) were tested. The optimum learning rate value (specific to this study) was found to be 0.0001.

1.9.3 Mini batch size

Performing gradient update for the whole training set after each iteration can cause the computational cost to increase rapidly when the training data size m is large. Mini batch gradient descent can efficiently overcome this problem (Goodfellow et al., 2016). Mini batch gradient descent makes updates for the mini batch size of n, as opposed to updating gradients after training on the whole training data, with a size of m (Ruder, 2016).

Considering the fact that our training data is relatively large ($m = 250,120$), we implemented Adam with mini batch size of $n = 512$. This means that, during the neural network training, after each iteration 512 examples from the full training data set ($m = 250,120$) is randomly sampled. Based on this 512 samples, we then calculate the loss and make gradient updates. When all the examples are sampled from the full training data, this means one epoch of the training phase is complete and the next epoch can start.

1.9.4 Regularization

Regularization is a technique that prevents the problem of overfitting to the training data by adding a penalty term to the cost function. It is an important technique because it can help the trained model to generalize better to the data that it has never seen. In this study, training and validation loss were monitored during the training phase. After some iteration, the model started to fit to the training data too well and the performance of the model on the validation was poorer. Therefore, L^2 norm regularization was applied in order to overcome the problem of overfitting.

L^2 regularization is a penalty term that is directly added to the defined cost function. In general, given a cost function J, the L^2 penalty is implemented as shown in the Equation (1.9).

$$\tilde{J}(\theta) = J(\theta) \; + \; \lambda w^T w \tag{1.9}$$

In the Equation (1.9) $\tilde{J}$ is called regularized cost function and it tries to minimize the cost J and the regularization term added together. The term λ is L^2 regularization factor and it controls the amount of penalty that is added. For example, when $\lambda = 0$ there is no any constraint added and $\tilde{J} = J$. The larger the λ, the more penalty we add, and therefore the flexibility of the weights to fit to the training data is more limited. The L^2 regularization does not affect bias term b and only regularizes weight term w. λ is another hyperparamter that needs to be tuned. In this study, most commonly used values (0.1, 0.01, 0.001 etc.) were tested and $\lambda = 0.0001$ was found to work well in terms of improving the generalizability of the model to the validation set.

The cost function in this study was updated from the Equation (1.4) to the Equation (1.10):

$$\tilde{MSE} = \frac{1}{N}\sum_{i=1}^{n}\left(y_i - f\left(x_i;\ \theta\right)\right)^2 + 0.0001 w^T w \tag{1.10}$$

Here, $\tilde{MSE}$ is the regularized cost function that our model tries to minimize.

1.9.5 Early stopping

Early stopping is a simple technique that monitors the loss and it halts the training when the model starts to overfit to the training data or when it does not improve its accuracy on the validation set over iterations. In this study, validation loss was monitored during the training after each iteration. The parameter *patience* controls when to stop the training and it was set to 50 in this research. This means that if there is no improvement on the validation loss during the last 50 iterations of the training, the training process must stop. This helps with the overfitting problem as well as it halts the unnecessary training steps (if there is no any improvement after each iteration).

1.9.6 Activation function

In neural networks, the units of the layers are activated by using activation functions. In this study, rectified linear unit activation (ReLU) function was utilized. The units in the hidden layers receive a vector x from the previous layer, and transforms it using a weight term W and a bias term b as follows:

$$z \;=\; W^T x \;+\; b \tag{1.11}$$

And then, we can use a non-linear activation function ReLU to activate z:

$$g\left(z\right) \;=\; \max\left\{0,\; z\right\} \tag{1.12}$$

As we can see, ReLU gives a value of 0 when z is smaller than 0, and it returns z otherwise. ReLU is undefined at $z = 0$.

1.9.7 Weight initialization

Neural network training is an iterative process and initial weights need to be given. Goodfellow et al. (2016) indicate that, the initial weight values can sometimes have a large impact on the neural network performance and depending on how the initial weights are defined the neural network moedl may not converge at all. In this research the He initialzation (He et al., 2015) method was applied. This initialization technique was designed to work with NN that make use of the ReLU activation functions. He initialization technique draws initial weights randomly from normal distribution, where the mean $\mu = 0$, and standard deviation $\sigma = \sqrt{\frac{2}{n_l}}$. Here, n_l refers to the number of nodes in the previous layer $l - 1$, that is connected to the nodes in the current layer l.

1.9.8 Processing of input and target variables

In this study, input variables are the 5 PCs for the first neural network model and 231 simulated PRISMA image bands for the second neural network model. Simulated

PRISMA image bands were normalized and standardized so that all the bands have the same mean ($\mu = 0$) and standard deviation ($\sigma = 1$) (see Equation (1.1)).

Our target variables are INFORM parameters we want to predict. These are C_{ab}, C_{cw}, C_{cm} and LAI_s. The ranges of these variables are different. For example, minimum and maximum for C_{ab} is 20 and 60 and for C_{cw} these are 0.0035 and 0.035. The cost function with these output variables would be largely driven by variables that have higher magnitudes and interpreting the loss after each iteration would be difficult. Therefore, the output variables were normalized using the Equation (1.13).

$$Y_{scaled} = \frac{Y - \mu_Y}{\sigma_Y} \tag{1.13}$$

Here, Y is the output variable, μ_Y is the mean and σ_Y is the standard deviation of the output variable Y.

1.9.9 Neural Network Architecture design

Designing NN architecture is an important step of building neural network models. Here, number of hidden layers and number of units in each layers are hyperparameters and they need to be tuned. In this research, several neural network architectures were tested. We started with the simplest architecture (1 input layer, 1 hidden layer with 1 hidden unit and output layer) and gradually increased the depth (1, 2, 3) and width (2^1, 2^2, 2^3, 2^4 etc.) of the NN to assess the impact of the design on the performance. In general, it was observed that the more complicated the neural network architecture is, the better it performs. Increasing the number of hidden layers significantly increased the generalizability of the model to the testing data when the number of hidden units were kept constant at each layer. Optimal NN architecture was chosen based on the time it took for the model to converge, training, validation and testing accuracy.

The Figure 1.1 shows the chosen neural network arthitecture for the first model (where PCs were used as inputs):

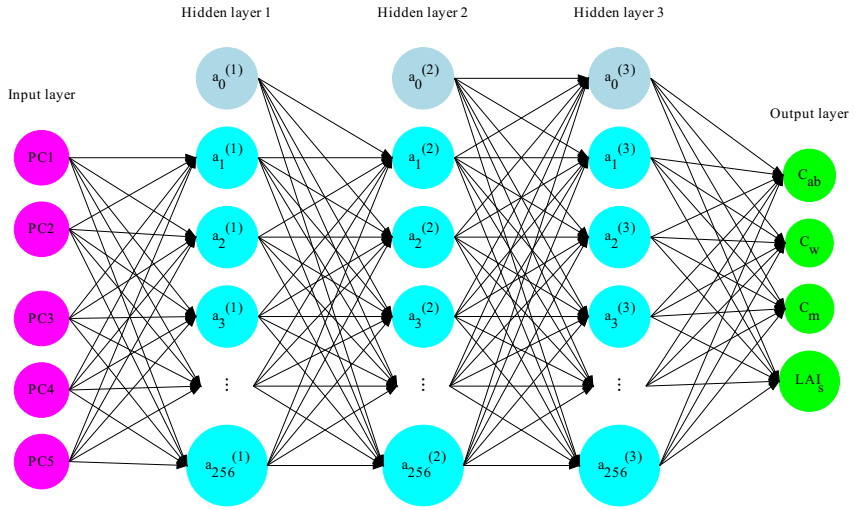

Figure 1.1: Architecture of the Neural Network with Principal Components as inputs

In the Figure 1.1, a_n^l refers to the nth unit in the lth hidden layer. Among all the tested neural network architectures when PCs as an input layer were used, this architecture yielded the best results in terms of the time it took the model to converge, the minimized loss, performance on training, validation and testing sets. However, it is important to note that a different neural network architecture that was not tested in this study could yield better results. Also, slightly simpler architecture could have yielded a similar result if it was trained much longer. But, in this study, simpler architectures (e.g. NN with 1 or 2 hidden layers) did not perform as well as the chosen architecture (Figure 1.1) when trained for the same number of epochs. It should also be noted that, NN with deeper than 3 hidden layers were not tested in this study.

The Figure 1.2 shows the chosen NN architecture for the NN model where simulated 231 PRISMA image bands were used as input.

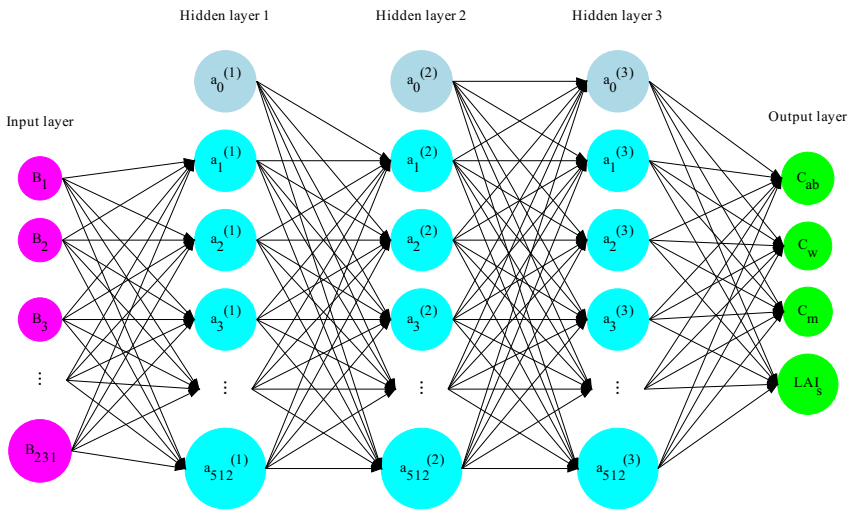

Figure 1.2: Architecture of the Neural Network with simulated PRISMA image bands as inputs

In the Figure 1.2, B_n refers to the simulated nth PRISMA image band and a_n^l refers to the nth unit in the lth hidden layer. When simulated PRISMA bands were used as an input layer, varying number of hidden layers and number of units had a much more noticeable change. In general, the more complex architectures yielded more significant results. And, when the NN model with less hidden layers were run for the same or more number of epochs they yielded significantly poorer results. This may potentially indicate that even the more complicated NN architecture could perform better than the chosen architecture.

2 Results

2.1 Local sensitivity analysis

Figure 2.1 shows the result of sensitivity analysis. Chlorophyll content (C_{ab}) appears to almost exclusively impact the visible spectra. Some effect can also be noticed in the red-edge, but there is not a significant effect of varying C_{ab} on the simulated spectra within the near-infrared (NIR) and short wave infrared (SWIR) (Figure 2.1.a). Conversely, equivalent water thickness (C_w) (Figure 2.1.b) and leaf dry matter content (C_m) (Figure 2.1.c) both have large effects on simulated spectra within the NIR and SWIR but no significant effect within the visible spectra. Leaf Area Index (single) (LAI_s) (Figure 2.1.d), Crown diameter (CD) (Figure 2.1.e)) and Stem density (Figure 2.1.f) all have noticeable effect on the simulated canopy reflectance almost all over the spectra.

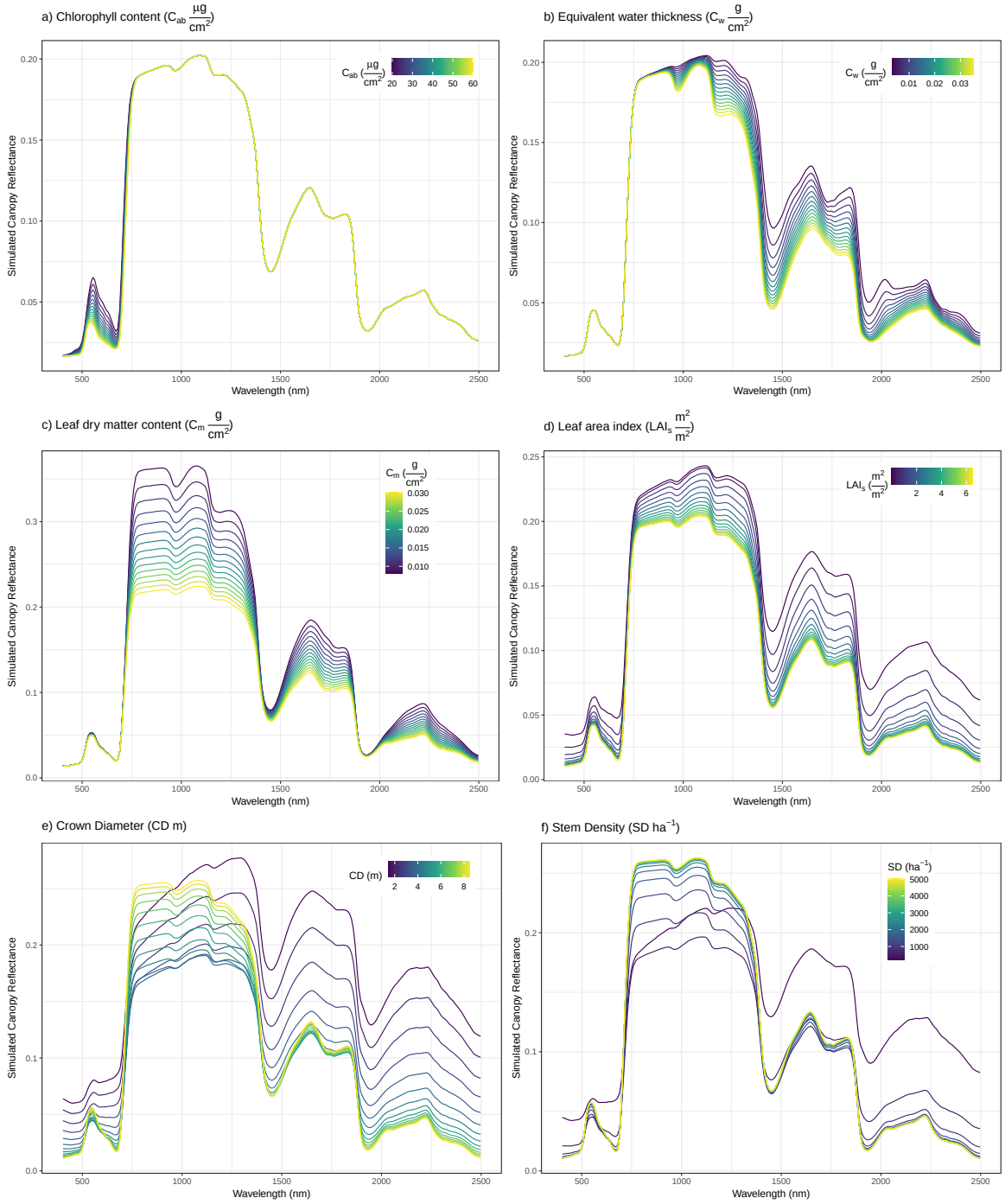

Figure 2.1: Effects of varying the chosen parameters on the simulated spectra

2.2 RTM simulation (INFORM)

Synthetic canopy reflectance data set were produced and stored in a LUT containing all 316,800 simulations. In this research, LUT was defined as a matrix. Each row of this matrix is a different simulated spectra and columns are simulated reflectance of wavelengths with the range of 400nm-2500nm with 1nm spectral resolution and 6 additional columns containing values of the corresponding variables C_{ab}, C_w, C_m, LAI_s, CD and SD that were used for each simulation. Hence the dimensions of the LUT matrix is 316,800 rows (number of simulations) by 2107 columns (2101 simulated "bands" + 6 INFORM variables):

$$\begin{bmatrix} 400nm_1 & \dots & 2500nm_1 & Cab_1 & Cw_1 & Cm_1 & LAIs_1 & CD_1 & SD_1 \\ 400nm_2 & \dots & 2500nm_2 & Cab_2 & Cw_2 & Cm_2 & LAIs_2 & CD_2 & SD_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 400nm_{316,800} & \dots & 2500nm_{316,800} & Cab_{316,800} & Cw_{316,800} & Cm_{316,800} & LAIs_{316,800} & CD_{316,800} & SD_{316,800} \end{bmatrix}$$

In this matrix, $400nm_n$, ..., $2500nm_n$ refer to the simulated reflectance for the corresponding wavelength in the simulation number n. Cab_n, Cw_n, Cm_n, $LAIs_n$, CD_n and SD_n are values of the INFORM parameters that were used in the nth simulation.

2.3 Spectral resampling

The output of INFORM simulations were resampled to 231 PRISMA bands. The LUT matrix was used for spectral resampling and the resulting matrix has a dimension of 316,800 rows (number of simulations) by 237 columns (231 PRISMA image bands + 6 INFORM variables):

$$\begin{bmatrix} Band1_1 & \dots & Band231_1 & Cab_1 & Cw_1 & Cm_1 & LAIs_1 & CD_1 & SD_1 \\ Band1_2 & \dots & Band231_2 & Cab_2 & Cw_2 & Cm_2 & LAIs_2 & CD_2 & SD_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ Band1_{316,800} & \dots & Band231_{316,800} & Cab_{316,800} & Cw_{316,800} & Cm_{316,800} & LAIs_{316,800} & CD_{316,800} & SD_{316,800} \end{bmatrix}$$

In this matrix, $Band1_n$, ..., $Band231_n$ correspond to the simulated reflectance for the corresponding image band in the simulation number n. Cab_n, Cw_n, Cm_n, $LAIs_n$, CD_n and SD_n refer to the values of the INFORM parameters that were used in the nth simulation.

2.4 Statistics of simulated data and PRISMA image

The Figure 2.2 shows statistical information (mean and mean ± standard deviation) calculated from the LUT and PRISMA image:

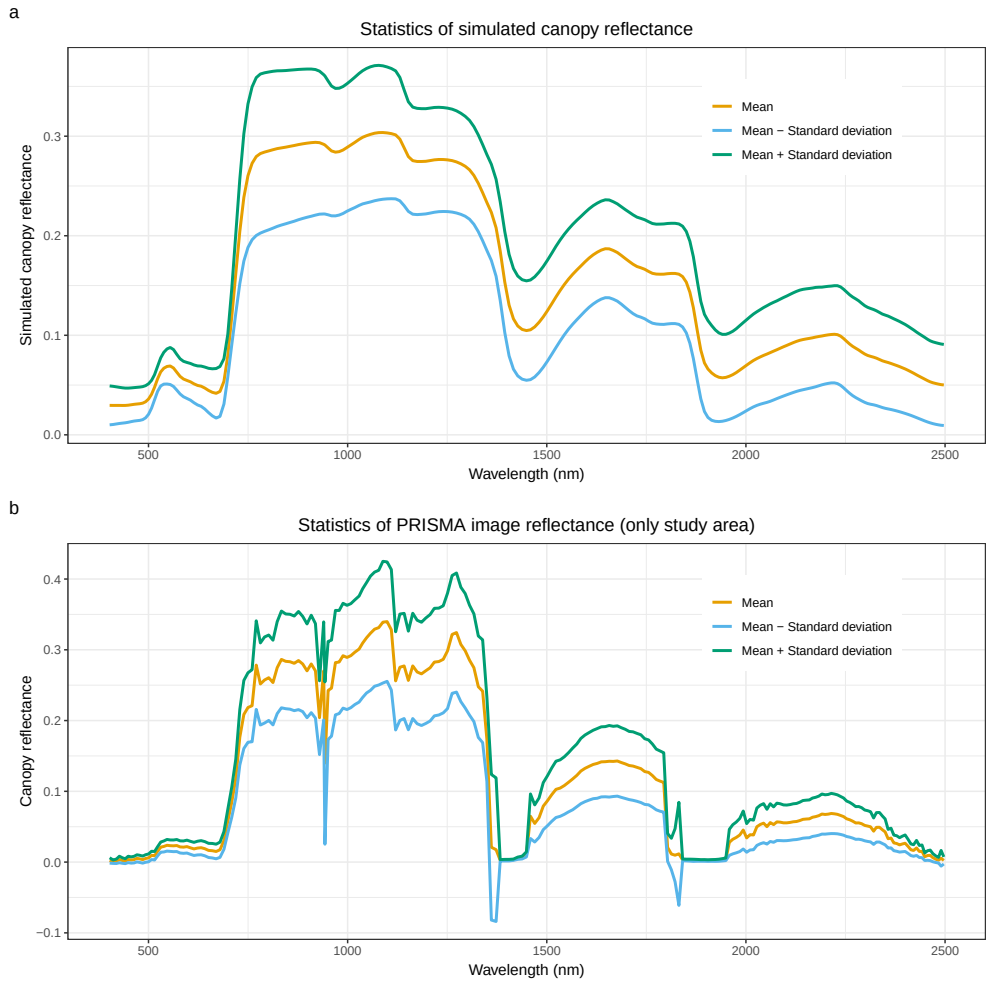

Figure 2.2: Mean and mean ± standard deviation in the a) LUT and b) PRISMA image

Mean and standard deviation within the LUT are much smoother compared to mean and standard deviation within the PRISMA image spectra. This is due to the fact that INFORM model does not add noise during the simulation which can commonly exist in remote sensing images. There is a noticeable amount of noise in the PRISMA image spectra. Some of the noise in the image spectra could potentially be due to the fact that the PRISMA image contained cloud and shadow

within the study area and although most of the cloud and shadow pixels were masked, the nearby pixels could still be affected.

The Figure 2.3 shows the difference between averaged reflectance within the simulated database (LUT) and PRISMA image.

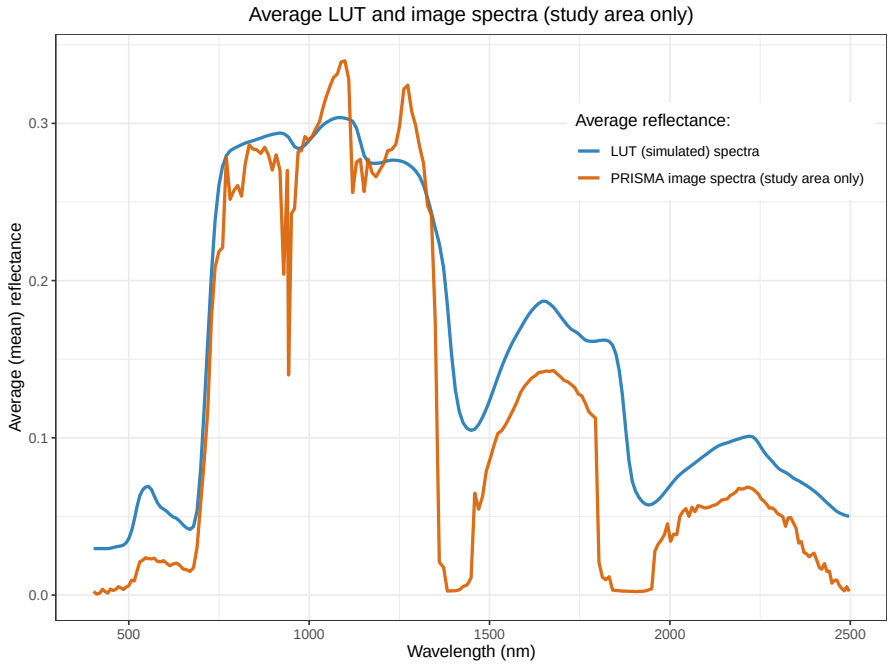

Figure 2.3: Difference between averaged LUT and PRISMA image reflectance

The LUT appears to have higher average reflectance within the visible spectra compared to the PRISMA image spectra. Differences within the water absorbtion bands can also be clearly seen. There is relatively good agreement within the NIR spectrum.

2.5 Gaussian noise

The Figure 2.4 shows the effect of adding 3% Gaussian noise to the simulated data.

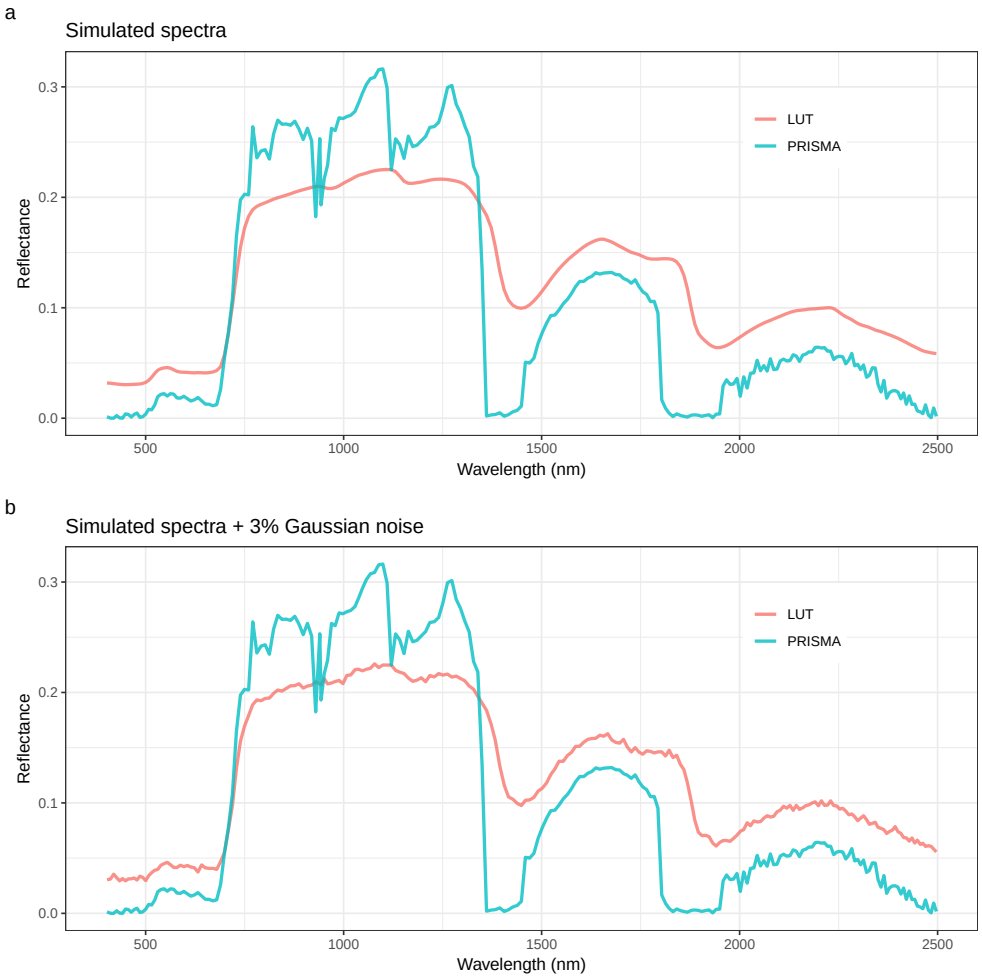

Figure 2.4: Effect of adding 3% Gaussian noise to the simulated spectra. The randomly chosen pixel from the PRISMA data was plotted to illustrate the noise found typically in the image

The Figure 2.4.a shows a simulated spectra that seems perfectly smooth. However, after adding 3% Gaussian noise, the simulated spectra is not as smooth anymore and contains random noise all over the whole spectra (Figure 2.4.b). This also makes the simulated spectra more similar to the pixel extracted from the PRISMA image.

2.6 Principal Component Analysis (PCA)

The result of PCA show that most of the variation in the simulated data can be explained by much fewer variables (PCs):

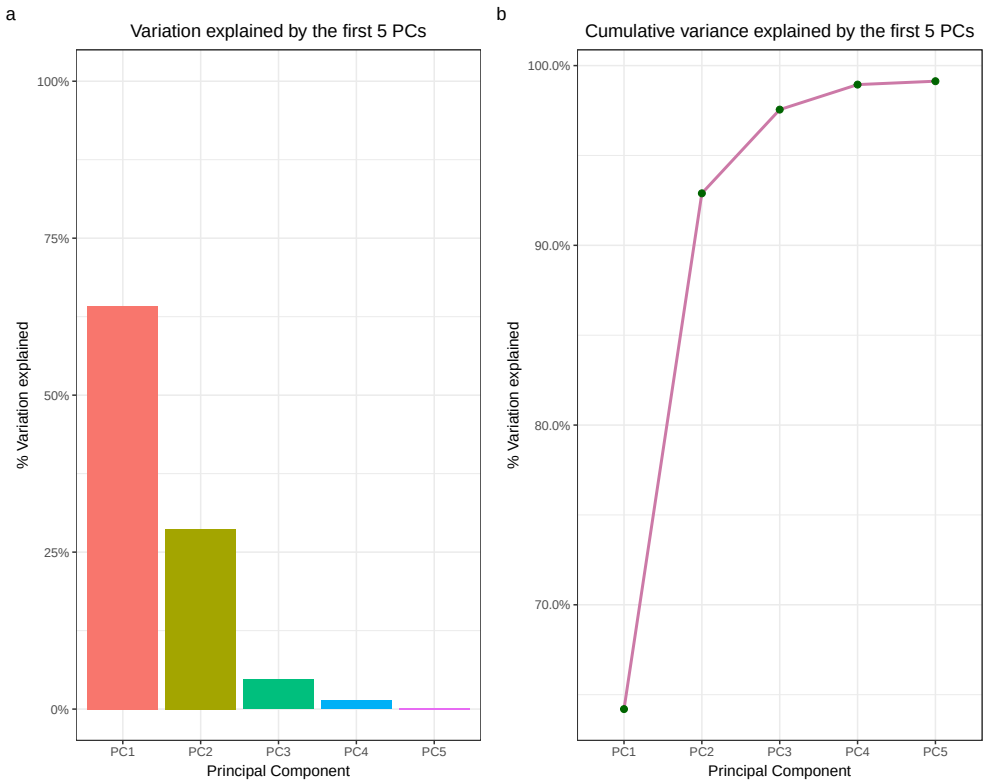

Figure 2.5: Principal Component Analysis: a) Screeplot, b) Cumulative variance explained by the first 5 PCs

The Figure 2.5.a shows the screeplot of the PCA result. The first PC explains the most of the variation and together with the next 4 PCs we can capture more than 99% of the variation that is present in the original data (Figure 2.5.b). This, once again shows the multicollinearity problem with hyperspectral data.

References

Ali, A. M., Darvishzadeh, R., Skidmore, A., Gara, T. W., and Heurich, M. Machine learning methods' performance in radiative transfer model inversion to retrieve plant traits from sentinel-2 data of a mixed mountain forest. *International Journal of Digital Earth*, pages 1–15, 2020.

Allaire, J. *Deep Learning with R*. Simon and Schuster, 2018.

Bro, R. and Smilde, A. K. Principal component analysis. *Analytical methods*, 6(9): 2812–2831, 2014.

Corporation, M. and Weston, S. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2020. URL `https://CRAN.R-project.org/package=doParallel`. R package version 1.0.16.

Danner, M., Berger, K., Wocher, M., Mauser, W., and Hank, T. Efficient rtm-based training of machine learning regression algorithms to quantify biophysical & biochemical traits of agricultural crops. *ISPRS Journal of Photogrammetry and Remote Sensing*, 173:278–296, 2021.

Darvishzadeh, R., Skidmore, A., Abdullah, H., Cherenet, E., Ali, A., Wang, T., Nieuwenhuis, W., Heurich, M., Vrieling, A., O'Connor, B., et al. Mapping leaf chlorophyll content from sentinel-2 and rapideye data in spruce stands using the invertible forest reflectance model. *International Journal of Applied Earth Observation and Geoinformation*, 79:58–70, 2019.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Hinton, G. Neural networks for machine learning coursera video lecturesgeoffrey hinton. 2012.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kuhn, M. and Wickham, H. *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles.*, 2020. URL `https://www.tidymodels.org`.

Kuhn, M. and Wickham, H. *recipes: Preprocessing Tools to Create Design Matrices*, 2021. URL `https://CRAN.R-project.org/package=recipes`. R package version 0.1.16.

Laurent, V. C., Verhoef, W., Clevers, J. G., and Schaepman, M. E. Inversion of a coupled canopy–atmosphere model using multi-angular top-of-atmosphere radiance data: A forest case study. *Remote Sensing of Environment*, 115(10):2603–2612, 2011.

Lehnert, L. W., Meyer, H., Obermeier, W. A., Silva, B., Regeling, B., Thies, B., and Bendix, J. Hyperspectral data analysis in R: The hsdar package. *Journal of Statistical Software*, 89(12):1–23, 2019. doi: 10.18637/jss.v089.i12.

Microsoft and Weston, S. *foreach: Provides Foreach Looping Construct*, 2020. URL `https://CRAN.R-project.org/package=foreach`. R package version 1.5.1.

Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL `https://www.R-project.org/`.

Rivera-Caicedo, J. P., Verrelst, J., Muñoz-Marí, J., Camps-Valls, G., and Moreno, J. Hyperspectral dimensionality reduction for biophysical variable statistical retrieval. *ISPRS journal of photogrammetry and remote sensing*, 132:88–101, 2017.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Schlerf, M. and Atzberger, C. Vegetation structure retrieval in beech and spruce forests using spectrodirectional satellite data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5(1):8–17, 2012.

Visser, M. D. *ccrtm: Coupled Chain Radiative Transfer Models*, 2021. URL `https://CRAN.R-project.org/package=ccrtm`. R package version 0.2.

Wickham, H. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL `https://ggplot2.tidyverse.org`.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., and Yutani, H. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. doi: 10.21105/joss.01686.