9.Random Forest on Lab data -genes

Fay

2022-11-04

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)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.1
```

```
## Warning: package 'tibble' was built under R version 4.2.1
```

```
## Warning: package 'tidyr' was built under R version 4.2.1
```

```
## Warning: package 'readr' was built under R version 4.2.1
```

```
## Warning: package 'purrr' was built under R version 4.2.1
```

```
## Warning: package 'dplyr' was built under R version 4.2.1
```

```
## Warning: package 'stringr' was built under R version 4.2.1
```

```
## Warning: package 'forcats' was built under R version 4.2.1
```

```
library(tidyr)
library(dplyr)
library(cowplot)
library(randomForest)
library(ggplot2)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.1
```

```
library(ggpubr)
library(rfUtilities) # Implements a permutation test cross-validation for
```

```
## Warning: package 'rfUtilities' was built under R version 4.2.2
```

```
# Random Forests models
```

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

```
Gene_lab   <- c("IFNy", "CXCR3", "IL.6", "IL.13", "IL.10",
                "IL1RN","CASP1", "CXCL9", "IDO1", "IRGM1", "MPO",
                "MUC2", "MUC5AC", "MYD88", "NCR1", "PRF1", "RETNLB", "SOCS1",
                "TICAM1", "TNF") #"IL.12", "IRG6")



Genes_wild   <- c("IFNy", "CXCR3", "IL.6", "IL.13", "IL.10",
                  "IL1RN","CASP1", "CXCL9", "IDO1", "IRGM1", "MPO",
                  "MUC2", "MUC5AC", "MYD88", "NCR1", "PRF1", "RETNLB", "SOCS1",
                  "TICAM1", "TNF") #, "IL.12", "IRG6")



Facs_lab <- c("Position", "CD4", "Treg", "Div_Treg", "Treg17", "Th1",
                    "Div_Th1", "Th17", "Div_Th17", "CD8", "Act_CD8",
                    "Div_Act_CD8", "IFNy_CD4", "IFNy_CD8","Treg_prop",
                    "IL17A_CD4")

Facs_wild <- c( "Treg", "CD4", "Treg17", "Th1", "Th17", "CD8",
                     "Act_CD8", "IFNy_CD4", "IL17A_CD4", "IFNy_CD8")
```

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

Cross validation in R

k - fold cross validation: the data set is divided into k subsets. Each time, one of the k subsets is used as the test set and

Random forest for predicting percentage of maximum weight loss

Dividing data into training and testing

```
# prepare the lab data
lab <- hm %>%
  dplyr::filter(origin == "Lab")


#select the imputed gene columns
gene <-  lab %>%
  dplyr::select(c(Mouse_ID, all_of(Gene_lab)))

gene <- unique(gene)

genes <- gene %>%
  dplyr::select(-Mouse_ID)

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

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

# select the same rows from the gene data
gene <- gene[row.names(genes),]

# select the same rows from the lab data
lab <- lab[row.names(genes),]


gene <- lab %>%
  dplyr::select(c(Mouse_ID, WL_max)) %>%
  right_join(gene, by = "Mouse_ID")

gene <- unique(gene) %>%
  dplyr::select(-Mouse_ID)
```

Cross validation

mtry: Number of variable is randomly collected to be sampled at each split time ntree: Number of trees you are going to grow in this random forest

https://rpubs.com/jvaldeleon/forest_repeat_cv

cross validation

The general procedure is like this.

We shuffle the data by random.

We split it into k-groups. Note that k is an arbitrary parameter. There's no specific criteria to choose the value for k. Typically, it's 5 or 10.

For each unique group:

1. Take the group as a hold out or test data set
2. Take the remaining groups as a training data set
3. Fit a model on the training set and evaluate it on the test set
4. Retain the evaluation score and discard the model

```r
repeat_cv <- trainControl(method = "repeatedcv", #repeated cross validation
                          number = 5, # 5 fold cross validation
                          repeats = 3)
```

Splitting into training and testing

```r
# split data into training and test
set.seed(333) # this will help us reproduce this random assignment

# in this way we can pick the random numbers
```

```
training.samples <- createDataPartition(y = gene$WL_max, 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 <- gene[training.samples, ]
test.data <- gene[-training.samples, ]
```

Building the model

```
set.seed(333)


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

```
##
## 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: 6
##
##           Mean of squared residuals: 44.28613
##                     % Var explained: 27.7
```

Plotting the WL_predict_gene 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.

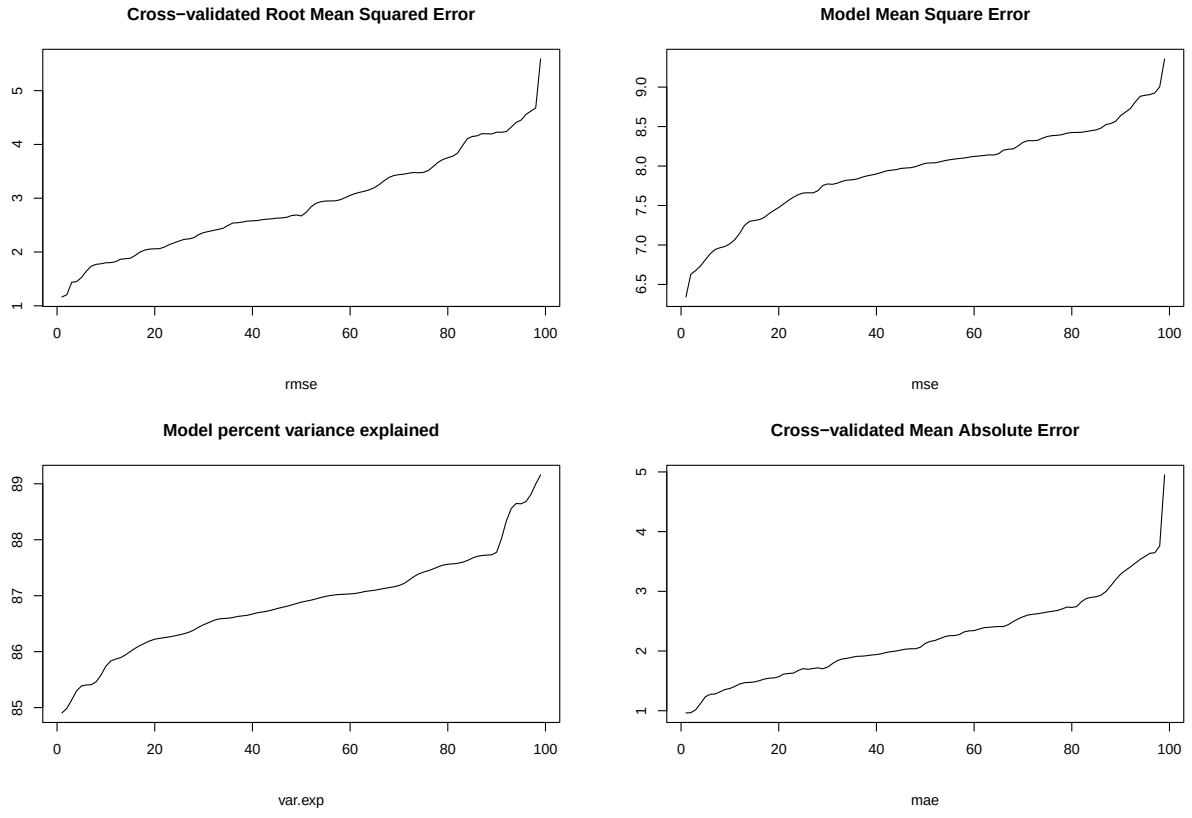
```
predict_WL_cv <- rf.crossValidation(x = WL_predict_gene, xdata = train.data,
                                    p = 0.10, n = 99, ntree = 501)
```

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

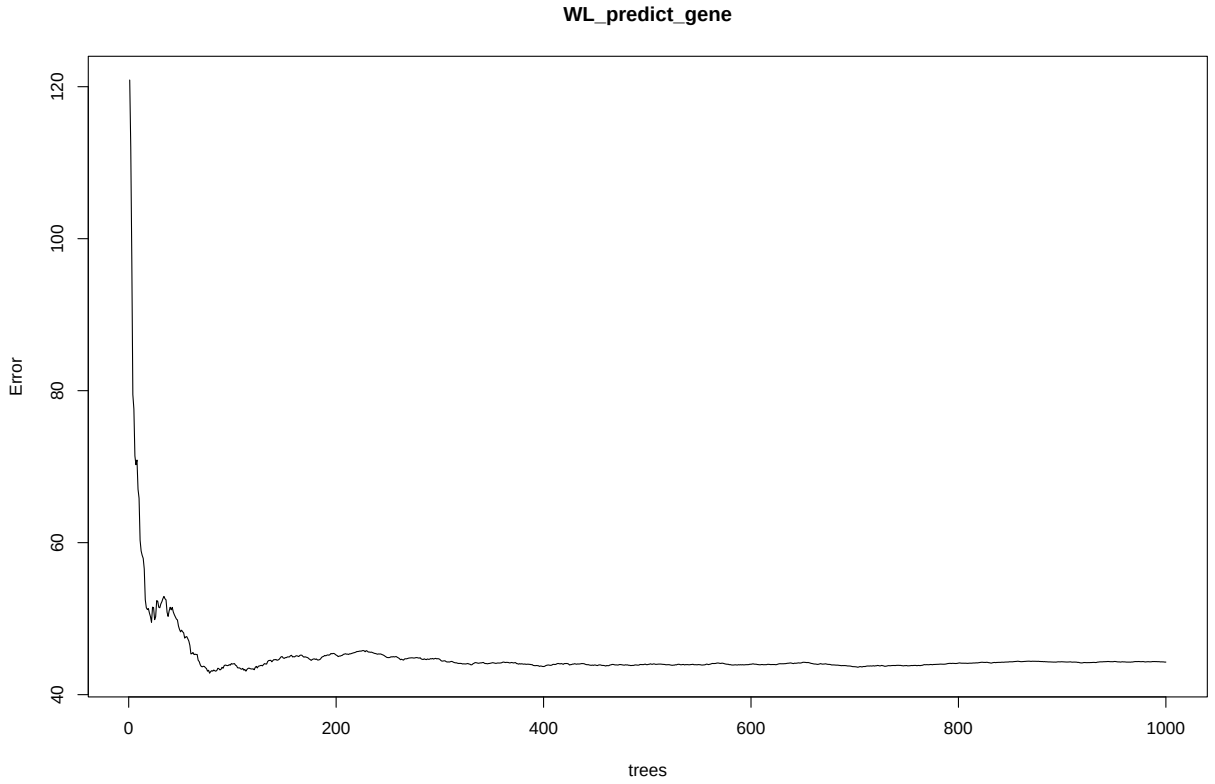
```
predict_WL_cv$fit.var.exp
```

```
## [1] 27.7
```

```
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(WL_predict_gene)
```



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(WL_predict_gene$mse)
```

```
## [1] 78
```

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

```
## [1] 6.543936
```

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 WL_predict_gene

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(WL_predict_gene, 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] 0.8448673
```

```
test_lab <- lab %>%
  left_join(result, by = c("WL_max", "IFNy", "CXCR3", "IL.6", "IL.13", "IL.10",
                "IL1RN","CASP1", "CXCL9", "IDO1", "IRGM1", "MPO",
                "MUC2", "MUC5AC", "MYD88", "NCR1", "PRF1", "RETNLB", "SOCS1",
                "TICAM1", "TNF"))

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))

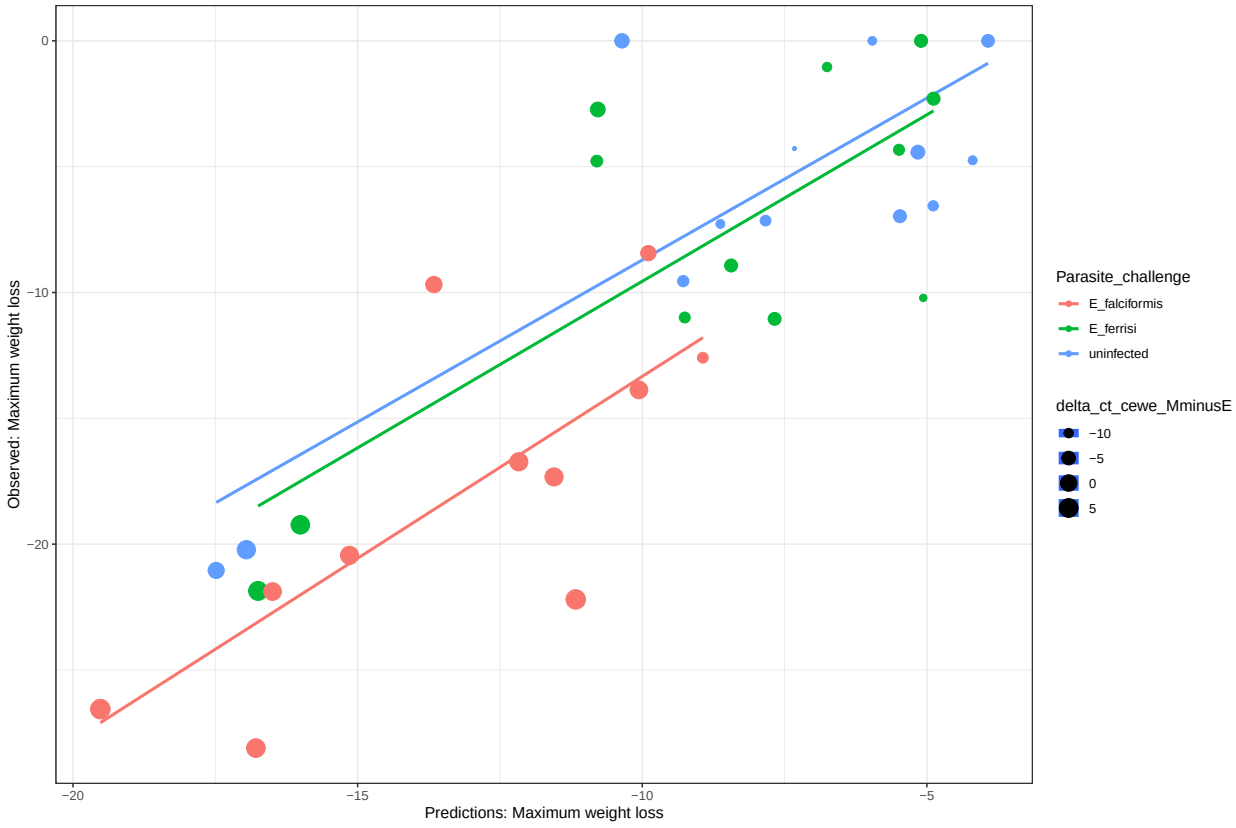
```
# what is the correlation between predicted and actual data?
cor(result$WL_max, result$predictions,
    method = c("pearson", "kendall", "spearman"))
```

```
## [1] 0.8448673
```

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

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```




```
cor(test_lab$predictions, test_lab$WL_max, method = "spearman")
```

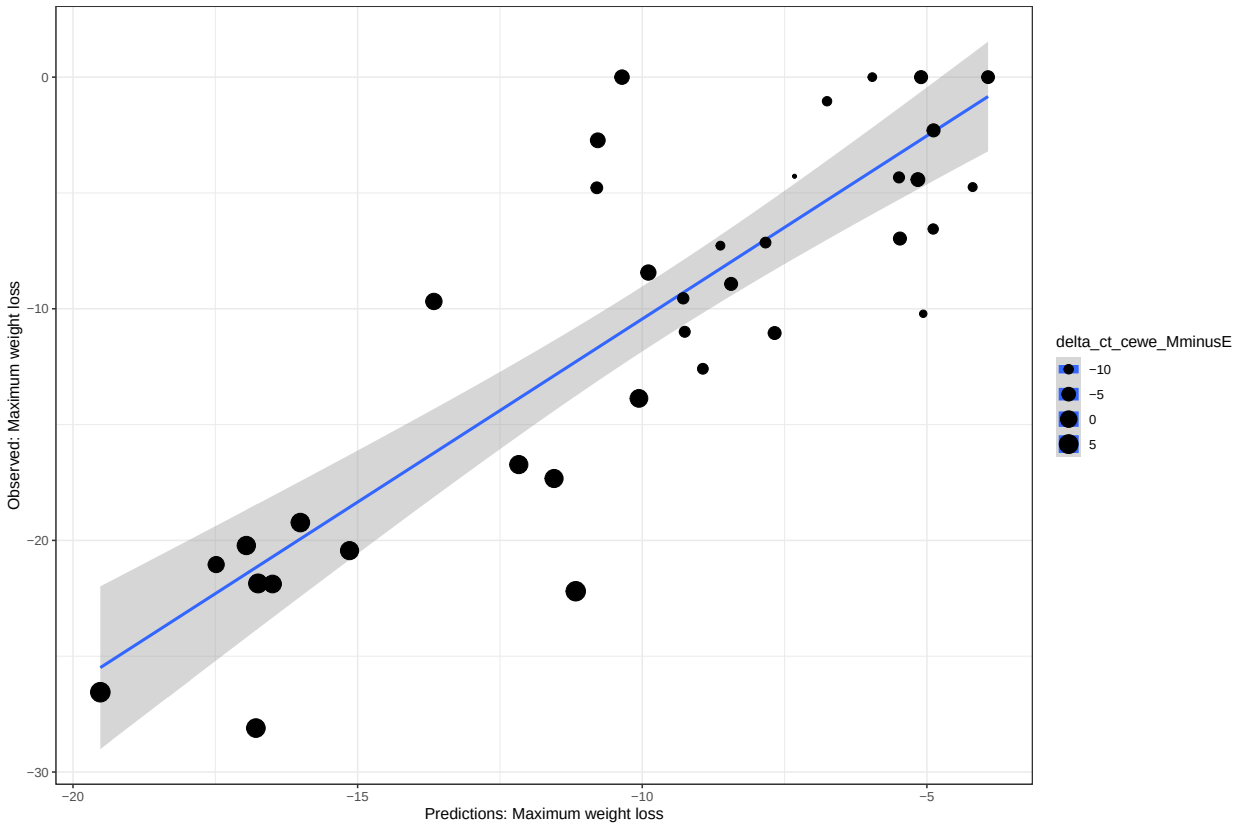
```
## [1] 0.750352
```

```
test_lab   %>%
  ggplot(aes(x = predictions, y = WL_max,
```

```
                size = delta_ct_cewe_MminusE)) +
  geom_smooth(method = lm, se = TRUE) +
  labs(x = "Predictions: Maximum weight loss",
       y = "Observed: Maximum weight loss") +
  geom_point(aes(x = predictions, y = WL_max,  size = delta_ct_cewe_MminusE)) +
  labs(x = "Predictions: Maximum weight loss",
       y = "Observed: Maximum weight loss") +
    theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 4 rows containing missing values (geom_point).
```



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

Building the model

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

```r
# save the model
# toa = trained on all
saveRDS(WL_predict_gene, "r_scripts/models/predict_WL.rds")

print(WL_predict_gene)
```

```
##
## Call:
##  randomForest(formula = WL_max ~ ., data = gene, 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: 33.37116
##                     % Var explained: 46.15
```

other models

Predicing parasite: splliting into training and testing

```r
lab$Parasite_challenge <- as.factor(lab$Parasite_challenge)

gene_curr <- lab %>%
  dplyr::select(c(Mouse_ID, all_of(Gene_lab), Parasite_challenge))



gene <- gene_curr %>%
  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 <- gene$Parasite_challenge%>%
  createDataPartition(p = .7, list = FALSE)
train.data_parasite <- gene[training.samples, ]
test.data_parasite <- gene[-training.samples, ]
```

Building the model_Parasite

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

# save the model
save(model_Parasite, file =  "r_scripts/models/predict_infecting_parasite.RData")

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: 31.25%
## Confusion matrix:
##               E_falciformis E_ferrisi uninfected class.error
## E_falciformis            10         7          4   0.5238095
## E_ferrisi                 5        27          5   0.2702703
## uninfected                3         6         29   0.2368421
```

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
```

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(lab[row.names(result_parasite), ], predictions_parasite)
```

Visualizing predictions_parasite

```r
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         2          1
##   E_ferrisi                 4         9          4
##   uninfected                3         4         11
##
## Overall Statistics
##
##                Accuracy : 0.55
##                  95% CI : (0.3849, 0.7074)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : 0.03917
##
##                   Kappa : 0.2885
##
##  Mcnemar's Test P-Value : 0.64437
##
## Statistics by Class:
##
##                      Class: E_falciformis Class: E_ferrisi Class: uninfected
## Sensitivity                        0.2222           0.6000            0.6875
## Specificity                        0.9032           0.6800            0.7083
## Pos Pred Value                     0.4000           0.5294            0.6111
## Neg Pred Value                     0.8000           0.7391            0.7727
## Prevalence                         0.2250           0.3750            0.4000
## Detection Rate                     0.0500           0.2250            0.2750
## Detection Prevalence               0.1250           0.4250            0.4500
## Balanced Accuracy                  0.5627           0.6400            0.6979
```

```r
conf_matrix_parasite$table
```

```
##                Reference
## Prediction      E_falciformis E_ferrisi uninfected
##   E_falciformis             2         2          1
##   E_ferrisi                 4         9          4
##   uninfected                3         4         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="darkturquoise") +
        labs(x = "Predictions",y = "Reference")
```