13.Random_forest_lab_facs

Fay

2022-11-08

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.

Load 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(ggpubr)
library(rfUtilities) # Implements a permutation test cross-validation for
# Random Forests models
library(FactoMineR)
```

Laboratory data

Importing the data

We start with the data from experimental lab infections.

```
#import data
hm <- read.csv("output_data/imputed_mice.csv")
```

vectors for selecting

```
Facs_lab <- c("CD4", "Treg", "Div_Treg", "Treg17", "Th1",
                    "Div_Th1", "Th17", "Div_Th17", "CD8", "Act_CD8",
```

```
                    "Div_Act_CD8", "IFNy_CD4", "IFNy_CD8") #"Treg_prop", removed due to many missing va
                    #"IL17A_CD4"
Facs_wild <- c( "Treg", "CD4", "Treg17", "Th1", "Th17", "CD8",
                     "Act_CD8", "IFNy_CD4", "IFNy_CD8") #"IL17A_CD4",
```

Data cleaning / preparation

```
# we need to change the  in challenge infections to a factor
hm$Parasite_challenge <- as.factor(hm$Parasite_challenge)
hm$MC.Eimeria <- as.factor(hm$MC.Eimeria)
# Here I create a new column, where we get the actual infection status
# According to the melting curve for eimeria
hm <- hm %>%
  dplyr::mutate(current_infection = case_when(
    Parasite_challenge == "E_ferrisi" & MC.Eimeria == "TRUE" ~ "E_ferrisi",
    Parasite_challenge == "E_ferrisi" & MC.Eimeria == "FALSE" ~ "uninfected",
    Parasite_challenge == "E_falciformis" & MC.Eimeria == "TRUE" ~ "E_falciformis",
    Parasite_challenge == "E_falciformis" & MC.Eimeria == "FALSE" ~ "uninfected",
    Parasite_challenge == "uninfected" & MC.Eimeria == "TRUE" ~ "infected_eimeria",
    Parasite_challenge == "uninfected" & MC.Eimeria == "FALSE" ~ "uninfected",
    TRUE ~ ""
  ))
```

Splitting data into training and testing sets

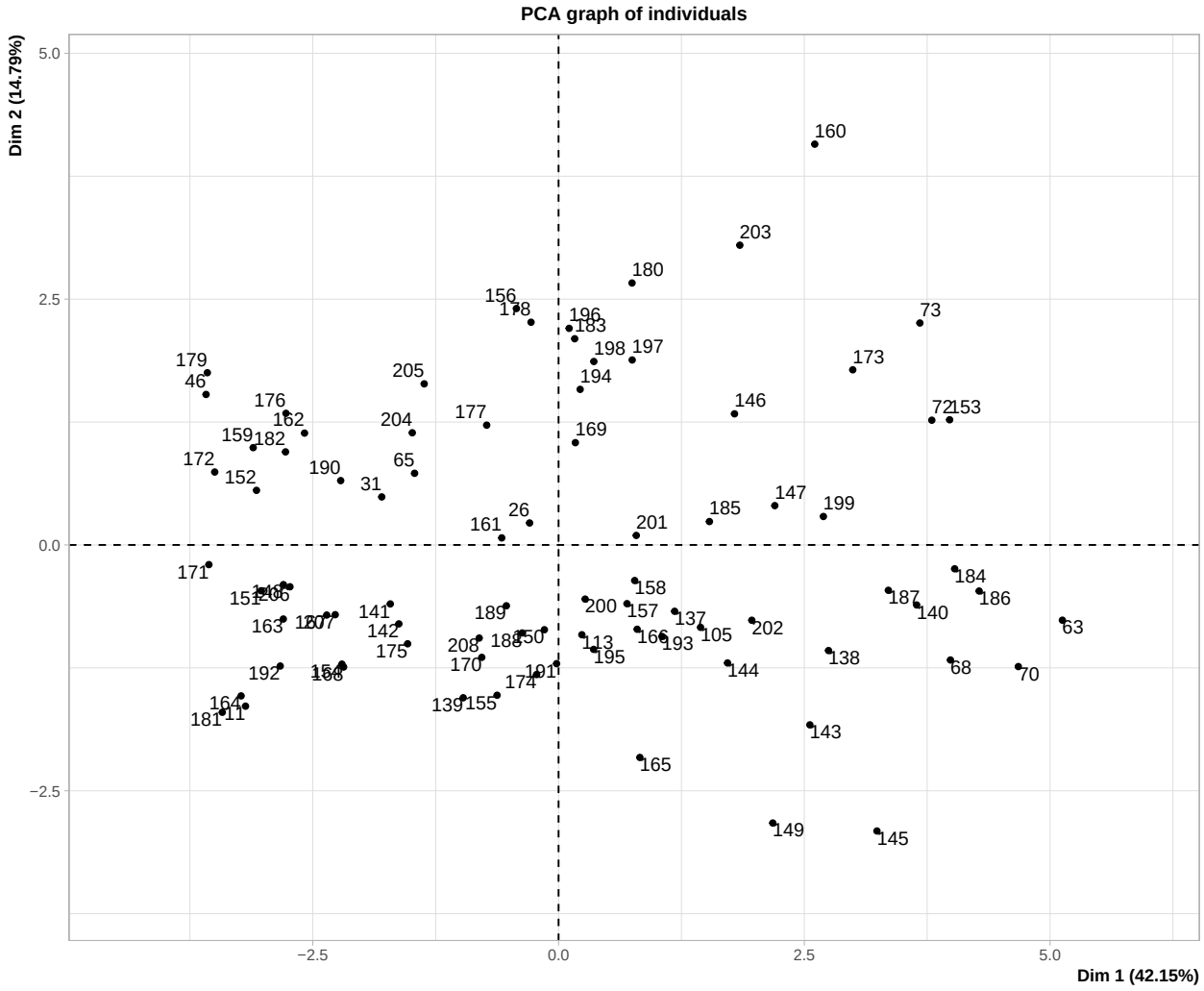
Splitting between training and testing: - Assess model performance on unseen data - Avoid over-fitting

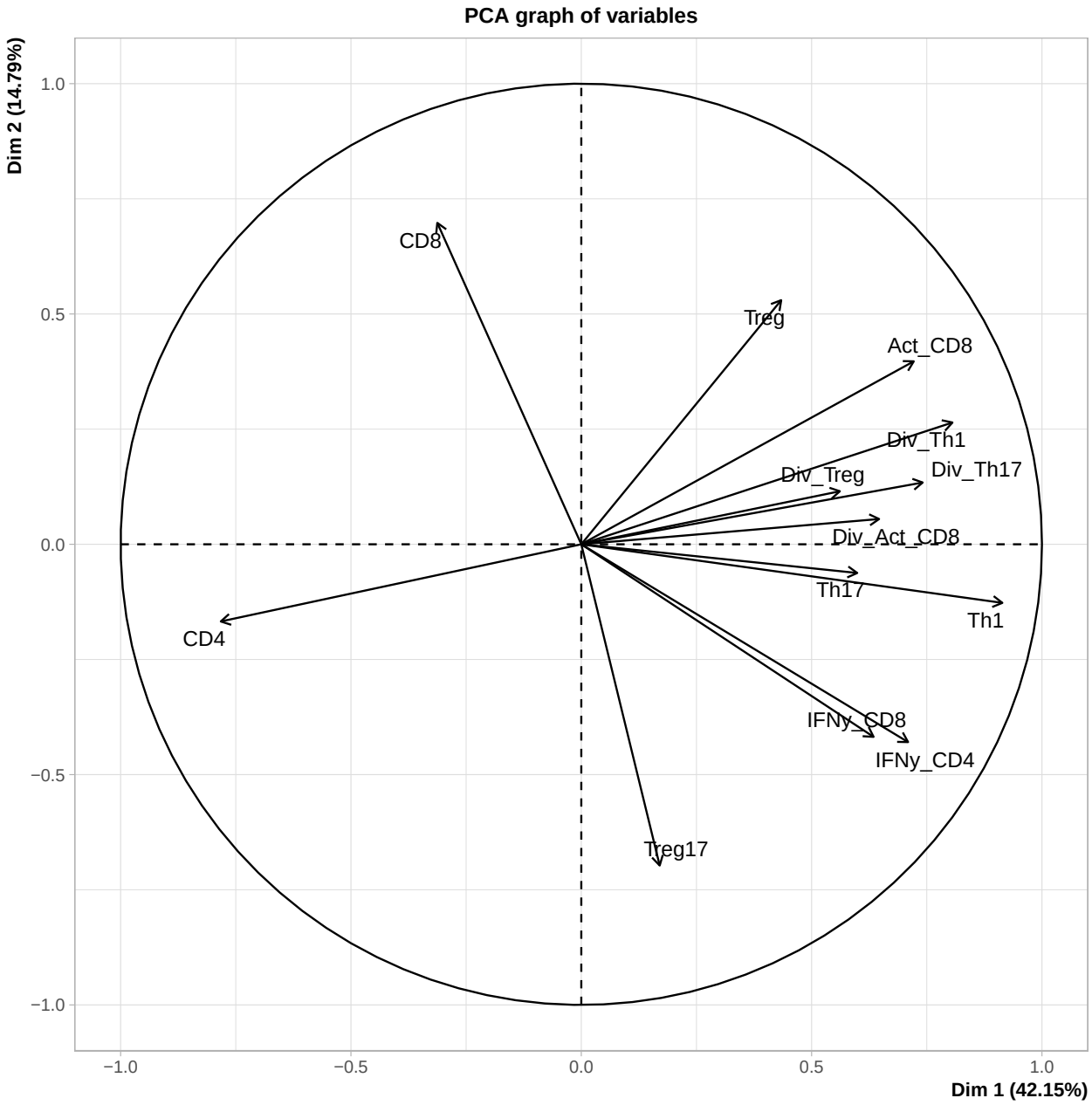
Random forest for predicting percentage of maximum weight loss

Dividing data into training and testing

```
#select the genes and lab muce
lab <- hm %>%
  dplyr::filter(origin == "Lab", Position == "mLN") #selecting for mln to avoid

# duplicates

lab <- unique(lab)

facs_mouse <- lab %>%
  dplyr::select(c(Mouse_ID, all_of(Facs_lab)))


facs <- facs_mouse[, -1]

#remove rows with only nas
facs <- facs[,colSums(is.na(facs))<nrow(facs)]

#remove colums with only nas
facs <- facs[rowSums(is.na(facs)) != ncol(facs), ]
```

```
#select same rows in the first table
facs_mouse <- facs_mouse[row.names(facs), ]


# we can now run a normal pca on the complete data set
res.pca <- PCA(facs)
```






```
# select the same rows from the lab data
lab <- lab[row.names(facs),]
M_W <- lab %>%
  dplyr::select(c(Mouse_ID, WL_max))
facs <- cbind(M_W, facs)
facs <- unique(facs) %>%
  dplyr::select(-Mouse_ID)
# 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 <- facs$WL_max%>%
  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 <- facs[training.samples, ]
test.data <- facs[-training.samples, ]
```

Building the model

```
#train the model
weight_loss_predict <- randomForest(WL_max ~., data = train.data,
                                    proximity = TRUE, ntree = 1000)
# ntree = number of trees
# save the model
save(weight_loss_predict, file =  "r_scripts/models/predict_weight_loss.RData")
print(weight_loss_predict)
```

```
##
## Call:
##  randomForest(formula = WL_max ~ ., data = train.data, proximity = TRUE,      ntree = 1000)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 33.78522
##                     % Var explained: 9.61
```

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.

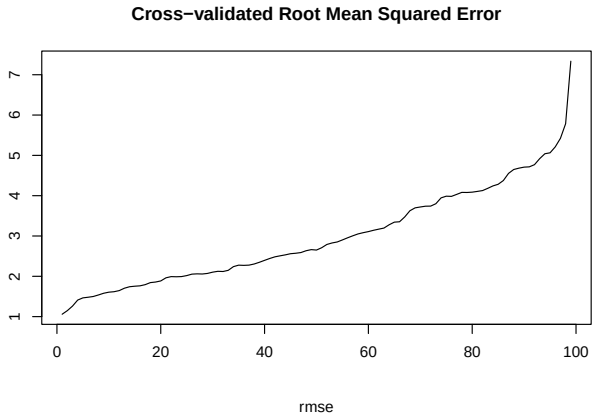
```
predict_WL_cv <- rf.crossValidation(x = weight_loss_predict, xdata = train.data,
                                    p = 0.10, n = 99, ntree = 501)
```

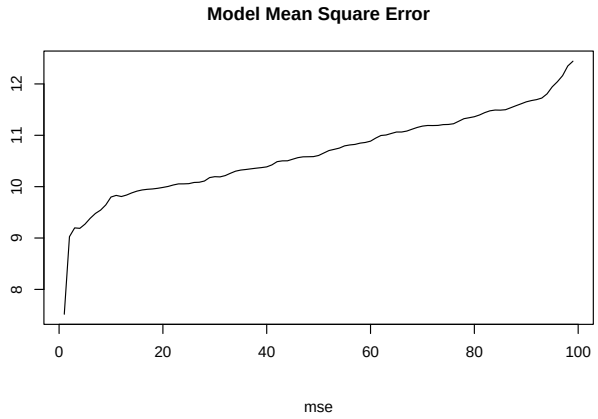
```
## running: regression cross-validation with 99 iterations
```

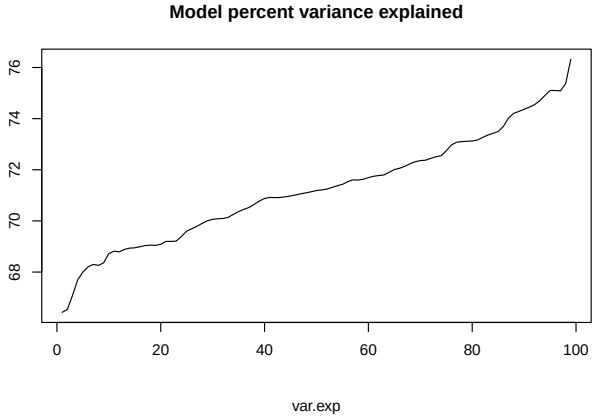
```
predict_WL_cv$fit.var.exp
```

```
## [1] 9.61
```

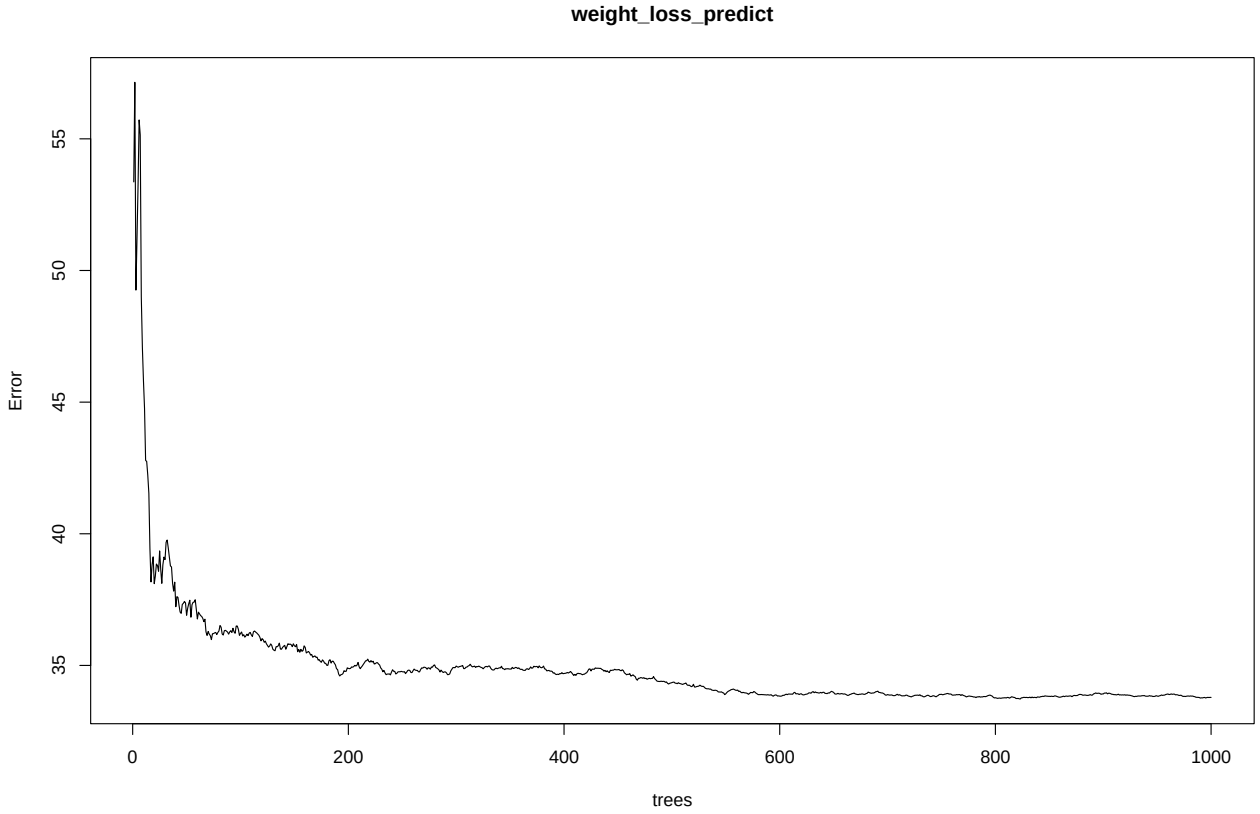
```
par(mfrow=c(2,2))
plot(predict_WL_cv)
# Root Mean Squared Error (observed vs. predicted) from each Bootstrap
# iteration (cross-validation)
plot(predict_WL_cv, stat = "mse")
#Percent variance explained from specified fit model
plot(predict_WL_cv, stat = "var.exp")
#Mean Absolute Error from each Bootstrapped 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

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

```
## [1] 823
```

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

```
## [1] 5.807175
```

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

```
## [1] 0.5288994
```

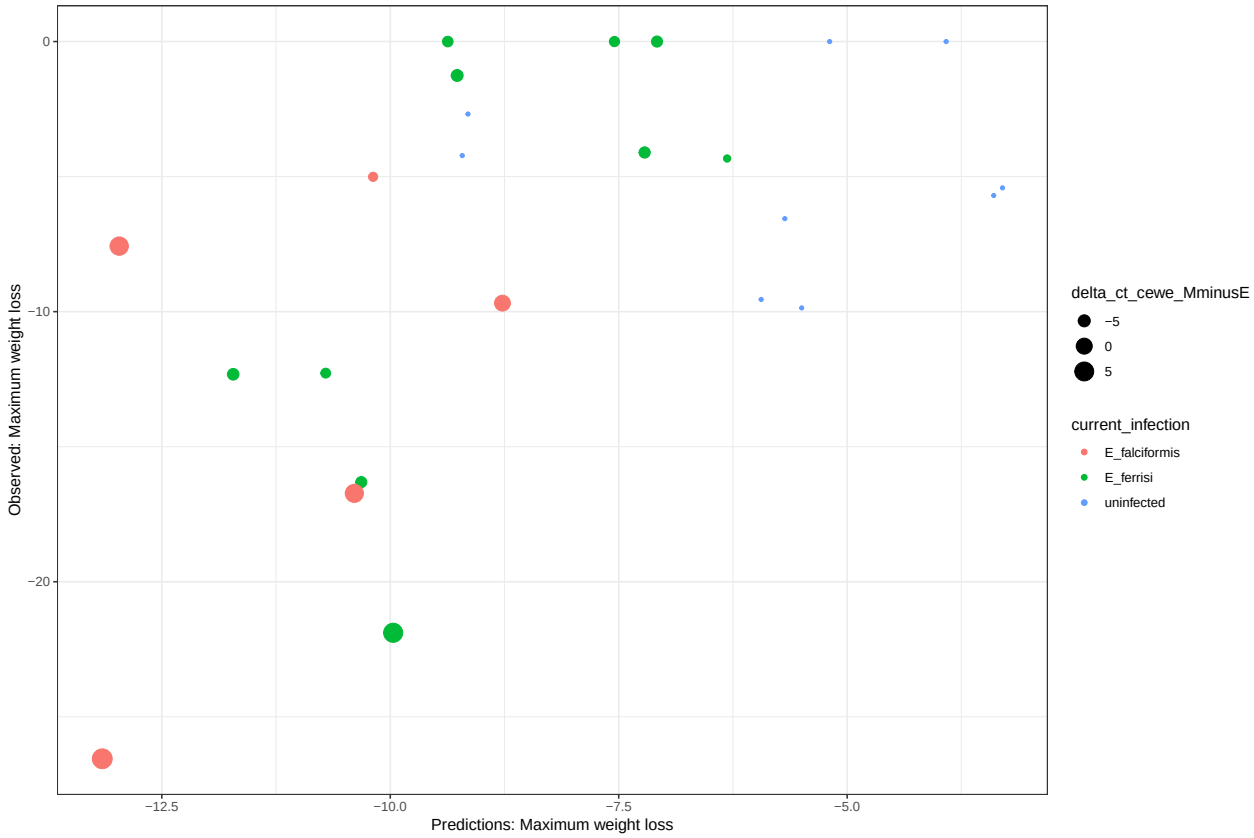
```
test_lab <- lab %>%
  left_join(result, by = c("WL_max", "Treg", "CD4", "Treg17", "Th1", "Th17", "CD8",
                      "Act_CD8", "IFNy_CD4", "IFNy_CD8"))
test_lab <- test_lab %>%
  drop_na(predictions)
```

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
test_lab <- test_lab %>%
    dplyr::mutate(Infection_intensity = case_when(
       Parasite_challenge == "uninfected" ~ -9,
       TRUE ~ delta_ct_cewe_MminusE
    ))
cor(test_lab$predictions, test_lab$WL_max)
```

```
## [1] 0.5288994
```

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




```
cor.test(result$predictions, result$WL_max, method = "spearman")
```

```
## Warning in cor.test.default(result$predictions, result$WL_max, method =
## "spearman"): Cannot compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  result$predictions and result$WL_max
## S = 1171.1, p-value = 0.01488
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.4908344
```

Repeating the process using the whole data set as a training data set