Yield gap decomposition in R

João Vasco Silva CIMMYT-Zimbabwe

May 26, 2023

Contents

This website contains all the material of the training on yield gap decomposition organized by the OneCGIAR initiative on Excellence in Agronomy! The training will cover the main methods available for yield gap decomposition using farmer field data, namely stochastic frontier analysis, boundary line analysis, and machine learning. It will also provide you with a data collection tool meeting the data requirements to run such analysis.

For this training, you will need the R software installed in your machine. We also recommend you to install Rstudio as a user friendly interface to R. Please refer to the menu on the left for further information on how to install these software. You also find there good resources to refresh your knowledge of R.

Background on the yield gap decomposition add-on and training objectives and expectations can be downloaded here. The schedule for the training is as follows:

- **Day 1**: Concepts & definitions for yield gap decomposition
- **Day 2**: Data collection tool
- **Day 3**: Hands-on stochastic frontier analysis
- **Day 4**: Hands-on machine learning
- **Day 5**: Introduction to boundary line analysis, and wrap-up

For questions contact João Vasco Silva (j.silva@cgiar.org).

1 Installing R

Robert J. Hijmans, UC-Davis

1.1 Background R software

R is perhaps the most powerful computer environment for data analysis that is currently available. *R* is both a computer *language*, that allows you to write instructions, and a *program* that responds to these instructions. *R* has core functionality to read and write files, manipulate and summarize data, run statistical tests and models, make fancy plots, and many more things like that. This core functionality is extended by hundreds of packages (plug-ins). Some of these packages provide more advanced generic functionality, others provide cutting-edge methods that are only used in highly specialized analysis.

Because of its versatility, *R* has become very popular across data analysts in many fields, from agronomy to bioinformatics, ecology, finance, geography, pharmacology and psychology. You can read about it in this article in Nature or in the New York Times. So you probably should learn *R* if you want to be good in data analysis, be a successful researcher, collaborate, get a high paying data science job, ... If you are not that much into *data analysis and visualization* but want to learn programming for more general tasks, you may want to start with python instead.

This document provides a concise introduction to *R*. It emphasizes what you need to know to be able to use the language in any context. There is no fancy statistical analysis here. We just present the basics of the *R* language itself. We do not assume that you have done any computer programming before (and if that is the case, we do assume that you think it is about time you did). Experienced *R* users obviously need not read this. However, the material may be useful if you want to refresh your memory, if you have not used *R* much, or if you feel confused.

When using this resource, it is very important to follow Norman Matloff's advice: *"When in doubt, try it out!"*. That is, test yourself. Copy the examples shown, and then make modifications to see if you can predict what will happen. Only then will you really understand what is going on. You are learning a language, and you will have to use it a lot to become good at it. So express yourself and accept that for a while you will be stumbling a lot and sometimes feel lost.

To work with *R* on your own computer, you need to download the program and install it. I recommend that you also install R-Studio. R-Studio is a separate program that makes *R* easier to use. Here is a video that shows how to work in R-Studio.

After having gone through the chapters presented here, you should could consult additional resources to learn *R*. It is very helpful to read several of these introductions to *R* while you use it in your work. Each time you will pick up new things, and feel accomplished about how much you already understand. There are many free resources on the web, including R for Beginners by Emmanuel Paradis and this tutorial by Kelly Black that is similar to the one you are reading now. Or consult this brief overview by Ross Ihaka (one of the originators of R) from his Information Visualization course. You can also consult the more formal "official" introduction or get a copy of Norman Matloff's book The Art of R Programming.

There is also a lot of good stuff on rstatistics.net

If you want to take it easy, or perhaps learn about R while you commute on a packed train, you could watch some Google Developers videos.

If none of this appeals to you, and you already are experienced with *R*, or you have done a lot of programming with other languages, skip all of this and perhaps have a look at Hadley Wickham's Advanced R.

1.2 Installing R on Windows

Download the latest R installer (.exe) for Windows. Install the downloaded file as any other windows app.

1.3 Installing Rstudio on Windows

Now that R is installed, you need to download and install RStudio. First download the installer for Windows. Run the installer (.exe file) and follow the instructions.

2 Concepts and definitions

2.1 (1) Global Yield Gap Atlas

Martin van Ittersum, WUR

Download slides here.

This talk highlights the current and future applications of the Global Yield Gap Atlas. You will learn about the platform and the types of data that it provides to inform food security and resource use efficiency assessments globally. Examples of different applications out of the data from the Global Yield Gap Atlas are provided.

2.2 (2) Yield levels and yield gaps

Kindie Tesfaye, CIMMYT

Download slides here.

This talk highlights the different yield levels defined so far in the context of production ecology. Those yield levels are explained in detail in this talk. An overview of how to estimate the different yield levels using farm surveys, field experiments, and crop simulation models is provided.

2.3 (3) Yield gap decomposition

João Vasco Silva, CIMMYT

Download slides here.

This talk highlights the methods and approaches most widely used so far for yield gap decomposition using farmer field data. An overview of how crop models and data-driven approaches (e.g., frontier analysis and machine learning) have been used for yield gap decomposition is provided and illustrated with examples from the literature.

3 Data collection & requirements

Information about **minimum data requirements** for yield gap decomposition can be found here. This minimum data requirement is required to link primary point-based data to secondary spatial data on growth-defining (climate) and -limiting factors (soils), which is needed for yield gap analysis in addition to primary data on crop management practices (remember again these should cover growth-defining, -limiting, -reducing factors). A group discussion, mixing members of different use cases, is undertaken to answer the following questions: (1) what type of data are required to conduct to conduct yield gap decomposition, (2) what type of data do you have with you, (3) are the data you have enough for yield gap decomposition, and (4) what additional data do you need for yield gap decomposition. After this discussion, we know the data available for each use case, and the additional data requirements, for yield gap decomposition.

The **data collection tool** developed by Excellence in Agronomy for yield gap decomposition is introduced here. The instrument to collect the required data can be accessed in the link below. This instrument is compatible with the Open Data Kit (ODK) and KoboCollect Toolbox. You can request support from members of the global team of Excellence in Agronomy to help you deploy this form, so that the collected data is centralized in a common database system. This is important to ensure that custom-made scripts can be used for retrieving secondary data, and for cleaning, curating, and analysing primary data across different use cases. A training on the data collection tool is provided after the tool is introduced, so you have the chance to get familiar with tool.

- Download the ODK form here.
- Slides with experiences from Ethiopia.

4 Boundary line analysis

4.1 Workflow of Fermont et al. (2009)

- **Tomás Roquette Tenreiro**, SISTAGRO
- **João Vasco Silva**, CIMMYT-Zimbabwe

4.1.1 Introduction

Boundary line analysis is a powerful concept in agronomy (Webb, 1972) as it can be used to explain yield gaps in farmers' fields. From a conceptual standpoint, different approaches have been proposed for boundary line analysis over the past decades, which can be grouped in three main methodological families. The first estimates the boundary lines "manually and by eye" (e.g., French & Schultz, 1984). The second relies on statistical methods, encompassing grouping data and fitting 'splines', to estimate the boundary line (e.g., Fermont et al., 2009). Lastly, econometric techniques including quantile regression and stochastic frontier analysis have also been used to estimate boundary lines (e.g., Silva et al., 2017).

Herewith, we focused on the boundary line approach encompassing grouping data and fitting of 'splines' to farmer field data. The general features of this approach include (1) identification of boundary points, (2) estimation of a continuous function characterizing the boundary points, and (3) yield gap analysis for each factor and across factors. This type of boundary line analysis has been used widely in the past years across a number of crops and geographies (e.g., Casanova et al., 1999; Shatar & McBratney, 2004; Fermont et al., 2009; Wairegi et al., 2010). Its statistical nature (e.g., Fermont et al., 2009) makes it suitable for reproducible research and easy to interpret by agronomists.

The use of the method for yield gap decomposition was best described by Wairegi et al. (2010). After estimating boundary lines for individual production factors, von Liebig's law of the minimum is used to identify the production factor limiting crop yield most. This is done for each field as Ymin = min(Yx1, Yx2, . . . , Yxn), where Yxn is the yield

predicted with the boundary line for a given production factor. Estimation of Ymin allows identifying the limiting factor to crop yield and decomposing the yield gap into an 'identified yield gap' (IYG) and an 'unexplained yield gap' (UYG) in each field. IYG is defined as the difference between the water-limited yield and the Ymin. UYG is defined as the difference between Ymin and the reported actual yield.

This workflow documents in a reproducible way the concept of boundary line analysis of Fermont et al. (2009) on the original dataset for cassava cropping systems in Eastern Africa. Fermont et al., (2009) conducted a number of household surveys and on-farm trials in Uganda and western Kenya to unravel the importance of abiotic, biotic and associated crop management constraints to cassava production. Clear boundary lines between cassava yield and growing season rainfall, soil fertility, soil texture, and pest, disease and weed management practices, which were identified as the key constraints to cassava production in East Africa.

4.1.2 Load required R packages

First, we need to load the R packages needed to run this workflow.

```
# package names
packages <- c("splines", "Metrics", "dplyr", "tidyr", "knitr", "reshape2", "ggplot2")
#
# install packages
installed_packages <- packages %in% rownames(installed.packages())
if(any(installed_packages == FALSE)){
  install.packages(packages[!installed_packages], repos="http://cran.us.r-project.org
↪", quiet=T)}
#
# load packages
invisible(lapply(packages, function(x) suppressMessages(require(x, character.only=T,␣
↪quietly=T, warn.conflicts=F))))
```

4.1.3 Crop field data

The first step is to load the crop data to be used for yield gap analysis. The dataset contains measured cassava yields, management practices, and biophysical conditions for a number of fields in Central/Eastern Uganda and Western Kenya (Fermont et al., 2009). The sites were chosen to represent a range of environments and management practices of cassava-based cropping systems in Eastern Africa. In total, 13 different yield-explaining variables (x-variables) were collected for a total number of 122 field-year combinations.

```
# read .csv file with data
file <- 'https://raw.githubusercontent.com/jvasco323/EiA_BLA_workflow/main/fermont_
↪etal_final.csv'
data <- read.csv(url(file))
#
# list variables of interest for visualization
str(data)
## 'data.frame':    122 obs. of  22 variables:
##  $ year           : int  2005 2005 2005 2005 2005 2005 2005 2005 2005 2005 ...
##  $ Trial          : chr  "NOT" "NOT" "NOT" "NOT" ...
##  $ Site           : chr  "Nambale" "Nambale" "Nambale" "Nambale" ...
##  $ Farm.no.       : chr  "41" "42" "43" "44" ...
##  $ yld_t0         : num  17.8 9.5 33.8 18 18.5 ...
##  $ pH_soil        : num  5.2 5.3 5.3 5.3 5.2 5.2 5.4 5.2 5.9 5.5 ...
```

(continues on next page)

(continued from previous page)

```
##  $ SOC             : num  9.66 8.85 9.4 9.72 9.14 ...
##  $ totN_soil       : num  0.516 0.509 0.539 0.545 0.602 ...
##  $ P_soil          : num  3.15 2.29 2.36 3.41 3.28 ...
##  $ K_soil          : num  0.124 0.14 0.124 0.142 0.512 ...
##  $ Ca_soil         : num  0.409 0.313 0.365 1.412 1.634 ...
##  $ Mg_soil         : num  0.236 0.555 0.165 0.1 0.915 ...
##  $ CBB_T0_9        : num  93.8 147.5 172.5 117.9 151.5 ...
##  $ CGM_T0_9        : num  120 210 128 131 183 ...
##  $ CAD_T0_9        : num  0 0 3.75 0 13.24 ...
##  $ RF_tot          : int  2460 2460 2460 2460 2460 2460 2460 2460 2460 2377 ...
##  $ RF0_6M          : int  1107 1107 1107 1107 1107 1107 1107 1107 1107 1306 ...
##  $ RF_.d           : int  144 144 144 144 144 144 144 144 144 165 ...
##  $ sand            : num  54 64 52 50 52 66 58 58 60 58 ...
##  $ clay            : num  33 23 33 33 31 21 27 27 20 23 ...
##  $ silt            : num  13 13 15 17 17 13 15 15 20 19 ...
##  $ days_to_harvest: int  397 397 397 397 397 397 397 397 397 398 ...
```

4.1.4 Data manipulation

In the chunk of code below, we exclude observations with cassava yields above 30 t/ha based on the yield distribution observed in the data.

```
# remove outliers
data <- subset(data, yld_t0 < 30)
```

4.1.5 Descriptive statistics

Descriptive statistics of actual cassava yield and crop management practices over the two cropping seasons used in the analysis are provided below. The actual yield of cassava across the study region was on average 13.9 t/ha. Local soils were moderately acidic (pH < 5.8) and growing season rainfall was on average 1728 mm. Soil clay content and soil organic C were on average 27.2% and 10.5 mg/kg, respectively.

Variable	Mean 2004	Mean 2005	StDev 2004	StDev 2005
Soil clay content (%)	26.5	27.8	8.4	9.5
Days to harvest (d)	368.6	387.2	37.7	11.7
Soil pH	5.8	5.7	0.5	0.4
Growing season rainfall (mm)	1642.8	1814.1	318.6	469.6
Soil organic C (mg/kg)	10.7	10.3	3.9	4.2
Cassava yield (t/ha)	12.9	14.9	6	6.4

4.1.6 Step 1: Field selection

The function in the next chunk of code was developed to categorize the x-variable into 10 groups with the same number of observations and to obtain the maximum yield (ymax) for each group. The latter can be obtained with two approaches. The first, **maximum**, defines ymax as the maximum yield observed in each of the 10 groups. The second, **95_quantile**, defines ymax as the average actual yield in the top 5th quantile of the yield distribution. The selection of the most appropriate approach depends on the yield variability within each of the 10 groups and on the error associated with the fitted boundary line (see next section).

In our view, the selection of ymax based on the absolute maximum yield observed in each group is more appropriate for denser scatters where no clear outliers in crop yield can be identified. Defining ymax based on the top 5th quantile of actual yields for each group is a more conservative approach, suitable for datasets with potential outliers in crop yield. In such cases, the mean actual yield in the top 5th quantile will reduce the individual weight of single observations, which might be overestimated.

It is important to note that boundary lines estimated with splines tend to underestimate ymax for most initial x-intervals (i.e., where 'y' shows a large response to a variation of 'x'). Thus, the use of the mean actual yield of the top 5th quantile should be avoided every time there is enough confidence on the reported 'y' values.

```
bl_points <- function(df, xvar, approach){
   #
   # select the x variable of interest
   df <- df[,c("yld_t0", xvar)]
   #
   # modify column names for generic use
   colnames(df)[1] = "Y"
   colnames(df)[2] = "X"
   df <- subset(df, X>0, select=c(Y, X)) # why x > 0?
   #
   # 'NULL' values are excluded to avoid data transformation problems and calculation␣
↪failures
   # correct NA values for both Y and X variables
   df$Y[is.na(df$Y)] <- mean(df$Y, na.rm=T)
   df$X[is.na(df$X)] <- mean(df$X, na.rm=T)
   #
   # split X variable in 10 quantiles (Fermont et al., 2009)
   x_0.1 <- subset(df, X <= quantile(X, 0.1))
   x_0.2 <- subset(df, X > quantile(X, 0.1) & X <= quantile(X, 0.2))
   x_0.3 <- subset(df, X > quantile(X, 0.2) & X <= quantile(X, 0.3))
   x_0.4 <- subset(df, X > quantile(X, 0.3) & X <= quantile(X, 0.4))
   x_0.5 <- subset(df, X > quantile(X, 0.4) & X <= quantile(X, 0.5))
   x_0.6 <- subset(df, X > quantile(X, 0.5) & X <= quantile(X, 0.6))
   x_0.7 <- subset(df, X > quantile(X, 0.6) & X <= quantile(X, 0.7))
   x_0.8 <- subset(df, X > quantile(X, 0.7) & X <= quantile(X, 0.8))
   x_0.9 <- subset(df, X > quantile(X, 0.8) & X <= quantile(X, 0.9))
   x_1.0 <- subset(df, X > quantile(X, 0.9) & X <= quantile(X, 1.0))
   #
   # define boundary points for each quantile based on maximum value
   if(approach == 'maximum'){
    blp_0.0 <- subset(x_0.1, X == min(X)) #  briefly explain why needed
    blp_0.1 <- subset(x_0.1, Y == max(Y))
    blp_0.2 <- subset(x_0.2, Y == max(Y))
    blp_0.3 <- subset(x_0.3, Y == max(Y))
    blp_0.4 <- subset(x_0.4, Y == max(Y))
    blp_0.5 <- subset(x_0.5, Y == max(Y))
    blp_0.6 <- subset(x_0.6, Y == max(Y))
    blp_0.7 <- subset(x_0.7, Y == max(Y))
    blp_0.8 <- subset(x_0.8, Y == max(Y))
```

(continues on next page)

(continued from previous page)

```
    blp_0.9 <- subset(x_0.9, Y == max(Y))
    blp_1.0 <- subset(x_1.0, Y == max(Y))
   #
   # define boundary points for each quantile based on yields in given quantile
   } else if(approach == '95_quantile'){
    blp_0.0 <- subset(x_0.1, X == min(X))
    blp_0.1 <- subset(x_0.1, Y > quantile(Y, 0.95))
    blp_0.2 <- subset(x_0.2, Y > quantile(Y, 0.95))
    blp_0.3 <- subset(x_0.3, Y > quantile(Y, 0.95))
    blp_0.4 <- subset(x_0.4, Y > quantile(Y, 0.95))
    blp_0.5 <- subset(x_0.5, Y > quantile(Y, 0.95))
    blp_0.6 <- subset(x_0.6, Y > quantile(Y, 0.95))
    blp_0.7 <- subset(x_0.7, Y > quantile(Y, 0.95))
    blp_0.8 <- subset(x_0.8, Y > quantile(Y, 0.95))
    blp_0.9 <- subset(x_0.9, Y > quantile(Y, 0.95))
    blp_1.0 <- subset(x_1.0, Y > quantile(Y, 0.95))}
   #
   # bind subsets
   blp_df <- rbind(blp_0.0, blp_0.1, blp_0.2, blp_0.3, blp_0.4, blp_0.5,
                   blp_0.6, blp_0.7, blp_0.8, blp_0.9, blp_1.0)
   return(blp_df)}
```

4.1.7 Step 2: Boundary lines

The first step to fit a boundary line is to select the independent variables of interest. Independent variables with a discrete distribution should be avoided as these won't allow for a reliable estimation of ymax for a range of x-values. The variables "CAD_T0_9", "RF_tot", "RF0_6M", "RF_d" and "days_to_harvest" in the dataset of Fermont et al. (2009) do not have a continuous distribution of values, which is problematic for boundary line analysis because many observations would be associated with the same ymax. Therefore, we decided not to include these variables from the analysis presented here.

We recommend the following two 'rules of thumb' when selecting independent variables for boundary line analysis. Firstly, the use of non-continuous x-variables (i.e., discrete, partially classified, excessive skewness of x-data) should be avoided. Secondly, the error of the fitted boundary line models should be similar across independent variables. Variables for which the boundary line has an excessive root mean square error should thus be avoided, as the fitted model does not describe the variability in the boundary points properly. In this specific case, we established a minimum root mean square error of 3.5 t/ha for inclusion of a variable in the yield gap analysis.

```
# four variables considered for illustrative purposes
variables <- c("pH_soil", "totN_soil", "P_soil", "clay")
```

Before estimating the boundary lines in the next chunk of code, we create empty dataframes to bind all the new data generated in the for loop used for this purpose.

```
blp_new <- c()
data_new <- c()
rmse_df <- c()
```

A generic six-step procedure, documented in the chunk of code below, was implemented to estimate boundary lines for individual production factors. In **step 1**, a new column is created in the dataframe of each production factor. In **step 2**, this new column stores the corresponding boundary lines points selected through the approach "maximum". A statistical model is fitted to the data using glm() in **step 3** for each group of boundary line points. The fitted model is further used in **step 4** to estimate ymax and to store its values in the dataframe containing the original data

("data_subset") and in the dataframe created for each production factor ("data.bla"). After adding a new column ('y_pred') to both dataframes, the root mean square error ("rmse") is computed in **step 5** to compare the maximum observed yields against the maximum predicted yields by the boundary line. Finally, dataframes are bound together in **step 6**. Three dataframes are produced with this procedure:

- "blp_new" includes only the boundary line points (one column with the observed yields plus n columns, each showing the predicted ymax for each i variable);
- "data_new" includes all observations as per the original data (one column with the observed yields plus n columns, each showing the predicted ymax for each i variable);
- "rmse_df" including the root mean square errors estimated for the boundary lines fitted for each production factor (one column associated with each boundary line for each i-variable).

```
for(i in unique(variables)){
  # print(i)
  #
  # 1) select data
  data_subset <- data[,c("year", "Trial", "Site", "Farm.no.", "yld_t0", i)]
  colnames(data_subset)[5] = "Y"
  colnames(data_subset)[6] = "X"
  data_subset$variable <- i
  #
  # 2) estimate boundary line points
  data.bla <- bl_points(data, i, approach="maximum")
  data.bla$variable <- i
  #
  # 3) create predictive model
  model <- glm(Y ~ ns(X, df = 2), data = data.bla)
  # print(model)
  #
  # 4) predict y_max
  # boundary points only
  data.bla$y_pred <- predict(model, newdata = data.bla)
  # raw data
  data_subset$y_pred <- predict(model, newdata = data_subset)
  #
  # 5) error assessment
  rmse_value <- rmse(data.bla$Y, data.bla$y_pred)
  #
  # 6) bind all data together
  blp_new <- rbind(blp_new, data.bla)
  data_new <- rbind(data_new, data_subset)
  rmse_df <- rbind(rmse_df, cbind(i, rmse_value))}
```

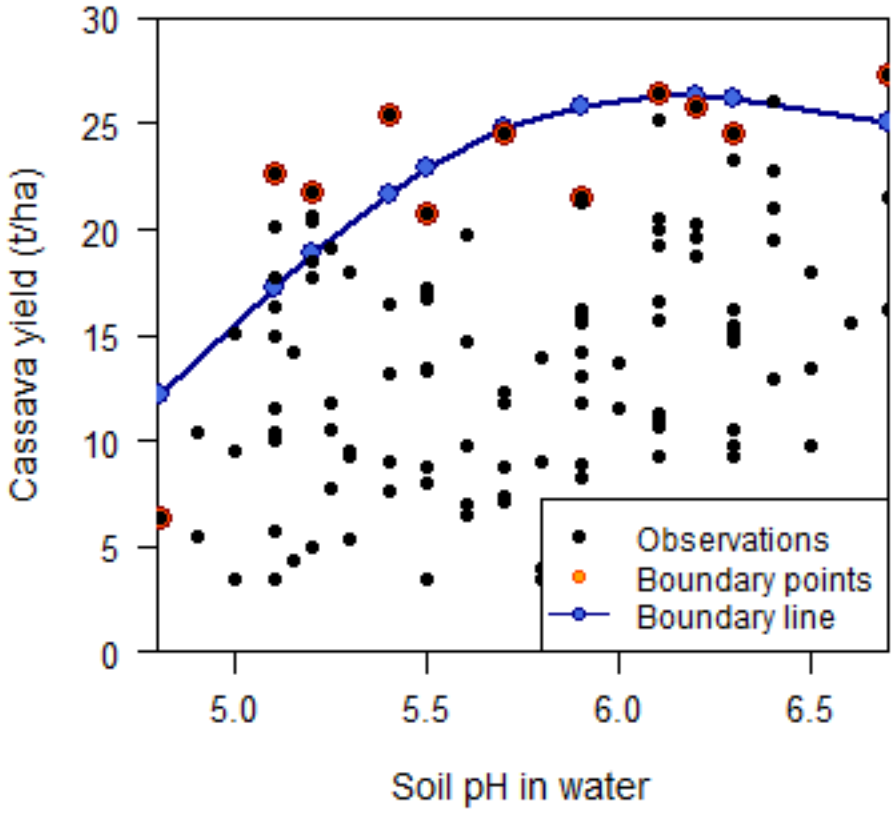
The boundary line fitted between cassava yield and soil pH in water is illustrated below.

```
# subset for variable of interest
pH <- subset(blp_new, variable =='pH_soil')
pH <- pH[order(pH$X),]
pH_original <- subset(data_new, variable =='pH_soil')
#
# make plot
par(mfrow=c(1,1), mar=c(5,5,1,1), las=1, xaxs='i', yaxs='i', cex.axis=1.1, cex.lab=1.
↪2)
plot(pH$X, pH$y_pred, ylim=c(0,30), xlab='Soil pH in water', ylab='Cassava yield (t/
↪ha)')
lines(pH$X, pH$y_pred, lty=1, col='darkblue', lwd=2.5)
points(pH$X, pH$y_pred, pch=21, cex=1.5, col='darkblue', bg='royalblue')
```

(continues on next page)

(continued from previous page)

```
points(pH$X, pH$Y, pch=21, cex=1.75, col='darkred', bg='orangered')
points(pH_original$X, pH_original$Y, pch=21, cex=1, col='black', bg='black')
legend('bottomright', legend=c('Observations', 'Boundary points', 'Boundary line'),
       lty=c(NA,NA,1), pch=c(21,21,21),
       col=c('black', 'orangered', 'darkblue'), pt.bg=c('black', 'orange', 'royalblue
↪'))
```



4.1.8 Step 3: Yield gap analysis

The first step for the yield gap analysis is to obtain ymin and the corresponding limiting factor (i.e., the corresponding x-variable with the lowest BL predicted yield for a certain observed yield) for each field. This is done with the chunk of code below.

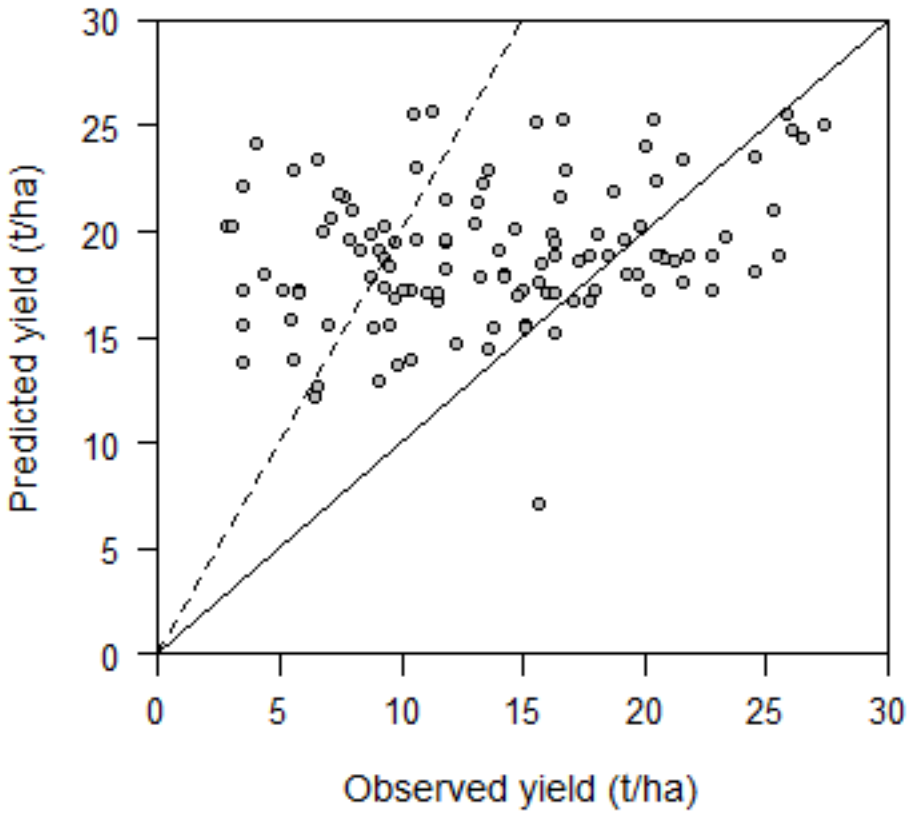
```
# reshape df
data_ygd <- dcast(data_new, year + Trial + Site + Farm.no. + Y ~ variable, value.var=
↪'y_pred')
data_ygd$y_pred_min <- apply(data_ygd[6:9], 1, min)
data_ygd$limiting_factor <- names(data_ygd[6:9])[apply(data_ygd[6:9], 1, which.min)]
```

Following, we estimate the 'identified yield gap' (IYG) and the 'unexplained yield gap' (UYG) according to Wairegi et al. (2010).

```
# decompose yg
data_ygd$IYG <- max(data_ygd$Y, na.rm=T) - data_ygd$y_pred_min
data_ygd$UYG <- data_ygd$y_pred_min - data_ygd$Y
```

Now that we have the IYG and UYG for each field, we can plot and interpret the results. The plot below shows the observed and the predicted cassava yields for the dataset of Fermont et al. (2009). Predicted yields correspond to the minimum yield predicted by all the boundary lines estimated in this workflow. The diagonal solid lines show the 1:1 and 1:2 lines, differentiating fields with no yield gaps (below the 1:1 line), fields with moderate yield gaps (between 1:1 and 1:2 lines), and fields with large yield gaps (above the 1:2 line). The table shows the number of fields affected by a given limiting factor.

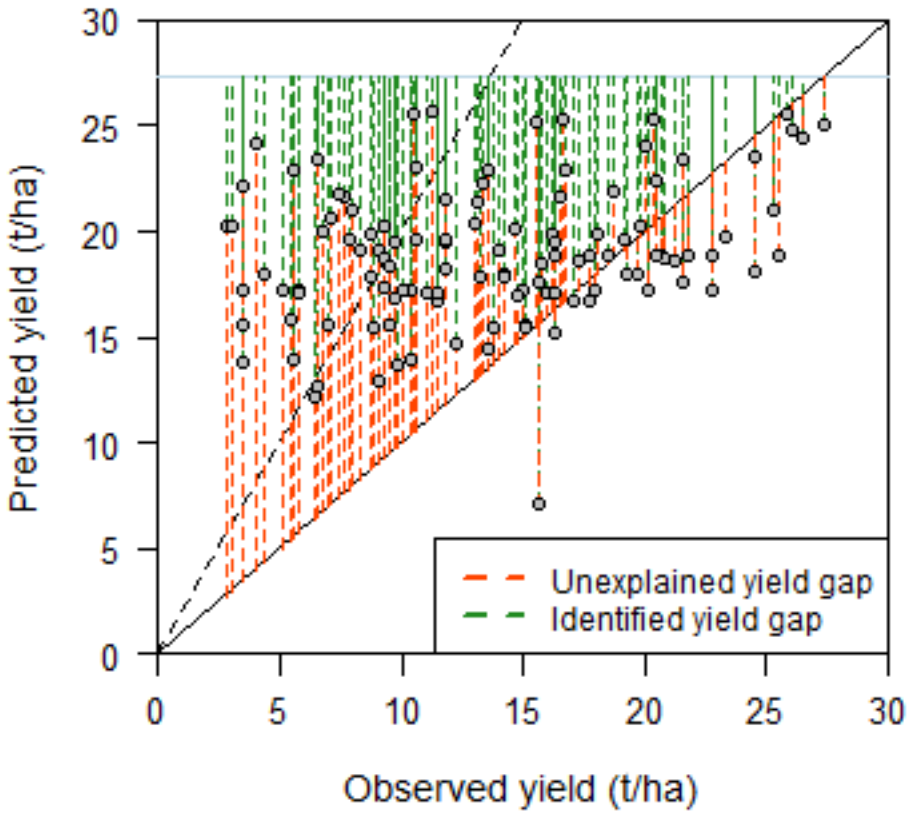
```
# plot yield gaps
par(mfrow=c(1,1), mar=c(5,5,1,1), las=1, xaxs='i', yaxs='i', cex.axis=1.1, cex.lab=1.
↪2)
plot(data_ygd$Y, data_ygd$y_pred_min, xlim=c(0, 30), ylim=c(0, 30), xlab='Observed␣
↪yield (t/ha)', ylab='Predicted yield (t/ha)')
points(data_ygd$Y, data_ygd$y_pred_min, pch=21, cex=1, col='black', bg='grey')
abline(a=0, b=1, lty=1)
abline(a=0, b=2, lty=2)
```




```
#
# see limiting factors
table(data_ygd$limiting_factor)
##
##      clay    P_soil   pH_soil totN_soil
##        16        34        29        39
```

We now plot the same figure as above, but highlighting the **IYG** (green segments) and the **UYG** (red segments). The larger the relative distance of the red line in comparison to the green one, the greater the fraction of total yield gap that was explained by the production factors considered in the analysis. Please note the maximum yield is assumed to be the maximum observed yield (upper horizontal line); yet, such value should be replaced by the respective potential or water-limited yield for irrigated or rainfed crops.

```
# plot IYG and UYG
par(mfrow=c(1,1), mar=c(5,5,1,1), las=1, xaxs='i', yaxs='i', cex.axis=1.1, cex.lab=1.
↪2)
plot(data_ygd$Y, data_ygd$y_pred_min, xlim=c(0, 30), ylim=c(0, 30), xlab='Observed␣
↪yield (t/ha)', ylab='Predicted yield (t/ha)')
abline(a=0, b=1, lty=1)
abline(a=0, b=2, lty=2)
abline(h=max(data_ygd$Y), lty=1, col=2, lwd=1.5)
segments(x0=data_ygd$Y, x1=data_ygd$Y, y0=data_ygd$y_pred_min, y1=data_ygd$y_pred_
↪min+data_ygd$IYG, lwd=0.7, lty=2, col='forestgreen')
segments(x0=data_ygd$Y, x1=data_ygd$Y, y0=data_ygd$y_pred_min-data_ygd$UYG, y1=data_
↪ygd$y_pred_min, lwd=0.7, lty=2, col='orangered')
points(data_ygd$Y, data_ygd$y_pred_min, pch=21, cex=1, col='black', bg='grey')
legend('bottomright', legend=c('Unexplained yield gap', 'Identified yield gap'),␣
↪lty=2, lwd=2, col=c('orangered', 'forestgreen'))
```



4.1.9 Recommendations

The main **advantages** of boundary lines for yield gap analysis can be summarized as follows:

- The method is empirically based and of easy interpretation to agronomists, allowing for reproducible research.
- Robust statistical techniques can be used to fit boundary lines to farmer field data with little computation requirements.
- A minimum of two variables are needed to apply the method (i.e., crop yield plus an additional explaining variable).
- The approach does not require a large dataset, if biophysical and management conditions are properly captured with data collection.
- Our experience indicates that sample size should be in the range of hundreds of field-year observations when dealing with homogeneous biophysical conditions.

The main **disadvantages** of boundary line anaysis can be summarized as follows:

- The method requires a thorough control of outliers, particularly on crop yield. Thus, the error of measured yields must be minimal. If this cannot be ensured then boundary points should be identified as the top 5th quantile of actual yields, rather than as the maximum yield, within each group of the independent variable.
- The interpretability of the data cannot be purely statistical as the patterns of the 'splines' need to be contextualized through agronomic knowledge.
- Independent variables must cover a wide range of variation, i.e., they must show a considerable range of values and accommodate a large variation in crop yield after being segmented in smaller groups.
- The segmentation of x-variables into groups with a similar number of observations is somewhat arbitrary. The influence of such decision on final results must be checked.
- Independent variables with discrete distributions (or with a lot of zero values) cannot be considered in this type of analysis. These may still be used in partitioning the dataset into homogeneous groups (e.g., irrigated vs. rainfed) depending on the sample of each subset.

4.1.10 Acknowledgments

We thank Ken Giller (WUR-PPS) and Piet van Asten (Olam International) for sharing the original data showcased in this workflow. The development of this notebook was possible thanks to the financial support from the OneCGIAR initiative on *Excellence in Agronomy*. For further support and questions on how to implement this workflow to other data sets, please contact J.V. Silva (**j.silva@cgiar.org**).

4.2 Workflow for Miti et al. (in prep.)

Chawezi Miti, University of Nottingham

4.2.1 Introduction

The censored bivariate normal model (Lark and Milne, 2016) is a statistical method of fitting the boundary line. It fits the boundary line using the maximum likelihood approach on a censored bivariate distribution. This removes the subjectivity of selecting boundary points to which the boundary line is fitted. It also gives evidence for presence of a boundary in the data set. This method has been previously used to fit boundary lines to data on nitrous oxide emission as a function of soil moisture (Lark and Milne, 2016) and, Wheat yield as a function of Nutrient concentration (Lark et al., 2020). An R package, called 'BLA', contains exploratory and boundary line fitting functions that can be used in the process of fitting a boundary line to data using the censored bivariate normal model. This workflow provides a step by step process for fitting the boundary line with this approach.

4.2.2 Installation of BLA package

Below, we need install the R packages required to run this workflow.

```
# problem with this package, install from source
install.packages('https://cran.r-project.org/src/contrib/aplpack_1.3.5.tar.gz',␣
↪repos=NULL, type="source", quiet=T)
library(aplpack)
#
# package names
packages <- c("MASS", "mvtnorm")
#
# install packages
installed_packages <- packages %in% rownames(installed.packages())
if(any(installed_packages == FALSE)){
  install.packages(packages[!installed_packages], repos="http://cran.us.r-project.org
↪", quiet=T)}
#
# load packages
invisible(lapply(packages, function(x) suppressMessages(require(x, character.only=T,␣
↪quietly=T, warn.conflicts=F))))
```

You also need to install the 'BLA' package (under development) from GitHub:

```
file <- 'https://raw.githubusercontent.com/jvasco323/eia-yg-training-ppt/master/BLA_1.
↪0.6.zip'
install.packages(file, repos=NULL)
## Installing package into 'C:/Users/JSILVA/AppData/Local/R/win-library/4.3'
## (as 'lib' is unspecified)
## package 'BLA' successfully unpacked and MD5 sums checked
library(BLA)
```

4.2.3 Load data set

The data set used for this illustration in this workflow consists of wheat yield and soil phosphorus data. Boundary line analysis was applied this data set in a previous study by Lark et al., (2020).

The data can be loaded into R software using the chunk of code below:

```
file <- 'https://raw.githubusercontent.com/jvasco323/eia-yg-training-ppt/master/my_
↪data_trimed.csv'
data <- read.csv(url(file), header=TRUE)
head(data)
##   yield.t.ha. Phosphorus_ppm
## 1     9.55787             10
## 2     8.88999              9
## 3    11.88140             12
## 4    10.35740             10
## 5    12.62530             14
## 6    11.64170             11
```

We wish to fit a boundary line to wheat yield data as function of soil phosphorus concentration.

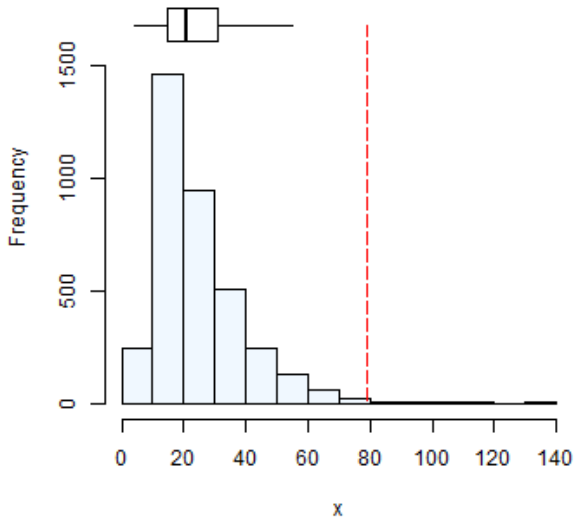
4.2.4 Data exploration

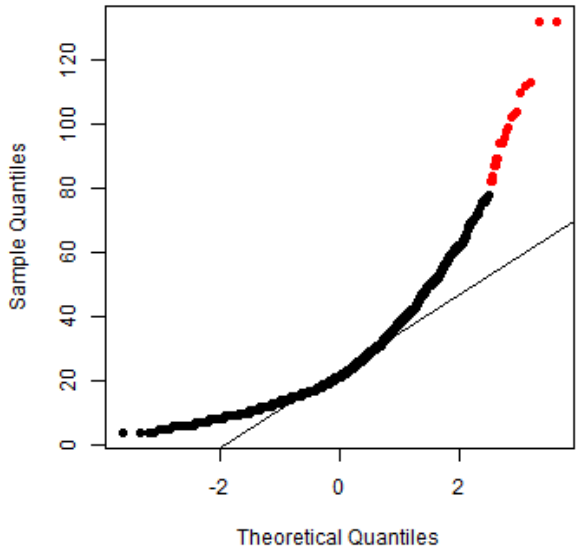
Exploratory analysis is an essential initial step in fitting a boundary line to data. This step ensures that the assumptions for the censored bivariate normal model are met. Three exploratory procedures are performed on the data which include (1) testing for normality of variables, (2) removal of outliers, and (3) testing for evidence of boundary in the data.

1. Testing for normality of independent and dependent variable

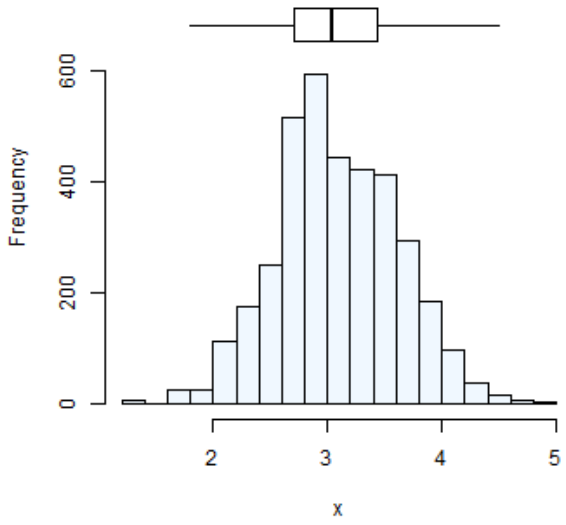
Boundary line model fitting using *cbvn* requires that the independent (x) and dependent (y) variables are normally distributed. The *summa()* function gives indices of the distribution of a variable. A variable is assumed to be normally distributed if the skewness index is between -1 and 1. The *summaplot()* function provides a visual description of the data distribution.

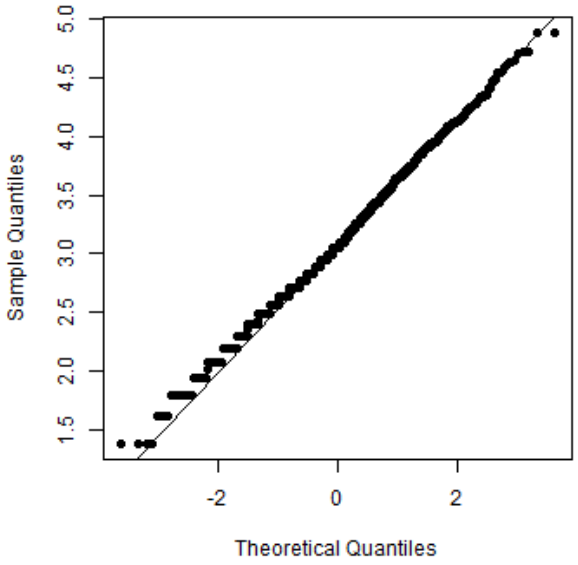
```
x <- data$Phosphorus_ppm
y <- data$yield.t.ha.
#
summaplot(x) # histogram and qqplot for distribution of x
```



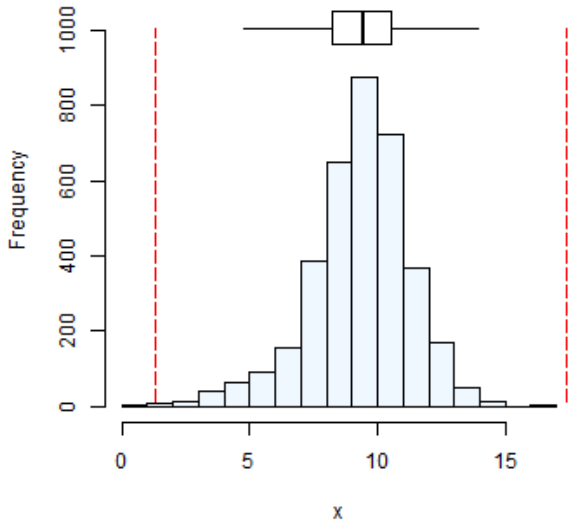


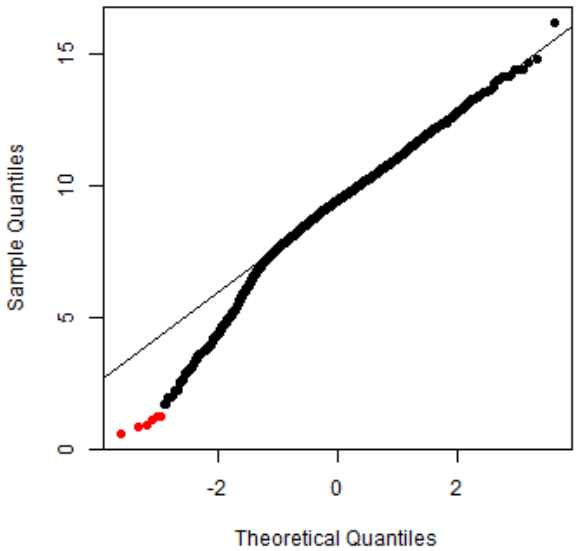

```
summa(x) # gives indices for skewness; the skewness is 1.84 which is outside the␣
↪normal distribution range of -1 to 1: data transformation is thus required
##          Mean Median Quartile.1 Quartile.3 Variance      SD Skewness
## [1,] 25.22307     21         15         31 203.4532 14.2637 1.786871
##      Octile skewness Kurtosis No. outliers
## [1,]       0.3793103 5.303787           22
summa(log(x))
##          Mean   Median Quartile.1 Quartile.3  Variance        SD  Skewness
## [1,] 3.091277 3.044522    2.70805   3.433987 0.2681185 0.5178016 0.1260458
##      Octile skewness   Kurtosis No. outliers
## [1,]      0.08906724 -0.1011968            0
summaplot(log(x))
```






```
#
# distribution of the y variable
summa(y) # gives indices for skewness; the skewness is -0.48 which is within the␣
↪normal distribution range of -1 to 1
##          Mean   Median Quartile.1 Quartile.3 Variance       SD   Skewness
## [1,] 9.290768 9.45873   8.231085    10.5264 3.820238 1.954543 -0.6748272
##      Octile skewness Kurtosis No. outliers
## [1,]     -0.07020916 1.306502            6
summaplot(y) # histogram and qqplot for distribution of y
```

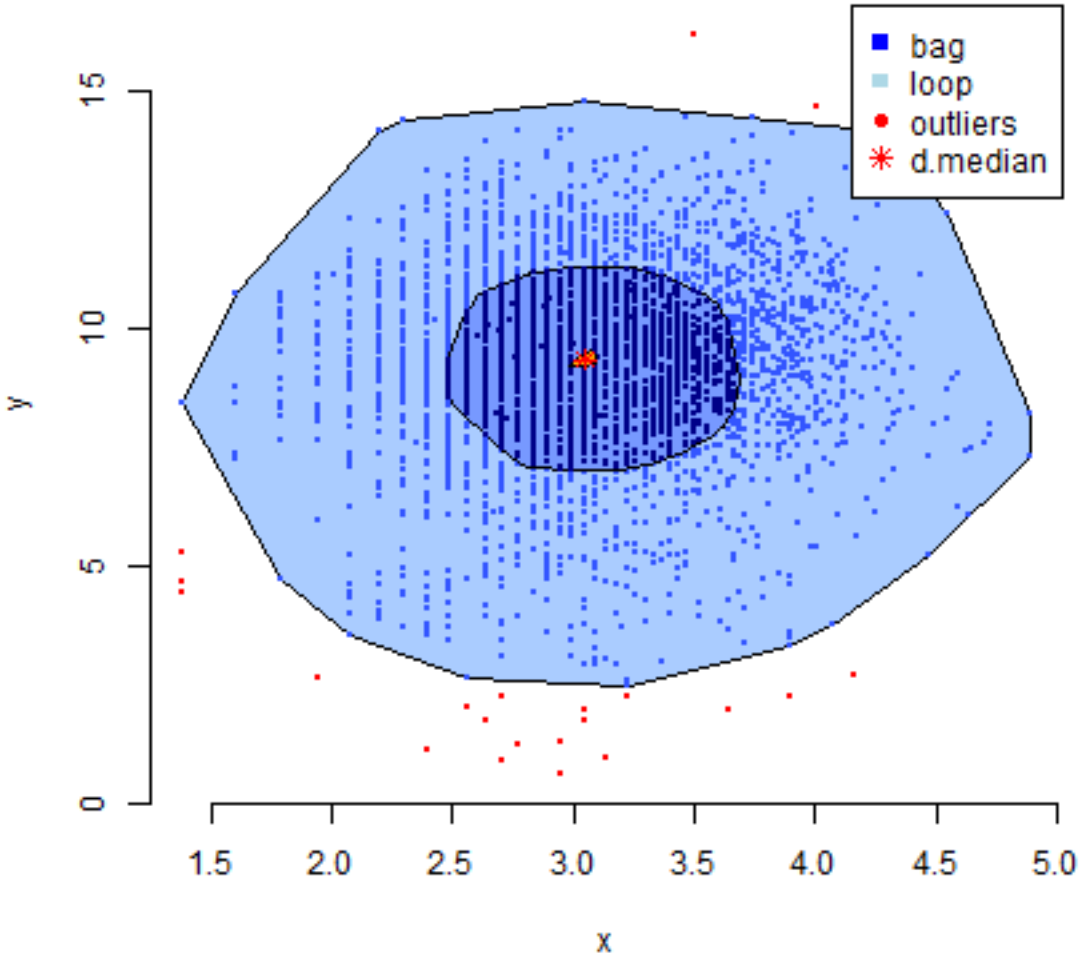




2. Removal of outliers

Boundary line analysis is sensitive to outlying values and hence it is required that they are identified and excluded prior to fitting a boundary line. A bag plot (a bivariate box plot) is used to identify outliers. The bag plot has four main components: (1) a depth median (equivalent to the median in a boxplot) which represents the center of the data set, (2) the bag which contains 50% of the data points (equiavalent to the interquartile range), (3) a 'fence' that separates probable outliers, and (4) a loop which contains points outside the bag which are not outliers.

```
x <- log(x) # since we requred a transformation
df <- data.frame(x,y)
bag <- bagplot(df,show.whiskers = FALSE)
legend("topright", c("bag","loop","outliers", "d.median"),
           pch = c(15,15,16,8),col=c("blue","lightblue","red","red"))
```




```
#
# combine data points from "bag" and within the loop
dat <- rbind(bag$pxy.bag,bag$pxy.outer)
#
# output is a matrix, we can pull out x and y variables for next stage
```

(continues on next page)

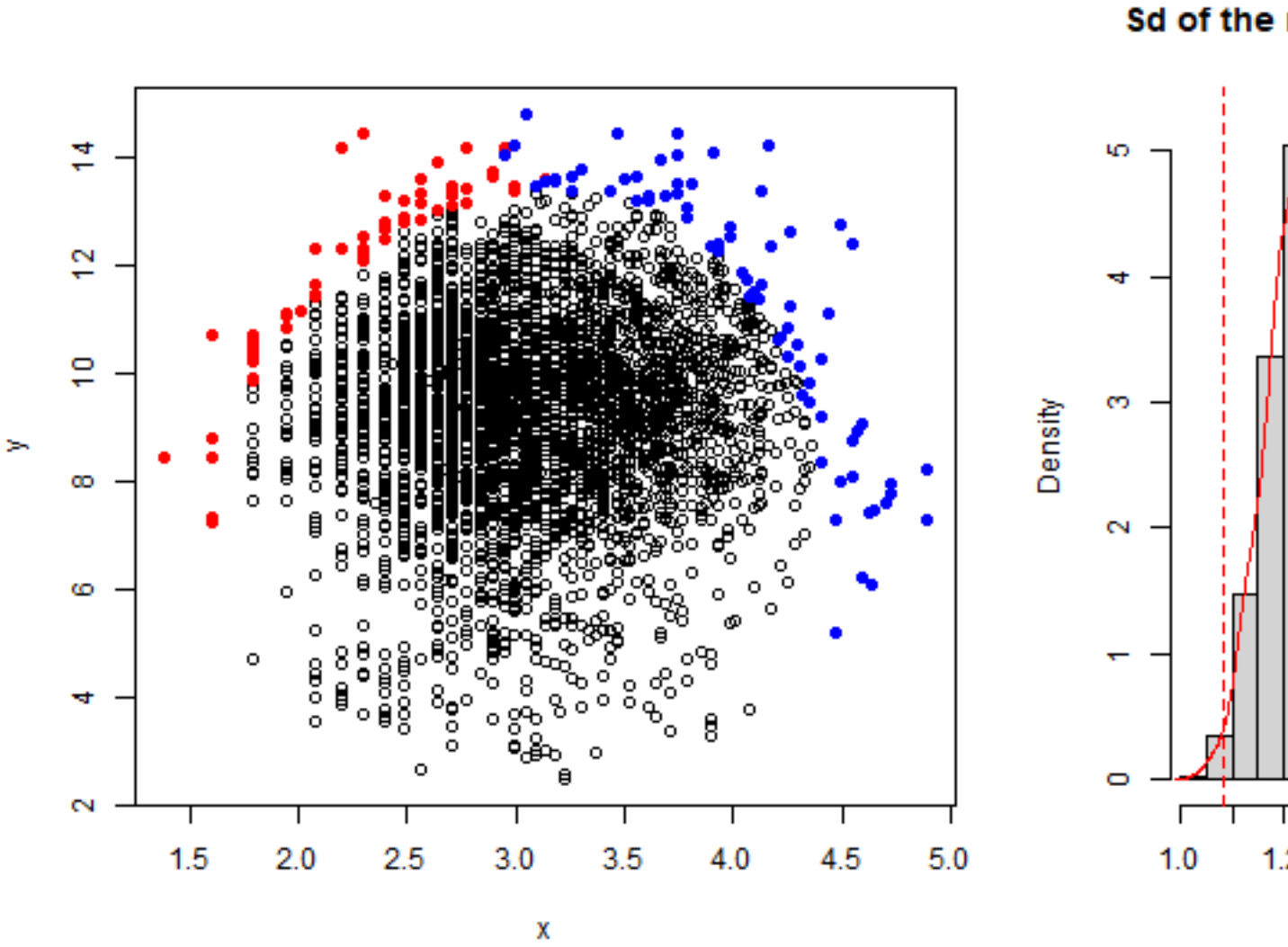
(continued from previous page)

```
x <- dat[,1]
y <- dat[,2]
```

3. Testing data for presence of boundary

It is important to note that not all data are suitable for boundary line analysis. Data used in boundary line analysis should exhibit some form of boundary on its edges (to satisfy the assumption that this is an actual response when other factors are not limiting). If a boundary exists in a data set, it is expected that points at the upper edges of the data cloud will be close to each other compared to a bivariate normally distributed data for which data points at the upper edges are only due to random error. A boundary can be assumed in a data set if there is evidence of clustering at the upper edges. The *expl.boundary() function,* which is based on the convex hull, can be used to access presence of boundary (Milne et al., 2006). This function checks probability of the observed clustering if it came from a bivariate normal distribution (p-value).

```
expl.boundary(x,y) # may take about 2 minutes to complete
```



```
##      Index Section    value
## 1       sd    Rise 1.082113
## 2       sd    Fall 1.195613
## 3 Mean sd    Rise 1.248132
## 4 Mean sd    Fall 1.360894
## 5 p_value    Rise 0.009000
## 6 p_value    Fall 0.023000
```

From the results, the probability (p-value) of the having observations close to each other in our data, assuming it follows a bivariate normal distribution, is less than 5%. Therefore, there is evidence of bounding effects in the data. Note that, in the plot, the data is split into right and left sections to get more information on the clustering nature of points.

4.2.5 Fitting the boundary line

The exploratory tests previously conducted provide evidence of a boundary in the data set, given that outliers were identified and excluded, and the variables *x* and *y* are normally distributed. We therefore, proceed to fit a boundary line model to the data set using the censored bivariate normal model. The *cbvn()* function fits the boundary line to the data. For more information about the arguments of this function, check:

```
?cbvn
```

Argument values for the function *cbvn()* need to be set. First, let's create a data-frame containing *x* and *y*:

```
vals <- data.frame(x,y) # this is an input dataframe containing the variables
```

Secondly, the *cbvn()* function requires initial starting values, *theta*, which are parameters of the boundary line (the censor) and the bivariate normal distribution. Starting values of the boundary line depend on the model that one wishes to fit to the data (see options in *?cbvn*). In this case, we shall fit a linear plateau model ('lp') and hence the parameters are the plateau value and the intercept and slope of the linear component. The boundary line start values can be obtained using the function *startValues()*. With a scatter plot of *y* against *x* active in the plot window in R, run the function *start.values(2)*, then click on the plot, the point you expect to be the lowest and the largest response of a linear model at the boundary.

```
# note this step will only work in R or Rstudio
plot(x,y)
startValues(n=2)
```

Parameters of the bivariate normal distribution include the means of the *x* and *y* variables, standard deviation of the *x* and *y* variables, and the correlation of *x* and *y*.

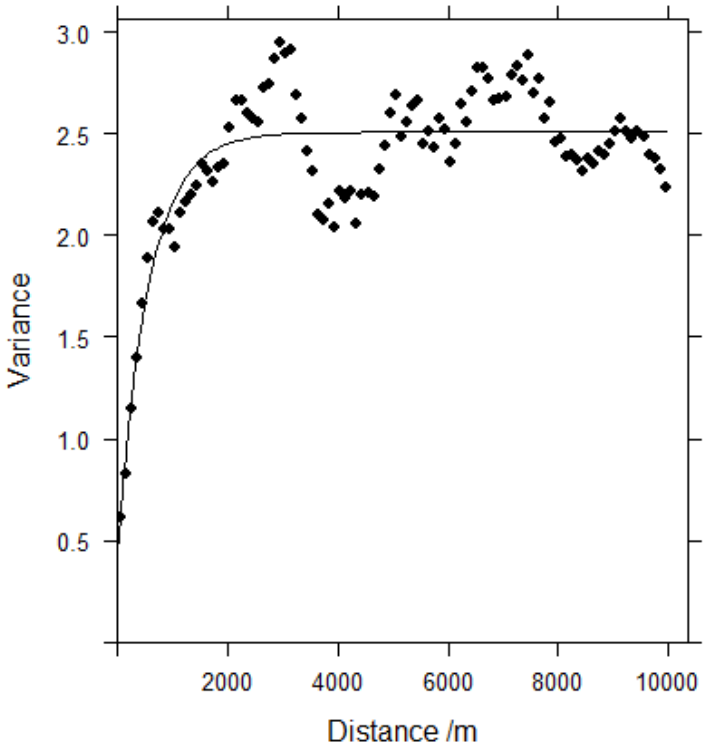
```
# required parameters
mean(x)
## [1] 3.092809
mean(y)
## [1] 9.324846
sd(x)
## [1] 0.5153963
sd(y)
## [1] 1.882962
cor(x,y)
## [1] 0.04729519
#
```

(continues on next page)

(continued from previous page)

```
# the parameters of the boundary line and the data can be combined in a vector theta
theta <- c(13.6, 4, 3, 3.13, 9.29, 0.5, 1.73, 0.03)
```

Another important argument is the standard deviation of the measurement error, *sigh*. This value can be obtained from standard deviation of repeated measurements of the sample if this is possible. However, in cases when this is not available, it can be estimated from the data. One option of estimation is to use a variogram if the location data (xy coordinates) is available. In this case, nugget variance which is the unexplained short distance variations can be taken as an estimate of the measurement error. The variogram below was obtained in the study for which the data used in this example is based.

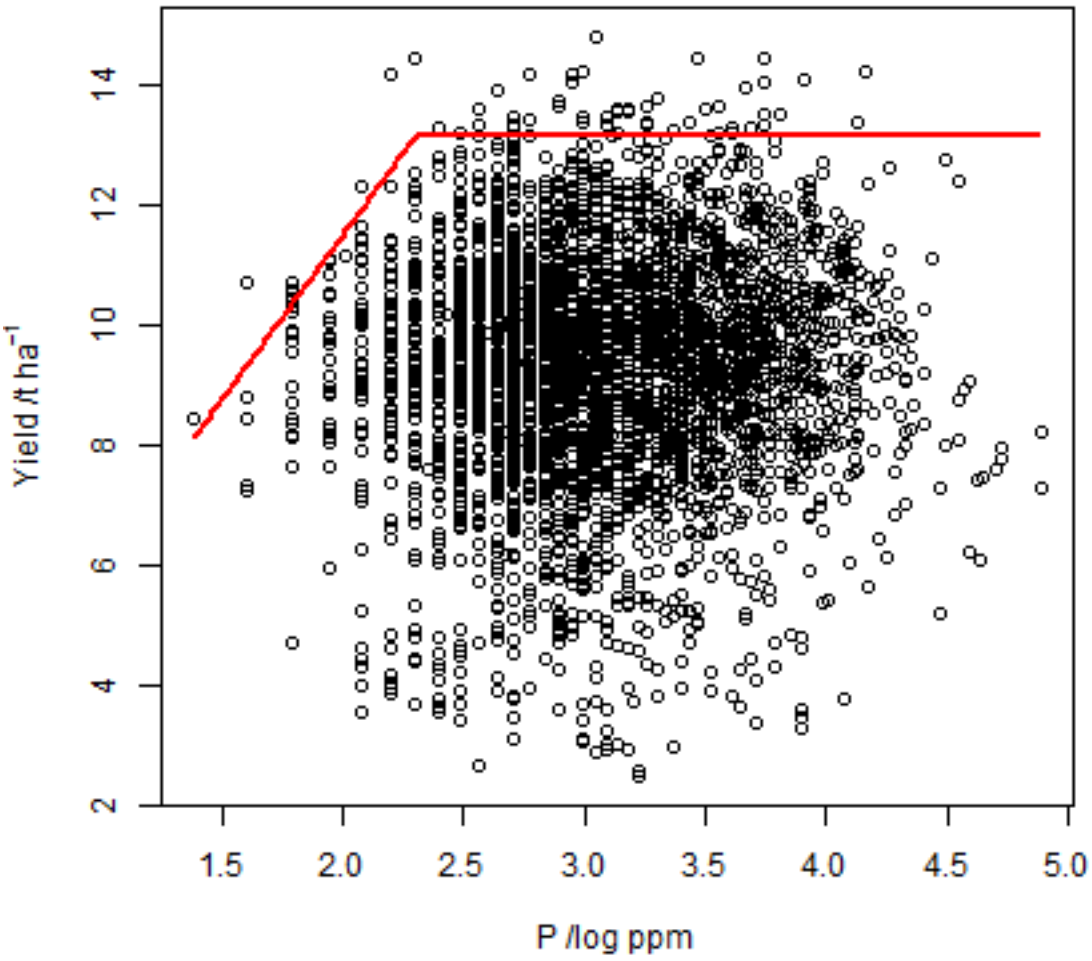


From the variogram, the nugget variance was found to be 0.435.

```
sigh <- sqrt(0.435) # standard deviation of the measurement error 0.66
```

All arguments are now set, the *cbvn()* can be fitted:

```
model1 <- cbvn(vals,theta,sigh,model = "lp", xlab=expression("P /log ppm"),␣
↪ylab=expression("Yield /t ha"^{-1}))
## Warning in sqrt(diag(covpar)): NaNs produced
```




```
model1
## $estimates
##          Estimate Standard error
## beta0 13.13820654    0.165731302
## beta1  0.51414702    0.100312067
## beta2  5.48249907            NaN
## mux    3.09278135    0.008601939
## muy    9.33714255    0.032071625
## sdx    0.51532725    0.006082414
## sdy    1.78781765    0.025362137
## rcorr  0.04415286    0.017695779
##
## $AIC
##
## constant max 20157.10
## mvn          20155.10
## BL           20139.98
```

Is the boundary line ideal for this data?

While fitting the BL model to the data, the *cbvn()* also fits a model with a constant boundary alone and also multivariate normal models with no boundary. From our output, the AIC value of the boundary line model is lower than that of the constant maximum and multivariate normal models. Therefore, the boundary line model is appropriate for this data set.

The parameters of the boundary line are obtained and hence can be used to predict the boundary yield for any given value of soil phosphorous, as per the chunk of code below.

```
yield_pred <- vector()
for(i in 1:length(data$P)){
  yield_pred[i] <- min(model1$estimates[2,1]+ model1$estimates[3,1]*log(data$P[i]),␣
↪model1$estimates[1,1])}
data$yield_pred <- yield_pred
head(data)
##   yield.t.ha. Phosphorus_ppm yield_pred
## 1     9.55787             10   13.13807
## 2     8.88999              9   12.56043
## 3    11.88140             12   13.13821
## 4    10.35740             10   13.13807
## 5    12.62530             14   13.13821
## 6    11.64170             11   13.13821
```

This section provides the workflow to conduct yield gap decomposition using boundary line analysis. Introductory slides can be downloaded here. You will learn (a) how to identify the boundary points, (b) how to fit a continuous function to the boundary points, (c) how to estimate the yield gap for single production factors, and (d) how to identify the most limiting factor, by analogy with von Liebig's law of the minimum, and decompose the yield gap for individual farm fields. Two workflows are available for this, one following Fermont et al. (2009) and another which is working progress by the University of Nottingham.

5 Stochastic frontier analysis

5.1 Workflow for Silva et al. (2017)

João Vasco Silva, CIMMYT-Zimbabwe

5.1.1 Introduction

Yield gap decomposition has been increasingly applied in agronomy to disentangle the impact of sub-optimal management on crop production and to identify agronomic measures to improve yields. To date, most applications refer to cereal crops (and some tuber and root crops) in a wide range of production systems worldwide, particularly in sub-Saharan Africa, South and Southeast Asia, and Northwest Europe. This notebook aims to formalize the R scripts used to decompose yield gaps across most of those applications making use of the framework introduced by Silva et al. (2017). Data collected by CIMMYT and EIAR for wheat in Ethiopia, previously used for yield gap analysis (Silva et al., 2021), are used here as an example. Before diving into the R scripts it is important to understand the key concepts and definitions involved in yield gap decomposition as these determine how the different yield levels and associated yield gaps are estimated.

The framework for yield gap decomposition described in this notebook considers four different yield levels (Silva et al., 2017). First, the **water-limited potential yield** (Yw) refers to the maximum yield that can be obtained under rainfed conditions in a well-defined, and relatively homogeneous, biophysical environment (van Ittersum et al., 2013). Yw

can be simulated with crop growth models or derived from field trials with non-limiting levels of nutrients and pests, diseases, and weeds fully controlled. Second, the **highest farmers' yield** (Y_{HF}) refer to the maximum yields (e.g., average above the 90th percentile of actual farmers' yields) observed in a representative sample of farmers sharing similar biophysical conditions (weather and soils) and technologies adopted (e.g., varieties). Third, the **technical efficient yield** (Y_{TEx}) comprises the maximum yield that can be achieved by farmers in a given production ecology for a given input level and can be computed using methods of frontier analysis in combination with concepts of production ecology (Silva et al., 2017). Finally, the **actual yield** (Ya) refers to the yield in farmers' fields as recorded in farm surveys.

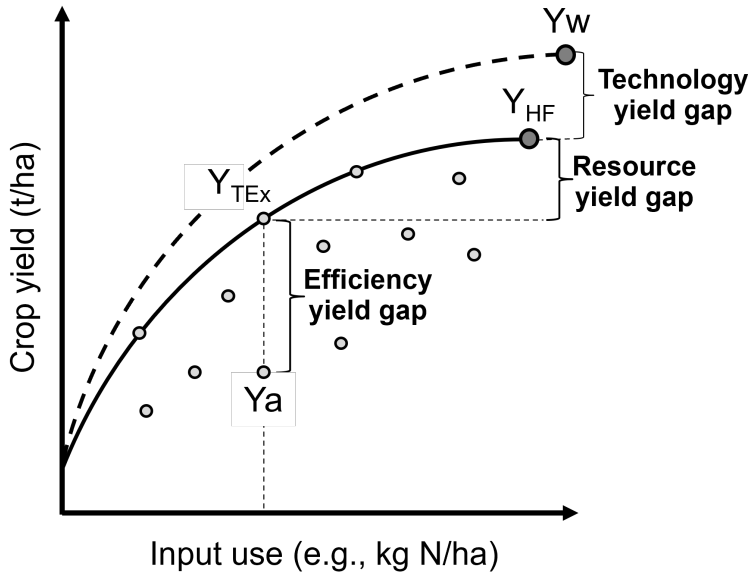


Figure 1. Visual illustration of the yield levels, and respective yield gaps, considered in the yield gap decomposition presented in this workflow. Source: Silva et al. (2017).

The total yield gap and three intermediate yield gaps can be estimated using the four yield levels previously described. The **total yield gap** is calculated as the difference between Yw and Ya. The **efficiency yield gap** is calculated as the difference between Y_{TEx} and Ya and it can be explained by crop management imperfections related to time, form, and/or space of the inputs applied. The **resource yield gap** is calculated as the difference between Y_{HF} and Y_{TEx} and captures the yield penalty due to a sub-optimal amount of inputs applied. Lastly, the **technology yield gap** is calculated as the difference between Yw and Y_{HF}, which can be explained by resource yield gaps of specific inputs and/or absence of certain technologies that would fully exploit the biophysical potential of the system. Please note Ya and the intermediate yield gaps can be expressed as a percentage of Yw, reflecting the current yield gap closure and the share of yield gap explained by each intermediate yield gap.

5.1.2 Load required R packages

First, we need to load the R packages needed to run this workflow.

```
# package names
packages <- c("frontier", "dplyr", "tidyr", "knitr", "car", "RColorBrewer")
#
# install packages
installed_packages <- packages %in% rownames(installed.packages())
if(any(installed_packages == FALSE)){
  install.packages(packages[!installed_packages], repos="http://cran.us.r-project.org
↪", quiet=T)}
```

(continues on next page)

(continued from previous page)

```
#
# load packages
invisible(lapply(packages, function(x) suppressMessages(require(x, character.only=T,␣
↪quietly=T, warn.conflicts=F))))
```

5.1.3 Farmer field data

The first step is to load the farmer field data to be used for yield gap analysis. The data contain (a) primary data on self-reported Ya, management practices, and biophysical conditions at field level obtained through a household survey, and (b) secondary data obtained from spatial products using the GPS coordinates of the individual households. The household survey is a panel of households over two growing seasons (2009 and 2013). Type and sources of secondary data include: climate zones from the Global Yield Gap Atlas (van Wart et al., 2013), soil water properties from AfSIS-GYGA (Leenaars et al., 2017), agro-ecological zones for Ethiopia, and farming systems classification for Africa.

```
# read .csv file with data
file <- 'https://raw.githubusercontent.com/jvasco323/EiA_YGD_workflow/main/data-wheat-
↪ethiopia.csv'
data <- read.csv(url(file))
#
# list variables of interest
str(data)
## 'data.frame':    3783 obs. of  42 variables:
##  $ zone                : chr  "WEST SHOA" "WEST SHOA" "WEST SHOA" "WEST SHOA" ...
##  $ zone_new            : chr  "W SHOA" "W SHOA" "W SHOA" "W SHOA" ...
##  $ farming_system      : chr  "6. Highland mixed farming system" "6. Highland␣
↪mixed farming system" "6. Highland mixed farming system" "6. Highland mixed farming␣
↪system" ...
##  $ aez                 : chr  "M2" "M2" "M2" "M2" ...
##  $ year                : int  2009 2009 2009 2009 2009 2009 2009 2009 2009 2009 ..
↪.
##  $ season_year         : chr  "Meher_2009" "Meher_2009" "Meher_2009" "Meher_2009"␣
↪...
##  $ hhid                : int  1 11 11 16 17 270 281 285 292 515 ...
##  $ plotid              : int  5 2 2 5 2 2 6 6 4 2 ...
##  $ subplotid           : int  1 4 3 1 3 1 1 1 1 1 ...
##  $ subplotsize_ha      : num  0.5 0.5 0.0625 1 1.5 0.5 0.25 0.25 1.5 0.5 ...
##  $ subplot_own         : chr  "Rented-in" "Owned" "Owned" "Rented-in" ...
##  $ subplot_manager     : chr  "Man" "Man" "Man" "Man" ...
##  $ plotdist_min        : num  10 5 2 15 10 5 15 30 5 5 ...
##  $ crop                : chr  "Wheat_br" "Wheat_br" "Wheat_br" "Wheat_br" ...
##  $ gyga_cz             : int  5501 5501 5501 5501 5501 5501 5501 5501 5501 5501 ..
↪.
##  $ gyga_gdd            : num  6539 6539 6539 6539 6539 ...
##  $ gyga_tseas          : int  1208 1208 1208 1208 1208 1285 1200 1200 1200 1208 ..
↪.
##  $ seed_kgha           : num  206 250 192 64 33.3 ...
##  $ variety             : chr  "Landrace" "unknown" "unknown" "unknown" ...
##  $ gyga_ai             : num  6544 6544 6544 6544 6544 ...
##  $ gyga_av_water       : int  9 9 9 9 9 7 7 7 7 9 ...
##  $ soil_depth          : chr  "Deep" "Deep" "Medium" "Medium" ...
##  $ soil_slope          : chr  "Medium" "Steep" "Steep" "Flat" ...
##  $ waterlogging_yn     : chr  "No" "No" "No" "No" ...
```

(continues on next page)

(continued from previous page)

```
##  $ drought_yn           : chr  "No" "No" "Yes" "No" ...
##  $ soilwatercons_yn     : chr  "Yes" "Yes" "Yes" "No" ...
##  $ oxplough_freq        : int  5 4 4 4 8 4 3 5 4 5 ...
##  $ oxplough_freq_cat    : chr  ">Five" "Four" "Four" "Four" ...
##  $ soil_fertility       : chr  "Poor" "Poor" "Poor" "Poor" ...
##  $ nfert_kgha           : num  32.3 41.3 51.7 32.3 21.5 ...
##  $ pfert_kgha           : num  10.04 20.07 16.06 10.04 6.69 ...
##  $ manure_yn            : chr  "No" "No" "No" "No" ...
##  $ residues_yn          : chr  "No" "Yes" "Yes" "Yes" ...
##  $ previous_crop        : chr  "Cereal" "Cereal" "Cereal" "Cereal" ...
##  $ herb_lha             : num  0.4 2 4 1 0.2 ...
##  $ handweeding_persdayha: num  24 8 64 20 16 0 16 28 0 22 ...
##  $ weeding_freq         : int  1 1 1 1 1 0 2 1 1 1 ...
##  $ weeding_freq_cat     : chr  "One" "One" "One" "One" ...
##  $ pesticide_yn         : chr  "No" "No" "No" "No" ...
##  $ disease_incidence_yn : chr  "No" "Yes" "No" "Yes" ...
##  $ pest_incidence_yn    : chr  "Yes" "No" "No" "No" ...
##  $ yield_tha            : num  1.2 2.4 3.2 1 0.267 ...
```

5.1.4 Data manipulation

Some data transformations need to be done prior to the analysis. These include (a) re-leveling and re-classification of categorical variables, (b) log-transformation of continuous variables so that model coefficients can be interpreted as elasticities, and (c) fill or drop data not available. If needed, missing data for specific observations of a given variable can be filled with the average value for that variable. This is documented in the chunk of code below.

```
# create final data
data <- subset(data, yield_tha > 0)
data <- subset(data, residues_yn == "No" | residues_yn == "Yes")
data <- subset(data, soil_slope == "Flat" | soil_slope == "Medium" | soil_slope ==
↪"Steep")
data <- subset(data, zone_new != "")
data <- subset(data, oxplough_freq_cat == "<Two" |
                     oxplough_freq_cat == "Three" |
                     oxplough_freq_cat == "Four" |
                     oxplough_freq_cat == ">Five")
data <- subset(data, weeding_freq_cat == "None" |
                     weeding_freq_cat == "One" |
                     weeding_freq_cat == "Two" |
                     weeding_freq_cat == "Three+")
#
# fill NA values
data$seed_kgha[is.na(data$seed_kgha)] <- mean(data$seed_kgha, na.rm=T)
data$nfert_kgha[is.na(data$nfert_kgha)] <- 0
data$herb_lha[is.na(data$herb_lha)] <- 0
data$handweeding_persdayha[is.na(data$handweeding_persdayha)] <- 0
#
# reclassify categorical variables
data$variety = ifelse(data$variety != 'Landrace' &  data$variety != 'unknown',
↪'Improved', data$variety)
data$variety = ifelse(data$variety == 'Landrace', 'unknown', data$variety)
data$nfert_yn = ifelse(data$nfert_kgha == 0, 'N0', 'N+')
data$weeding_yn = ifelse(data$herb_lha == 0 & data$handweeding_persdayha == 0, 'No',
↪'Yes')
```

(continues on next page)

(continued from previous page)

```
#
# copy df with transformed data
data_new <- data
#
# replace 0 with small value for log-transformation
data_new[data_new == 0] = 0.0001
#
# log-transform continuous variables
vars1 <- c('gyga_gdd', 'gyga_tseas', 'seed_kgha', 'gyga_ai', 'gyga_av_water', 'nfert_
↪kgha', 'pfert_kgha',
           'herb_lha', 'handweeding_persdayha', 'yield_tha')
log_f <- function(x){log(x)}
data_new[,vars1] <- lapply(data_new[,vars1], log_f)
#
# set categorical variables to factor
vars2 <- c('farming_system', 'aez', 'zone_new', 'season_year', 'variety', 'soil_depth
↪', 'soil_fertility',
           'waterlogging_yn', 'drought_yn', 'soilwatercons_yn', 'manure_yn',
↪'residues_yn', 'previous_crop',
           'oxplough_freq_cat', 'weeding_yn', 'pesticide_yn', 'disease_incidence_yn',
↪'pest_incidence_yn')
data_new[,vars2] <- lapply(data_new[,vars2], factor)
```

5.1.5 Descriptive statistics

Descriptive statistics of the actual yield and (continuous) crop management variables used in the analysis for the 2 years of the survey are provided below. Actual yield of wheat across Ethiopia was on average 1.76 t/ha in 2009 and 1.77 t/ha in 2013. N and P application rates were on average 48 kg N/ha and 20 kg P/ha, respectively. Plot sizes were on average 0.45 ha in 2009 and 0.40 ha in 2013.

```
## Warning: There was 1 warning in `summarise()`.
##  In argument: `across(...)`.
##  In group 1: `year = 2009`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

Variable	Mean 2009	Mean 2013	StDev 2009	StDev 2013
Hand-weeding (person-day/ha)	21.84	24.28	30.46	35.46
Herbicide use (L/ha)	0.5	0.59	0.83	0.88
N application rate (kg N/ha)	48.07	48.99	40.53	32.04
P application rate (kg P/ha)	19.63	20.34	13.93	12.36
Seed rate (kg/ha)	192.88	195.43	79.85	95.66
Plot size (ha)	0.45	0.4	0.39	0.3
Actual wheat yield (t/ha)	1.76	1.77	1.13	1.09

5.1.6 Efficiency yield gap

A stochastic frontier model is needed to estimate Y_{TEx} and the efficiency yield gap. Stochastic frontier analysis is an econometric technique widely used for benchmarking different production units (Kumbhakar & Lovell, 2000). Stochastic frontiers differentiate two random errors: technical inefficiency, *ui* (translated in agronomic terms as the efficiency yield gap), and random noise, *vi*, hence separating the effects of sub-optimal crop management (*ui*) from random noise (*vi*) in the response variable. For yield gap analysis, the variables used to estimate stochastic frontiers need to be selected based on principles of production ecology (van Ittersum & Rabbinge, 1997) to capture the impact of growth-defining, -limiting, and -reducing factors on crop yield.

It is important to fit an **ordinary least squares (OLS) regression** prior to fit a stochastic frontier. Although both models have the same structure, and should exhibit similar signs and effect sizes for the different variables, OLS regressions allow the computation of Variance Inflation Factors (VIF) and hence to assess multicolinearity between variables. As a rule-of-thumb, VIF values above 10 indicate multicolinearity between variables, which is helpful to screen the final set of variables to be included in the model (see: https://en.wikipedia.org/wiki/Variance_inflation_factor). Please note the VIF values and parameter estimates are not shown, but can be obtained by removing the symbol # in the chunk of code below.

```
# fit ols regression model
ols <-
  lm(yield_tha ~
     season_year + gyga_gdd + gyga_tseas + seed_kgha + variety +
     gyga_ai + gyga_av_water + soil_depth + soil_fertility + waterlogging_yn +␣
↪drought_yn + soilwatercons_yn +
     nfert_kgha + manure_yn + residues_yn + previous_crop + oxplough_freq_cat +
     herb_lha + handweeding_persdayha + weeding_yn + pesticide_yn + disease_incidence_
↪yn + pest_incidence_yn,
     data=data_new)
#
# check vif values
# vif(ols)
#
# see parameter estimates
# summary(ols)
```

The OLS regression fitted above can now be fitted as a **stochastic frontier model with a Cobb-Douglas functional form**. The Cobb-Douglas functional form takes only the first-order terms and is thus the simplest model that can be fitted to the data. Two functions from the R package *frontier* are used here: (1) the function *sfa()* is used to estimate model parameters and, (2) the function *efficiencies()* is used to estimate the technical efficiency scores (or the equivalent efficiency yield gap in agronomy). Further information about the theoretical background of these functions can be found in Battese & Coelli (1992). This is illustrated in the chunk of code below, which results are the same as those presented in Table 3 of Silva et al. (2021) for the pooled data. Please refer to that manuscript for further interpretation of model coefficients.

```
# fit cobb-douglas stochastic frontier
sfa_cd <-
  sfa(yield_tha ~
    season_year + gyga_gdd + gyga_tseas + seed_kgha + variety +
    gyga_ai + gyga_av_water + soil_depth + soil_fertility + waterlogging_yn + drought_
↪yn + soilwatercons_yn +
    nfert_kgha + manure_yn + residues_yn + previous_crop + oxplough_freq_cat +
    herb_lha + handweeding_persdayha + weeding_yn + pesticide_yn + disease_incidence_
↪yn + pest_incidence_yn,
    data=data_new)
#
# add technical efficiency score to data frame
```

(continues on next page)

(continued from previous page)

```
data_new$te_score_cd = efficiencies(sfa_cd, asInData=T)
#
# see parameter estimates
summary(sfa_cd)
## Error Components Frontier (see Battese & Coelli 1992)
## Inefficiency decreases the endogenous variable (as in a production function)
## The dependent variable is logged
## Iterative ML estimation terminated after 33 iterations:
## log likelihood values and parameters of two successive iterations
## are within the tolerance limit
##
## final maximum likelihood estimates
##                            Estimate Std. Error  z value  Pr(>|z|)
## (Intercept)               9.7733888  1.1050740   8.8441 < 2.2e-16 ***
## season_yearMeher_2013    -0.0539923  0.0264810  -2.0389  0.041460 *
## gyga_gdd                 -0.5840446  0.0909876  -6.4189 1.372e-10 ***
## gyga_tseas               -0.3252022  0.0525343  -6.1903 6.006e-10 ***
## seed_kgha                 0.0976523  0.0123652   7.8974 2.849e-15 ***
## varietyunknown           -0.0018340  0.0231824  -0.0791  0.936944
## gyga_ai                  -0.3311850  0.0594151  -5.5741 2.488e-08 ***
## gyga_av_water            -0.0080901  0.0412886  -0.1959  0.844658
## soil_depthMedium         -0.0742006  0.0232508  -3.1913  0.001416 **
## soil_depthShallow        -0.0812014  0.0291497  -2.7857  0.005342 **
## soil_fertilityMedium     -0.0568591  0.0203025  -2.8006  0.005101 **
## soil_fertilityPoor       -0.1627704  0.0309154  -5.2650 1.402e-07 ***
## waterlogging_ynYes       -0.3472614  0.0379737  -9.1448 < 2.2e-16 ***
## drought_ynYes            -0.4474776  0.0456593  -9.8004 < 2.2e-16 ***
## soilwatercons_ynYes       0.0585665  0.0275969   2.1222  0.033820 *
## nfert_kgha                0.2723653  0.0126521  21.5272 < 2.2e-16 ***
## manure_ynYes              0.0373056  0.0261372   1.4273  0.153494
## residues_ynYes            0.0316401  0.0270959   1.1677  0.242925
## previous_cropLegume       0.0220438  0.0244585   0.9013  0.367442
## previous_cropOther        0.1230667  0.0264684   4.6496 3.326e-06 ***
## oxplough_freq_cat>Five    0.0534308  0.0580652   0.9202  0.357475
## oxplough_freq_catFour    -0.0126670  0.0569799  -0.2223  0.824075
## oxplough_freq_catThree   -0.1015532  0.0573738  -1.7700  0.076722 .
## herb_lha                  0.0134322  0.0029268   4.5894 4.446e-06 ***
## handweeding_persdayha    -0.0038850  0.0020551  -1.8905  0.058697 .
## weeding_ynYes             0.0369644  0.0496531   0.7445  0.456602
## pesticide_ynYes           0.1208168  0.0495391   2.4388  0.014735 *
## disease_incidence_ynYes -0.3161391  0.0314308 -10.0582 < 2.2e-16 ***
## pest_incidence_ynYes     -0.0857481  0.0733460  -1.1691  0.242367
## sigmaSq                   0.5977253  0.0250709  23.8414 < 2.2e-16 ***
## gamma                     0.7421739  0.0246107  30.1565 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## log likelihood value: -3066.353
##
## cross-sectional data
## total number of observations = 3694
##
## mean efficiency: 0.633058
```

The simple Cobb-Douglas model fitted above can be made more complex by adding second-order terms in what is known as a **stochastic frontier model with a translog functional form**. This functional form is most flexible as it considers non-linear relations between variables (squared terms and interactions). Yet, the translog functional form should only be fitted last to assess non-linear effects on crop yield as the large number of parameters make it difficult

to interpret. The functions *sfa()* and *efficiencies()* are used again to estimate model parameters and technical efficiency scores, respectively. Please note this output is not displayed given the large number of parameters involved. To do so, remove the symbol # in the chunk of code below.

```
# fit translog stochastic frontier
sfa_tl <-
  sfa(yield_tha ~

   # 1st order terms (linear)
   season_year + gyga_gdd + gyga_tseas + seed_kgha + variety +
   gyga_ai + gyga_av_water + soil_depth + soil_fertility + waterlogging_yn + drought_
↪yn + soilwatercons_yn +
   nfert_kgha + manure_yn + residues_yn + previous_crop + oxplough_freq_cat +
   herb_lha + handweeding_persdayha + weeding_yn + pesticide_yn + disease_incidence_
↪yn + pest_incidence_yn +

   # 2nd order terms (squared)
   I(0.5*gyga_gdd^2) + I(0.5*gyga_tseas^2) + I(0.5*seed_kgha^2) +
   I(0.5*gyga_ai^2) + I(0.5*gyga_av_water^2) +
   I(0.5*nfert_kgha^2) +
   I(0.5*herb_lha^2) + I(0.5*handweeding_persdayha^2) +

   # 2nd order terms (interactions)
   I(gyga_gdd*gyga_tseas) + I(gyga_gdd*seed_kgha) + I(gyga_gdd*gyga_ai) + I(gyga_
↪gdd*gyga_av_water) +
     I(gyga_gdd*nfert_kgha) + I(gyga_gdd*herb_lha) + I(gyga_gdd*handweeding_
↪persdayha) +
   I(gyga_tseas*seed_kgha) + I(gyga_tseas*gyga_ai) + I(gyga_tseas*gyga_av_water) +␣
↪I(gyga_tseas*nfert_kgha) +
     I(gyga_tseas*herb_lha) + I(gyga_tseas*handweeding_persdayha) +
   I(seed_kgha*gyga_ai) + I(seed_kgha*gyga_av_water) + I(seed_kgha*nfert_kgha) +␣
↪I(seed_kgha*herb_lha) +
     I(seed_kgha*handweeding_persdayha) +
   I(gyga_ai*gyga_av_water) + I(gyga_ai*nfert_kgha) + I(gyga_ai*herb_lha) + I(gyga_
↪ai*handweeding_persdayha) +
   I(gyga_av_water*nfert_kgha) + I(gyga_av_water*herb_lha) + I(gyga_av_
↪water*handweeding_persdayha) +
   I(nfert_kgha*herb_lha) + I(nfert_kgha*handweeding_persdayha) +
   I(herb_lha*handweeding_persdayha),
   data=data_new)
#
# add technical efficiency score to data frame
data_new$te_score_tl = efficiencies(sfa_tl, asInData=T)
#
# see parameter estimates
# summary(sfa_tl)
```

The two chunks of code above added two new columns to the original data frame, namely **te_score_cd** and **te_score_tl**. These technical efficiency scores range between 0 and 1 and indicate how much extra yield could have been produced for the observed level of inputs. For instance, if the technical efficiency score is equal to 0.2 for a specific field, then that field only produced 20% of what it could have produced with the level of inputs it received. The efficiency yield gap is the agronomic equivalent of technical inefficiency when variables used in the stochastic frontier analysis are selected based on concepts of production ecology. Thus, Y_{TEx} and the efficiency yield gap can be estimated from the technical efficiency scores as follows.

```
# estimate efficiency yield gap (%)
data_new['efficiency_yg'] = 100 - (data_new['te_score_cd'] * 100)
```

(continues on next page)

(continued from previous page)

```
#
# select relevant columns
data_new <- data_new[c('zone_new', 'season_year', 'hhid', 'plotid', 'subplotid', 'te_
↪score_cd', 'te_score_tl',
                       'efficiency_yg')]
#
# merge the new columns to original data frame
data <- merge(data, data_new, by=c('zone_new', 'season_year', 'hhid', 'subplotid'),␣
↪all.x=T)
#
# estimate technical efficiency yield (t/ha)
data['ytex_tha'] = data['yield_tha'] / data['te_score_cd']
```

5.1.7 Resource yield gap

Before calculating Y_{HF}, it is useful to categorize the farm-fields into highest-, average-, and lowest-yielding based on the distribution of the actual yield. Highest-yielding fields are identified as the observations above the 90th percentile of Ya and the highest farmers' yields (Y_{HF}) were computed as the mean Ya for these fields. Similarly, the lowest-yielding fields were identified as the observations below the 10th percentile of Ya (Y_{LF}), and the average-yielding fields as the observations between the 10th and the 90th percentile of Ya (Y_{AF}).

The field classification described above needs to be done for a **given biophysical unit**, composed in this case of a unique year x climate zone x soil fertility combination, to avoid confounding between environmental conditions and crop management. For wheat in Ethiopia, 'year' refers to the Meher seasons of 2009 and 2013, 'climate zone' refers to the units included in the climate delineation scheme of GYGA, and 'soil fertility' refers to farmers' own assessment of the fertility of their soil. Variety type was not considered in the field classification because there was no significant yield difference between variety types (see results of the Cobb-Douglas stochastic frontier model above), but it should be considered otherwise.

```
# create an empty data frame
data_final <- data.frame()
#
# create loop per year
for(yr in unique(data$year)){
  subset_year <- subset(data, year == yr)
  #
  # create loop per climate zone
  for(cz in unique(subset_year$gyga_cz)){
    subset_cz <- subset(subset_year, gyga_cz == cz)
    #
    # create loop per soil type
    for(soil in unique(subset_cz$soil_fertility)){
      subset_soil <- subset(subset_cz, soil_fertility == soil)

      # create column with field class based on yield distribution
      subset_soil$field_class <- ifelse(subset_soil$yield_tha >= quantile(subset_soil
↪$yield_tha, 0.90),
                                        'YHF', '')
      subset_soil$field_class <- ifelse(subset_soil$yield_tha <= quantile(subset_soil
↪$yield_tha, 0.10),
                                        'YLF', subset_soil$field_class)
      subset_soil$field_class <- ifelse(subset_soil$yield_tha > quantile(subset_soil
↪$yield_tha, 0.10) &
```

(continues on next page)

(continued from previous page)

```
                                                subset_soil$yield_tha < quantile(subset_
↪soil$yield_tha, 0.90),
                                                'YAF', subset_soil$field_class)
      #
      # subset highest yielding fields only
      yhf <- subset(subset_soil, field_class == 'YHF')
      #
      # add column with yhf in t/ha to data frame
      subset_soil['yhf_tha'] <- mean(yhf$yield_tha, na.rm=T)
      #
      # bind all individual fields into single data frame
      data_final <- rbind(data_final, subset_soil)
}}}
```

The chunk of code above implements the classification of fields as highest-, average-, and lowest-yielding fields and the estimation of Y_{HF} for each field in the data set. The variables used for this classification (i.e., year, climate zone, and soil fertility class) were the most suitable for this specific example, meaning it is possible to use other types of variables to control for differences in biophysical conditions across fields (e.g., landscape position or slope), which will depend on the data set and cropping systems at stake.

5.1.8 Technology yield gap

The water-limited yield (Yw) is the yield benchmark for rainfed crops (van Ittersum et al., 2013), which is the case of wheat in Ethiopia. Spatial-explicit data on Yw can be obtained from different sources and a **tier-approach** is proposed for that. The most preferred data source on yield ceilings is GYGA from which data can be sourced through http://www.yieldgap.org (an API to GYGA can also be created on a demand basis). If the required data are not available in GYGA, yield ceilings can be simulated with crop models or obtained through literature review (least preferred option). Simulated yields in GYGA refer to most recently released high-yield crop cultivars, grown in pure stands. Yet, yield ceilings should be simulated for different varieties when data are available and different varieties are known to have different yield potential - see Silva et al. (2022) for an example of how to consider yield gaps due to variety choice in the yield gap decomposition framework used here.

The chunk of code below combines the water-limited yield data, retrieved from GYGA based on the GPS coordinates of the surveyed households, and the main database containing all field level data. The steps needed to retrieve these data from the GYGA API are shown in a **companion notebook**. Please note the water-limited yields presented here are slightly different than those presented in Silva et al., (2021) due to different criteria used to retrieve these data in the companion script.

```
# load dataframe with yw data
file <- 'https://raw.githubusercontent.com/jvasco323/EiA_YGD_workflow/main/data-gps-
↪coordinates-final.csv'
yw_data <- read.csv(url(file))
yw_data <- yw_data[c('hhid', 'GYGA_CZ', 'Yw_average', 'Yw_2009', 'Yw_2013')]
yw_data <- unique(yw_data)
#
# merge yw data with the rest of the data
data_final <- merge(data_final, yw_data, by='hhid', all.x=T)
#
# get yw per field
data_final$yw_tha <- ifelse(data_final$year == 2009, data_final$Yw_2009, data_final
↪$Yw_2013)
#
```

(continues on next page)

(continued from previous page)

```
# summarize of yw data
summary_yw <- unique(data_final[c('GYGA_CZ', 'Yw_average', 'Yw_2009', 'Yw_2013')])
summary_yw <- aggregate(summary_yw[c(2:4)], by=list('GYGA_CZ'=summary_yw$GYGA_CZ),␣
↪FUN=mean)
summary_yw[c(2:4)] <- round(summary_yw[c(2:3)], 1)
colnames(summary_yw)[1] <- 'Climate zone'
colnames(summary_yw)[2] <- 'Yw long-term (t/ha)'
colnames(summary_yw)[3] <- 'Yw 2009 (t/ha)'
colnames(summary_yw)[4] <- 'Yw 2013 (t/ha)'
```

5.1.9 Yield gap decomposition

The four yield levels needed for yield gap decomposition were calculated in the previous sections. At last, the intermediate yield gaps need to be estimated in t/ha and in terms of yield gap closure, i.e., relative to Yw. This is what the chunk of code below implements. With all this information in hand, a summary figure showing the **yield gap decomposition** for units of interest can be produced. This is exemplified here for different administrative regions and for different farming systems.

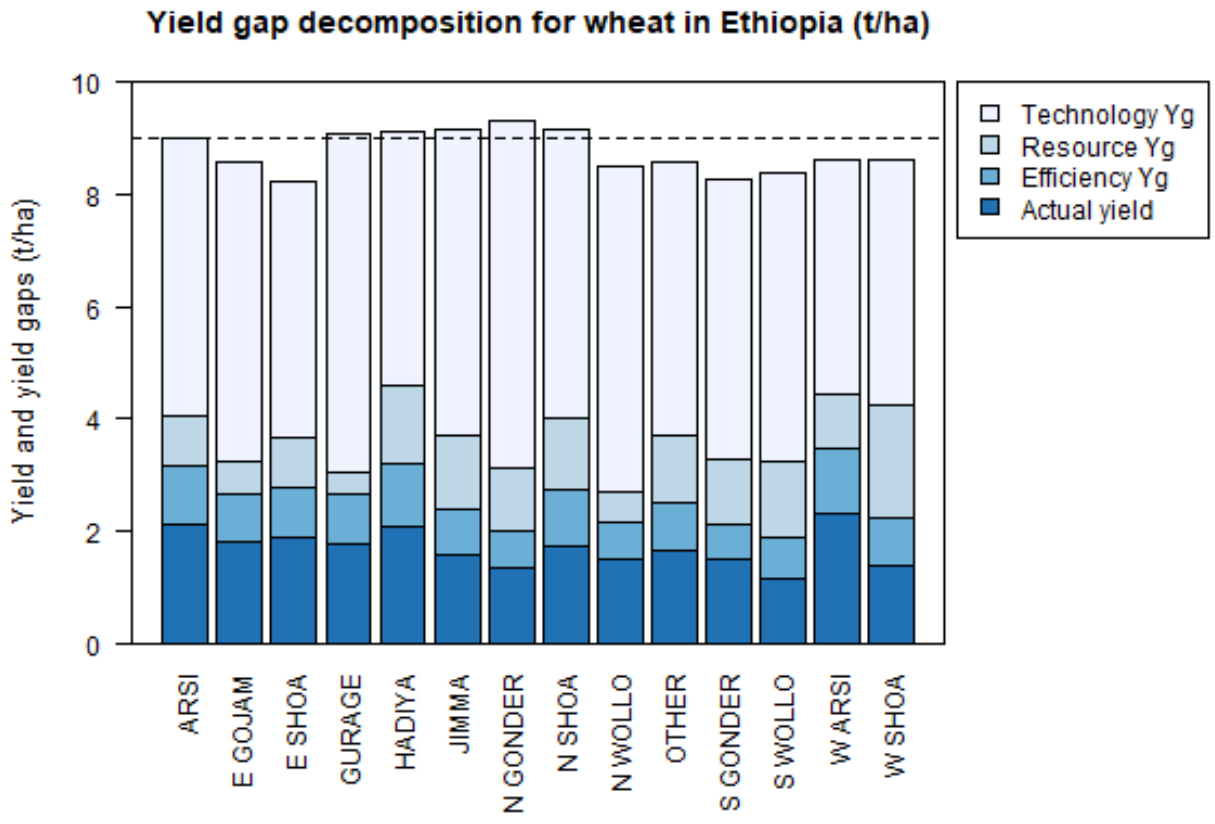
Below is the code for estimating the **yield gaps in t/ha**. Please note automation is needed for data sets spanning over many years, but the general principle is the same.

```
# total yield gap in t/ha
data_final['yg_total_2009'] <- data_final['Yw_2009'] - data_final['yield_tha']
data_final['yg_total_2013'] <- data_final['Yw_2013'] - data_final['yield_tha']
data_final['yg_total'] <- ifelse(data_final$year == 2009, data_final$yg_total_2009,
                                                          data_final$yg_total_2013)
#
# efficiency yield gap in t/ha
data_final['eff_yg_tha_2009'] <- data_final['ytex_tha'] - data_final['yield_tha']
data_final['eff_yg_tha_2013'] <- data_final['ytex_tha'] - data_final['yield_tha']
data_final['eff_yg_tha'] <- ifelse(data_final$year == 2009, data_final$eff_yg_tha_
↪2009,
                                                            data_final$eff_yg_tha_
↪2013)
#
# resource yield gap in t/ha
data_final['res_yg_tha_2009'] <- data_final['yhf_tha'] - data_final['ytex_tha']
data_final['res_yg_tha_2013'] <- data_final['yhf_tha'] - data_final['ytex_tha']
data_final['res_yg_tha'] <- ifelse(data_final$year == 2009, data_final$res_yg_tha_
↪2009,
                                                            data_final$res_yg_tha_
↪2013)
#
# technology yield gap in t/ha
data_final['tech_yg_tha_2009'] <- data_final['Yw_2009'] - data_final['yhf_tha']
data_final['tech_yg_tha_2013'] <- data_final['Yw_2013'] - data_final['yhf_tha']
data_final['tech_yg_tha'] <- ifelse(data_final$year == 2009, data_final$tech_yg_tha_
↪2009,
                                                             data_final$tech_yg_tha_
↪2013)
#
# aggregate absolute yield gaps by zone
absolute <- aggregate(data_final[c('yield_tha', 'eff_yg_tha', 'res_yg_tha', 'tech_yg_
↪tha')], by=list('zone_new'=data_final$zone_new), FUN=mean, na.rm=T)
```

(continues on next page)

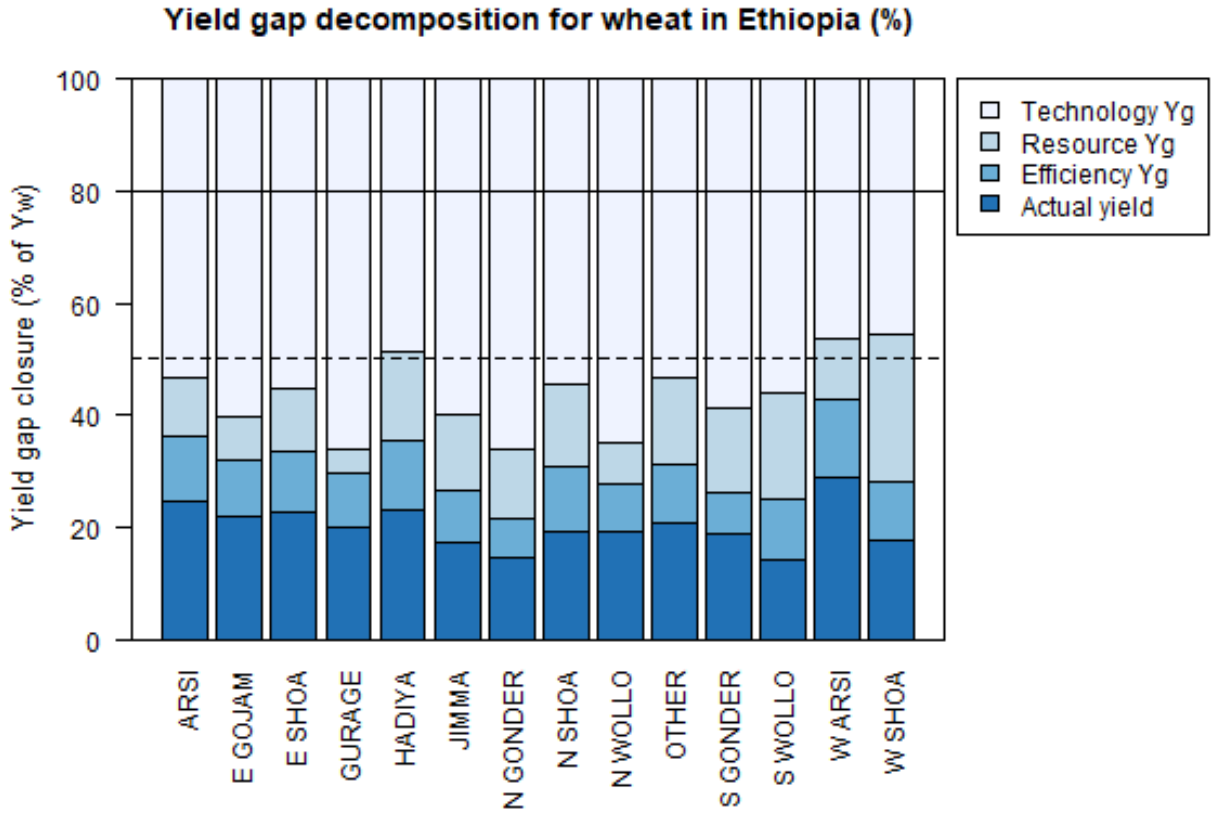
(continued from previous page)

```
absolute_t <- t(absolute)
colnames(absolute_t) <- absolute$zone_new
absolute_t <- absolute_t[-1, ]
#
# make barplot
pal <- rev(palette(brewer.pal(n=4, name="Blues")))
pal <- rev(palette(brewer.pal(n=4, name="Blues")))
par(mfrow=c(1,1), mar=c(7,5,4,9), yaxs='i')
{barplot(absolute_t,
         las=2,
         cex.lab=1.1,
         ylim=c(0, 10),
         ylab='Yield and yield gaps (t/ha)',
         main='Yield gap decomposition for wheat in Ethiopia (t/ha)',
         col=pal)
 abline(h=9, col="black", lty=2)
 legend("topright",
        inset=c(-0.325, 0),
        legend=c("Technology Yg", "Resource Yg", "Efficiency Yg", "Actual yield"),
        fill=c(palette(brewer.pal(n=4, name="Blues"))),
        xpd=TRUE)
 box()}
```



Below is the code for estimating the **yield gap closure relative to Yw**. Please note automation is needed for data sets spanning over many years, but the general principle is the same.

```
# yield gap closure relative to yw
data_final['yg_closure_2009'] <- (data_final['yield_tha'] / data_final['Yw_2009']) *␣
↪100
data_final['yg_closure_2013'] <- (data_final['yield_tha'] / data_final['Yw_2013']) *␣
↪100
data_final['yg_closure'] <- ifelse(data_final$year == 2009, data_final$yg_closure_
↪2009,
                                                            data_final$yg_closure_
↪2013)
#
# ytex relative to yw
data_final['eff_yg_2009'] <- (data_final['ytex_tha'] / data_final['Yw_2009']) * 100
data_final['eff_yg_2013'] <- (data_final['ytex_tha'] / data_final['Yw_2013']) * 100
data_final['ytex_closure'] <- ifelse(data_final$year == 2009, data_final$eff_yg_2009,
                                                              data_final$eff_yg_2013)
#
# yhf relative to yw
data_final['res_yg_2009'] <- (data_final['yhf_tha'] / data_final['Yw_2009']) * 100
data_final['res_yg_2013'] <- (data_final['yhf_tha'] / data_final['Yw_2013']) * 100
data_final['yhf_closure'] <- ifelse(data_final$year == 2009, data_final$res_yg_2009,
                                                             data_final$res_yg_2013)
#
# intermediate yield gaps
data_final$eff_yg <- data_final$ytex_closure - data_final$yg_closure
data_final$res_yg <- data_final$yhf_closure - data_final$ytex_closure
data_final$tech_yg <- 100 - data_final$yhf_closure
#
# aggregate absolute yield gaps by zone
relative <- aggregate(data_final[c('yg_closure', 'eff_yg', 'res_yg', 'tech_yg')],␣
↪by=list('zone_new'=data_final$zone_new), FUN=mean, na.rm=T)
relative_t <- t(relative)
colnames(relative_t) <- relative$zone_new
relative_t <- relative_t[-1, ]
#
# make barplot
par(mfrow=c(1,1), mar=c(7,5,4,9), yaxs='i')
{barplot(relative_t,
         las=2,
         cex.lab=1.1,
         ylim=c(0, 100),
         ylab='Yield gap closure (% of Yw)',
         main='Yield gap decomposition for wheat in Ethiopia (%)',
         col=pal)
 abline(h=50, col="black", lty=2)
 abline(h=80, col="black", lty=1)
 legend("topright",
        inset=c(-0.325, 0),
        legend=c("Technology Yg", "Resource Yg", "Efficiency Yg", "Actual yield"),
        fill=c(palette(brewer.pal(n=4, name="Blues"))),
        xpd=T)
 box()}
```



5.1.10 Recommendations

The work flow described in this notebook was applied to a wide range of (cereal) cropping systems worldwide. The relative importance of the three intermediate yield gaps, as well as inter-linkages between them, depend on the degree of yield gap closure (i.e., level of intensification) of the cropping system/dataset in question. The experiences learned so far with the application of this framework across contrasting cropping systems can be summarized as follows:

- **Low yielding cereal crops in Ethiopia and Zambia:** Large importance of the technology yield gap, which is often confounded with the resource yield gap, because inputs used in highest yielding fields are way below the inputs needed to reach the water-limited yield. Possible confounding between the technology and efficiency yield gaps is also possible when resource-use efficiency observed on-farm is way below what is agronomically possible (due to e.g., poor crop establishment or poor management of pests, diseases, and weeds). For further examples and information see Assefa et al. (2020), Silva et al., (2021), and Silva et al. (2023).
- **Intermediate yielding cereal crops in the Philippines:** Similar relative importance of efficiency, resource, and technology yield gaps. This means that yield gaps in such cropping systems are explained by a multitude of factors: sub-optimal time, space, form, and rate of input use and technologies used on-farm not being able to reach the potential or water-limited yield. Such patterns have also been observed for other rice cropping systems in Southeast Asia (e.g., Thailand). For further examples and information see Silva et al. (2017) and Silva et al. (2022).
- **High yielding cereal crops in the Netherlands and NW India:** Small resource yield gap and the also small total yield gap is equally explained by efficiency and technology yield gaps. It is questionable to focus only on explaining yield gaps in such cropping systems, as often there is scope to reduce input use without compromising

yield. It is thus important to complement such analysis with studies on resource-use efficiency, and to consider pests, diseases, and weeds as these are often responsible for the small yield gaps observed. For further examples and information see Silva et al. (2017) and Nayak et al. (2022).

The framework has **limitations** that users must be aware of. Firstly, the framework takes a production perspective and does not consider profitability or links to policy explicitly. An example on how to do so can be found in van Dijk et al. (2017). Secondly, yield ceilings only consider a limited set of climatic, edaphic, and management factors, which might overestimate the yields (and hence the technology yield gap) that can potentially be obtained in practice. For instance, soil acidity can be a serious constraint to crop yields in some areas and such effects are not captured in the yield ceilings simulated (Silva et al., 2023). It is thus recommended to compare the simulated yield ceilings against experimental trial data under optimal conditions. Thirdly, conclusions derived from the interpretation of resource yield gaps must be contextualized with knowledge about the farming system. For instance, promoting greater amounts of inputs might be detrimental to farmers in dryland areas with erratic rainfall (Silva et al., 2023) whereas in other regions promoting increased fertilizer use might lead to increased disease pressure (Silva et al., 2022). Lastly, it is difficult to derive concrete recommendations to narrow efficiency yield gaps as the associated crop management to do so is highly context specific. Further research is on-going to improve some of these aspects.

5.1.11 Acknowledgments

I thank Martin van Ittersum (WUR-PPS), Andy McDonald (Cornell CALS) and Johan Ninanya (CIP-Peru) for constructive comments in an earlier version of this workflow and Marloes van Loon (WUR-PPS) for retrieving the yield ceiling data from the Global Yield Gap Atlas. The development of this notebook was possible thanks to the financial support from the OneCGIAR initiative on *Excellence in Agronomy*. For further support and questions on how to implement this workflow to other data sets please contact J.V. Silva (**j.silva@cgiar.org**).

5.2 Retrieve data from GYGA

- **João Vasco Silva**, CIMMYT-Zimbabwe
- **Marloes van Loon**, WUR

5.2.1 Introduction

This notebook complements an earlier notebook describing the methodology for yield gap decomposition. That earlier notebook makes use of water-limited yield data to decompose yield gaps. Such data were derived using the scripts documented in this notebook. The reader is referred to that earlier notebook for further information about the concepts and definitions considered in yield gap analysis. To make the approach fully reproducible, it is explained here how to retrieve the water-limited yield data from the Global Yield Gap Atlas (GYGA) using available APIs for acquiring such data. Also here an example is provided for for wheat in Ethiopia.

5.2.2 Load required R packages

First, the R packages needed to run this workflow are loaded.

```
# package names
packages <- c("dplyr", "tidyr", "httr", "jsonlite", "sf", "reshape2")
#
# install packages
installed_packages <- packages %in% rownames(installed.packages())
if(any(installed_packages == FALSE)){
  install.packages(packages[!installed_packages], repos="http://cran.us.r-project.org
↪", quiet=T)}
#
# load packages
invisible(lapply(packages, function(x) suppressMessages(require(x, character.only=T,␣
↪quietly=T, warn.conflicts=F))))
```

5.2.3 Access to GYGA data

An example on how data can be accessed from GYGA, and linked to farmer field data, using an API specifically set for wheat water-limited yields in Ethiopia. The chunk of code below illustrates how to access from the GYGA API the following data for a given country: (a) crop yield data for all weather stations, (b) weather station coordinates and information and, (c) climate zones. Please note an internet connection is needed to access the API.

```
# retrieve data for all weather stations
request <- httr::GET(
  "https://www.yieldgap.org/apigyga/json/cropcountrystationyear?accesstoken=anonymous&
↪par1=30&par2=5",
  httr::add_headers())
wheat_data <- httr::content(request, as = "text")
wheat_data <- jsonlite::fromJSON(wheat_data)[["items"]]
wheat_data$harvestyear <- paste0('Yw_', wheat_data$harvestyear)
wheat_data <- dcast(wheat_data, country + country_id + climatezone +
                                station + station_id + crop + crop_id ~ harvestyear,
                    value.var='yw')
#
# get coordinates of weather stations
request <- httr::GET("https://www.yieldgap.org/apigyga/metadata/stations",
                     httr::add_headers())
json_data <- httr::content(request, as="text")
gyga_stations <- jsonlite::fromJSON(json_data)[["items"]]
gyga_stations <- gyga_stations[gyga_stations$station_id %in% unique(wheat_data
↪$station_id),]
#
# retrieve climate zones gyga
request <- httr::GET("https://www.yieldgap.org/apigyga/metadata/climatezones",
                     httr::add_headers())
json_data <- httr::content(request, as="text")
cz_data <- jsonlite::fromJSON(json_data)[["items"]]
cz_data <- cz_data[cz_data$country == unique(wheat_data$country),]
```

The next step entails merging the yield data from GYGA with the respective weather station coordinates and climate zone information. Once data are merged, the crop yield data is made spatially explicit using the *st_as_sf()* function. Please refer to the chunk of code below.

```
# merge wheat data with gyga stations
wheat_data <- merge(wheat_data[,c("country", "country_id", "station", "station_id",
                                  "climatezone", "crop", "crop_id",
                                  "Yw_1998", "Yw_1999", "Yw_2000", "Yw_2001",
                                  "Yw_2002", "Yw_2003", "Yw_2004", "Yw_2005",
                                  "Yw_2006", "Yw_2007", "Yw_2008", "Yw_2009",
                                  "Yw_2010", "Yw_2011", "Yw_2012", "Yw_2013",
                                  "Yw_2014", "Yw_2015", "Yw_2016", "Yw_2017")],
                    gyga_stations[,c("station_id", "latitude", "longitude")],
                    by="station_id", all.x=TRUE)
#
# merge wheat data with climate zones
wheat_data <- merge(wheat_data,
                    cz_data[,c("climatezone", "climatezone_id")],
                    by="climatezone", all.x=TRUE)
#
# convert to spatial dataframe
wheat_data <- st_as_sf(wheat_data, coords=c("longitude", "latitude")) %>%
  st_set_crs(4326)
```

5.2.4 Link to farmer field data

Once the GYGA data are retrieved, it is then possible to merge for each field available in the farmer field data. To do so, the data frame with the farmer field data is loaded and made spatially explicit based on the latitude and longitude values of each field. The climate zones for the country of interest are also loaded and added to the farmer field data using the *st_join()* function. This is described in the chunk of code below.

```
# read .csv file with data
file <- 'https://raw.githubusercontent.com/jvasco323/EiA_YGD_workflow/main/data-gps-
↪coordinates-original.csv'
data <- read.csv(url(file))
data <- unique(data)
#
# convert df to spatial dataframe
data <- data[!is.na(data$LON),]
data <- data[!is.na(data$LAT),]
data <- st_as_sf(data, coords=c("LON", "LAT")) %>%
  st_set_crs(4326)
#
# load climate zones and merge to field data
file <- 'https://raw.githubusercontent.com/jvasco323/EiA_YGD_workflow/main/data-
↪climate-zone-eth.rds'
climate_zone <- readRDS(url(file))
#
# join field data with climate zones
data <- st_join(data, climate_zone, join=st_intersects, left=FALSE)
```

The next step is to get the closest weather station to each field within the climate zone of the field. The closest weather station must be within the same climate zone of the field, but when climate zones are very fragmented, it is advisable to consider the closest weather station independently of the climate zone (more information on this is provided below). The chunk of code below adds the closest weather station to each field.

```
# get the closest station within each climate zone
data_rfwh  <- NULL
```

(continues on next page)

(continued from previous page)

```
for(i in unique(data$GYGA_CZ)){
  #
  # subset each climate zone
  dat <- subset(data, GYGA_CZ==i)
  dat_grid <- subset(wheat_data, climatezone==i)
  #
  # find the nearest station within each climate zone
  find_nearest_RWS_CZ <- st_join(dat, dat_grid, join=st_nearest_feature, left=FALSE)
  nearest <- st_nearest_feature(dat, dat_grid)
  dist <- st_distance(dat, dat_grid[nearest,], by_element=TRUE)
  dist <- data.frame(dist)
  #
  # conver m to km
  dist$dist <- as.numeric(dist$dist)/1000 #convert from m to km
  find_nearest_RWS_CZ <- cbind(find_nearest_RWS_CZ, dist)
  #
  # bind all data from the loop
  data_rfwh <- rbind(data_rfwh, find_nearest_RWS_CZ)
}

# add -99 to yw not available
data_rfwh[c(22:41)][is.na(data_rfwh[c(22:41)])] <- -99
## Warning in `[<-.data.frame`(`*tmp*`, c(22:41), value = structure(list(Yw_1998 =
## c(-99, : provided 21 variables to replace 20 variables
```

Now, the fields with water-limited yield data from the same climate zone of the weather station are identified. To do this, a subset is created containing the farmer field data for each the water-limited yield is greater than 0. A new column is added to that data frame to flag the source of the water-limited yield data.

```
# fields with yield data
data_rfwh_with_data <- subset(data_rfwh, Yw_2010 >= 0) # arbitrary year of the data
data_rfwh_with_data$data_from <- "same_cz"
```

Some fields do not fall inside climate zones for which crop model simulations were done and hence, they do not have a weather station coupled. These fields are identified through a subset of the data set containing the fields for which the water-limited yield is below 0 (i.e., equal to -99). For these fields, the distance from each field to the closest weather station is estimated using the chunk of code below.

```
# fields with no yield data
data_rfwh_no_data <- subset(data_rfwh, Yw_2010 < 0) # arbitrary year of the data
data_rfwh_no_data <- data_rfwh_no_data[,c('hhid')]
#
# check GYGA station closer than 30km
nearest <- st_nearest_feature(data_rfwh_no_data, wheat_data)
dist <- st_distance(data_rfwh_no_data, wheat_data[nearest,], by_element=TRUE)
dist <- data.frame(dist)
dist$dist <- as.numeric(dist$dist)/1000 #convert from m to km
data_rfwh_no_data <- cbind(data_rfwh_no_data, dist)
```

Fields with a weather station closest than 30km are identified first. For those fields, the yield data from the respective weather station are used. Also here a new column is added to the data frame to flag the source of the water-limited yield data. This is done in the chunk of code below.

```
# fields closer than 30km: get data from respective station
data_rfwh_no_data_30 <- subset(data_rfwh_no_data, dist <= 30)
data_rfwh_no_data_30 <- st_join(data_rfwh_no_data_30, wheat_data, join=st_nearest_
↪feature,
```

(continues on next page)

(continued from previous page)

```
                                    left=FALSE)
data_rfwh_no_data_30$data_from <- "cz_station"
```

Lastly, crop yield data need to be retrieved for the fields located more than 30km away from any weather station. The national average is used for these fields. The chunk of code below first subsets the fields with no weather station closer than 30km, then retrieves the national average for the country of interest using the GYGA API and, finally, merges that data with the field data.

```
# fields further than 30km: use country average
data_rfwh_no_data_m30 <- subset(data_rfwh_no_data, dist > 30)
data_rfwh_no_data_m30$country <- 'Ethiopia'
#
# retrieve wheat data Ethiopia
request <- httr::GET(
  "https://www.yieldgap.org/apigyga/json/cropcountryyear?accesstoken=anonymous&
↪par1=30&par2=5",
  httr::add_headers())
wheat_data_eth <- httr::content(request, as="text")
wheat_data_eth <- jsonlite::fromJSON(wheat_data_eth)[["items"]]
wheat_data_eth$harvestyear <- paste0('Yw_', wheat_data_eth$harvestyear)
wheat_data_eth <- dcast(wheat_data_eth, country + country_id +
                                        crop + crop_id ~ harvestyear, value.var='yw')
#
# join yw data of whole country with fields which have no data
data_rfwh_no_data_m30 <- merge(data_rfwh_no_data_m30,
                               wheat_data_eth[,c("country", "country_id", "crop",
↪"crop_id",
                                                 "Yw_1998", "Yw_1999", "Yw_2000", "Yw_
↪2001",
                                                 "Yw_2002", "Yw_2003", "Yw_2004", "Yw_
↪2005",
                                                 "Yw_2006", "Yw_2007", "Yw_2008", "Yw_
↪2009",
                                                 "Yw_2010", "Yw_2011", "Yw_2012", "Yw_
↪2013",
                                                 "Yw_2014", "Yw_2015", "Yw_2016", "Yw_
↪2017")],
                               by="country", all.x=TRUE)
data_rfwh_no_data_m30$data_from <- "country_average"
```

5.2.5 Export file with GYGA data

The last step in this workflow is to bring back the different subsets of fields into a single data frame. Recall: the different subsets of fields are as follows: (a) fields with a weather station within the respective climate zone, (b) fields without a climate zone for which crop model simulations were done but with a weather station closer than 30km and, (c) fields without a climate zone for which crop model simulations were done and with no weather station closer than 30km. This is implemented in the chunk of code below, where this final data frame is saved to disk as a csv file.

```
# final data frame
data_rfwh_final <- rbind(data_rfwh_with_data[,c("hhid", "country", "crop",
                                                "data_from", "geometry",
                                                "Yw_1998", "Yw_1999", "Yw_2000", "Yw_
↪2001",
```

(continues on next page)

(continued from previous page)

```
                                                "Yw_2002", "Yw_2003", "Yw_2004", "Yw_
↪2005",
                                                "Yw_2006", "Yw_2007", "Yw_2008", "Yw_
↪2009",
                                                "Yw_2010", "Yw_2011", "Yw_2012", "Yw_
↪2013",
                                                "Yw_2014", "Yw_2015", "Yw_2016", "Yw_
↪2017")],
                          data_rfwh_no_data_30[,c("hhid", "country", "crop",
                                                  "data_from", "geometry",
                                                  "Yw_1998", "Yw_1999", "Yw_2000", "Yw_
↪2001",
                                                  "Yw_2002", "Yw_2003", "Yw_2004", "Yw_
↪2005",
                                                  "Yw_2006", "Yw_2007", "Yw_2008", "Yw_
↪2009",
                                                  "Yw_2010", "Yw_2011", "Yw_2012", "Yw_
↪2013",
                                                  "Yw_2014", "Yw_2015", "Yw_2016", "Yw_
↪2017")],
                          data_rfwh_no_data_m30[,c("hhid", "country", "crop",
                                                   "data_from", "geometry",
                                                   "Yw_1998", "Yw_1999", "Yw_2000",
↪"Yw_2001",
                                                   "Yw_2002", "Yw_2003", "Yw_2004",
↪"Yw_2005",
                                                   "Yw_2006", "Yw_2007", "Yw_2008",
↪"Yw_2009",
                                                   "Yw_2010", "Yw_2011", "Yw_2012",
↪"Yw_2013",
                                                   "Yw_2014", "Yw_2015", "Yw_2016",
↪"Yw_2017")])
#
# get cimate zones for each field
data_rfwh_final <- st_join(data_rfwh_final, climate_zone, join=st_intersects,␣
↪left=FALSE)
#
# remove geometry
data_rfwh_final <- st_drop_geometry(data_rfwh_final)
#
# calculate average Yw
data_rfwh_final$Yw_average <- rowMeans(data_rfwh_final[c(5:24)], na.rm=TRUE)
#
# save csv file
write.csv(data_rfwh_final, 'data-gps-coordinates-final.csv')
```

5.2.6 Final remarks

Please note this script does not add data to fields without GPS coordinates reported. The national average could also be considered for those fields, but this is not implemented in this script.

Data from GYGA can be made available to the OneCGIAR initiative on **Excellence in Agronomy** on a demand basis. Requests should be made to prof. Martin van Ittersum (**martin.vanittersum@wur.nl**). The script documented here is reproducible and can be used to retrieved GYGA data for other crop x country combinations. Any questions or suggestions for improving the scripts presented in this document should be addressed to **j.silva@cgiar.org** or **marloes.vanloon@wur.nl**.

This section provides the workflow to conduct yield gap decomposition using stochastic frontier analysis. Introductory slides can be downloaded here. You will learn (a) how to fit and interpret a stochastic frontier to farmer field data, (b) how to estimate the efficiency yield gap from the fitted model, (c) how to estimate the highest farmers' yields and the resource yield gap, and (d) how to estimate the technology yield gap using the water-limited yields from the Global Yield Gap Atlas. Please refer to the complementary workflow to access yield data from the Global Yield Gap Atlas.

6 Interpretable machine learning

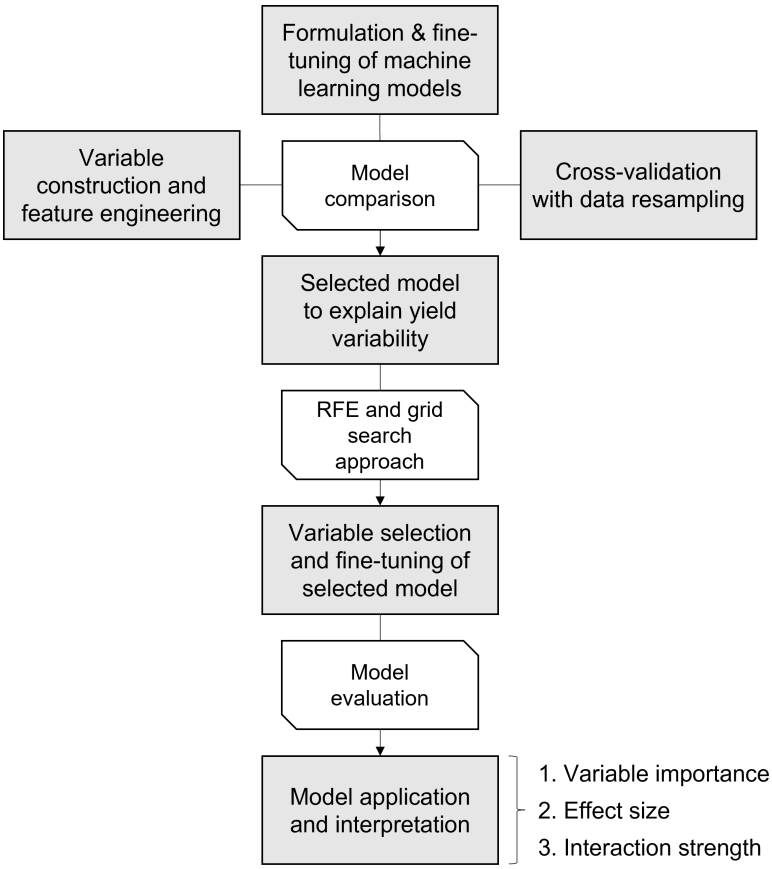
6.1 Workflow for Nayak et al. (2022)

- **Hari Sankar Nayak**, Cornell University
- **João Vasco Silva**, CIMMYT-Zimbabwe

6.1.1 Introduction

Interest in the application of machine learning in agronomic science is increasing with the growing availability of geo-referenced farmer field data and spatially explicit environmental data in a diversity of cropping systems. Different types of machine learning methods can be identified based on their **interpretability** and **level of complexity**: (i) regression-based methods (e.g., linear, ridge, and lasso regression), (ii) single tree- or multiple tree-based methods (e.g., classification and regression trees, gradient boosting, extreme gradient boosting, and random forest), and (iii) decision boundary-based support vector regression, amongst others.

Regression-based methods are parametric, and their coefficients are obtained through ordinary least-squares. **Tree-based methods** such as classification and regression tree, random forest, and gradient boosting rely on decision trees, and a series of if-then rules to arrive at a particular prediction or classification. **Distance-based methods**, like K-nearest mean, find the K-nearest neighbors in the feature space and provide predictions based on those K data point's outcomes. Finally, **decision boundary-based methods** create a decision boundary with data projected in a higher dimension (suppose there are three variables, then their values will be projected in a 3-dimensional space) to derive a particular prediction. Regression-based methods are less complex and have easier and direct interpretation compared to tree-based or distance and decision boundary-based methods.



Some of the methods referred above have been used in agronomy research over the recent years. One such example comparing different machine learning models can found in Nayak et al. (2022), which provide an application for wheat crops in Northwest India (Punjab and Haryana). We will reproduce the key steps proposed in the study in this workflow (see figure above) namely (1) comparison of different models, (2) recursive feature elimination, (3) tuning of model hyper-parameters, (4) assessing model performance, and (5) fit a final model and interpreted it using different techniques. Estimation of Shapely values, which will provide a useful means to decompose yield gaps in the future, is provided at the end.

6.1.2 Load required R packages

First, we need to load the R packages needed to run this workflow.

```
# install packages
install.packages('randomForest', repos="http://cran.us.r-project.org", dependencies=T,
↪ quiet=T)
## package 'randomForest' successfully unpacked and MD5 sums checked
install.packages('caret', repos="http://cran.us.r-project.org", dependencies=T,␣
↪quiet=T)
## package 'caret' successfully unpacked and MD5 sums checked
install.packages('caretEnsemble', repos="http://cran.us.r-project.org",␣
↪dependencies=T, quiet=T)
## package 'caretEnsemble' successfully unpacked and MD5 sums checked
install.packages('iml', repos="http://cran.us.r-project.org", dependencies=T, quiet=T)
## package 'iml' successfully unpacked and MD5 sums checked
```

(continues on next page)

(continued from previous page)

```
install.packages('tidyverse', repos="http://cran.us.r-project.org", dependencies=T,␣
↪quiet=T)
## package 'tidyverse' successfully unpacked and MD5 sums checked
install.packages('dplyr', repos="http://cran.us.r-project.org", dependencies=T,␣
↪quiet=T)
## Warning: package 'dplyr' is in use and will not be installed
#
# install packages
packages <- c("elasticnet", "partykit", "rpart", "rpart.plot")
installed_packages <- packages %in% rownames(installed.packages())
if(any(installed_packages == FALSE)){
  install.packages(packages[!installed_packages], repos="http://cran.us.r-project.org
↪", quiet=T)}
#
# load packages
packages <- c("elasticnet", "partykit", "rpart", "rpart.plot", "randomForest", "caret
↪", "caretEnsemble", "iml", "tidyverse", "dplyr")
invisible(lapply(packages, function(x) suppressMessages(require(x, character.only=T,␣
↪quietly=T, warn.conflicts=F))))
```

6.1.3 Farmer field data

The chunk of code below loads the data that we will use for this workflow. The data used to illustrate how much learning can be used for yield gap decomposition refers to wheat crops in Northwest India (Punjab and Haryana). Further details about the data set can be found in Nayak et al. (2022).

```
# read .csv file with data
file <- 'https://raw.githubusercontent.com/jvasco323/eia-yg-training-ppt/master/
↪pooled_data_wheat_phd_fcr.csv'
pooled_data <- read.csv(url(file), header=TRUE)
#
# list variables of interest for visualization
str(pooled_data)
## 'data.frame':    6139 obs. of  24 variables:
##  $ X                    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ cc_yield             : num  3475 5086 5086 5548 6275 ...
##  $ total_nitrogen_ha    : num  182 182 178 178 126 ...
##  $ weed_severity        : chr  "Low" "No" "No" "Medium" ...
##  $ insect_severity      : chr  "None" "Medium" "Medium" "Medium" ...
##  $ disease_severity     : chr  "None" "Medium" "Medium" "Medium" ...
##  $ rice_till            : chr  "<=4" "6" "<=4" "<=4" ...
##  $ till_int             : chr  "ZT/MT" "ZT/MT" "ZT/MT" "ZT/MT" ...
##  $ irri_num             : chr  "<=2" "3" "<=2" "<=2" ...
##  $ variety_nameIn       : chr  "WH 1105" "HD 2967" "Other" "HD 2967" ...
##  $ residue              : chr  "80_100" "80_100" "80_100" "80_100" ...
##  $ duration             : int  177 176 167 163 155 160 174 167 164 166 ...
##  $ seed_amount          : int  50 40 50 40 40 40 40 40 40 40 ...
##  $ tot_p2o5_ha          : num  69 69 57.5 57.5 57.5 57.5 57.5 57.5 57.5 57.5 ..
↪.
##  $ lodging_cat          : chr  "30" "No" "No" "No" ...
##  $ date_sowin           : int  297 300 311 308 315 312 305 307 305 296 ...
##  $ dat_urea1            : int  28 30 30 28 25 25 25 25 25 20 ...
##  $ dat_urea2            : int  45 45 45 39 35 35 35 35 35 35 ...
```

(continues on next page)

(continued from previous page)

```
##  $ fallow_days               : num  13 4 6 35 25 35 35 35 25 0.1 ...
##  $ Max_temperature           : num  22.1 22.1 22.3 23.2 22.3 ...
##  $ Min_temperature           : num  10.6 10.6 10.4 11.6 10.4 ...
##  $ Precipitation_Accumulated: num  236 243 221 207 270 ...
##  $ radiation                 : num  74.8 75.2 76.9 79.8 78.6 ...
##  $ Texture                   : chr  "loam" "clay_loam" "loam" "loam" ...
```

We select a random sample of 30% of the total data, to ease computation power. We need to use the 'set.seed()' function to make sure the random sample is the same every time we re-run the code.

```
set.seed(1234)
pooled_data1 <- pooled_data %>% sample_frac(size = 0.3)
```

We need to some data manipulation, particularly re-levelling categorical variables, prior to model fitting. This is done with the chunk of code below.

```
# leveling categorical variables
pooled_data1$lodging_cat <- as.factor(pooled_data1$lodging_cat)
pooled_data1$lodging_cat <- relevel(pooled_data1$lodging_cat, ref="No")
pooled_data1$irri_num <- as.factor(pooled_data1$irri_num)
pooled_data1$irri_num <- relevel(pooled_data1$irri_num, ref="<=2")
pooled_data1$rice_till <- as.factor(pooled_data1$rice_till)
pooled_data1$rice_till <- relevel(pooled_data1$rice_till, ref="<=4")
pooled_data1$residue <- as.factor(pooled_data1$residue)
pooled_data1$residue <- relevel(pooled_data1$residue, ref="No")
pooled_data1$till_int <- as.factor(pooled_data1$till_int)
pooled_data1$till_int <- relevel(pooled_data1$till_int, ref="Intensive")
pooled_data1$Texture <- as.factor(pooled_data1$Texture)
pooled_data1$Texture <- relevel(pooled_data1$Texture, ref="clay")
pooled_data1$insect_severity <- as.factor(pooled_data1$insect_severity)
pooled_data1$insect_severity <- relevel(pooled_data1$insect_severity, ref="None")
pooled_data1$disease_severity <- as.factor(pooled_data1$disease_severity)
pooled_data1$disease_severity <- relevel(pooled_data1$disease_severity, ref="None")
pooled_data1$weed_severity <- as.factor(pooled_data1$weed_severity)
pooled_data1$weed_severity <- relevel(pooled_data1$weed_severity, ref="No")
pooled_data1$X = NULL
```

6.1.4 Step 1: Model comparison

In this section, we use the functions of the *caret* R package to compare the performance of different machine learning models. First, the data are split into training and test set. THis is done with *createDataPartition()* function, which ensure the same yield distribution between both train and test data sets. Further, the *nearZeroVar()* function is used check whether categorical variables have a balanced number of observations between per category. None of the variables considered here has near zero variance. So, we can proceed with all of them for further analysis. That not being the case, we need to exclude variables with unbalanced number of observations for different categories.

For example, if 98% of the fields reported residue incorporation and only 2% do not, then that variable has to be excluded from further analysis due to lack variability in the data to check the effects of residue management with such data distribution.

```
# partitioning data into train and test set
set.seed(458)
train_seq <- createDataPartition(pooled_data1$cc_yield, p=0.7, list=F)
```

(continues on next page)

(continued from previous page)

```
train_data <- pooled_data1[train_seq,]
test_data <- pooled_data1[-train_seq,]
#
# check near zero variance predictor
(nzv <- nearZeroVar(train_data, saveMetrics=T))
##                           freqRatio percentUnique zeroVar   nzv
## cc_yield                   1.656250    33.8497289   FALSE FALSE
## total_nitrogen_ha          2.496296     9.3725794   FALSE FALSE
## weed_severity              1.992405     0.3098373   FALSE FALSE
## insect_severity            1.150562     0.2323780   FALSE FALSE
## disease_severity           1.036613     0.2323780   FALSE FALSE
## rice_till                  1.143284     0.3098373   FALSE FALSE
## till_int                   1.225728     0.2323780   FALSE FALSE
## irri_num                   1.389016     0.3098373   FALSE FALSE
## variety_nameIn             1.388325     0.8520527   FALSE FALSE
## residue                    1.293532     0.3098373   FALSE FALSE
## duration                   1.030303     3.7180480   FALSE FALSE
## seed_amount                1.157773     1.3942680   FALSE FALSE
## tot_p2o5_ha                6.539683     1.6266460   FALSE FALSE
## lodging_cat                4.302857     0.3872967   FALSE FALSE
## date_sowin                 1.031579     3.3307514   FALSE FALSE
## dat_urea1                  1.639456     1.6266460   FALSE FALSE
## dat_urea2                  1.138158     2.0139427   FALSE FALSE
## fallow_days                1.346154     2.7885360   FALSE FALSE
## Max_temperature            1.166667    84.0433772   FALSE FALSE
## Min_temperature            1.166667    84.3532146   FALSE FALSE
## Precipitation_Accumulated  1.789474    48.7219210   FALSE FALSE
## radiation                  1.166667    84.3532146   FALSE FALSE
## Texture                    5.987805     0.2323780   FALSE FALSE
```

The chunk of code below implements fits four different machine learning models namely random forest (ranger), classification and regression trees (rpart), lasso regressions (lasso), and multiple linear regressions (lm) to the training data set (see *algorithmList*). This is done with *caretList()* function. This done considering a five-fold cross-validation scheme with random sampling repeated once, as specified with the *trainControl()* function.

You might get some warnings here, just ignore them for now. Moreover, the model selection provides a summary of the performance of each models based on the mean absolute error (MAE), root mean square error (RMSE), and r2 (Rsquared). The table shows the mean and variability of each statistical indicator.

```
# five-fold cross-validation repeated once with random sampling
control <- trainControl(method="repeatedcv", number=5, repeats=1, search="random")
algorithmList <- c("ranger", "rpart", "lasso", "lm")
#
# fit the models
models <- caretList(cc_yield ~ ., data=train_data, trControl=control,␣
↪methodList=algorithmList)
## Warning in trControlCheck(x = trControl, y = target): trControl$savePredictions
## not 'all' or 'final'.  Setting to 'final' so we can ensemble the models.
## Warning in trControlCheck(x = trControl, y = target): indexes not defined in
## trControl.  Attempting to set them ourselves, so each model in the ensemble
## will have the same resampling indexes.
#
# check model performance and select the final model
model_selection <- summary(resamples(models))
model_selection
##
```

(continues on next page)

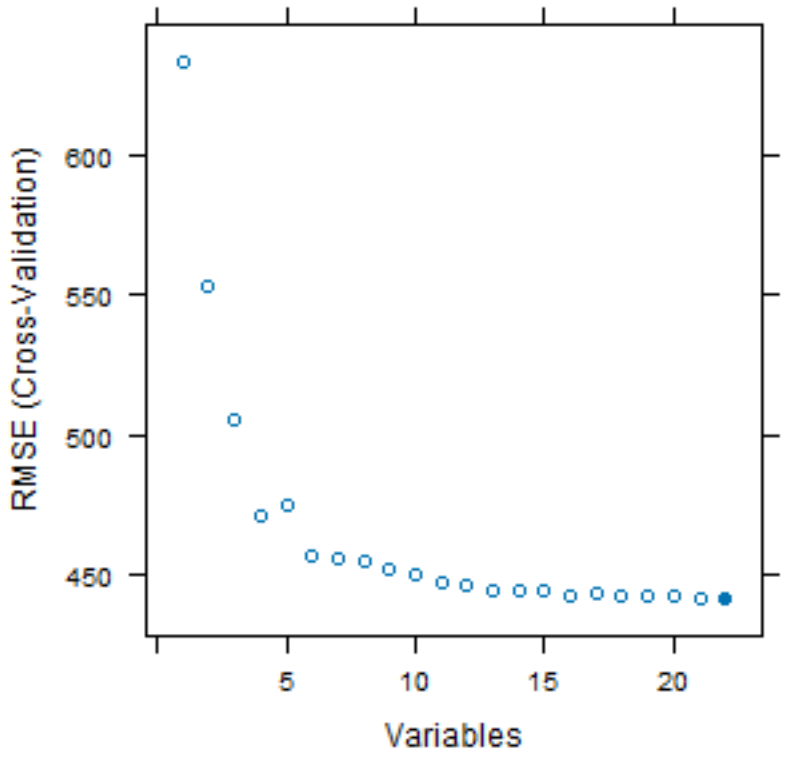
(continued from previous page)

```
## Call:
## summary.resamples(object = resamples(models))
##
## Models: ranger, rpart, lasso, lm
## Number of resamples: 5
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ranger 325.6882 329.0564 340.2513 336.8692 341.9962 347.3539    0
## rpart  385.9707 396.8829 404.5381 403.9548 411.9435 420.4389    0
## lasso  362.7796 367.1223 381.6161 381.3467 390.0721 405.1432    0
## lm     360.0128 371.1747 373.3334 374.2182 376.1845 390.3857    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## ranger 423.8816 441.4632 454.4603 456.8959 471.2636 493.4108    0
## rpart  515.0038 524.3552 544.7379 549.7969 578.7274 586.1599    0
## lasso  473.3300 490.0915 504.2732 501.6990 518.9239 521.8766    0
## lm     474.2202 484.3892 500.1749 495.1089 507.5487 509.2114    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ranger 0.3361684 0.4138663 0.4409527 0.4484798 0.5173389 0.5340726    0
## rpart  0.1914681 0.2335654 0.2682762 0.2709623 0.3302742 0.3312278    0
## lasso  0.2724807 0.3220751 0.3279141 0.3347506 0.3607098 0.3905734    0
## lm     0.2845403 0.3340153 0.3650927 0.3511480 0.3850839 0.3870076    0
```

6.1.5 Step 2: Recursive feature elimination

Not all variables (also known in machine learning as features) are equally important in a given machine learning model. Most often, most variability in the output variable is described by few features only. The benefits of a reduced feature space include (1) improved, targeted, data collection in subsequent field activities and (2) the model is fitted without variables that don't add additional information/explanatory power. **Recursive feature elimination** is a technique to reduce the number of features prior to model fitting. To do so, a given functional form is needed to establish the relationship between input and output variables. The functional form assumed in this example is random forest.

```
# define five-fold cross-validation scheme
control_feature <- rfeControl(functions=rfFuncs, method="cv", number=5)
#
# recursive feature elimination; 'sizes' stands for the number of variables
feature_selection <- rfe(train_data[,2:23], train_data[,1], sizes=c(1:23),␣
↪rfeControl=control_feature)
#
# rmse vs. number of variables (ordered from very to not important)
plot(feature_selection)
```




```
#
# r2 vs. number of variables (ordered from very to not important)
# plot(feature_selection$results$Rsquared)
```

The final number of variables to keep in the model is a subjective user decision. From the plot above, including more than variables results in small changes in the root mean square error of the training data set. So, we can pick any number of variables greater than 10. For this example, we pick the 10 most important variables only.

```
# list all variables
list_of_var <- feature_selection$optVariables
list_of_var <- append("cc_yield", list_of_var)
#
# select 10 most important variables only
train_data <- train_data[,list_of_var[1:11]]
test_data <- test_data[,list_of_var[1:11]]
```

6.1.6 Step 3: Tuning hyper-parameters

The structure of underlying regression trees can lead to the formulation of different random forest models, which validity and performance can be evaluated as explained in this section. Different random forest models can be created through manipulation of so-called **hyper-parameters** and the hyper-parameters leading to best model performance can be identified using a **grid search approach**. Three hyper-parameters can be tuned in random forest: **ntree** (number of trees build), **nodesize** (number of observations allowed per terminal node controlling the depth of each tree), and **mtry** (the number of variables randomly sampled as candidates at each split).

The chunk of code below creates the combinations of parameters for which the model performance will be assessed using a grid search approach. Next, it defines the cross-validation scheme for the hyper-parameter tuning and fits a random forest model for each combination using the *train()* function.

```
# parameter combinations for grid search approach
ntrees   <- c(100, 500)
nodesize <- seq(10,130, 15)
tuneGrid <- expand.grid(.mtry = c(4,6,8,10,12))
params   <- expand.grid(ntrees = ntrees, nodesize = nodesize)
#
# cross-validation scheme for the hyper-parameter tuning
control <- trainControl(method="repeatedcv", number=5, repeats=1, search='grid',␣
↪verboseIter=F)
#
# create vector to store the 18 combinations of nodesize x ntree, at the best mtry␣
↪value
all_model <- vector("list", nrow(params))
#
# fit the model for different combinations of hyper-parameters
for(i in 1:nrow(params)){
  nodesize <- params[i,2]
  ntree <- params[i,1]
  set.seed(65)
  rf_model <- train(cc_yield ~ .,          # y = f(x)
                    data=train_data,    # select training data
                    method="rf",        # choose random forest
                    tuneGrid=tuneGrid, # evaluate mtry for each combination
                    trControl=control, # set cross-validation scheme
                    ntree=ntree,        # set number of trees
                    nodesize=nodesize) # set node size
  all_model[[i]] <- rf_model             # store model outputs
  }
#
# set dataframe names
names(all_model) <- paste("ntrees:", params$ntrees, "nodesize:", params$nodesize)
```

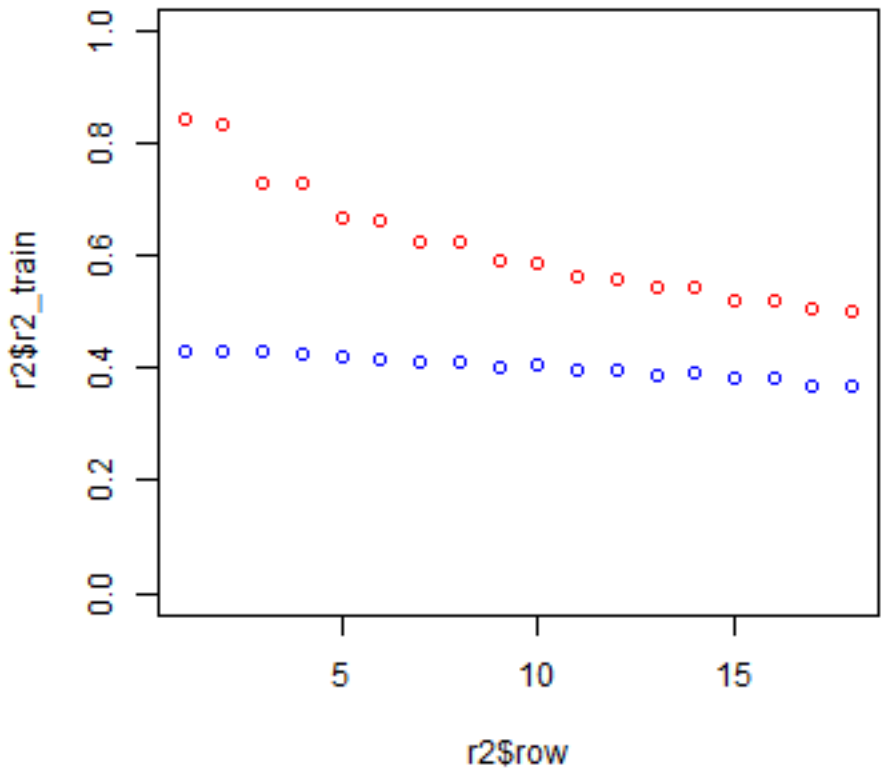
6.1.7 Step 4: Model performance

After fitting random forest models with different structure with the chunk of code above, we can now evaluate the performance of those models based on the cross-validated and training R2. This is necessary to select the "best" combination of hyper-parameters to be used in the final model from which interpretation will be done.

There is no rule of thumb regarding which model (i.e., combination of hyper-parameters) to choose, which thus remains a subjective user decision. For instance, if the model is intended for **explanatory purposes**, then the hyper-parameter combination with highest training R2 can be selected. Conversely, if the model is intended for **predictive purposes**, then the hyper-parameter combination with highest cross-validated, out-of-bag, R2 can be selected. Finally, if the model **interpretation** is the final goal, then the hyper-parameter combination with most similar training and cross-validated R2 can be selected. Such decision can be made based on the plot below.

Please note there is a rule of thumb to tune the *mtry* value in random forest. That is: the best *mtry* values lies between square root of the number of variables or the number of variables divided by three.

```
# extract cross-validated r2 (proxy of test set r2)
rsquared <- sapply(all_model, function(object) object$results["Rsquared"])
names(rsquared) <- paste("ntrees:", params$ntrees, "nodesize:", params$nodesize)
rsq_df = data.frame(matrix(unlist(rsquared), nrow=length(rsquared), byrow=T))
rsq_df$nodesize = paste(params$nodesize)
rsq_df$ntree    = paste(params$ntrees)
rsq_df$mean_r2 = rowMeans(rsq_df[1:5])
#
# extract the training r2 (optional)
store_validation <- vector("list", nrow(params))
for(i in 1:nrow(params)){
  store_validation[[i]] <- list(predict(all_model[[i]],
                                        newdata=all_model[[i]][["trainingData
↪"]][2:11]),
                                all_model[[i]][["trainingData"]]$.outcome)
  }
rsq_train = vector("list", nrow(params))
for(i in 1:nrow(params)){
  rsq_train[[i]] <- cor(store_validation[[i]][[1]], store_validation[[i]][[2]])^2
  }
rsq_train_df = data.frame(matrix(unlist(rsq_train)))
rsq_train_df$nodesize =  paste(params$nodesize)
rsq_train_df$ntree    =  paste(params$ntrees)
names(rsq_train_df)[1] <- 'r2_train'
#
# merge training and cross-validated r2 into single data frame
r2 <- merge(rsq_train_df, rsq_df, by=c('nodesize', 'ntree'))
r2 <- r2[order(r2$r2_train, decreasing=T),]
r2$row <- seq(1,18,1)
plot(r2$row, r2$r2_train, col='red', ylim=c(0,1))
points(r2$row, r2$mean_r2, col='blue')
```



6.1.8 Step 5: Final random forest

Once the combination of hyper-parameters to be used is defined, then the model can be fitted considering those hyper-parameter values. The chunk of code below defines the hyper-parameters to be used in the final random forest model and fits the respective model to the training data. This model will be further interpreted with the *iml* R package in the next sections.

```
# set final hyper-parameter values
ntrees_model <- c(500)
nodesize_model <- c(55)
tuneGrid_model <- expand.grid(.mtry=c(4))
#
# define cross-validation scheme
control_model <- trainControl(method="repeatedcv", number=10, repeats=5,␣
↪verboseIter=F)
#
# fit the final random forest model
rf_model_final <- train(cc_yield ~ .,
                        data=train_data,
                        method="rf",
                        tuneGrid=tuneGrid_model,
```

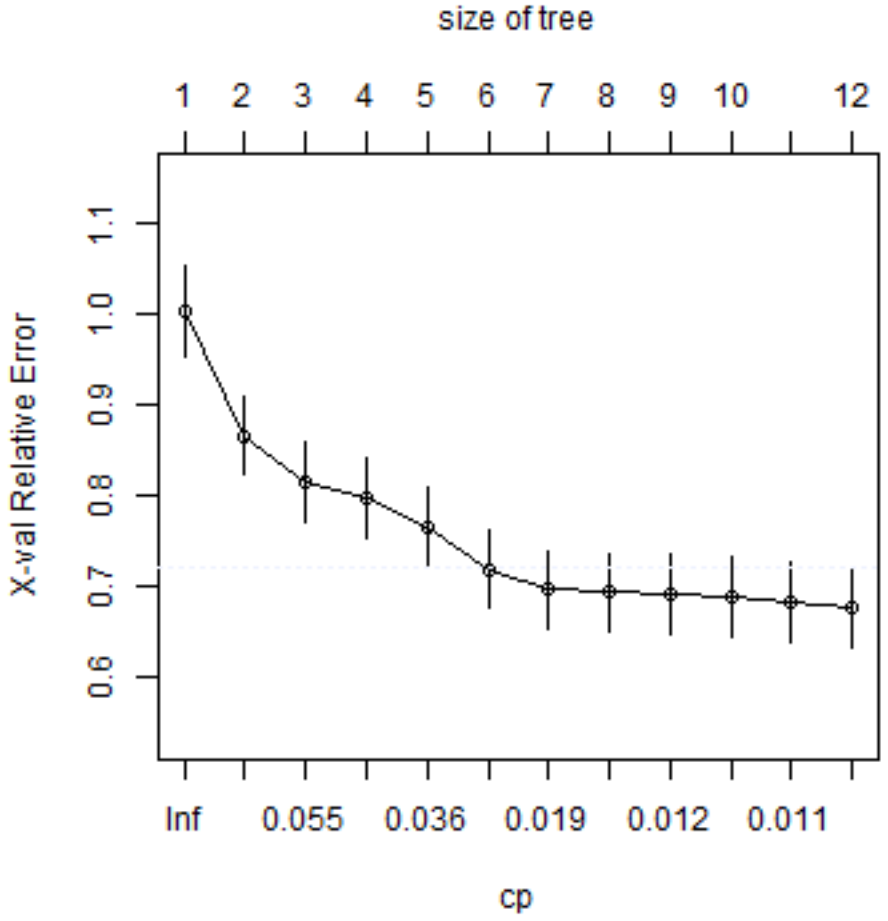
(continues on next page)

(continued from previous page)

```
                    trControl=control_model,
                    ntree=ntrees_model,
                    nodesize=nodesize_model)
```

Before proceeding to the direct interpretation of the random forest model, a classification and regression tree can be fitted to the data due to its high interpretability. This provides a first-order indication of the important variables and pathways leading to high crop yield.

```
# create data frame for the tree
train_data_tree <- train_data
#
# change variable names
names(train_data_tree)[1] = c("Yield")
#
# fit the regression tree
yield.model <- rpart(Yield ~ ., data=train_data_tree)
#
# get a good cp value by ploting the relative error vs. tree size
# the figure shows cp of 0.015 or 0.019 being optimum
plotcp(yield.model)
```



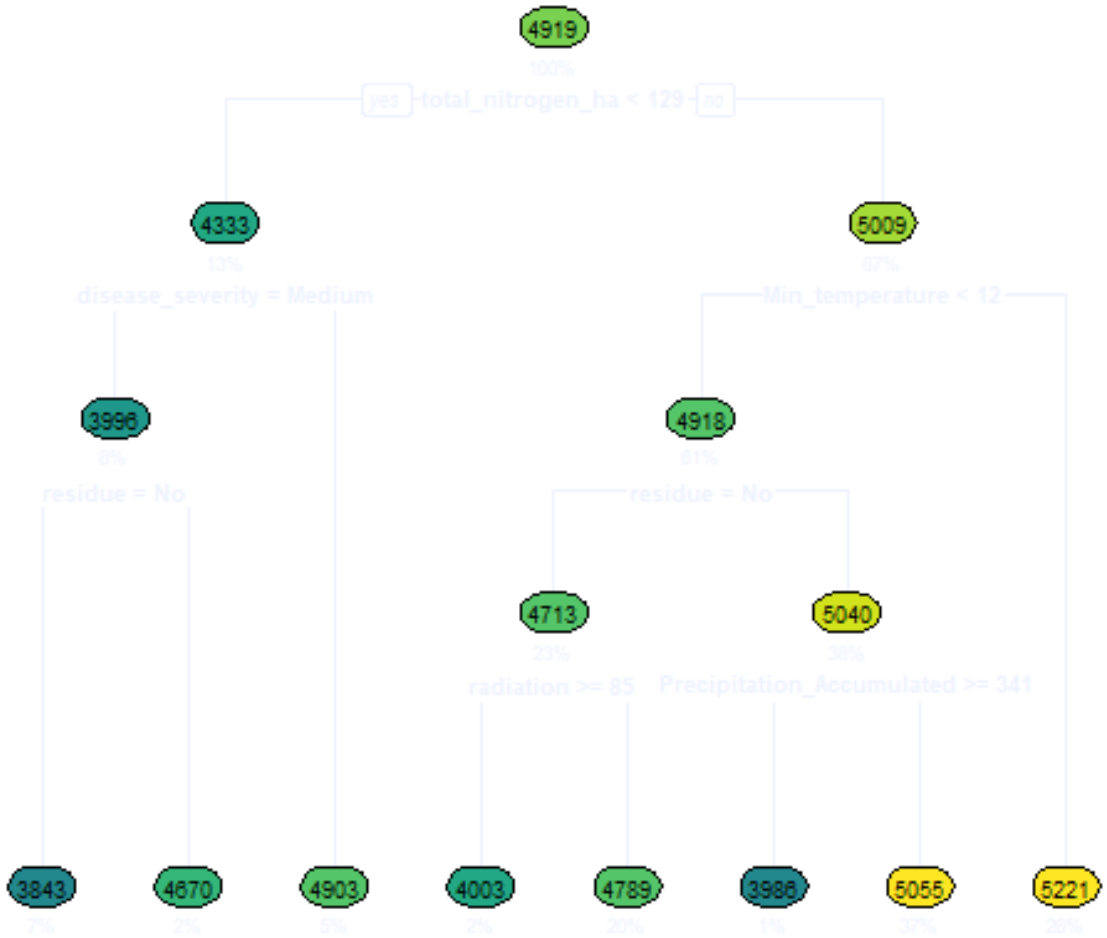

```
#
# fit the regression tree with the best cp value
```

(continues on next page)

(continued from previous page)

```
yield.model.opt = rpart(Yield ~ ., data=train_data_tree, cp=0.015)
```

```
# plot the regression tree
rpart.plot(yield.model.opt, type=2, extra="auto", round=0, under=T, box.palette=
↪"BlGnYl")
```



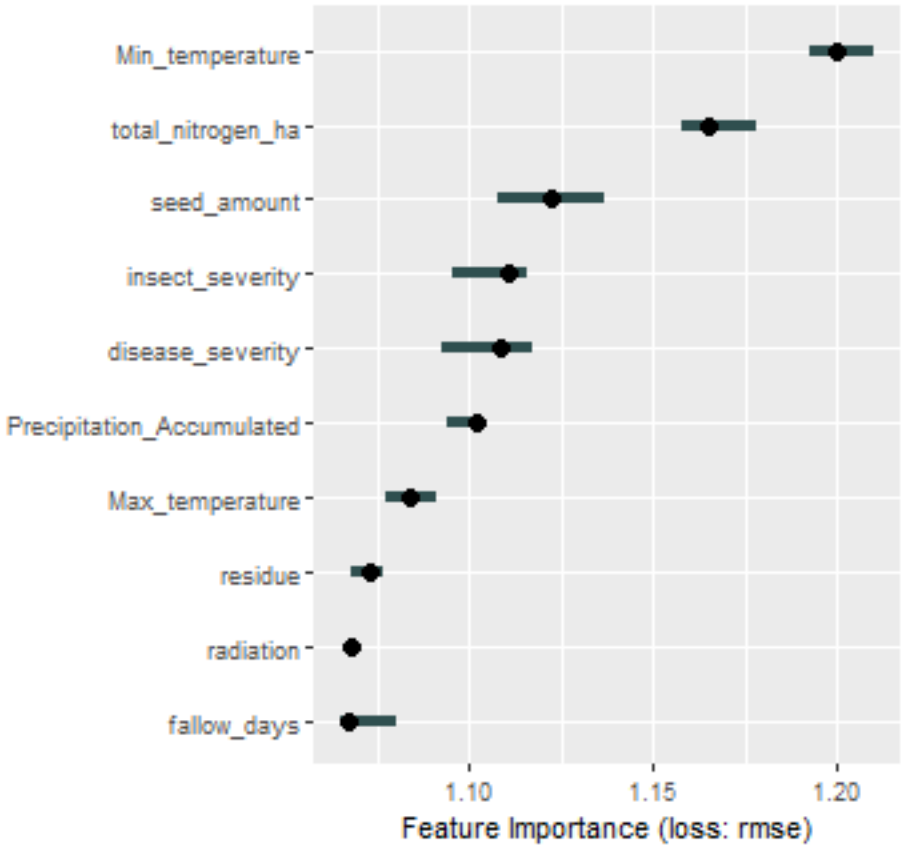
6.1.9 Step 6: Global interpretation

Model agnostic interpretation techniques can be used to interpret the tuned random forest model fitted in the previous section. Several techniques can be used for this purpose such as partial dependency plots, two-way interactions, and surrogate models, as demonstrated below.

Variable importance

The most common model interpretation technique is to visualize the variable importance plot. This plot indicates the most important variable governing yield variability, as shuffling that variable leads to considerable changes in model performance. In our data set, the most important variables are nitrogen applied and number of irrigations. Please refer to the code below to compute the variable importance plot.

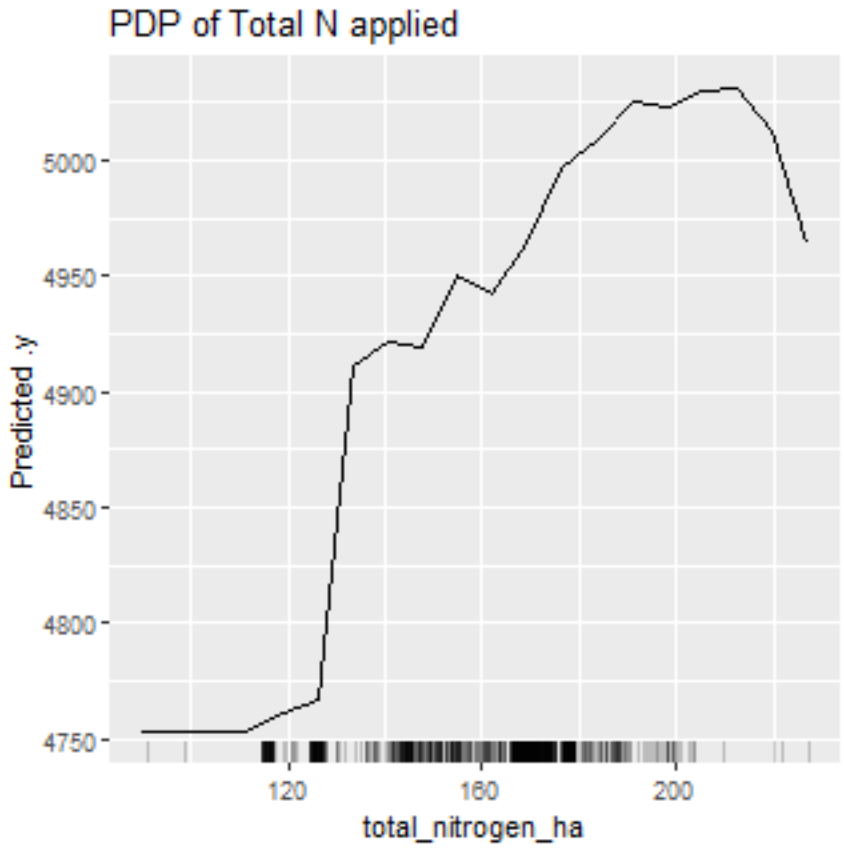
```
# select xy variables
X <- train_data[which(names(train_data) != "cc_yield")]
y <- train_data$cc_yield
#
# check variable importance based on model rmse
model_imp <- Predictor$new(rf_model_final, data=X, y=train_data$cc_yield)
imp_rf <- FeatureImp$new(model_imp, loss="rmse")
plot(imp_rf)
```



Partial dependency plots

Partial dependency plots for single variables are also commonly used for the interpreation of machine learning models. The interpretation of, what are the variable responsible for low yield vs high yield gives a direction how to brings the farmers from low yielding group to high yield.

```
# example partial dependency plot for nitrogen applied
pdp_totN <-
  FeatureEffect$new(model_imp, feature=c("total_nitrogen_ha"), method="pdp") %>%
↪plot() +
  ggtitle("PDP of Total N applied")
pdp_totN
```



Two-way interactions

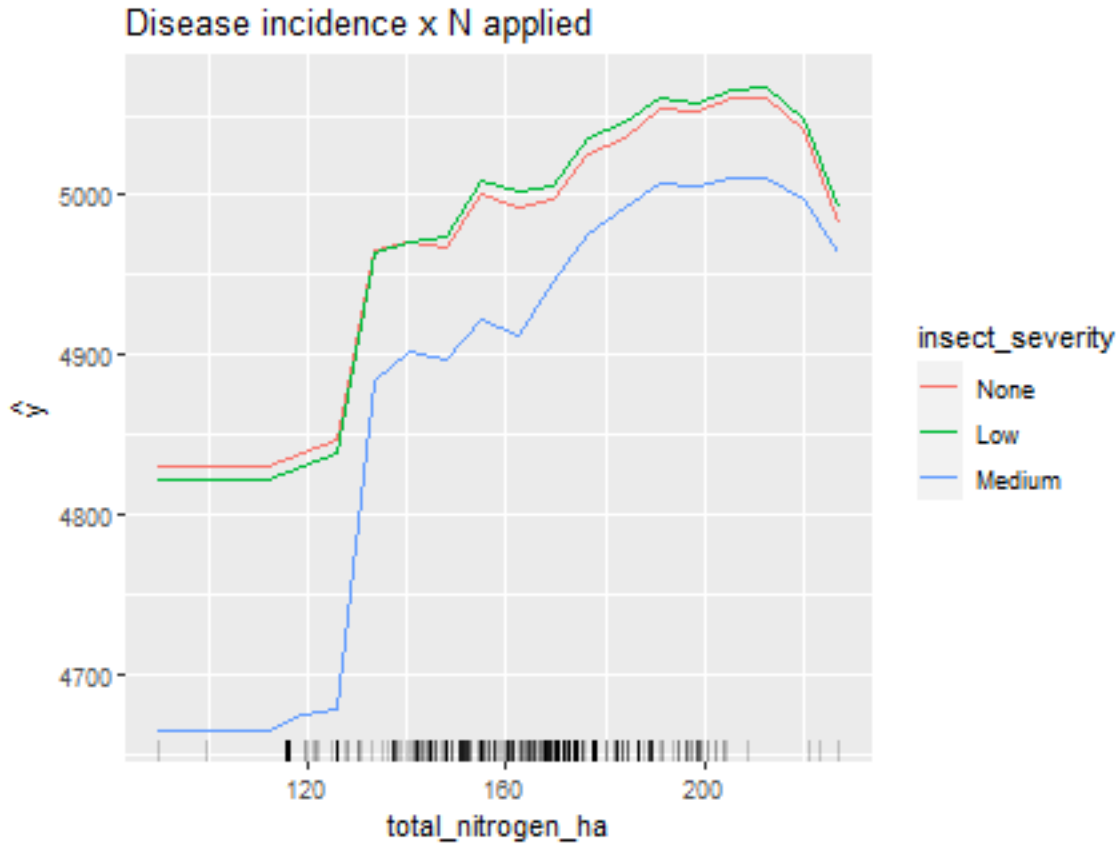
Another way to interpret the model is to visualize the impact of two-way interactions between important variables on crop yield. First we check interactions between a categorical and a continuous variable. Second we check the interaction between two continuous variables. This is done with the chunk of code below.

```
# overall two-way interaction strength
ia <- Interaction$new(model_imp)
#
# interaction between production practices
ia_disease_severity <- Interaction$new(model_imp,feature="insect_severity") # changed
↪from disease severity...
```

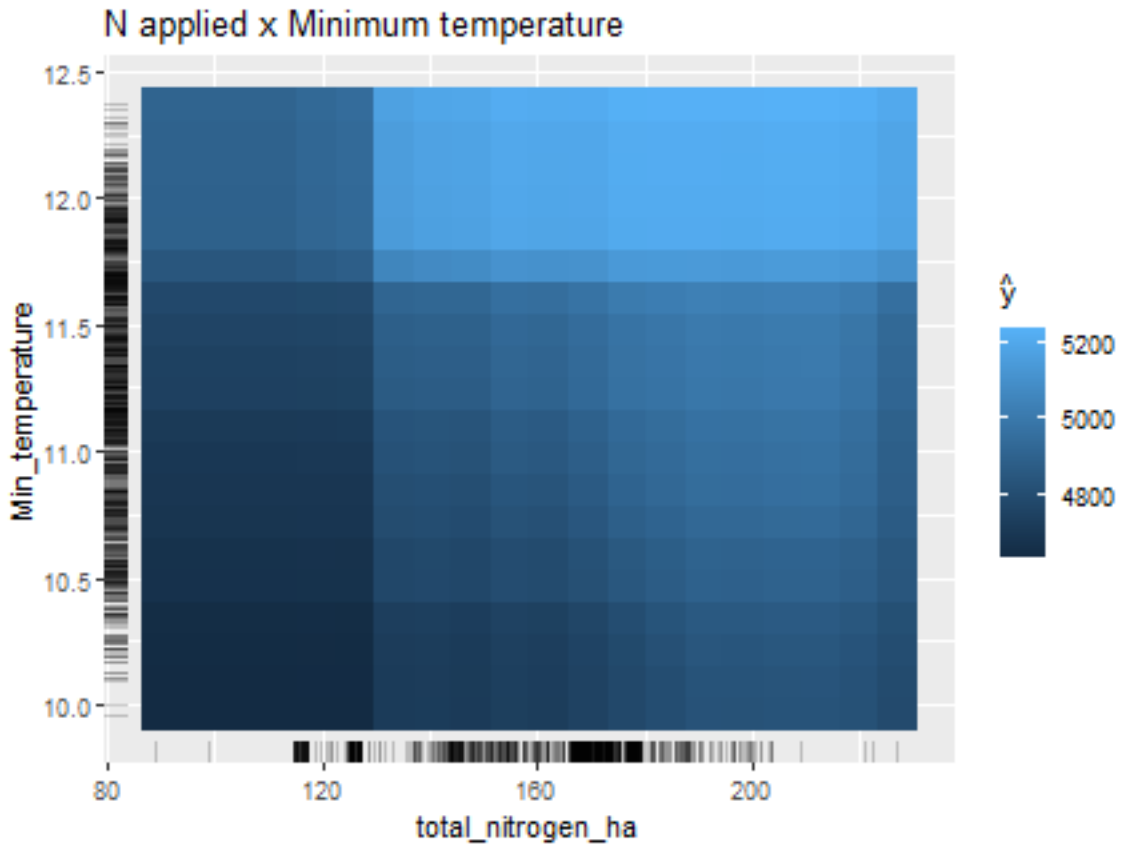
(continues on next page)

(continued from previous page)

```
#
# continuous x categorical variable
pdp_dis_N <-
  FeatureEffect$new(model_imp, c("insect_severity", "total_nitrogen_ha"), method="pdp
↪") %>% plot() +
  ggtitle("Disease incidence x N applied")
pdp_dis_N
```



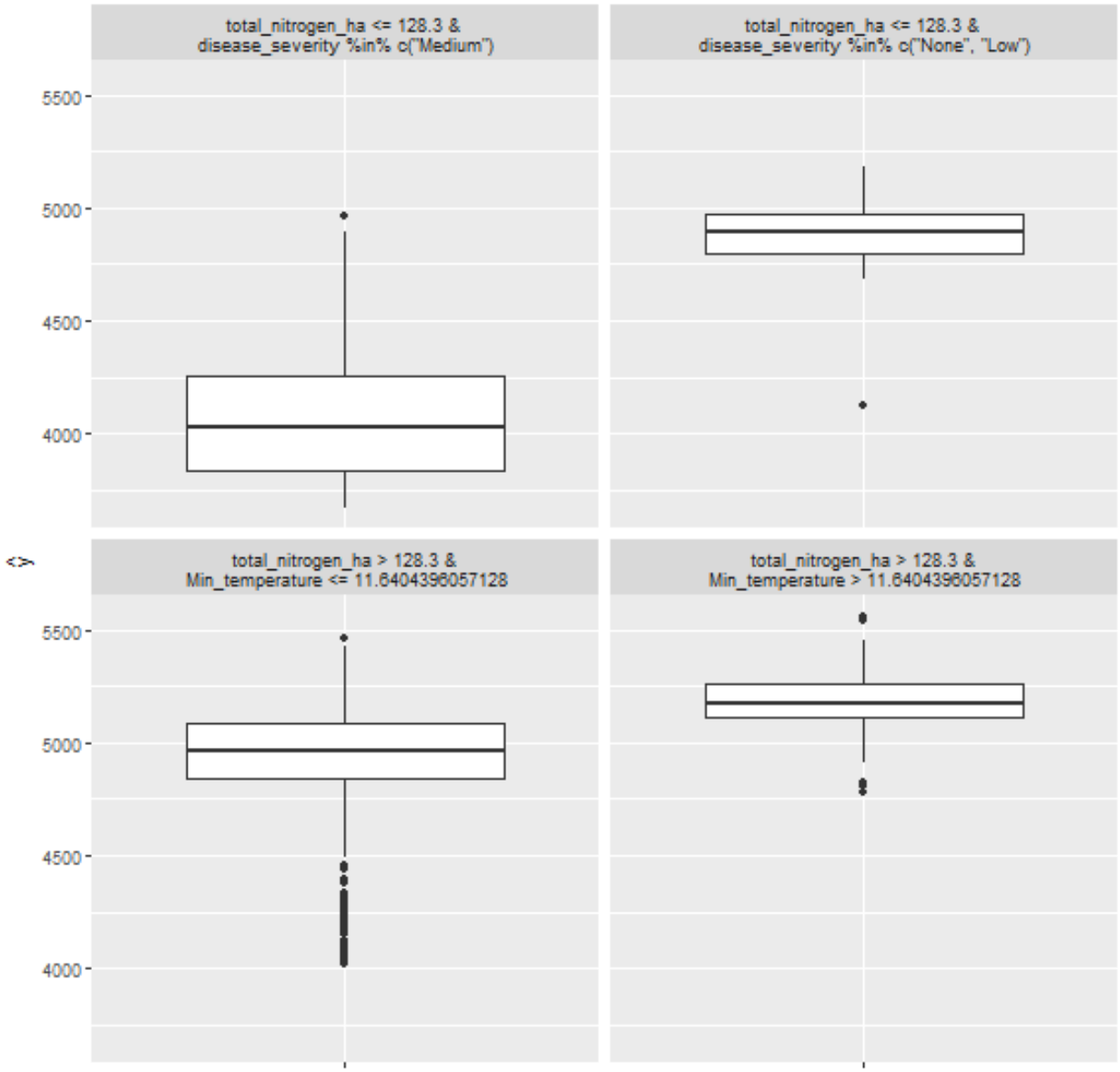

```
#
# continuous x continuous variable
pdp_N_temp <-
  FeatureEffect$new(model_imp, c("total_nitrogen_ha", "Min_temperature"), method="pdp
↪") %>% plot() +
  ggtitle("N applied x Minimum temperature")
pdp_N_temp
```



Surrogate model

Finally, boxplots can be produced to visualize the yield variability for the most important 'conditions' captured by the regression tree representing random forest model fitted. This can be visualized with the chunk of code below.

```
surrigate_tree <- TreeSurrogate$new(model_imp)
plot(surrigate_tree)
```



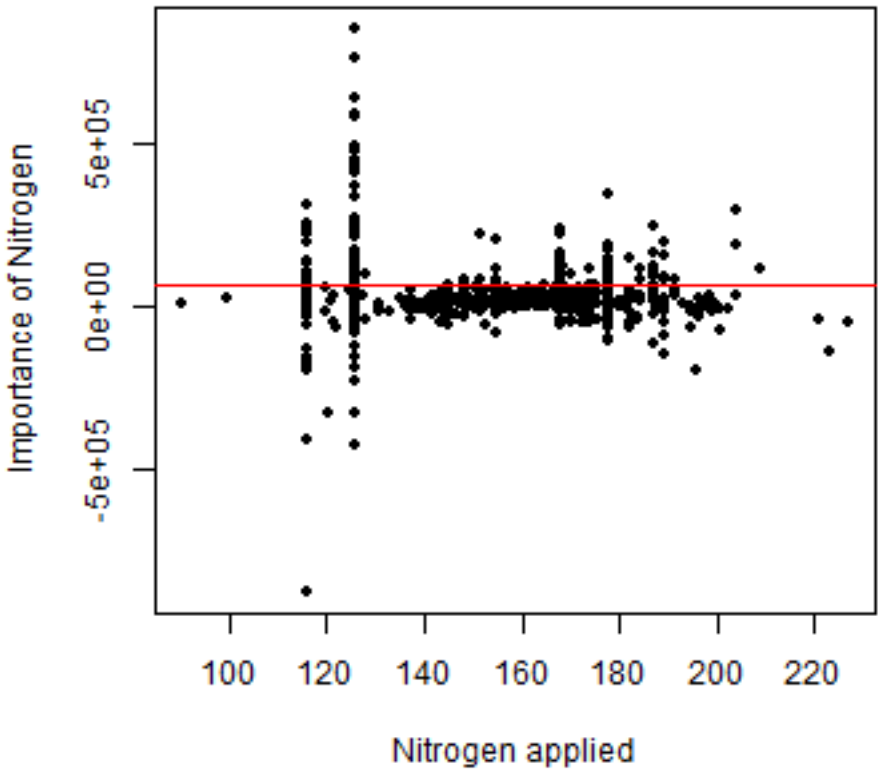
6.1.10 Step 7: Local interpretation

The above model interpretation relied on a single global interpretation using all the data, i.e., total N applied is important to explain on-farm wheat variability in the study region, followed by the number of irrigations and other factors. Yet, a single variable may not be equally important for all individual farms. Also, the extent to which a factor of production affects crop yield varies across the population, i.e., N applied may be more important for some fields and not that important for other fields. Local model interpretation techniques can thus help identifying find which variables are most important for specific subsets of the data. There are numerous techniques available for local model interpretation, one of which is illustrated in the chunk of code below.

```
# fit model with local importance
yield_model_localImp <- ranger::ranger(cc_yield ~ ., data=train_data, importance=
↪'permutation', local.importance=T, scale.permutation.importance=T, mtry=3)
#
# transform to data frame and print summary
local_imp <- data.frame(yield_model_localImp$variable.importance.local)
#
# global importance data of each variable
global_imp <- data.frame(ranger::importance(yield_model_localImp))
global_imp
##                          ranger..importance.yield_model_localImp.
## Min_temperature                                        115487.44
## residue                                                 74745.05
## total_nitrogen_ha                                       65529.60
## Precipitation_Accumulated                               82812.65
## seed_amount                                             46300.81
## Max_temperature                                         72325.22
## disease_severity                                        41882.88
## radiation                                               67767.79
## insect_severity                                         38022.74
## fallow_days                                             43972.72
```

The chunk of code below is used to visualize how the local importance changes across a range of inputs use. These results are compared with the global importance seen above, as they both have the same unit.

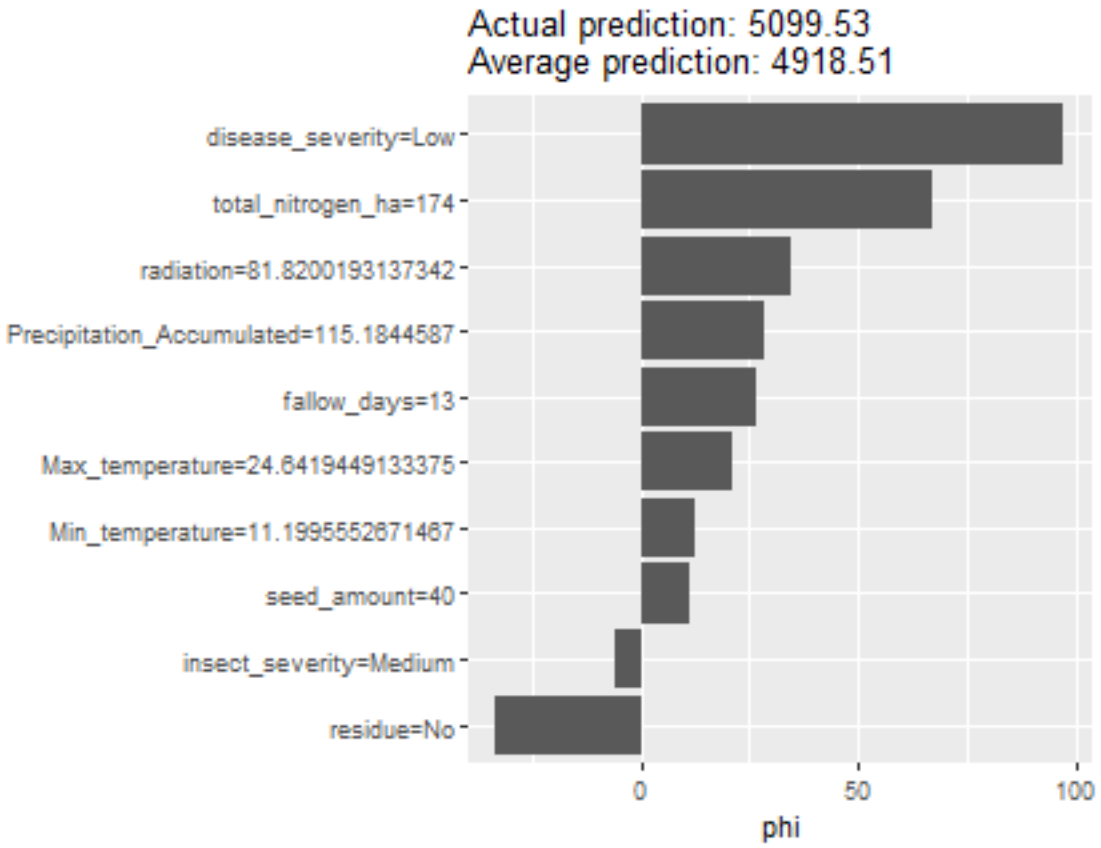
```
plot(train_data$total_nitrogen_ha, yield_model_localImp$variable.importance.local[,
↪"total_nitrogen_ha"], xlab="Nitrogen applied", ylab="Importance of Nitrogen", pch =␣
↪20)
abline(h=ranger::importance(yield_model_localImp)["total_nitrogen_ha"], col='red')
```



6.1.11 Step 8: Shapely values

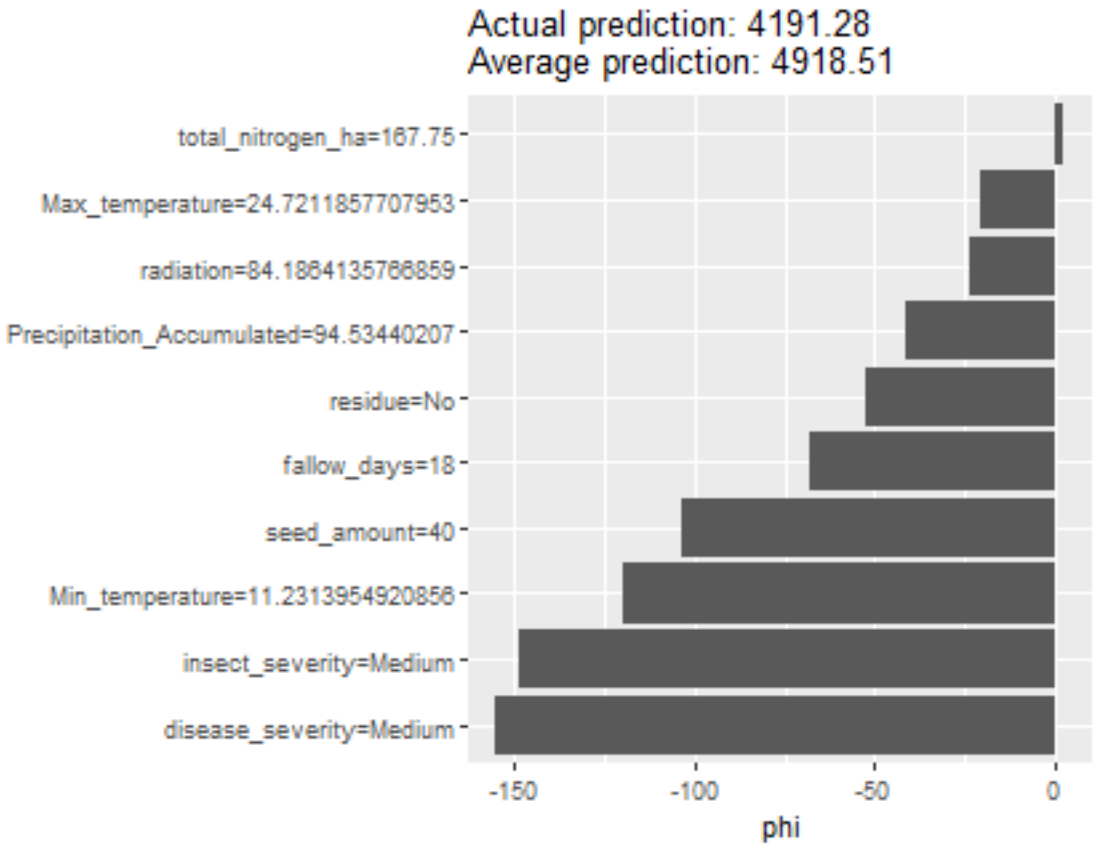
The most important feature for yield gap decomposition is the estimation of shapely values from a tuned random forest model. The **Shapely additive explanation technique** ranks individual variables and identifies the yield contribution of each variable as deviations from mean predicted yield by the model. More information about Shapely values can be found here. More recently, Shapely values have been applied to machine learning models to estimate the contribution of individual variables to the final model prediction. Thus, Shapely values indicate the marginal contribution of each predictor to the final yield prediction as compared to a baseline prediction. Shapely values can be estimated using the chunk of code below.

```
# used model prediction
X <- train_data[which(names(train_data) != "cc_yield")]
y <- train_data$cc_yield
model_imp <- Predictor$new(rf_model_final, data=X, y=train_data$cc_yield)
#
# get shapely values
shap.explain <- iml::Shapley$new(model_imp, x.interest=X[1,])
plot(shap.explain)
```



The chunk of code below finds the minimum observed yield and describes with Shapely the cause of that minimum yield. This can be generalized to any field, for which row indices need to be modified.

```
ix.minyld <- which.min(y)
shap.explain.minyld <- iml::Shapley$new(model_imp, x.interest=X[ix.minyld,])
plot(shap.explain.minyld)
```



6.1.12 Recommendations

Keep calm and let the machine learn =)

This section provides the workflow to conduct yield gap decomposition using interpretable machine learning techniques. Machine learning lies at the intersection of computer science and statistics and has been used to unravel patterns in large datasets that are challenging to analyze with more conventional statistical approaches. In agriculture, machine learning is being increasingly used to predict crop yield, although the research community remains divided on which methods are most appropriate given different data types and contexts. With this workflow, you will learn how to (a) perform comparisons of machine learning models, (b) conduct recursive feature elimination, (c) tune models hyper-parameters, (d) assess model performance, and (e) fit and interpret a tuned machine learning model. Estimation of Shapely values will be explained at the end. A companion slide deck can be assessed here.