2.Gene_expresion

Fay

2022-05-27

Aim: - Predicting health impact of infections utilizing immune parameters as predictors - Predicted variable: WL as a proxy of health - To do that we are using immune data from experimental lab infections. - We are training random forest models on the immune data from experimental lab infections - And we test them on the field. - We then compare the differences in the predicted health impact among non-hybrid and hybrid mice.

In this document I am preparing the models using the lab data only.

Libraries:

```
#install.packages("optimx", version = "2021-10.12") # this package is required for
#the parasite load package to work
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidyr)
library(dplyr)
library(cowplot)
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(VIM) # visualizing missing data
```

```
## Loading required package: colorspace

## Loading required package: grid

## VIM is ready to use.

## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues

##
## Attaching package: 'VIM'

## The following object is masked from 'package:datasets':
##
##     sleep
```

```
library(mice) # imputing missing data without predictors
```

```
##
## Attaching package: 'mice'

## The following object is masked from 'package:stats':
##
##     filter

## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
library(ggpubr)
```

```
##
## Attaching package: 'ggpubr'

## The following object is masked from 'package:cowplot':
##
##     get_legend
```

```
library(optimx)
```

Import the data:

We start with the data from experimental lab infections.

```
# Here we import the cleaned data set from the previous script derived from the
# data set challenge infections
g <- read.csv("https://raw.githubusercontent.com/fayweb/Eimeria_mouse_immunity/main/output_data/gene_ex


# vectors for selecting gene columns
Genes <- c("IFNy", "CXCR3_bio", "IL.6", "IL.10", "IL.13", "IL.10", "IL.13", "IL1RN",
           "CASP1", "CXCL9", "IDO1", "IRGM1", "MPO", "MUC2", "MUC5AC", "MYD88",
           "NCR1", "PRF1", "RETNLB", "SOCS1", "TICAM1", "TNF")
```

Data cleaning

```
# we need to change the  in challenge infections to a factor
g$Parasite_challenge <- as.factor(g$Parasite_challenge)
g$Eim_MC <- as.factor(g$Eim_MC)

# Here I create a new column, where we get the actual infection status
# According to the melting curve for eimeria
g <- g %>%
  dplyr::mutate(current_infection = case_when(
    Parasite_challenge == "E_ferrisi" & Eim_MC == "TRUE" ~ "E_ferrisi",
    Parasite_challenge == "E_ferrisi" & Eim_MC == "FALSE" ~ "uninfected",
    Parasite_challenge == "E_falciformis" & Eim_MC == "TRUE" ~ "E_falciformis",
    Parasite_challenge == "E_falciformis" & Eim_MC == "FALSE" ~ "uninfected",
    Parasite_challenge == "uninfected" & Eim_MC == "TRUE" ~ "infected_eimeria",
    Parasite_challenge == "uninfected" & Eim_MC == "FALSE" ~ "uninfected",
    TRUE ~ ""
  ))


# how to impute delta? Replacing with 0 the ones with negative melting curve
# open for other solutions!
g <- g %>%
  dplyr::mutate(Intensity = case_when(
    Eim_MC == "TRUE" ~ delta,
    Eim_MC == "FALSE" ~ 0))

# create variable maximum weight loss instead of maximum relative weight loss
g <- g %>% dplyr::mutate(max_WL = 100 - max_WL)
```

Imputing missing data + cleaning

```
#Start by selecting only the genes and the maximum weight loss for each mouse
# Apparently the relative end weight doesn't work so well for predictions

g.1 <- g %>%
  dplyr::select(c(max_WL, all_of(Genes)))

# to get reproducible results we use a seed
set.seed(42)

# We want the maximum weight loss to be predicted by the data ina ll of the other columns

# iter = how many random forests are needed, in theory 6 are enough
g.imputed <- rfImpute(max_WL ~ ., data = g.1, iter = 6)
```

```
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |     26.1    61.11 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    27.08    63.39 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
```

```
##  300 |    27.97    65.49 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    28.27    66.19 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    28.26    66.17 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    28.24    66.11 |
```

```
g.imputed <- g.imputed %>% dplyr::select(-max_WL)

g_minus <- g %>%
  dplyr::select(-all_of(Genes))

#full data set containing the imputed gene expression data
g.imputed <- cbind(g_minus, g.imputed)
```

How many mice are in the infection planning?

```
g.imputed %>%
  filter(infection == "challenge") %>%
  group_by(Parasite_challenge) %>%
  summarize(length(EH_ID))
```

```
## # A tibble: 3 x 2
##   Parasite_challenge `length(EH_ID)`
##   <fct>                         <int>
## 1 E_falciformis                    22
## 2 E_ferrisi                        47
## 3 uninfected                       47
```

How many mice are indeed infected?

```
g.imputed %>%
  filter(infection == "challenge") %>%
  group_by(current_infection) %>%
  summarize(length(EH_ID))
```

```
## # A tibble: 4 x 2
##   current_infection `length(EH_ID)`
##   <chr>                       <int>
## 1 E_falciformis                  22
## 2 E_ferrisi                      39
## 3 infected_eimeria                9
## 4 uninfected                     46
```

I guess mice got mixed up here?

Splitting data into training and testing sets

Splitting between training and testing: - Assess model performance on unseen data - Avoid over-fitting

```
g.imputed_full <- g.imputed

#select the relevant columns:
g.imputed <- g.imputed %>%
```

```
  dplyr::select(c(max_WL, all_of(Genes))) %>%
  dplyr::select(- CXCR3_bio) # remove this as not included in field

# split data into training and test

set.seed(123) # this will help us reproduce this random assignment

# in this way we can pick the random numbers

training.samples <- g.imputed$max_WL%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return

train.data <- g.imputed[training.samples, ] #include all the randomly selected rows
test.data <- g.imputed[-training.samples, ]
```

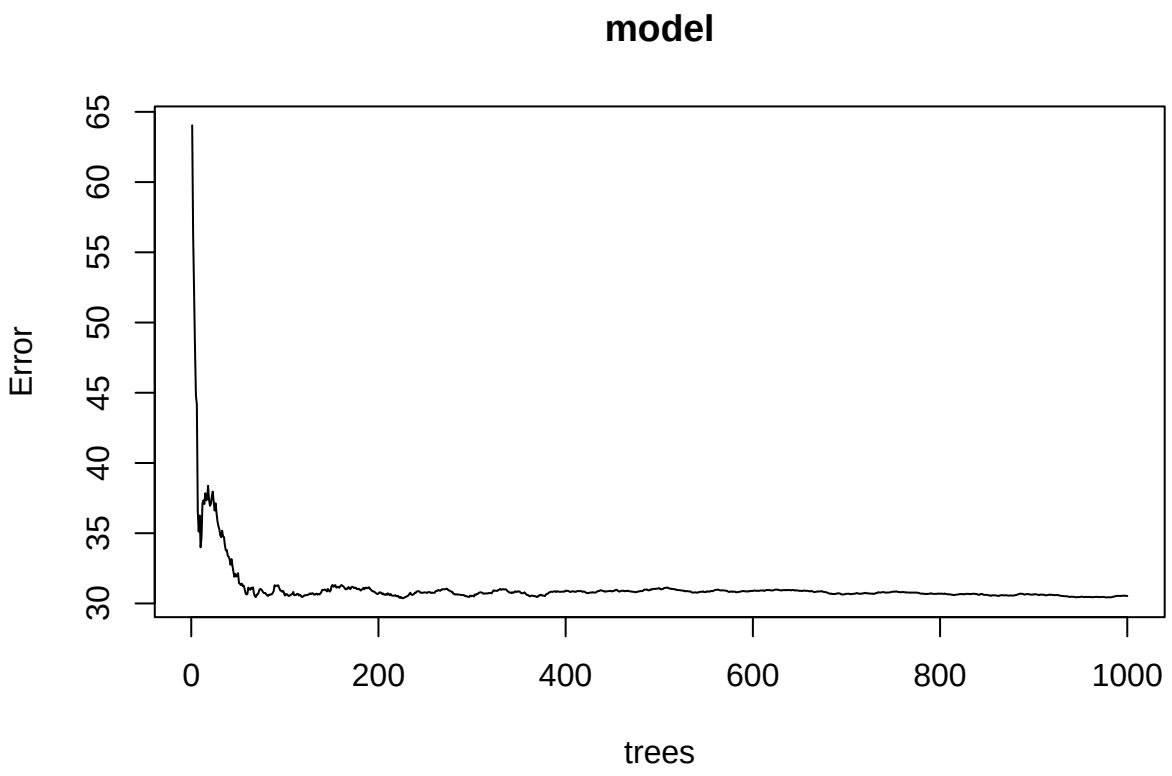
Building the model

```
#train the model
model <- randomForest(max_WL ~., data = train.data, proximity = TRUE,
                      ntree = 1000) # number of trees


print(model)
```

```
##
## Call:
##  randomForest(formula = max_WL ~ ., data = train.data, proximity = TRUE,      ntree = 1000)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 30.52161
##                     % Var explained: 27
```

Plotting the model will illustrate the error rate as we average across more trees and shows that our error rate stabalizes with around 200 trees.

```
plot(model)
```



The plotted error rate above is based on the OOB sample error and can be accessed directly at m1$mse. Thus, we can find which number of trees providing the lowest error rate, which is 257 trees providing an weight error of 5.024738.

```
# number of trees with lowest MSE
which.min(model$mse)
```

```
## [1] 226
```

```
## [1] 257

# RMSE of this optimal random forest
sqrt(model$mse[which.min(model$mse)])
```

```
## [1] 5.510924
```

```
## [1] 5.024738
```

https://uc-r.github.io/s

RandomForest also allows us to use a validation set to measure predictive accuracy if we did not want to use the OOB samples.

Tutorial: https://hackernoon.com/random-forest-regression-in-r-code-and-interpretation Random forest regression in R provides two outputs: decrease in mean square error (MSE) and node purity. Prediction error described as MSE is based on permuting out-of-bag sections of the data per individual tree and predictor, and the errors are then averaged. In the regression context, Node purity is the total decrease in residual sum of squares when splitting on a variable averaged over all trees (i.e. how well a predictor decreases variance). MSE is a more reliable measure of variable importance. If the two importance metrics show different results, listen to MSE. If all of your predictors are numerical, then it shouldn't be too much of an issue

Mean Decrease Gini (IncNodePurity) - This is a measure of variable importance based on the Gini impurity index used for the calculating the splits in trees.

Improving Your Model Your model depends on the quality of your dataset and the type of Machine Learning algorithm used. Therefore, to improve the accuracy of your model, you should:

Check what attributes affect our model the most and what variables to leave out in future analysis Find out what other attributes affect a person's wage; we can use as predictors in future analysis Tweak the algorithm (e.g. change the ntree value) Use a different machine learning algorithm If any of these reduces the RMSE significantly, you have succeeded in improving your model!

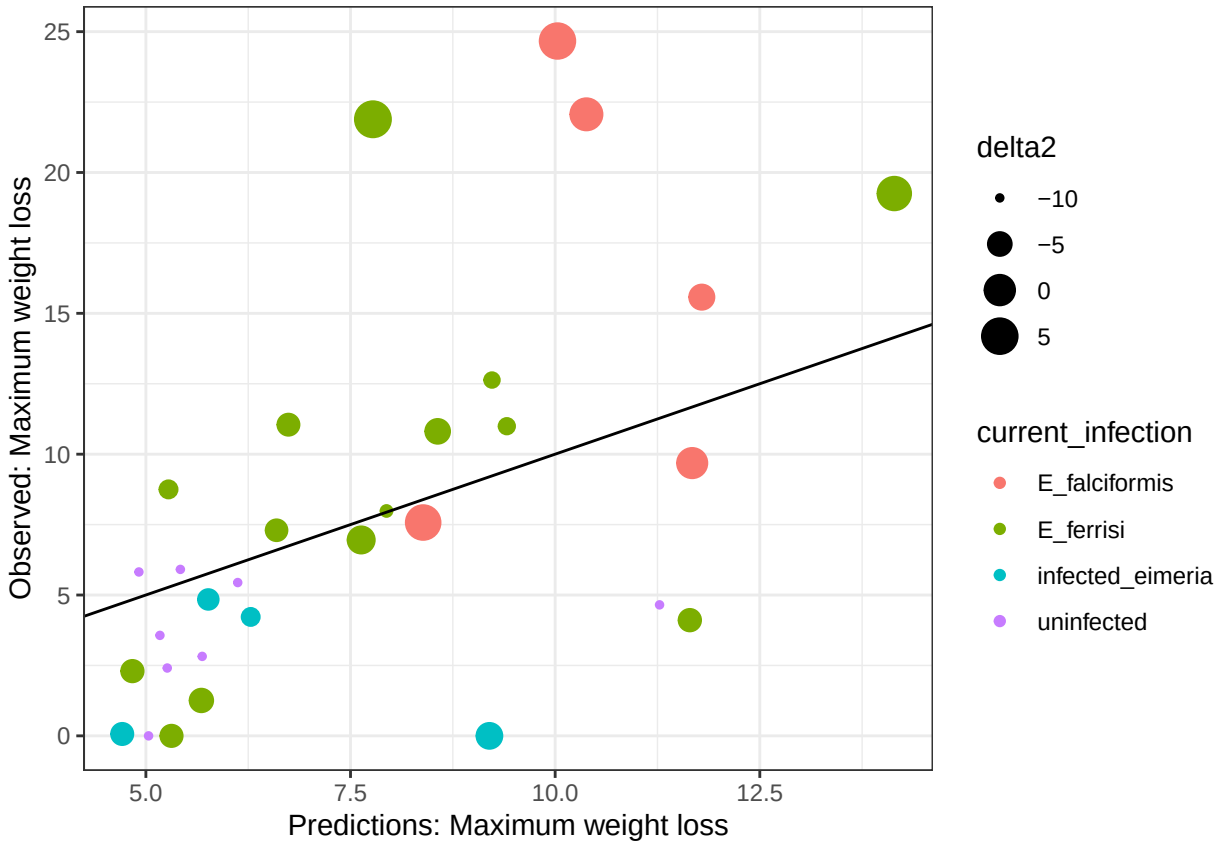
Making predictions

Let's now make some predictions using our test data.

```
#The predict() function in R is used to predict the values based on the input data.
predictions <- predict(model, test.data)

# assign test.data to a new object, so that we can make changes
result <- test.data

#add the new variable of predictions to the result object
result <- cbind(result, predictions)

#add the results to a data frame containing test data and the prediction
result <- cbind(g[row.names(result), ], predictions)
```

Visualizations

```
# trying to find a way to represent the delta ct for the negative ones
# please find a better way to do this
result <- result %>%
    dplyr::mutate(delta2 = case_when(
       current_infection == "uninfected" ~ -10,
       TRUE ~ delta
    ))

result   %>%
  ggplot() +
  geom_point(aes(x = predictions, y = max_WL, color = current_infection, size = delta2)) +
  geom_abline() +
  labs(x = "Predictions: Maximum weight loss", y = "Observed: Maximum weight loss") +
    theme_bw()
```



Predicting eimeria species according to gene expression

```
g$Parasite_challenge <- as.factor(g$Parasite_challenge)

#now select the genes and the actual infection of the mice in the new mutate column
#infection
g.2 <- g %>%
    dplyr::select(c(Parasite_challenge, delta, all_of(Genes)))

# to get reproducible results we use a seed
set.seed(42)
# We want the current infection to be predicted by the data ina ll of the other columns
# iter = how many random forests are needed, in theory 6 are enough


#now we can impute our data
g.imputed_parasite <- rfImpute(Parasite_challenge ~ ., data = g.2, iter = 6)
```

```
## ntree      OOB      1      2      3
##   300:  31.03% 54.55% 31.91% 19.15%
## ntree      OOB      1      2      3
##   300:  30.17% 50.00% 29.79% 21.28%
## ntree      OOB      1      2      3
##   300:  27.59% 45.45% 29.79% 17.02%
## ntree      OOB      1      2      3
##   300:  27.59% 50.00% 27.66% 17.02%
```

```
## ntree      OOB      1      2      3
##   300:  29.31% 45.45% 31.91% 19.15%
## ntree      OOB      1      2      3
##   300:  27.59% 50.00% 27.66% 17.02%
```

```
g.imputed_parasite <- g.imputed_parasite %>% dplyr::select(- Parasite_challenge)

g_minus <- g %>% dplyr::select(-c((all_of(Genes)), delta))

#full data set containing the imputed gene expression data
g.imputed_parasite <- cbind(g_minus, g.imputed_parasite)
```

Now split the data again into training and testing

```
g.imputed_parasite$Parasite_challenge <- as.factor(g.imputed_parasite$Parasite_challenge)

#select the relevant columns:
g.imputed_parasite <- g.imputed_parasite %>%
  dplyr::select(c(Parasite_challenge, all_of(Genes)))

# split data into training and test
set.seed(123) # this will help us reproduce this random assignment
# in this way we can pick the random numbers
training.samples_parasite <- g.imputed_parasite$Parasite_challenge%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return
train.data_parasite <- g.imputed_parasite[training.samples, ] #include all the randomly selected rows
test.data_parasite <- g.imputed_parasite[-training.samples, ]
```

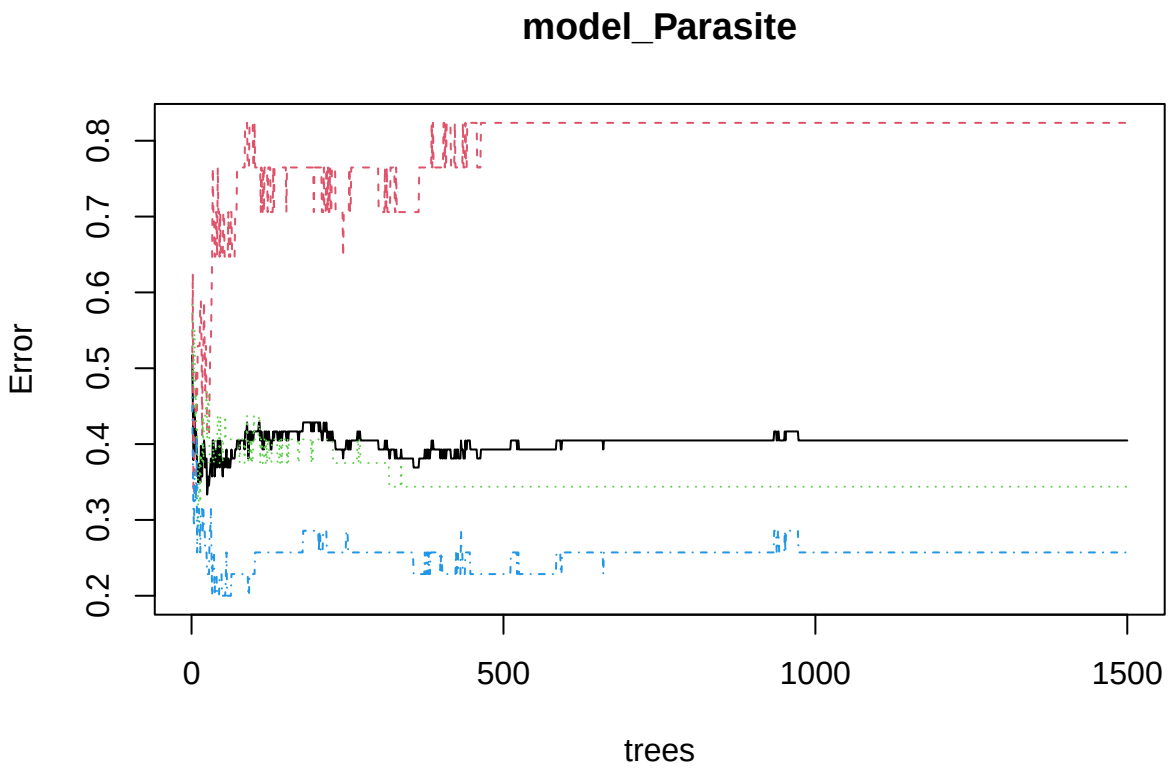
Building the model

```
#train the model
model_Parasite <- randomForest(Parasite_challenge ~., data = train.data_parasite, proximity = TRUE,
                       ntree = 1500) # number of trees

print(model_Parasite)
```

```
##
## Call:
##  randomForest(formula = Parasite_challenge ~ ., data = train.data_parasite,      proximity = TRUE, n
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 40.48%
## Confusion matrix:
##               E_falciformis E_ferrisi uninfected class.error
## E_falciformis             3         8          6   0.8235294
## E_ferrisi                 5        21          6   0.3437500
## uninfected                2         7         26   0.2571429
```

OOB = 46.43, this means that only 53 % of our predictions are accurate

```
plot(model_Parasite)
```



Test the model

Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions_parasite <- predict(model_Parasite, test.data_parasite)
# assign test.data to a new object, so that we can make changes
result_parasite <- test.data_parasite
#add the new variable of predictions to the result object
result_parasite <- cbind(result_parasite, predictions_parasite)
#add the results to a data frame containing test data and the prediction
result_parasite <- cbind(g[row.names(result_parasite), ], predictions_parasite)
```

Predicting eimeria species according to gene expression

```
g$Parasite_challenge <- as.factor(g$Parasite_challenge)

#now select the genes and the actual infection of the mice in the new mutate column
#infection
g.2 <- g %>%
    dplyr::select(c(Parasite_challenge, delta, all_of(Genes)))

# to get reproducible results we use a seed
set.seed(42)
# We want the current infection to be predicted by the data ina ll of the other columns
# iter = how many random forests are needed, in theory 6 are enough
```

```
#now we can impute our data
g.imputed_parasite <- rfImpute(Parasite_challenge ~ ., data = g.2, iter = 6)
```

```
## ntree      OOB      1      2      3
##   300:  31.03% 54.55% 31.91% 19.15%
## ntree      OOB      1      2      3
##   300:  30.17% 50.00% 29.79% 21.28%
## ntree      OOB      1      2      3
##   300:  27.59% 45.45% 29.79% 17.02%
## ntree      OOB      1      2      3
##   300:  27.59% 50.00% 27.66% 17.02%
## ntree      OOB      1      2      3
##   300:  29.31% 45.45% 31.91% 19.15%
## ntree      OOB      1      2      3
##   300:  27.59% 50.00% 27.66% 17.02%
```

```
g.imputed_parasite <- g.imputed_parasite %>% dplyr::select(- Parasite_challenge)

g_minus <- g %>% dplyr::select(-c((all_of(Genes)), delta))

#full data set containing the imputed gene expression data
g.imputed_parasite <- cbind(g_minus, g.imputed_parasite)
```

Now split the data again into training and testing

```
g.imputed_parasite$Parasite_challenge <- as.factor(g.imputed_parasite$Parasite_challenge)

#select the relevant columns:
g.imputed_parasite <- g.imputed_parasite %>%
  dplyr::select(c(Parasite_challenge, Eim_MC, all_of(Genes)))

# to use in the next model
parasite_data <- g.imputed_parasite

g.imputed_parasite <- g.imputed_parasite %>%
  dplyr::select(-Eim_MC)

# split data into training and test
set.seed(123) # this will help us reproduce this random assignment
# in this way we can pick the random numbers
training.samples_parasite <- g.imputed_parasite$Parasite_challenge%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return
train.data_parasite <- g.imputed_parasite[training.samples, ] #include all the randomly selected rows
test.data_parasite <- g.imputed_parasite[-training.samples, ]
```

Building the model

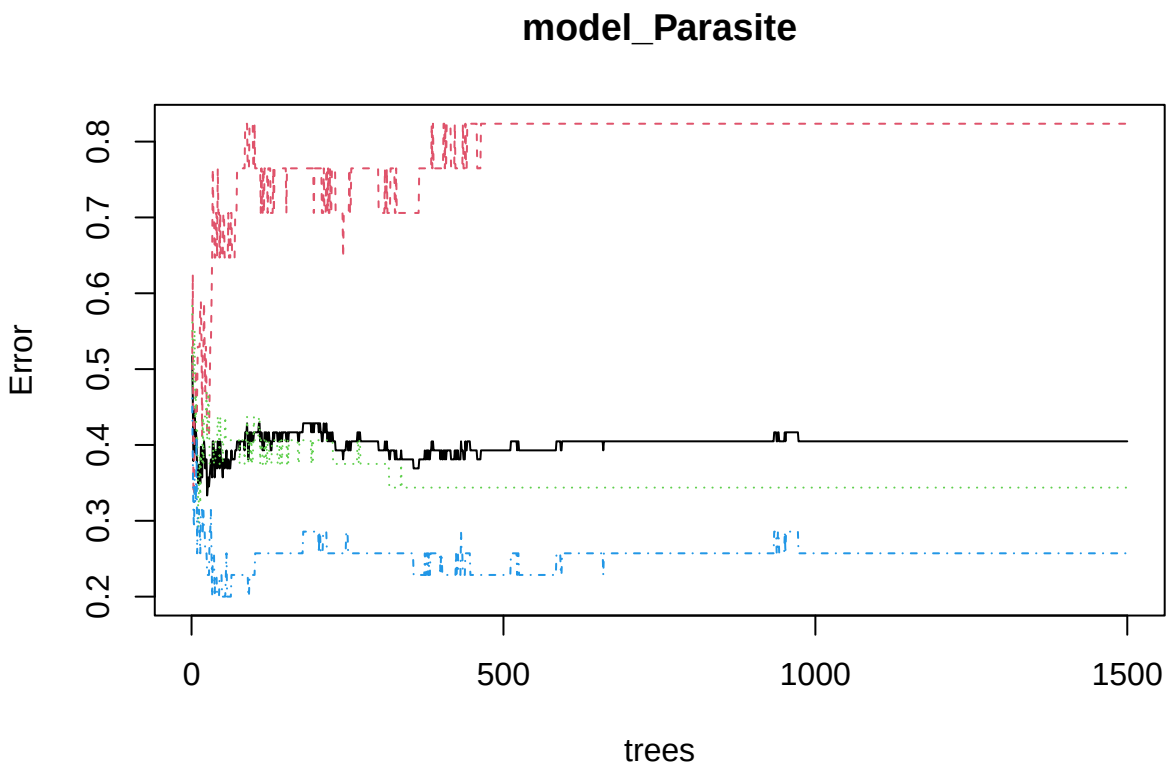
```
#train the model
model_Parasite <- randomForest(Parasite_challenge ~., data = train.data_parasite, proximity = TRUE,
                      ntree = 1500) # number of trees
```

```
print(model_Parasite)
```

```
##
## Call:
##  randomForest(formula = Parasite_challenge ~ ., data = train.data_parasite,      proximity = TRUE, n
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 40.48%
## Confusion matrix:
##               E_falciformis E_ferrisi uninfected class.error
## E_falciformis             3         8          6   0.8235294
## E_ferrisi                 5        21          6   0.3437500
## uninfected                2         7         26   0.2571429
```

OOB = 46.43, this means that only 53 % of our predictions are accurate

```
plot(model_Parasite)
```



Test the model

Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions_parasite <- predict(model_Parasite, test.data_parasite)
# assign test.data to a new object, so that we can make changes
result_parasite <- test.data_parasite
#add the new variable of predictions to the result object
result_parasite <- cbind(result_parasite, predictions_parasite)
```

```r
#add the results to a data frame containing test data and the prediction
result_parasite <- cbind(g[row.names(result_parasite), ], predictions_parasite)
```

Visualizations

```r
conf_matrix_parasite <- confusionMatrix(result_parasite$predictions_parasite, reference = result_parasi

print(conf_matrix_parasite)
```

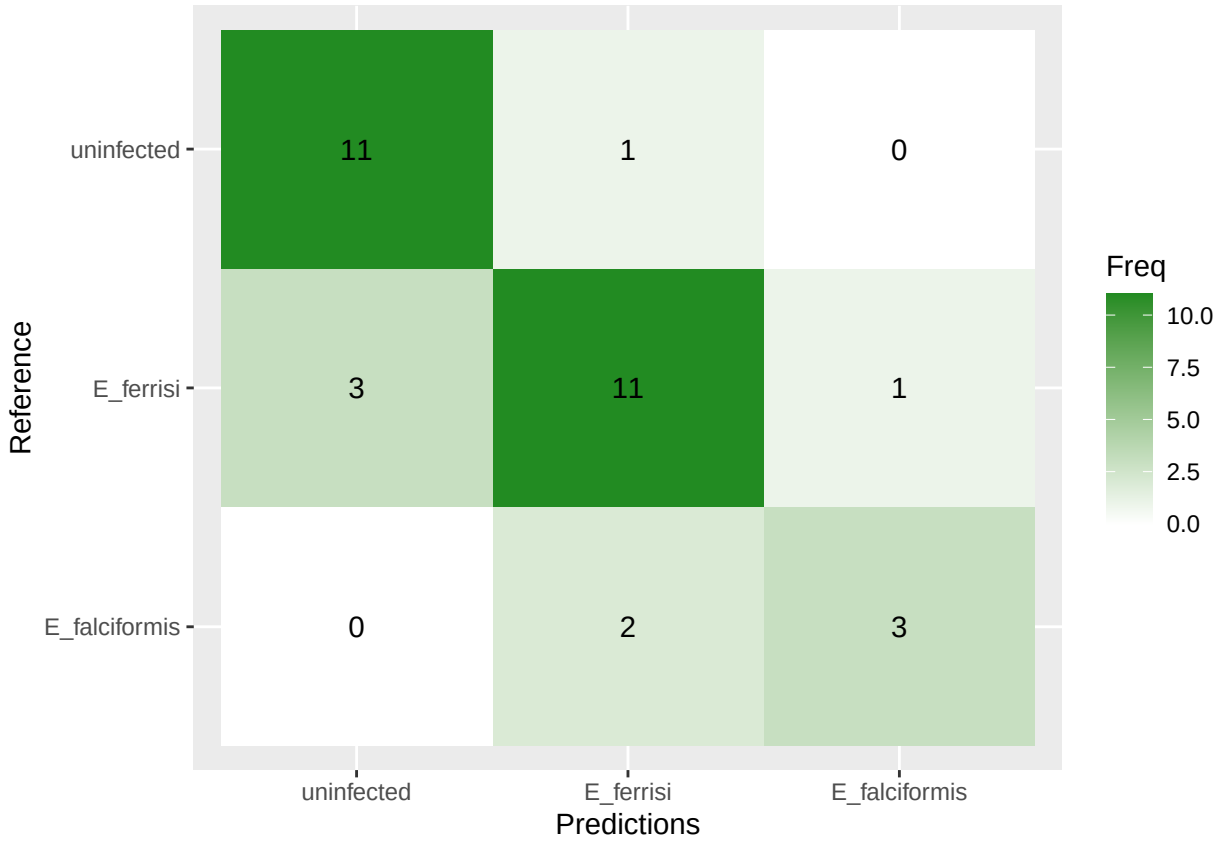
```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             3         1          0
##   E_ferrisi                 2        11          1
##   uninfected                0         3         11
##
## Overall Statistics
##
##                Accuracy : 0.7812
##                  95% CI : (0.6003, 0.9072)
##     No Information Rate : 0.4688
##     P-Value [Acc > NIR] : 0.0003084
##
##                   Kappa : 0.6422
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: E_falciformis Class: E_ferrisi Class: uninfected
## Sensitivity                       0.60000           0.7333            0.9167
## Specificity                       0.96296           0.8235            0.8500
## Pos Pred Value                    0.75000           0.7857            0.7857
## Neg Pred Value                    0.92857           0.7778            0.9444
## Prevalence                        0.15625           0.4688            0.3750
## Detection Rate                    0.09375           0.3438            0.3438
## Detection Prevalence              0.12500           0.4375            0.4375
## Balanced Accuracy                 0.78148           0.7784            0.8833
```

```r
conf_matrix_parasite$table
```

```
##                Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             3         1          0
##   E_ferrisi                 2        11          1
##   uninfected                0         3         11
```

```r
plt <- as.data.frame(conf_matrix_parasite$table)
plt$Prediction <- factor(plt$Prediction, levels=rev(levels(plt$Prediction)))

ggplot(plt, aes(x = Prediction, y =  Reference, fill= Freq)) +
        geom_tile() + geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="forestgreen") +
```

```
        labs(x = "Predictions",y = "Reference")
```




```
train.data_parasite %>%
  group_by(Parasite_challenge) %>%
  summarize(length(Parasite_challenge))
```

```
## # A tibble: 3 x 2
##   Parasite_challenge `length(Parasite_challenge)`
##   <fct>                                     <int>
## 1 E_falciformis                                17
## 2 E_ferrisi                                    32
## 3 uninfected                                   35
```

Repeat the previous model, this time testing for

infected with Eimeria or not. ###

```
# to use in the next model
parasite_data <- parasite_data %>%
  dplyr::select(-Parasite_challenge)

parasite_data <- parasite_data %>%
    dplyr::select(-CXCR3_bio)


# split data into training and test
set.seed(123) # this will help us reproduce this random assignment
# in this way we can pick the random numbers
```

```r
training.samples_melting <- parasite_data$Eim_MC%>%
  createDataPartition(p = .7, # this is the partiicition! In this case 0.7 = training data and 0.3 = te
                      list = FALSE) # we don't want to get a list in return
train.data_melting <- parasite_data[training.samples, ] #include all the randomly selected rows
test.data_melting <- parasite_data[-training.samples, ]
```

Building the model

```r
#train the model
model_melting <- randomForest(Eim_MC ~., data = train.data_melting, proximity = TRUE,
                      ntree = 1500) # number of trees

print(model_melting)
```

```
##
## Call:
##  randomForest(formula = Eim_MC ~ ., data = train.data_melting,      proximity = TRUE, ntree = 1500)
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 30.95%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE    22   15   0.4054054
## TRUE     11   36   0.2340426
```

Test the model

Making predictions

```r
#The predict() function in R is used to predict the values based on the input data.
predictions_melting <- predict(model_melting, test.data_melting)


# assign test.data to a new object, so that we can make changes
result_melting <- test.data_melting
#add the new variable of predictions to the result object
result_melting <- cbind(result_melting, predictions_melting)
#add the results to a data frame containing test data and the prediction
result_melting <- cbind(g[row.names(result_melting), ], predictions_melting)
```

```r
conf_matrix_melting <- confusionMatrix(result_melting$predictions_melting, reference = result_melting$E

print(conf_matrix_melting)
```

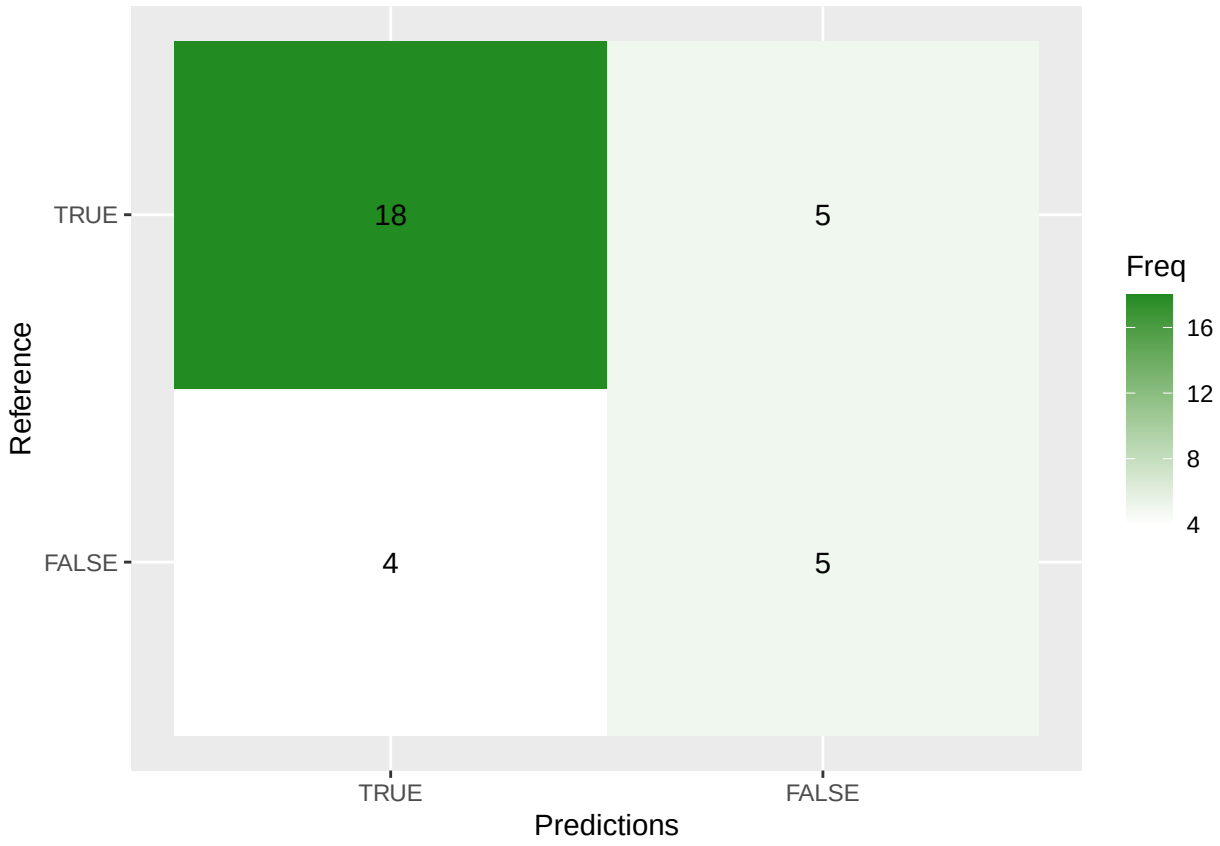
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE     5    5
##      TRUE      4   18
##
##                Accuracy : 0.7188
```

```
##                  95% CI : (0.5325, 0.8625)
##     No Information Rate : 0.7188
##     P-Value [Acc > NIR] : 0.5886
##
##                   Kappa : 0.3271
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.5556
##             Specificity : 0.7826
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.8182
##              Prevalence : 0.2812
##          Detection Rate : 0.1562
##    Detection Prevalence : 0.3125
##       Balanced Accuracy : 0.6691
##
##        'Positive' Class : FALSE
##
```

```
conf_matrix_melting$table
```

```
##           Reference
## Prediction FALSE TRUE
##      FALSE     5    5
##      TRUE      4   18
```

```
plt <- as.data.frame(conf_matrix_melting$table)
plt$Prediction <- factor(plt$Prediction, levels=rev(levels(plt$Prediction)))

ggplot(plt, aes(x = Prediction, y =  Reference, fill= Freq)) +
        geom_tile() + geom_text(aes(label=Freq)) +
        scale_fill_gradient(low="white", high="forestgreen") +
        labs(x = "Predictions",y = "Reference")
```


TRUE
FALSE
Reference
18
5
4
5
TRUE
FALSE
Predictions
Freq
16
12
8
4