2.Gene expression anaylsis: Creating random forest models on lab data

Fay Webster

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.

Install necessary libraries:

```
#install.packages("optimx", version = "2021-10.12") # this package is required for
#the parasite load package to work
library(tidyverse)
library(tidyr)
library(dplyr)
library(cowplot)
library(randomForest)
library(ggplot2)
library(caret)
library(VIM) # visualizing missing data
library(mice) # imputing missing data without predictors
library(ggpubr)
library(optimx)
library(rfUtilities) # Implements a permutation test cross-validation for
# Random Forests models
```

Laboratory data

Importing 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_expressi


# 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 / preparation

```
# 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 ~ ""
  ))


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

Imputation of missing data

Imputing missing data + cleaning

Here, I am using a function from the random forest package, rfImpute which utilizes random forests to impute missing data in the other variables.

The variables used for imputing mainly the immune gene expression are the current infection, the state of Eimeria infection, oocysts and the non-missing genes.

```
#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(all_of(Genes), max_WL,
                             primary_infection, challenge_infection,
                             mouse_strain, Parasite_primary, Parasite_challenge,
                             max_OOC, Eim_MC,  delta, Parasite_primary,
                             Parasite_challenge, OPG_0))

sapply(g.1, function(x) sum(is.na(x)))
```

```
##                IFNy          CXCR3_bio                IL.6               IL.10
##                  27                  0                  10                 10
##               IL.13               IL1RN              CASP1              CXCL9
##                  86                  0                   2                  0
##               IDO1               IRGM1                MPO              MUC2
##                  0                   0                  15                  0
##             MUC5AC              MYD88               NCR1               PRF1
##                  0                   0                  10                 23
##             RETNLB              SOCS1             TICAM1                TNF
```

```
##                   0                   0                   1                   2
##              max_WL   primary_infection challenge_infection        mouse_strain
##                   0                   0                   0                   0
##    Parasite_primary  Parasite_challenge             max_OOC              Eim_MC
##                   0                   0                   0                   0
##               delta               OPG_0
##                   6                  68
```

```
g.1$max_OOC[is.infinite(g.1$max_OOC)] <- NA


g.1  <- g.1 %>% mutate_if(is.character, as.factor)
g.1  <- g.1 %>% mutate_if(is.integer, as.numeric)


# 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 |    23.02    53.90 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |     23.8    55.72 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    23.61    55.27 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    24.08    56.38 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |     23.7    55.50 |
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  300 |    24.11    56.44 |
```

```
g_minus <- g %>%
  dplyr::select(-c(max_WL, primary_infection, challenge_infection,
                   mouse_strain, Parasite_primary, Parasite_challenge, max_OOC,
                   Eim_MC, delta, Parasite_primary, Parasite_challenge, OPG_0,
                   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

Random forest for predicting percentage of maximum weight loss

Dividing data into training and testing

```
Genes <- c("IFNy",  "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")

g.imputed_full <- g.imputed

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



train.data <- g.imputed[training.samples, ]
test.data <- g.imputed[-training.samples, ]
```

Building the model

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

print(weight_loss_predict)
```

```
##
## 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: 29.84162
##                     % Var explained: 28.63
```

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

Model - quality testing

Cross-validation

MSE: As a brief explanation, mean squared error (MSE) is the average of the summation of the squared difference between the actual output value and the predicted output value. Our goal is to reduce the MSE as much as possible.

Variance explained: %explained variance is a measure of how well out-of-bag predictions explain the target variance of the training set.

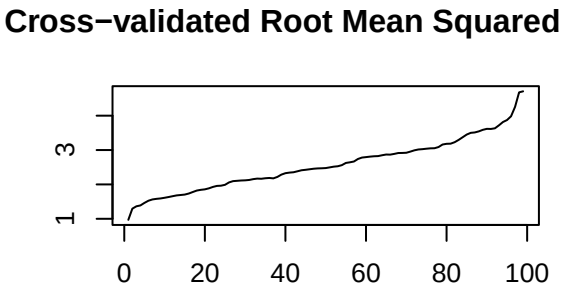
```
predict_WL_cv <- rf.crossValidation(x = weight_loss_predict, xdata = train.data,
                                    p = 0.10, n = 99, ntree = 501)
```

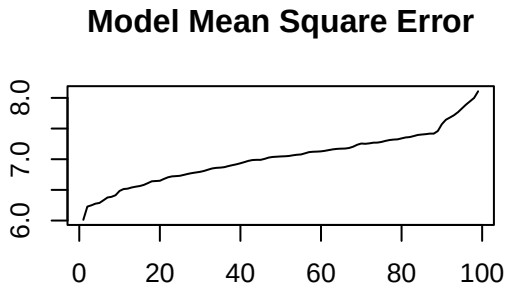
```
## running: regression cross-validation with 99 iterations
```

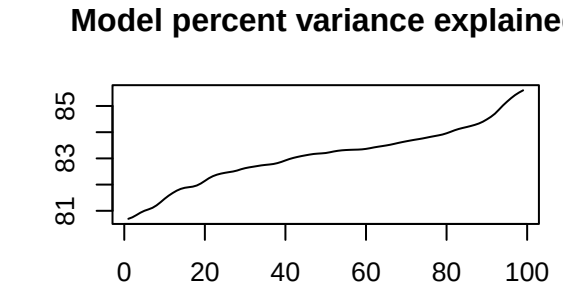
```
predict_WL_cv$fit.var.exp
```

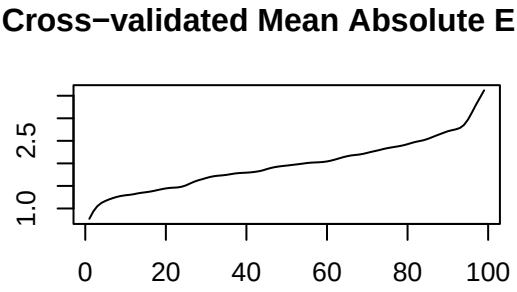
```
## [1] 28.63
```

```
par(mfrow=c(2,2))

plot(predict_WL_cv)

# Root Mean Squared Error (observed vs. predicted) from each Bootstrap
# iteration (cross-validation)
plot(predict_WL_cv, stat = "mse")

#Percent variance explained from specified fit model
plot(predict_WL_cv, stat = "var.exp")

#Mean Absolute Error from each Bootstrapped model
plot(predict_WL_cv, stat = "mae")
```

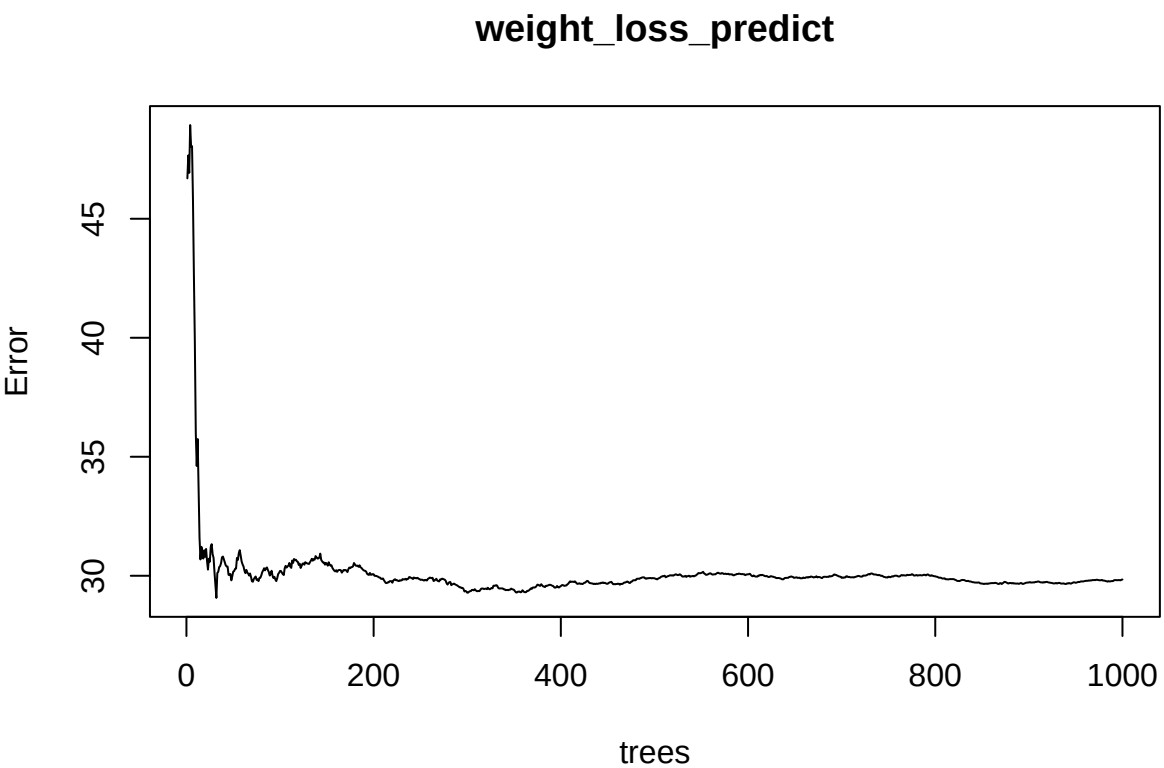









```
plot(weight_loss_predict)
```



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(weight_loss_predict$mse)
```

```
## [1] 32
```

```
## [1] 257

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

```
## [1] 5.391599
```

```
## [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!

Application of weight_loss_predict

Using the testing data

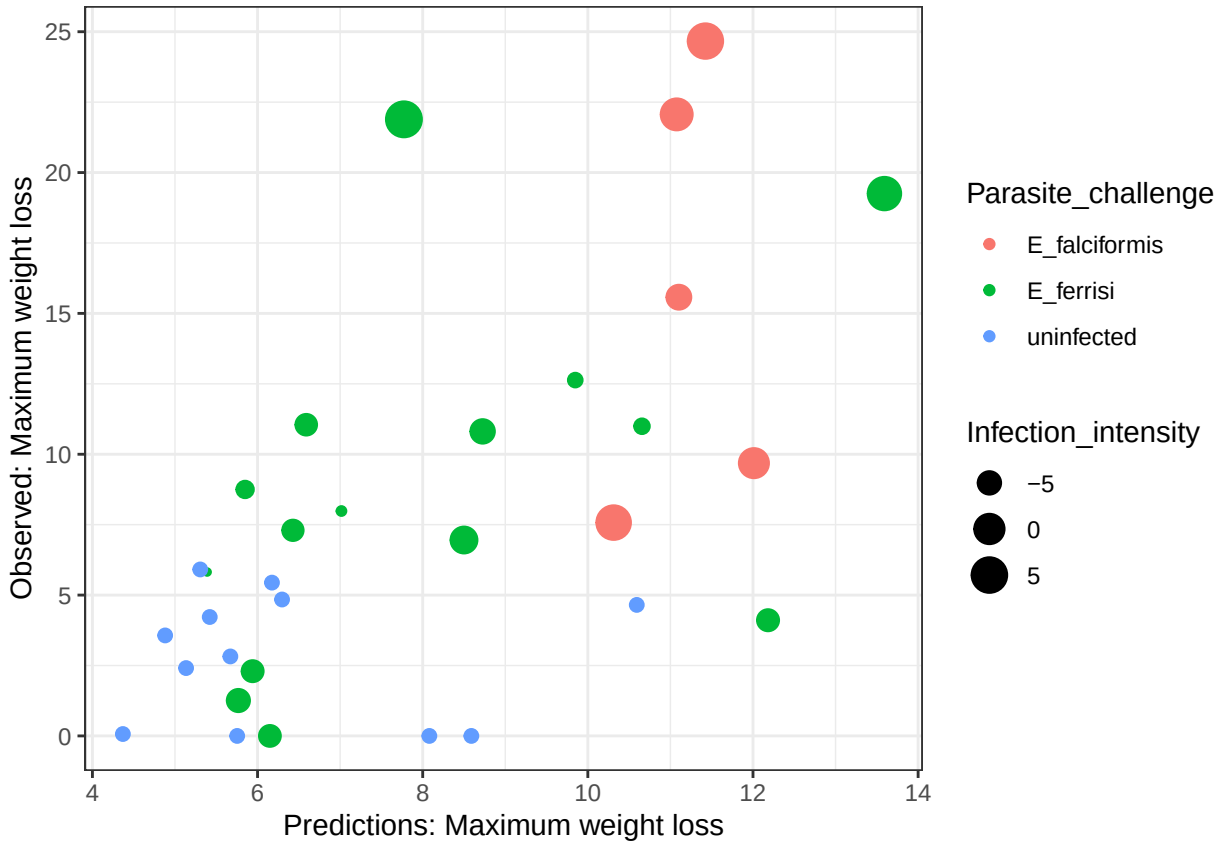
Let's now make some predictions using our test data.

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


# what is the correlation between predicted and actual data?
cor(result$max_WL, result$predictions,
    method = c("pearson", "kendall", "spearman"))
```

```
## [1] 0.6037497
```

Visualizing the predictions

```
# 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(Infection_intensity = case_when(
       Parasite_challenge == "uninfected" ~ -9,
       TRUE ~ delta
    ))

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



Using the same method to predict either Melting curve or infecting parasite

(2nd validation)

As a second part I am using the same method to predict either infection with Eimeria in general or the species of eimeria.

Predicting eimeria species

Predicing parasite: splliting into training and testing

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

#select the relevant columns:
g.imputed_parasite <- g.imputed_full %>%
  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, list = FALSE)
train.data_parasite <- g.imputed_parasite[training.samples, ]
test.data_parasite <- g.imputed_parasite[-training.samples, ]
```

Building the model_Parasite

```
#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, ntre
##                Type of random forest: classification
##                      Number of trees: 1500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 29.76%
## Confusion matrix:
##               E_falciformis E_ferrisi uninfected class.error
## E_falciformis             6         7          4  0.64705882
## E_ferrisi                 5        21          6  0.34375000
## uninfected                0         3         32  0.08571429
```

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

Quality checks

Cross-validation MSE: As a brief explanation, mean squared error (MSE) is the average of the summation of the squared difference between the actual output value and the predicted output value. Our goal is to reduce the MSE as much as possible.

Variance explained: %explained variance is a measure of how well out-of-bag predictions explain the target variance of the training set.

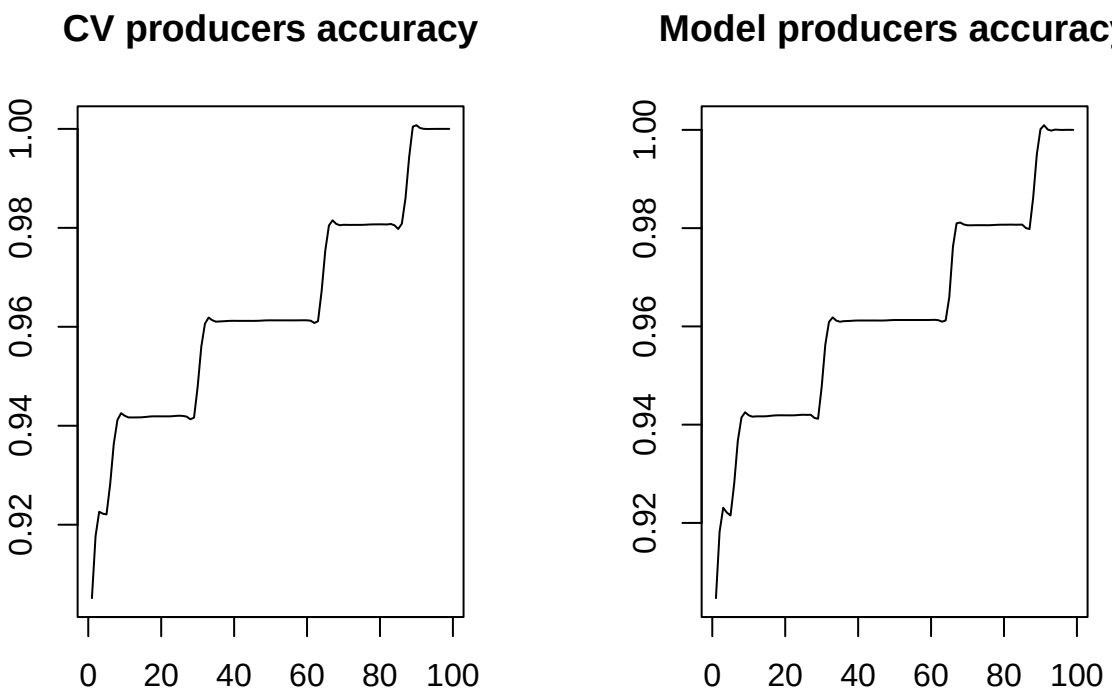
```
model_Parasite_cv <- rf.crossValidation(x = model_Parasite, xdata =
                                          train.data_parasite,
                                    p = 0.10, n = 99, ntree = 501)
```

```
## running: classification cross-validation with 99 iterations
```

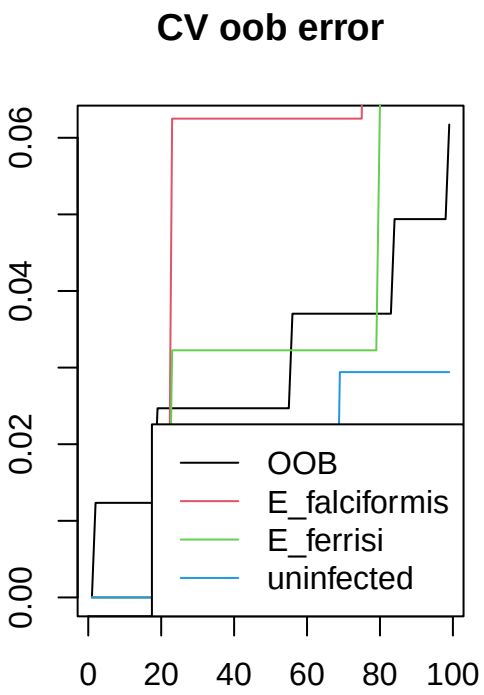
```
model_Parasite_cv$fit.var.exp
```

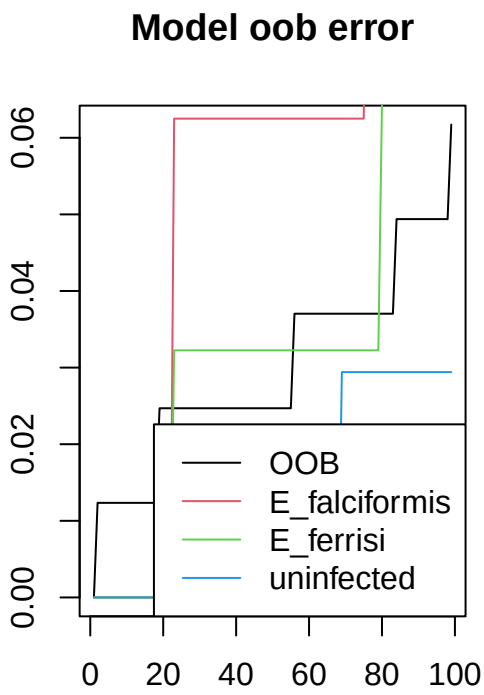
```
## NULL
```

```
# Plot cross validation versus model producers accuracy
par(mfrow=c(1,2))
  plot(model_Parasite_cv, type = "cv", main = "CV producers accuracy")
  plot(model_Parasite_cv, type = "model", main = "Model producers accuracy")
```

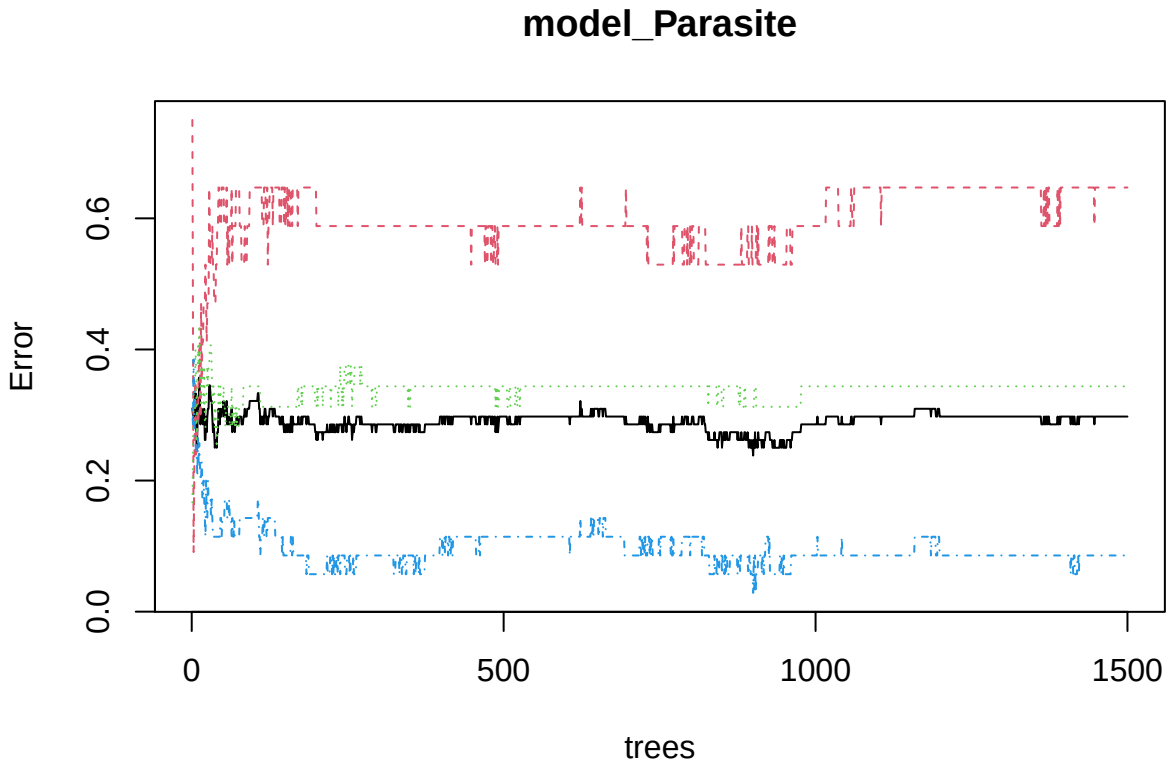



```
  # Plot cross validation versus model oob
par(mfrow=c(1,2))
  plot(model_Parasite_cv, type = "cv", stat = "oob", main = "CV oob error")
  plot(model_Parasite_cv, type = "model", stat = "oob",
       main = "Model oob error")
```






```
plot(model_Parasite)
```



Testing the model: 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)
```

Visualizing predictions_parasite

```
conf_matrix_parasite <-
  confusionMatrix(
    result_parasite$predictions_parasite,
    reference = result_parasite$Parasite_challenge)

print(conf_matrix_parasite)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             2         1          0
##   E_ferrisi                 2        11          2
##   uninfected                1         3         10
##
## Overall Statistics
##
##                Accuracy : 0.7188
##                  95% CI : (0.5325, 0.8625)
##     No Information Rate : 0.4688
##     P-Value [Acc > NIR] : 0.003708
##
##                   Kappa : 0.5325
##
##  Mcnemar's Test P-Value : 0.674599
##
## Statistics by Class:
##
##                      Class: E_falciformis Class: E_ferrisi Class: uninfected
## Sensitivity                       0.40000           0.7333            0.8333
## Specificity                       0.96296           0.7647            0.8000
## Pos Pred Value                    0.66667           0.7333            0.7143
## Neg Pred Value                    0.89655           0.7647            0.8889
## Prevalence                        0.15625           0.4688            0.3750
## Detection Rate                    0.06250           0.3438            0.3125
## Detection Prevalence              0.09375           0.4688            0.4375
## Balanced Accuracy                 0.68148           0.7490            0.8167
```

```
conf_matrix_parasite$table
```

```
##                Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             2         1          0
##   E_ferrisi                 2        11          2
##   uninfected                1         3         10
```

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

Predicting for Melting curve

Split the data again into training and testing

```
#select the relevant columns:
g.imputed_mc <- g.imputed_full %>%
  dplyr::select(c(Eim_MC, 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_mc <- g.imputed_mc$Eim_MC %>%
  createDataPartition(p = .7, list = FALSE)
train.data_mc <- g.imputed_mc[training.samples, ]
test.data_mc <- g.imputed_mc[-training.samples, ]
```

Building the model

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

print(model_mc)
```

```
##
```

```
## Call:
##  randomForest(formula = Eim_MC ~ ., data = train.data_mc, 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: 26.19%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE    25   12   0.3243243
## TRUE     10   37   0.2127660
```

Cross-validation MSE: As a brief explanation, mean squared error (MSE) is the average of the summation of the squared difference between the actual output value and the predicted output value. Our goal is to reduce the MSE as much as possible.

Variance explained: %explained variance is a measure of how well out-of-bag predictions explain the target variance of the training set.

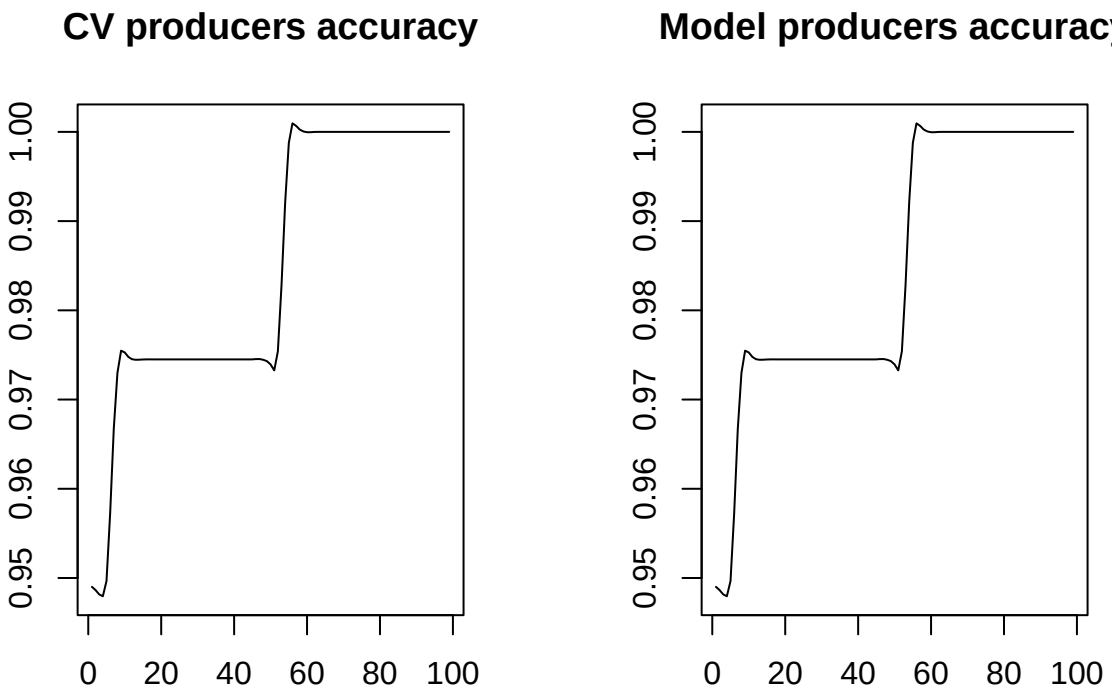
```
model_mc_cv <- rf.crossValidation(x = model_mc, xdata =  train.data_mc,
                                  p = 0.10, n = 99, ntree = 501)
```

```
## running: classification cross-validation with 99 iterations
```

```
model_mc_cv$fit.var.exp
```

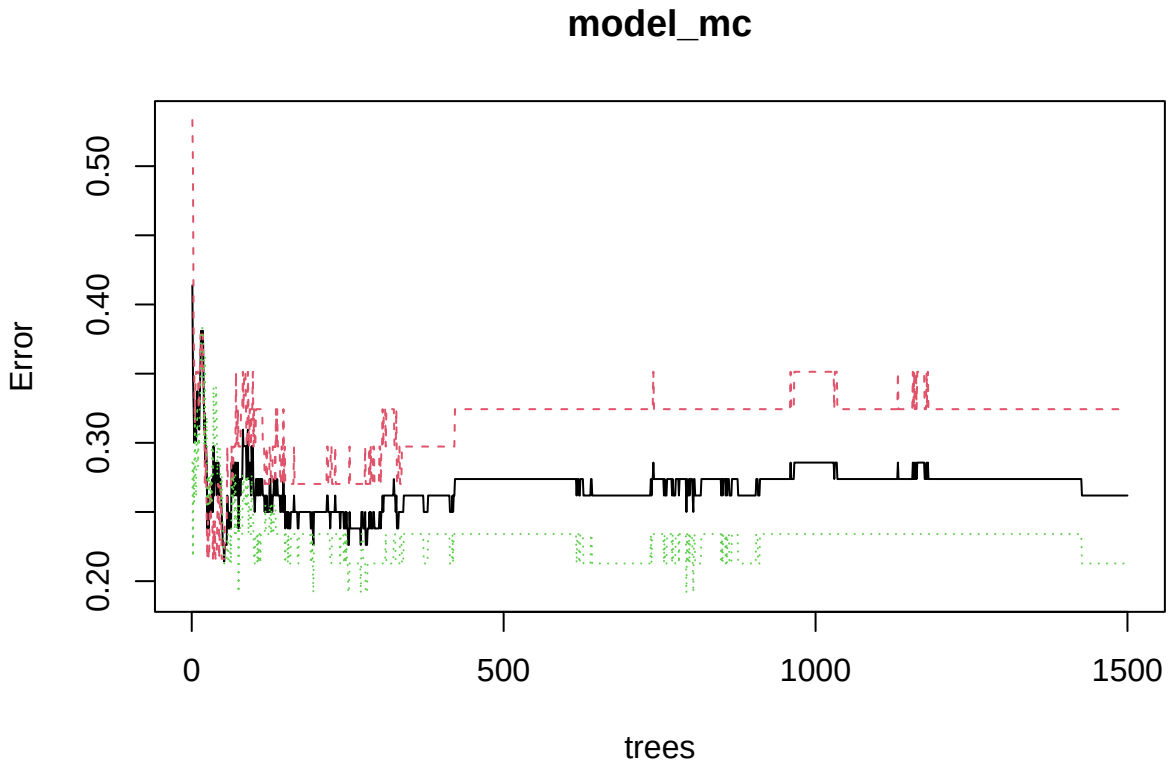
```
## NULL
```

```
# Plot cross validation versus model producers accuracy
par(mfrow=c(1,2))
  plot(model_mc_cv, type = "cv", main = "CV producers accuracy")
  plot(model_mc_cv, type = "model", main = "Model producers accuracy")
```




```
  # Plot cross validation versus model oob
par(mfrow=c(1,2))
  plot(model_mc_cv, type = "cv", stat = "oob", main = "CV oob error")
  plot(model_mc_cv, type = "model", stat = "oob", main = "Model oob error")
```


```
plot(model_mc)
```



Test the model

Making predictions

```
#The predict() function in R is used to predict the values based on the input data.
predictions_mc <- predict(model_mc, test.data_mc)
# assign test.data to a new object, so that we can make changes
result_mc <- test.data_mc
```

```r
#add the new variable of predictions to the result object
result_mc <- cbind(result_mc, predictions_mc)
#add the results to a data frame containing test data and the prediction
result_mc <- cbind(g[row.names(result_mc), ], predictions_mc)
```

Visualizations

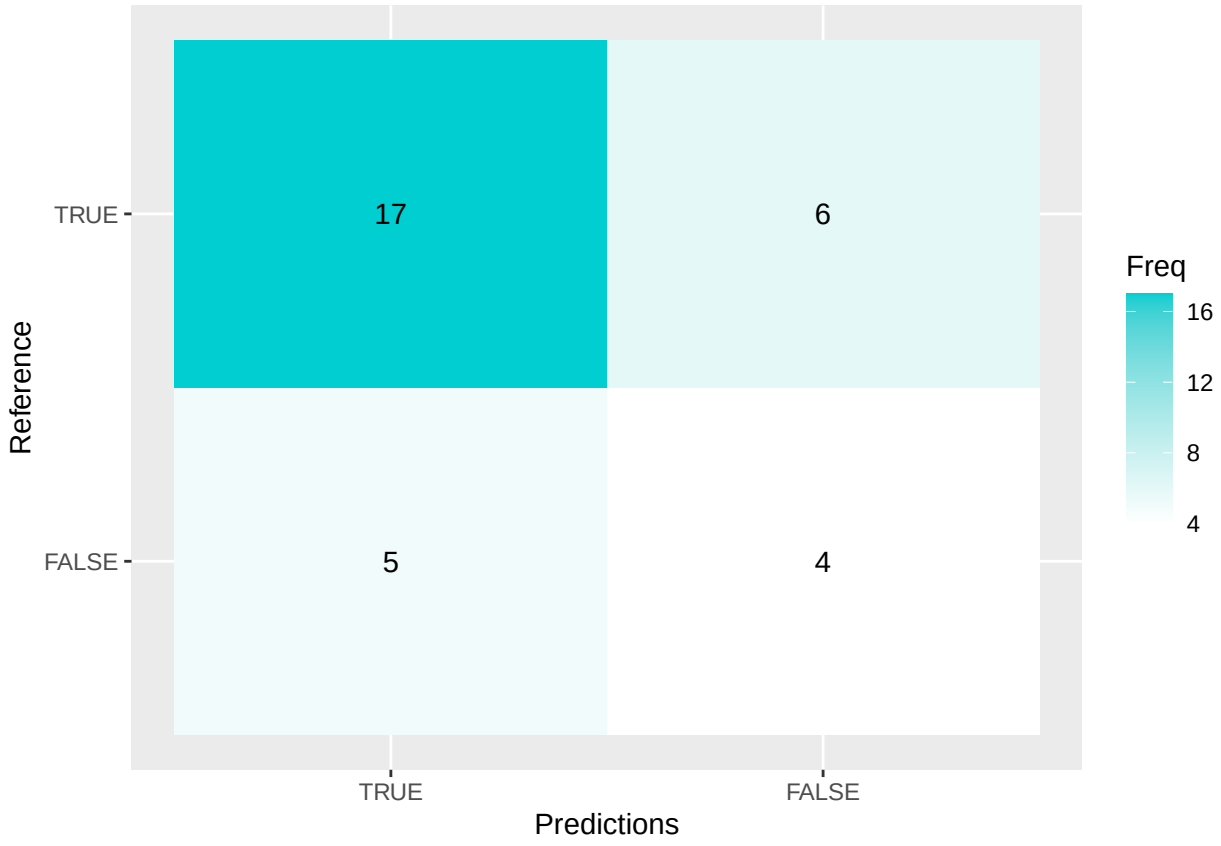
```r
conf_matrix_mc <-
  confusionMatrix(result_mc$predictions_mc, reference = result_mc$Eim_M)

print(conf_matrix_mc)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE     4    6
##      TRUE      5   17
##
##                Accuracy : 0.6562
##                  95% CI : (0.4681, 0.8143)
##     No Information Rate : 0.7188
##     P-Value [Acc > NIR] : 0.8377
##
##                   Kappa : 0.1776
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.4444
##             Specificity : 0.7391
##          Pos Pred Value : 0.4000
##          Neg Pred Value : 0.7727
##              Prevalence : 0.2812
##          Detection Rate : 0.1250
##    Detection Prevalence : 0.3125
##       Balanced Accuracy : 0.5918
##
##        'Positive' Class : FALSE
##
```

```r
conf_matrix_mc$table
```

```
##           Reference
## Prediction FALSE TRUE
##      FALSE     4    6
##      TRUE      5   17
```

```r
plt <- as.data.frame(conf_matrix_mc$table)
plt$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="darkturquoise") +
        labs(x = "Predictions",y = "Reference")
```


17
6
5
4
TRUE
FALSE
Reference
TRUE
FALSE
Predictions
Freq
16
12
8
4