Random Forests

Amber Lee

7/6/2021

Summary

This notebook presents random forest interpolation of total phosphorous, total nitrogen, and velocity from the LTRM UMR dataset. Each of these three response variables will be predicted by 14 other continuous and categorical variabes.

Set up

Load libraries

```
library(tidyverse)
library(lubridate)    # date types
library(kableExtra)   # presenting tables
library(rsample)      # data splitting
library(randomForest) # basic implementation
library(ranger)       # a faster implementation of randomForest
```

Read data

```
water20 <- read.csv("../../LTRM data/water_data_qfneg.csv", header = TRUE)
```

Data cleaning

- Add year and season variable, where year is categorical and season is a factor variable
- Change `FLDNUM` to be categorical (character)
- Remove latitude, longitude, and date from building RF

```
cleaning_for_forests <- function(df){
  # this function will be called after imputing

  df <- df %>%
    mutate(quarter = as.factor(quarter),
           FLDNUM = case_when(FLDNUM == 1 ~ "Lake City, MN",
                              FLDNUM == 2 ~ "Onalaska, WI",
                              FLDNUM == 3 ~ "Bellevue, IA",
                              FLDNUM == 4 ~ "Brighton, IL",
```

```
                              FLDNUM == 5 ~ "Jackson, MO",
                              FLDNUM == 6 ~ "Havana, IL"),
           FLDNUM = as.factor(FLDNUM),
           STRATUM = as.factor(STRATUM),
           TN = as.numeric(TN),
           TP = as.numeric(TP),
           TEMP = as.numeric(TEMP),
           DO = as.numeric(DO),
           TURB = as.numeric(TURB),
           COND = as.numeric(COND),
           VEL = as.numeric(VEL),
           SS = as.numeric(SS),
           WDP = as.numeric(WDP),
           CHLcal = as.numeric(CHLcal),
           SECCHI = as.numeric(SECCHI))

  return(df)

}

water20 <- water20 %>%
  filter(STRATUM != 9) %>% # 9 = Unexploded Ordinance Area - Pool 13???
  mutate(nice_date = mdy(DATE),
         year = year(nice_date),
         quarter = quarter(nice_date, fiscal_start = 3)) %>%
  cleaning_for_forests() %>%
  select(-SHEETBAR, -nice_date, -DATE, -LOCATCD, -LATITUDE, -LONGITUDE)

dim(water20)
```

```
## [1] 82442    15
```

Random forests notes

From the UC Riverside Programming Guide

- Individual decision trees can suffer from high variance and poor predictive performance.
- Random forests introduce randomness to a group of trees in two ways.
  - First, bagging (bootstrap aggregating) trees means that each tree is grown from a bootstrapped sample of the data. Bootstrapping is sampling with replacement, so each bootstrap sample is independent of the other.
  - However, bagged trees are still correlated because each bootstrap resample will have a similar structure of the original dataset. The most important splits for regression trees from different bootstrap samples will probably be similar.
- Second, random forests limit each tree's split to a random subset of the variables, called split-variable randomization. Let p be the number of predictor variables and m be the size of this random subset. Usually $m = p/3$ when the response variable is continuous. With these two steps, random forests reduce the correlation of the individual trees.
- Out of bag (OOB) error: as a result of the bootstrap resampling, the data that *aren't* sampled provide a natural validation set. This helps to decide on the number of trees to stabilize the error rate. This also provides an OOB RMSE without a train/test split.

- One disadvantage of RF is computational time.

Random forests do not handle missing values in the predictor values because the bagged trees *do not* use surrogates. As an ensemble method, random forests need all the variables.

One solution for missing values in the predictors is to impute those missing values with the median of the variable. This is done by setting `na.action = na.roughfix` and is described more in this post.

Random forests questions, July 7

1. Assessing the performance of a random forest with a train-test split.

a. Error metrics like RMSE for a random forest can be calculated with a train-test split, but they are more commonly calculated in the model-building process itself.

b. (This is called the "out of bag" (OOB) error. Recall that a random forest averages the predictions of individual bootstrapped trees. Each bootstrap resample of the data has a corresponding non-bootstrapped dataset (thus called OOB) which serves as a "test" set to evaluate RF predictions. The section about OOB error vs. test set error from this tutorial was helpful for me.)

c. The OOB RMSE look fairly promising, outperforming the other methods for TP and TN (0.09 and 1.802 respectively, compared to around 0.14 and 3 RMSE's from IDW and trees).

2. Dealing with missing values in the predictor variables.

a. RF's cannot take missing values in the predictors, so the algorithm imputes missing predictor variables with the median of the column/variable. The RF that got low OOB RMSE's indeed used this method of imputation in predictor values. I wanted to check that this is okay.

b. Also, in order to calculate error metrics with the train/test split, the RF model would only run on a test set with no missing values in the predictors. I wanted to check if this is okay to do, too.

c. There are also packages like missRanger that interpolate missing values with random forest modeled predictions. missRanger can interpolate multiple variables at once (the formula looks like . ~ ., where . refers to all the columns in a dataset). However, I'm not sure how to evaluate missRanger predictions using train/test splits.

RF Description

Random forests (RFs) are an ensemble machine learning algorithm that combines individual decision tree models, thus usually performing better than an individual tree otherwise would. RF are a popular model because they are simple to implement and can perform well with little to no tuning (Rashka). They can predict both categorical and continuous response variables. Here, I discuss RFs in the continuous setting; each individual tree model is a regression tree (RTs). In ecological applications like hydrology, water quality, and geology, RFs are used to interpolate continuous variables.

The RF algorithm works as follows. We first create a number of bootstrap samples (that is, sampling with replacement) of the data. Then, we fit regression trees on each of the bootstrap sample using split-variable randomization: each node of the tree is selected from a random subset of all the predictors. This modification, alongside the bootstrap samples, reduces the correlation among the trees. Then, the final RF prediction will be an average of the prediction of all the trees. The relevant parameters of the RF algorithm are the number of trees (the `randomForest` package sets this parameter to 500 by default) and the size of the random subset of predictors use in split-variable randomization.

The advantages of RF is as follows. The disadvantages include computational efficiency and its inability to handle missing values in the predictor variables.

Interpolating

The interpolation process is as follows, for each of the three response variables.

1. Filter the data for non-missing values in the response variable and create 80/20 train test splits, In the testing data, impute the median or mode for missing predictor values.

These are the functions that will be used to impute the missing predictor variables (along with `cleaning_for_forests`).

```
# which predictor variables have no missingness?
# names(water20)[sapply(water20, function(x) sum(!is.na(x)) == 82481)]
# "FLDNUM"  "STRATUM" "year"    "quarter"

replace_median_mode <- function(var_str, dataset, response_var){
  # for imputing test data with median/mode

  # iterate through each variable name var_str with sapply to
  # replace every variable except for response_var (no need to impute)


  # column of the dataset
  data_col <- eval(parse(text = paste("dataset$", var_str, sep = "")))

  if (var_str %in% c(response_var, "FLDNUM",  "STRATUM", "year",  "quarter")){
    # these other variables have no missingness
    return(data_col) # don't impute the response variable
  }

  if (class(data_col) == "factor"){
    # categorical imputation
    # take the mode

    counts_table <- table(data_col)

    # retrieve mode
    imputation <- names(counts_table)[counts_table == max(counts_table)]

  } else if (class(data_col) %in% c("numeric", "integer")){
    # continuous imputation
    # take the median

    imputation <- median(data_col, na.rm = TRUE)

  } else (return("error in class of variable"))

  data_col[is.na(data_col)] <- imputation

  if (sum(!is.na(data_col)) == length(data_col)){

    return(data_col)
  } else return("error in imputing NAs")

}
```

```r
impute_test_data <- function(df, response_var, df_train){

  df <- data.frame(sapply(names(df), replace_median_mode, df, response_var)) %>%
    cleaning_for_forests()

  # trick to fix incompatible types for random forest
  # source: https://stackoverflow.com/questions/24829674/r-random-forest-error-type-of-predictors-in-ne

  df <- rbind(df_train[1, ] , df)
  df <- df[-1,]

  return(df)

}
```

2. Fit the model with the `randomForest()` function with `na.action = na.roughfix`, which imputes missing predictor variables with the median or mode of the variable (depending on if it is continuous or categorical). Plot the performance of the model.
3. Apply the model on the test data, taking note of the RMSE and MAE. Also, take note of the distribution of the actual versus predicted values with a scatterplot and box plot. Finally, present a histogram of the residuals (the difference between predicted and actual).

Finally, I will include a table of the error metrics and size of the test data for each response variable.

```r
error_df <- data.frame("Response" = as.character(),
                       "RSME" = as.numeric(),
                       "MAE" = as.numeric(),
                       `Test data size` = as.numeric())

add_errors <- function(error_df, response_var, test_data){
  # the test_data has a column called predicted
  # that has RF predictions without NAs

  test_data <- test_data %>%
    mutate(residual_sq = (!!sym(response_var) - predicted)^2,
         abs_residual = abs(!!sym(response_var)- predicted))

  n <- round(dim(test_data)[1], 3)
  rsme <- round(sqrt(sum(test_data$residual_sq)/n), 3)
  mae <- round(sum(test_data$abs_residual)/n, 3)

  return(rbind(error_df, data.frame("Response" = response_var,
                                    "RSME" = rsme,
                                    "MAE" = mae,
                                    `Test data size` = n)))

}

distribution_df <- data.frame("Response" = as.character(),
                              "Mean" = as.numeric(),
                              "Minimum" = as.numeric(),
                              "Quartile 1" = as.numeric(),
                              "Median" = as.numeric(),
                              "Quartile 3" = as.numeric(),
```

```
                              "Maximum" = as.numeric())

add_distribution <- function(distribution_df, response_var, test_data){

  predictions <- test_data$predicted

  return(rbind(distribution_df,
               data.frame("Response" = response_var,
                          "Mean" = round(mean(predictions), 3),
                          "Minimum" = round(min(predictions), 3),
                          "Quartile 1" = round(quantile(predictions, 0.25), 3),
                          "Median" = round(median(predictions), 3),
                          "Quartile 3" = round(quantile(predictions, 0.50), 3),
                          "Maximum" = round(max(predictions), 3))))
}
```

Total Phosphorous

1. 80/20 split

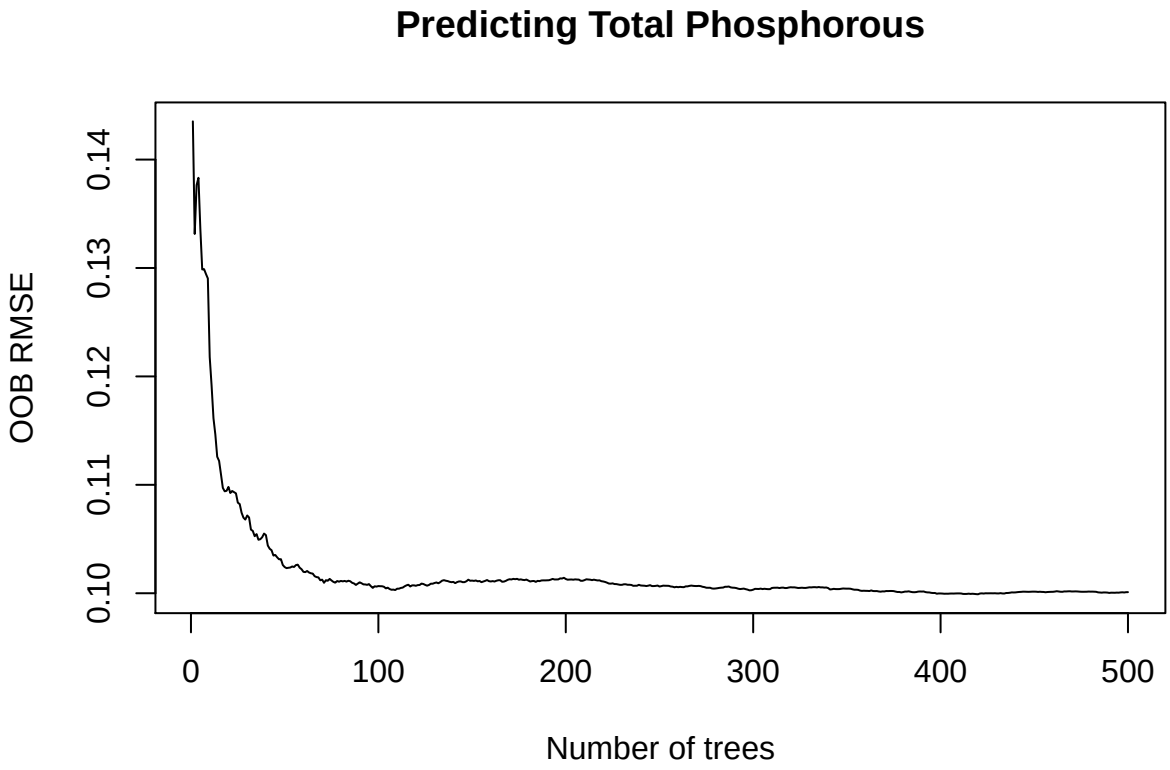
```
set.seed(4774)

# train and test RF on dataset with full TP values
fullTP <- water20 %>% filter(!is.na(TP))

# these splits will have missing values in the predictor
TP_split <- initial_split(fullTP, prop = 0.8)
TP_train <- training(TP_split)
TP_test <- testing(TP_split)
TP_test <- impute_test_data(TP_test, "TP", TP_train)
```

2. Fit the model

```
TP_rf <- randomForest(
  formula = TP ~ .,
  data    = TP_train,
  na.action = na.roughfix # impute missing predictor values with the median
)
```

```
TP_rf
```

```
##
## Call:
##  randomForest(formula = TP ~ ., data = TP_train, na.action = na.roughfix)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 0.01002035
##                     % Var explained: 73.11
```

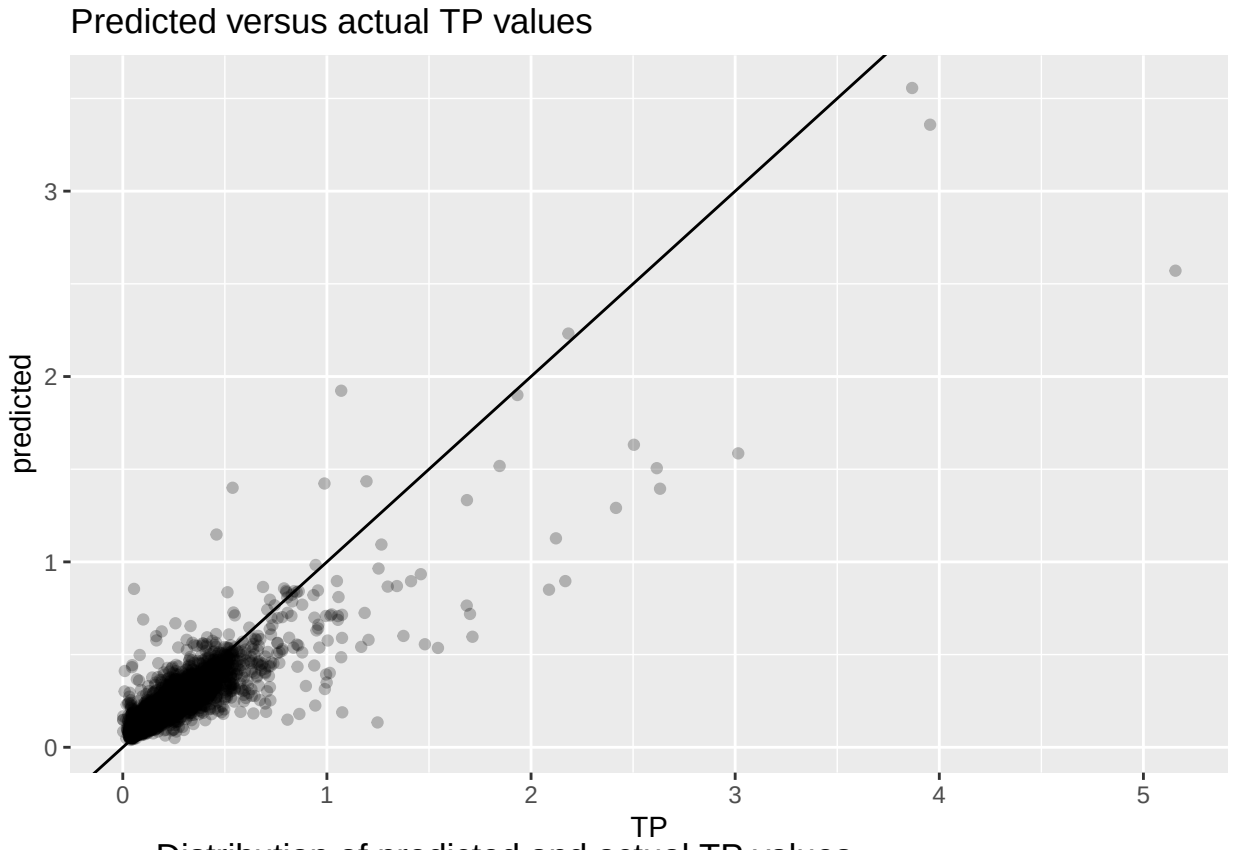


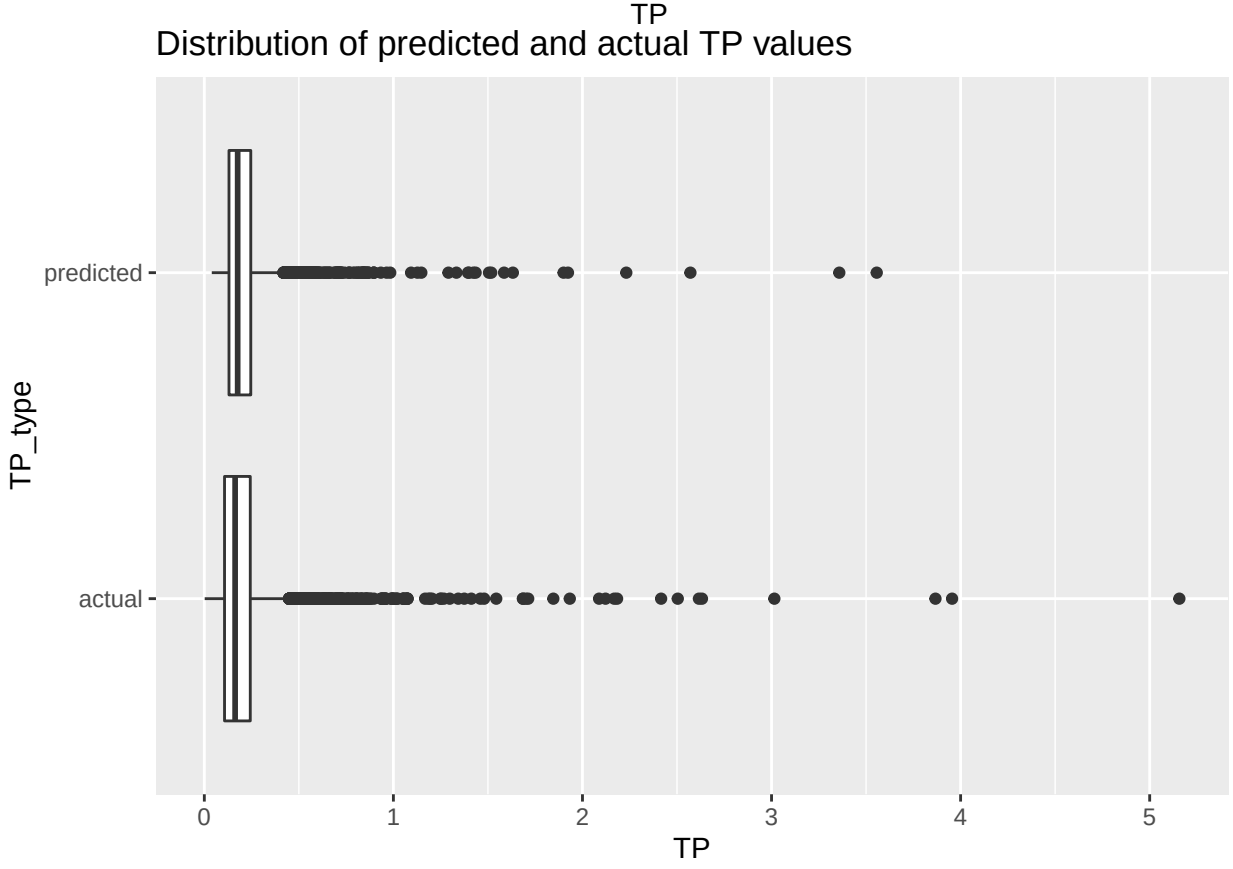
For `TP`, a RF with 420 trees minimizes the mean squared error.

Evaluate

I run the same `TP_rf` model on the test data.

```
# predict.all gives $aggregate and $individual
TP_pred_all <- predict(TP_rf, newdata = TP_test, predict.all = TRUE)

# https://gist.github.com/DexGroves/d6c055addf870b30d678862cc0fa8a88
# sapply(1:length(TP_pred_all$individual), function(n) mean(TP_pred_all$individual[1:n]))

TP_test$predicted <- TP_pred_all$aggregate

error_df <- add_errors(error_df, "TP", TP_test)
distribution_df <- add_distribution(distribution_df, "TP", TP_test)
```


Predicted versus actual TP values
predicted
TP
0
1
2
3
4
5


Distribution of predicted and actual TP values
TP_type
predicted
actual
TP
0
1
2
3
4
5

Total Nitrogen

1. 80/20 split

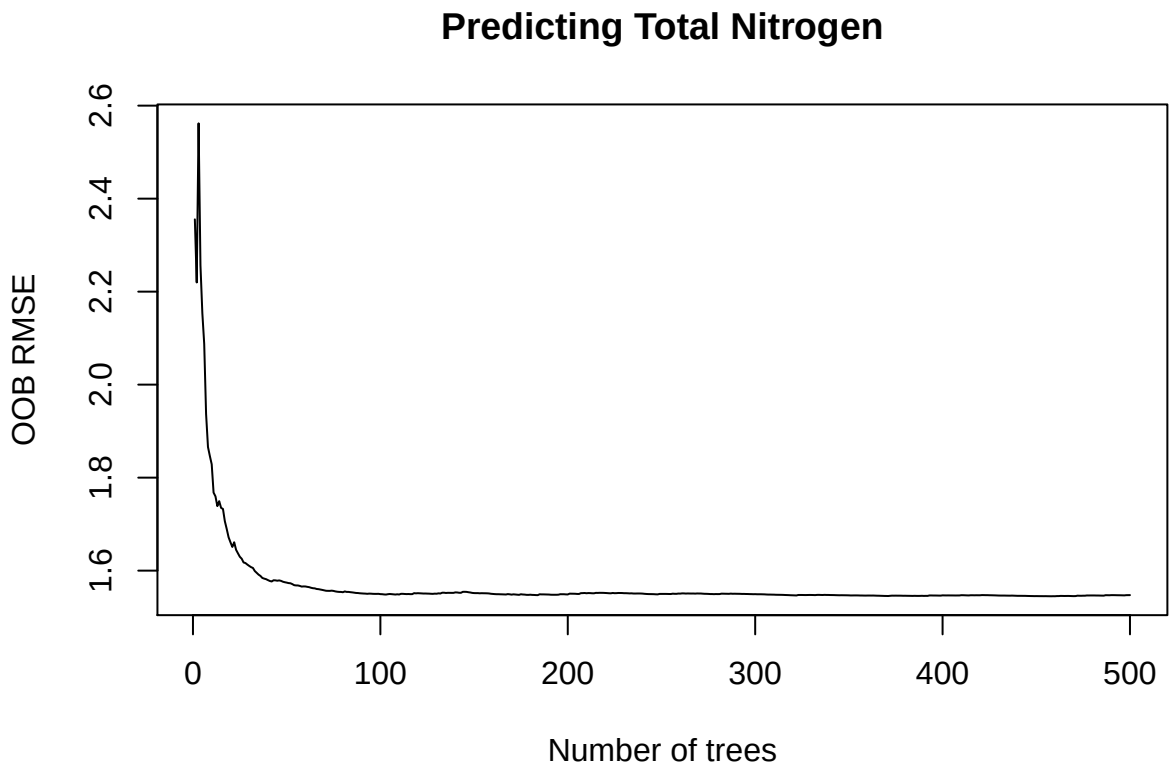
```
set.seed(4774)

fullTN <- water20 %>% filter(!is.na(TN))
TN_split <- initial_split(fullTN, prop = 0.8)
TN_train <- training(TN_split)
TN_test <- testing(TN_split)
TN_test <- impute_test_data(TN_test, "TN", TN_train)
```

2. Fit the model

```
# default RF model
TN_rf <- randomForest(
  formula = TN ~ .,
  data    = TN_train,
  na.action = na.roughfix
)
```

```
TN_rf
```

```
##
## Call:
##  randomForest(formula = TN ~ ., data = TN_train, na.action = na.roughfix)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 2.394229
##                     % Var explained: 39.91
```

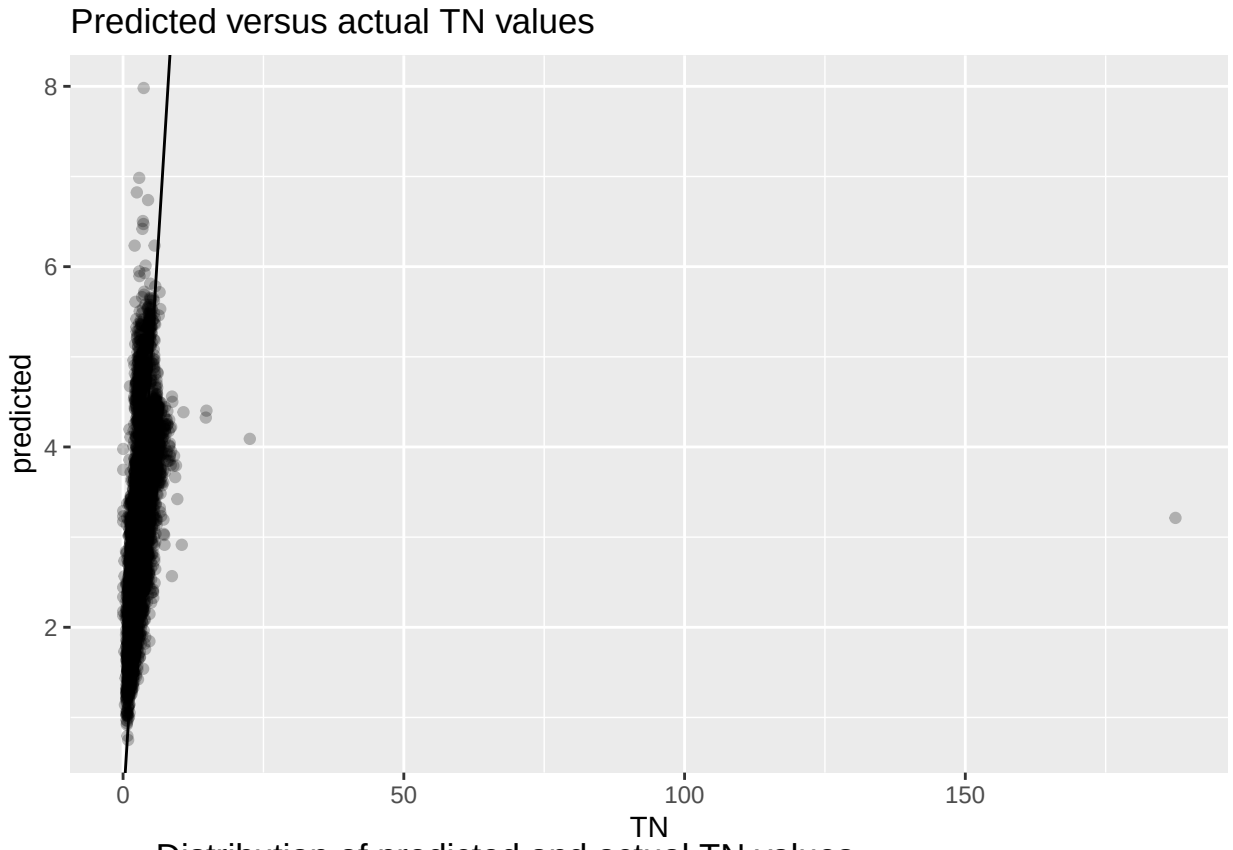


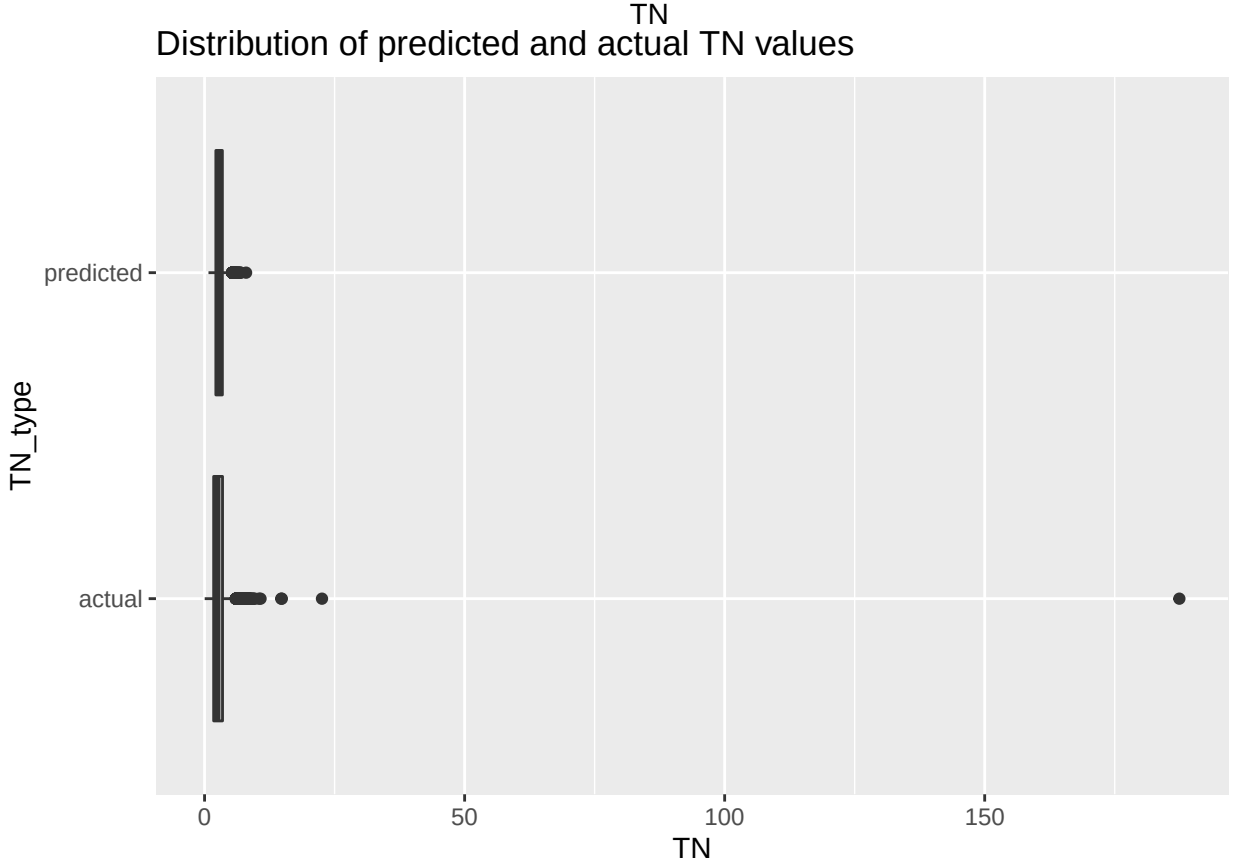
For `TN`, a RF with 458 trees minimizes the mean squared error.

Evaluate

I run the same `TN_rf` model on the test data.

```
# predict.all gives $aggregate and $individual
TN_pred_all <- predict(TN_rf, newdata = TN_test, predict.all = TRUE)

TN_test$predicted <- TN_pred_all$aggregate

error_df <- add_errors(error_df, "TN", TN_test)
distribution_df <- add_distribution(distribution_df, "TN", TN_test)
```


Predicted versus actual TN values
predicted
8
6
4
2
0
50
100
150
TN


Distribution of predicted and actual TN values
TN_type
predicted
actual
0
50
100
150
TN

Velocity

1. 80/20 split

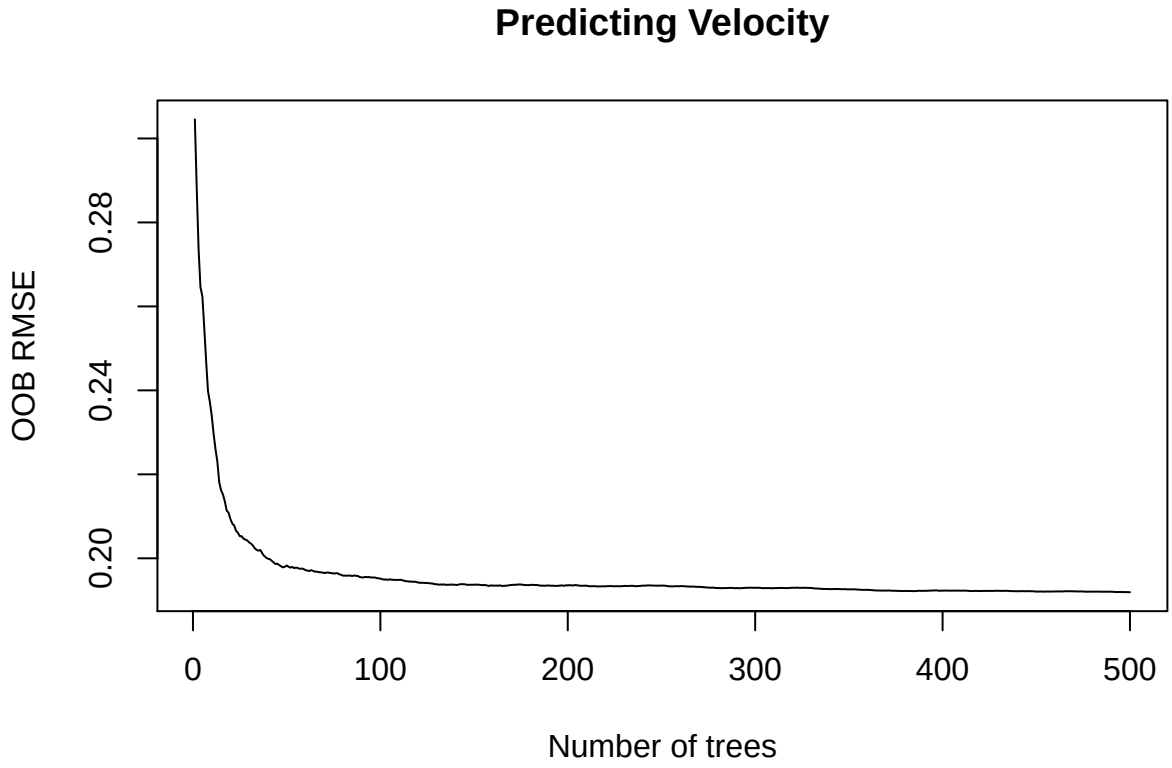
```
set.seed(4774)

fullVEL <- water20 %>% filter(!is.na(VEL))
VEL_split <- initial_split(fullVEL, prop = 0.8)
VEL_train <- training(VEL_split)
VEL_test <- testing(VEL_split)
VEL_test <- impute_test_data(VEL_test, "VEL", VEL_train)
```

2. Fit the model

```
# default RF model
VEL_rf <- randomForest(
  formula = VEL ~ .,
  data    = VEL_train,
  na.action = na.roughfix
)
```

```
VEL_rf
```

```
##
## Call:
##  randomForest(formula = VEL ~ ., data = VEL_train, na.action = na.roughfix)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 0.03683128
##                     % Var explained: 76.19
```

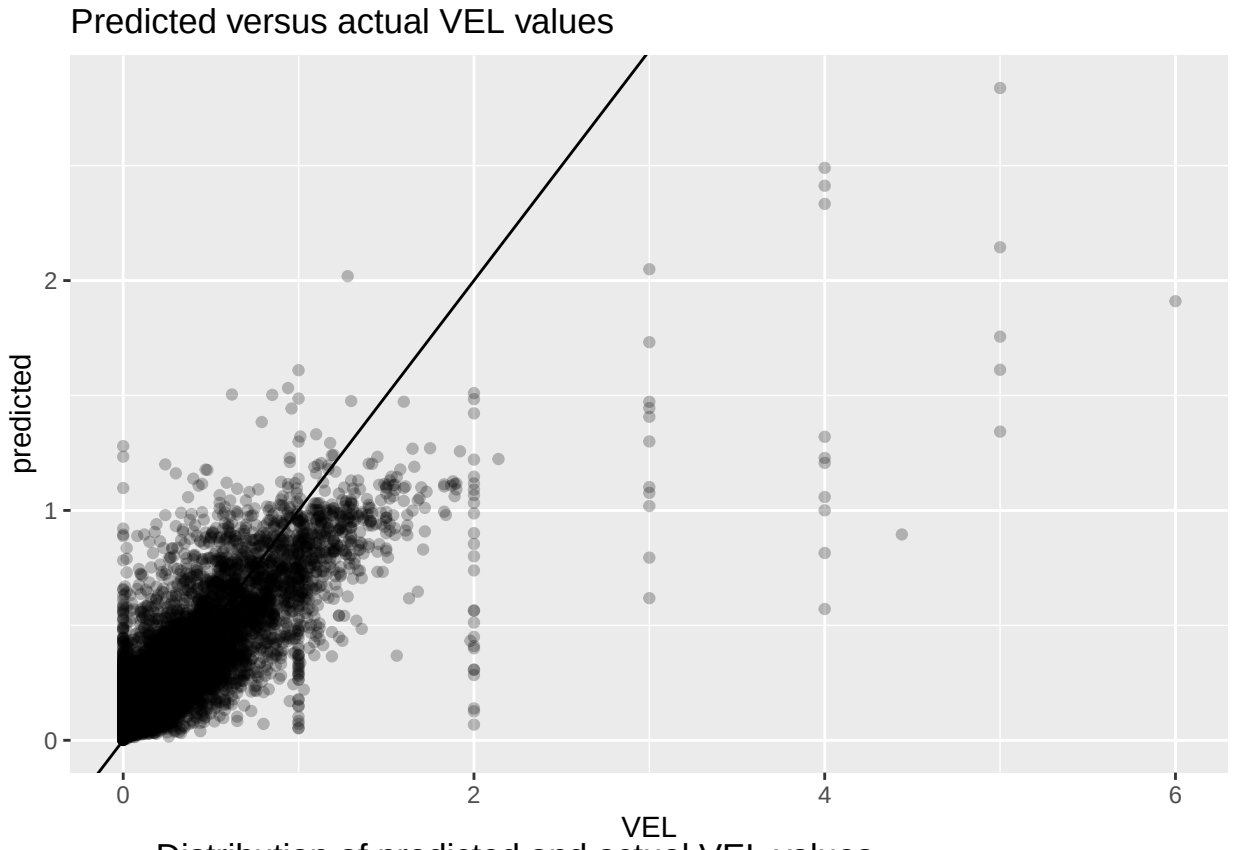


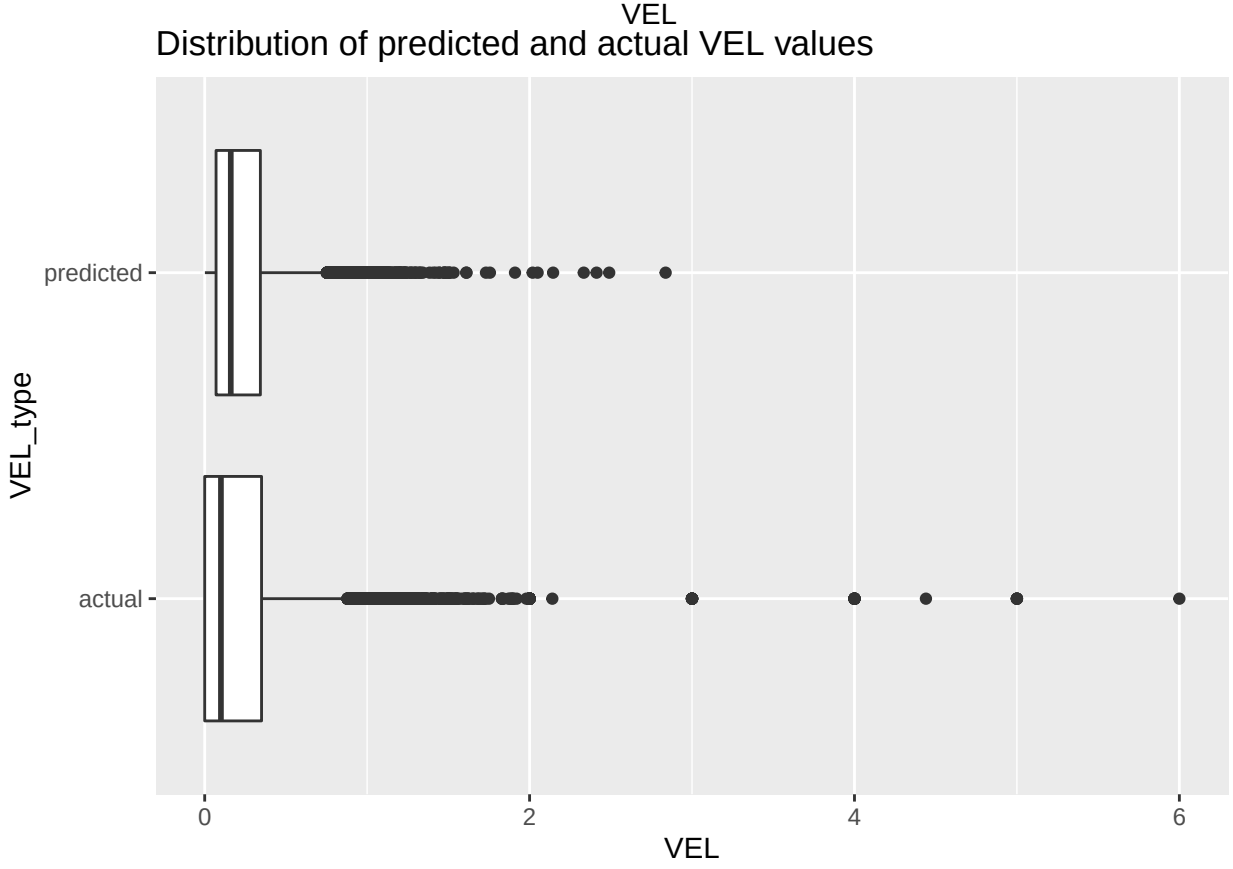
For `VEL`, a RF with 500 trees minimizes the mean squared error.

Evaluate

I run the same `VEL_rf` model on the test data.

```
# predict.all gives $aggregate and $individual
VEL_pred_all <- predict(VEL_rf, newdata = VEL_test, predict.all = TRUE)

VEL_test$predicted <- VEL_pred_all$aggregate

error_df <- add_errors(error_df, "VEL", VEL_test)
distribution_df <- add_distribution(distribution_df, "VEL", VEL_test)
```


Predicted versus actual VEL values
predicted
VEL
0
1
2
0
2
4
6


Distribution of predicted and actual VEL values
VEL_type
predicted
actual
VEL
0
2
4
6

Comparing results

```
error_df %>% kbl(booktabs = T)
```

Response	RSME	MAE	Test.data.size
TP	0.099	0.047	6286
TN	2.531	0.778	6434
VEL	0.226	0.127	11229

```
data.frame(distribution_df, row.names = NULL) %>% kbl(booktabs = T)
```

Response	Mean	Minimum	Quartile.1	Median	Quartile.3	Maximum
TP	0.207	0.039	0.130	0.176	0.176	3.556
TN	2.865	0.750	2.175	2.713	2.713	7.982
VEL	0.253	0.001	0.070	0.160	0.160	2.838