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 and season are both factor variables
- Change `FLDNUM` and `STRATUM` to be factor variables
- Remove sheetbar, date, location code, latitude, and longitude, from building RF

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

  df <- df %>%
    mutate(quarter = as.factor(quarter),
           year = as.factor(year),
           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 = case_when(STRATUM == 1 ~ "Main channel",
                               STRATUM == 2 ~ "Side channel",
                               STRATUM == 3 ~ "Backwater area contiguous to the main channel",
                               STRATUM == 4 ~ "Lake Pepin or Swan Lake",
                               STRATUM == 5 ~ "Impounded",
                               STRATUM == 6 ~ "Isolated",
                               STRATUM == 9 ~ "Unexploded Ordinance Area - Pool 13",
                               TRUE ~ as.character(STRATUM)),
           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 %>%
  mutate(nice_date = mdy(DATE),
         year = year(nice_date),
         quarter = quarter(nice_date, fiscal_start = 3)) %>%
  # year and quarter become factors later
  cleaning_for_forests() %>%
  select(-SHEETBAR, -nice_date, -DATE, -LOCATCD, -LATITUDE, -LONGITUDE)

dim(water20)
```

```
## [1] 82481    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 (for the `randomForests` package) 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.

Since the `ranger` package does not have a `na.action` parameter, I use median/mode imputation the training and test sets manually.

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

  #   print("running replace_median_mode")
  print(paste(var_str, "imputation is", as.character(imputation)))

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

    return(data_col)

  } else return("error in imputing NAs")

}

impute_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.

```
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 = !!sym(response_var)- predicted,
           residual_sq = residual^2,
           abs_residual = abs(residual))
```

```
  n <- round(dim(test_data)[1], 3)
  rsme <- round(sqrt(sum(test_data$residual_sq)/n), 5)
  mae <- round(sum(test_data$abs_residual)/n, 5)

  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), 5),
                          "Minimum" = round(min(predictions), 5),
                          "Quartile 1" = round(quantile(predictions, 0.25), 5),
                          "Median" = round(median(predictions), 5),
                          "Quartile 3" = round(quantile(predictions, 0.50), 5),
                          "Maximum" = round(max(predictions), 5))))
}
```

Total Phosphorous

1. 80/20 split

```
set.seed(4747)

# 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)
print("TP_train median/mode imputation")
```

```
## [1] "TP_train median/mode imputation"
```

```
TP_train <- impute_data(TP_train, "TP", TP_train)
```

```
## [1] "TN imputation is 2.548"
```

```
## [1] "TEMP imputation is 14.5"
## [1] "DO imputation is 9.7"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 462"
## [1] "VEL imputation is 0.1"
## [1] "SS imputation is 26"
## [1] "WDP imputation is 2.2"
## [1] "CHLcal imputation is 16.36344"
## [1] "SECCHI imputation is 42"
```

```
TP_test <- testing(TP_split)
print("TP_test median/mode imputation")
```

```
## [1] "TP_test median/mode imputation"
```

```
TP_test <- impute_data(TP_test, "TP", TP_train)
```

```
## [1] "TN imputation is 2.5145"
## [1] "TEMP imputation is 14.6"
## [1] "DO imputation is 9.7"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 460"
## [1] "VEL imputation is 0.1"
## [1] "SS imputation is 26.4"
## [1] "WDP imputation is 2.3"
## [1] "CHLcal imputation is 16.593295"
## [1] "SECCHI imputation is 42"
```

2. Fit the model

Using `ranger` instead of `randomForest`.

```
TP_rf <- ranger(
  formula = TP ~ .,
  data    = TP_train
)
```

```
TP_rf
```

```
## Ranger result
##
## Call:
##  ranger(formula = TP ~ ., data = TP_train)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      25160
## Number of independent variables:  14
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.0100633
## R squared (OOB):                  0.742759
```

On the training set, the RMSE is 0.100316 and the MAE is 0.0349623.

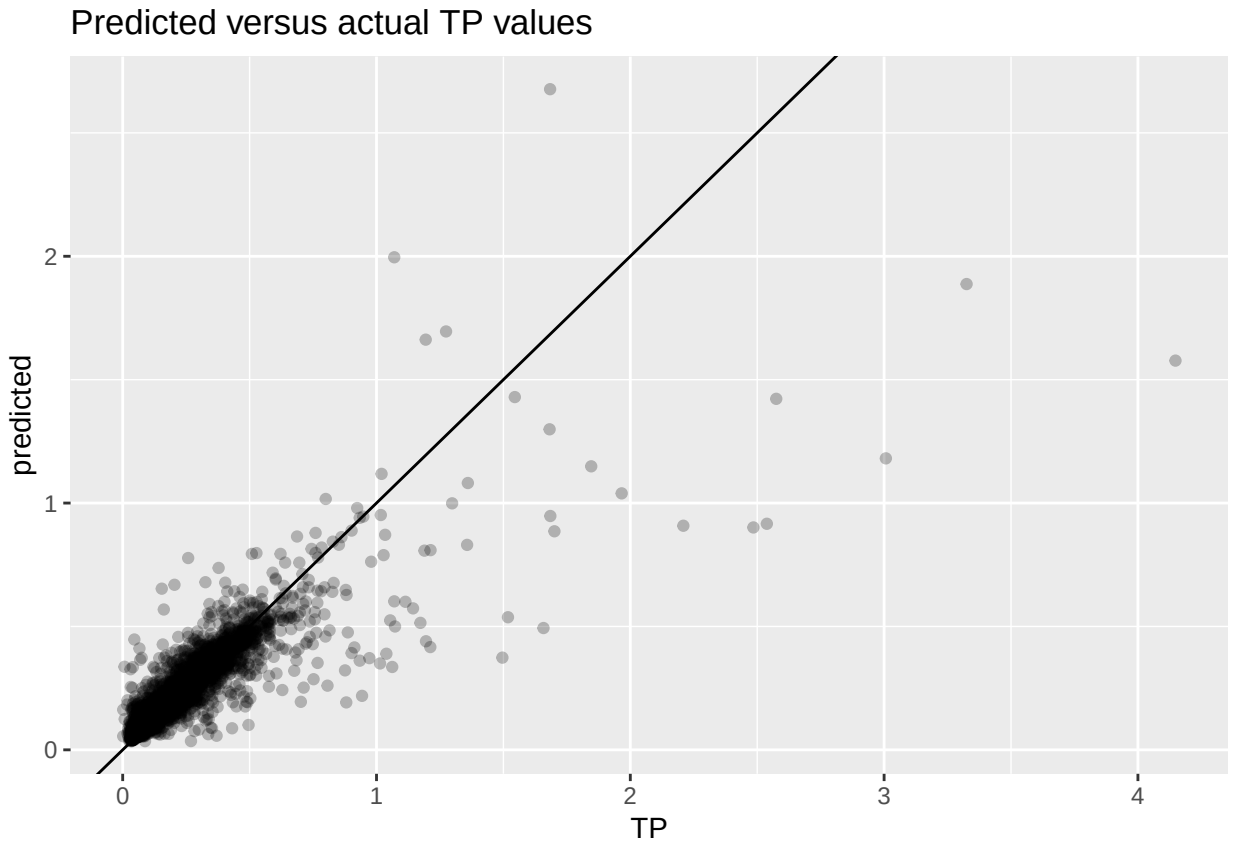
Evaluate

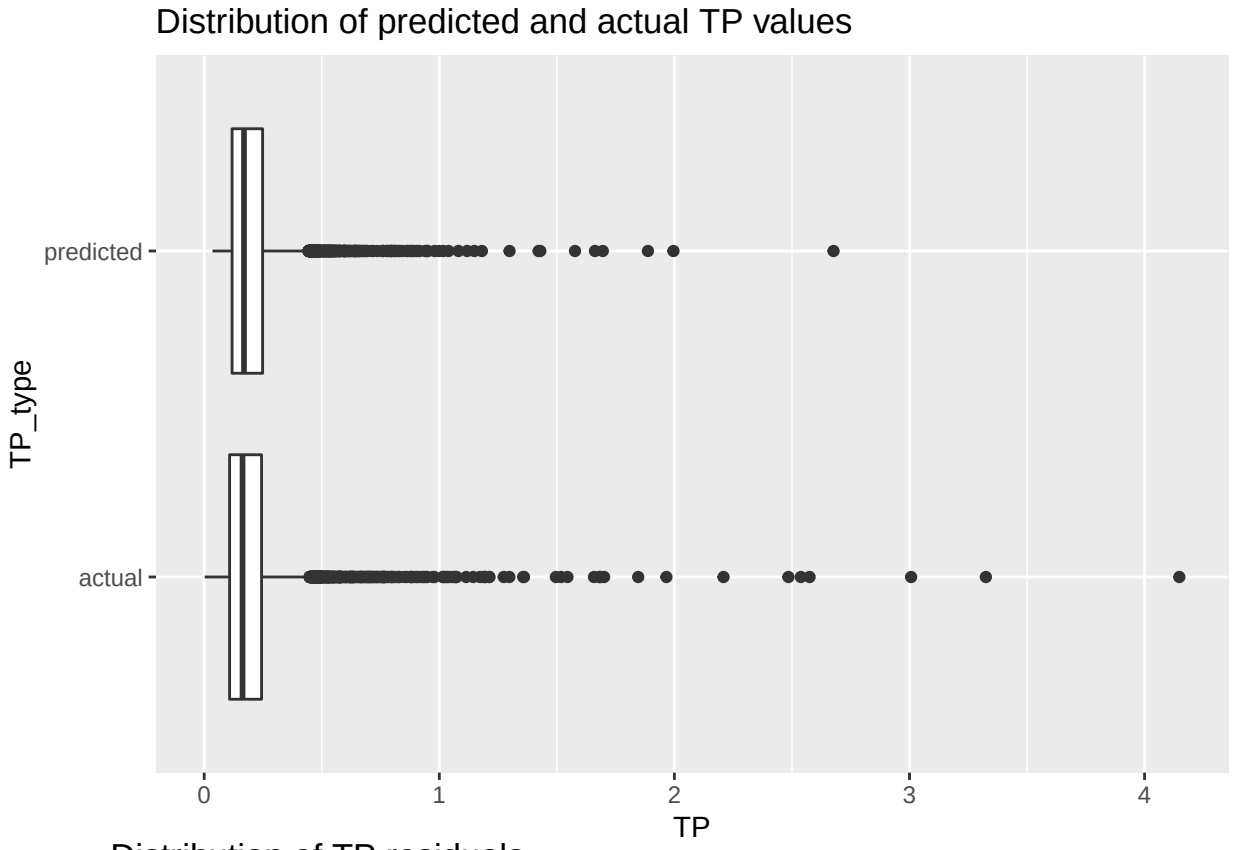
I run the same `TP_rf` model on the test data.

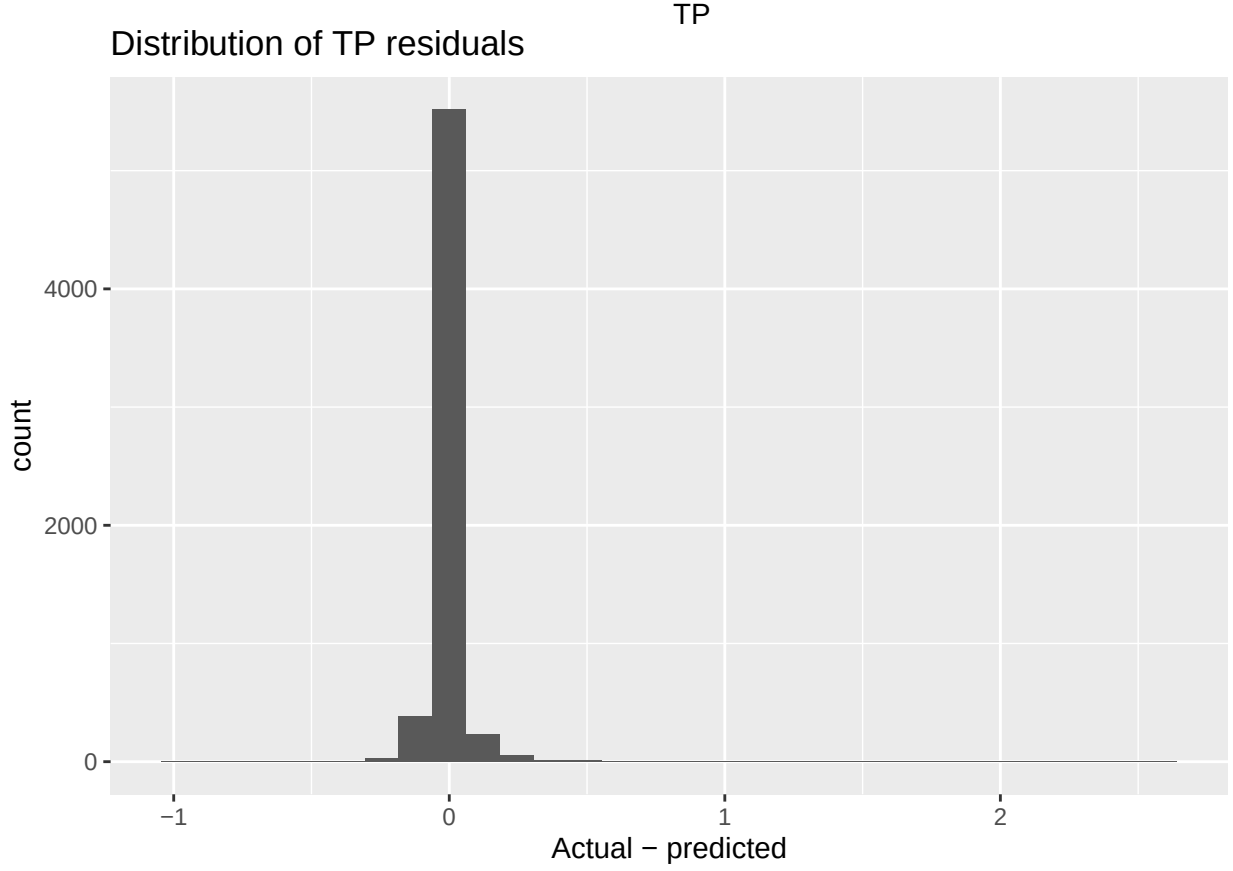
```
# https://gist.github.com/DexGroves/d6c055addf870b30d678862cc0fa8a88
# sapply(1:length(TP_pred_all$individual), function(n) mean(TP_pred_all$individual[1:n]))

TP_test$predicted <- predict(TP_rf, data = TP_test)$predictions

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




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


Distribution of TP residuals
4000
2000
0
count
−1
0
1
2
Actual − predicted

Total Nitrogen

1. 80/20 split

```
set.seed(4747)

fullTN <- water20 %>% filter(!is.na(TN))  %>% filter(TN < 100) # take away three values that are more t
TN_split <- initial_split(fullTN, prop = 0.8)
TN_train <- training(TN_split)
print("TN_train median/mode imputation")
```

```
## [1] "TN_train median/mode imputation"
```

```
TN_train <- impute_data(TN_train, "TN", TN_train)
```

```
## [1] "TP imputation is 0.163"
## [1] "TEMP imputation is 14.7"
## [1] "DO imputation is 9.7"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 461"
## [1] "VEL imputation is 0.1"
## [1] "SS imputation is 26.4"
## [1] "WDP imputation is 2.21"
## [1] "CHLcal imputation is 16.53175"
## [1] "SECCHI imputation is 41"
```

```
TN_test <- testing(TN_split)
print("TN_test median/mode imputation")
```

```
## [1] "TN_test median/mode imputation"
```

```
TN_test <- impute_data(TN_test, "TN", TN_train)
```

```
## [1] "TP imputation is 0.162"
## [1] "TEMP imputation is 14.6"
## [1] "DO imputation is 9.6"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 462"
## [1] "VEL imputation is 0.09"
## [1] "SS imputation is 26.4"
## [1] "WDP imputation is 2.24"
## [1] "CHLcal imputation is 16.36502"
## [1] "SECCHI imputation is 42"
```

2. Fit the model

```
# default RF model
TN_rf <- ranger(
  formula = TN ~ .,
  data    = TN_train
)
```

```
TN_rf
```

```
## Ranger result
```

```
##
## Call:
##  ranger(formula = TN ~ ., data = TN_train)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      25748
## Number of independent variables:  14
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.4732669
## R squared (OOB):                  0.7718334
```

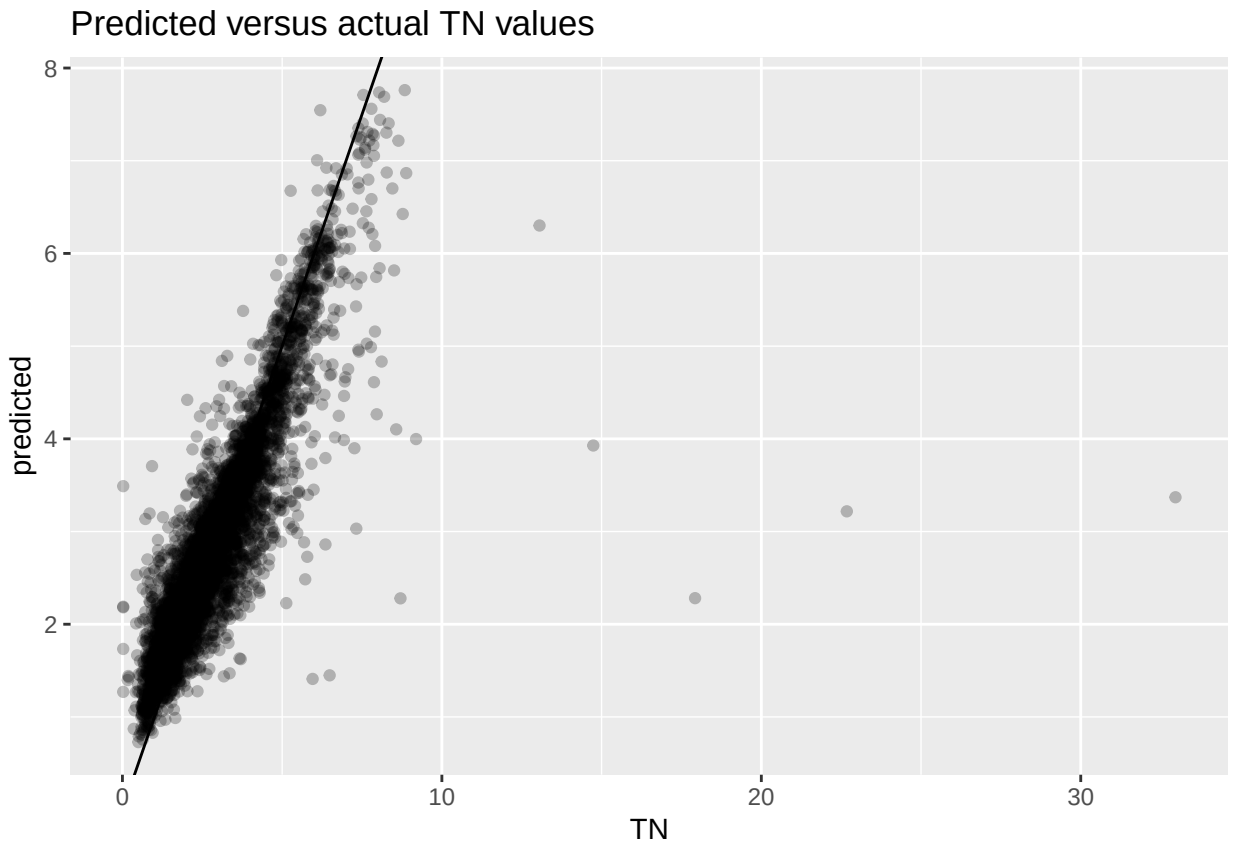
On the training set, the RMSE is 0.687944 and the MAE is 0.3790941.

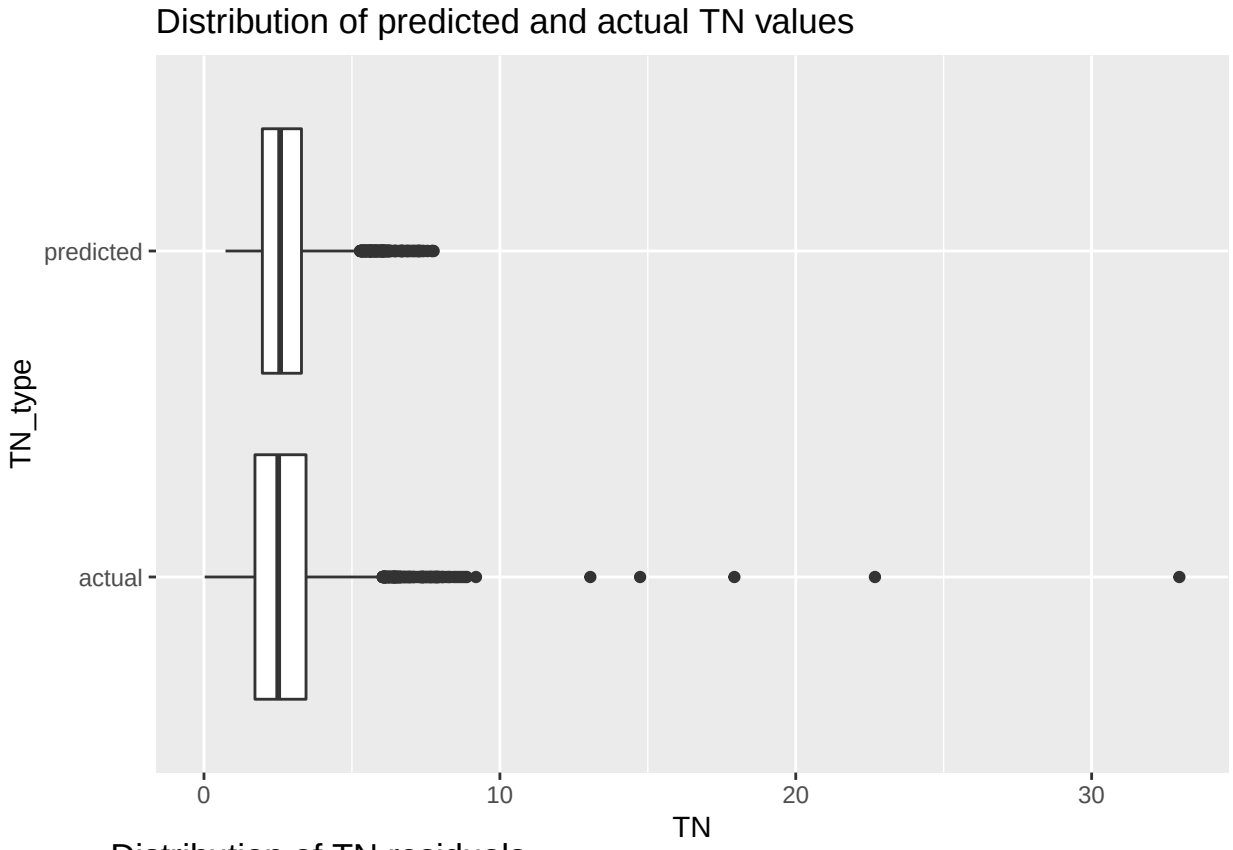
Evaluate

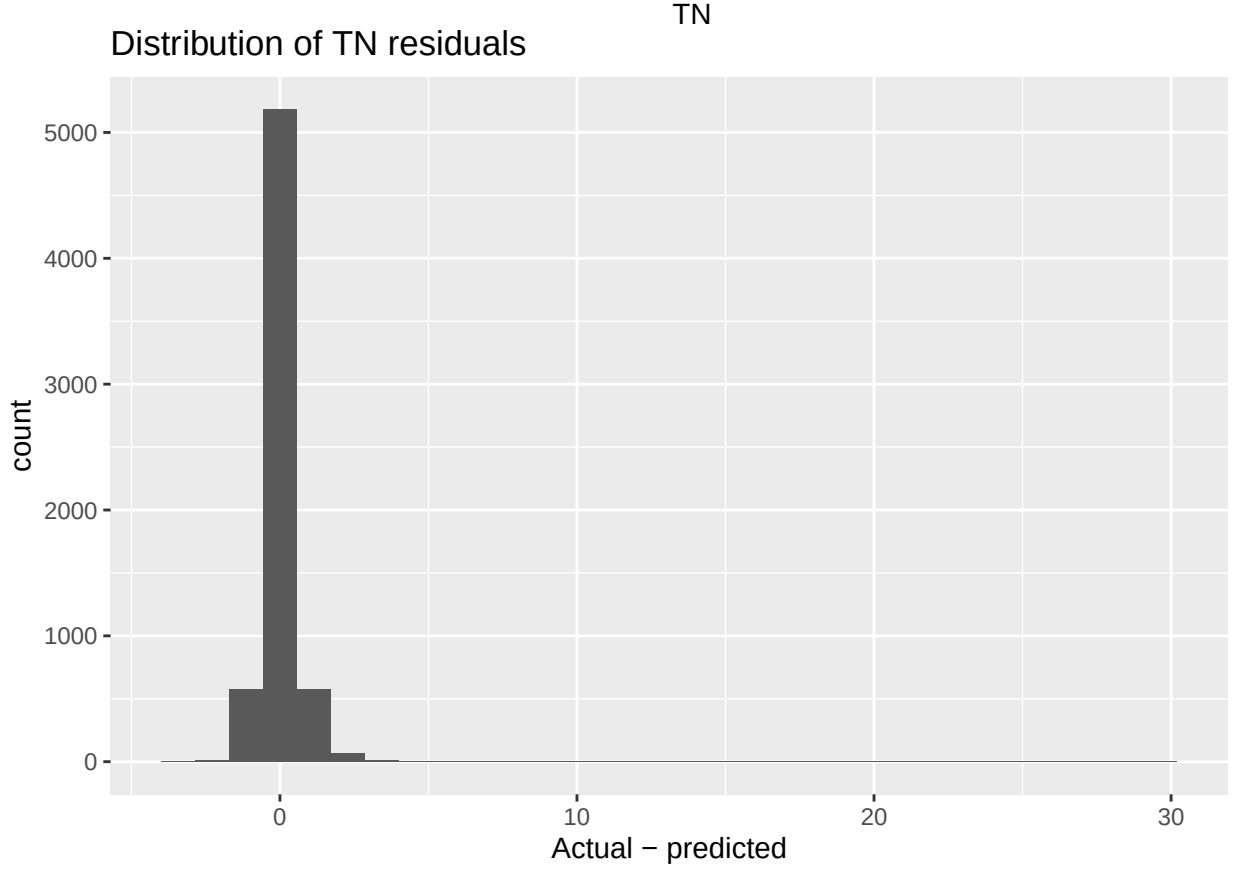
I run the same `TN_rf` model on the test data.

```
TN_test$predicted <- predict(TN_rf, data = TN_test)$predictions

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




Distribution of predicted and actual TN values
predicted
actual
TN_type
0
10
20
30
TN


Distribution of TN residuals
5000
4000
3000
2000
1000
0
count
0
10
20
30
Actual − predicted

Velocity

1. 80/20 split

```
set.seed(4747)

fullVEL <- water20 %>% filter(!is.na(VEL))
VEL_split <- initial_split(fullVEL, prop = 0.8)
VEL_train <- training(VEL_split)
print("VEL_train median/mode imputation")
```

```
## [1] "VEL_train median/mode imputation"
```

```
VEL_train <- impute_data(VEL_train, "VEL", VEL_train)
```

```
## [1] "TN imputation is 2.391"
## [1] "TP imputation is 0.15"
## [1] "TEMP imputation is 14.4"
## [1] "DO imputation is 9.9"
## [1] "TURB imputation is 16"
## [1] "COND imputation is 443"
## [1] "SS imputation is 20.4"
## [1] "WDP imputation is 1.5"
## [1] "CHLcal imputation is 16.62596"
## [1] "SECCHI imputation is 46"
```

```
VEL_test <- testing(VEL_split)
print("VEL_test median/mode imputation")
```

```
## [1] "VEL_test median/mode imputation"
```

```
VEL_test <- impute_data(VEL_test, "VEL", VEL_train)
```

```
## [1] "TN imputation is 2.3645"
## [1] "TP imputation is 0.149"
## [1] "TEMP imputation is 14.4"
## [1] "DO imputation is 9.9"
## [1] "TURB imputation is 16"
## [1] "COND imputation is 441"
## [1] "SS imputation is 20.3"
## [1] "WDP imputation is 1.5"
## [1] "CHLcal imputation is 16.2675"
## [1] "SECCHI imputation is 46"
```

2. Fit the model

```
# default RF model
VEL_rf <- ranger(
  formula = VEL ~ .,
  data    = VEL_train
)
```

```
VEL_rf
```

```
## Ranger result
```

```
##
## Call:
##  ranger(formula = VEL ~ ., data = VEL_train)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      44944
## Number of independent variables:  14
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       0.03814145
## R squared (OOB):                  0.7492412
```

On the training set, the RMSE is 0.1952984 and the MAE is 0.0985047.

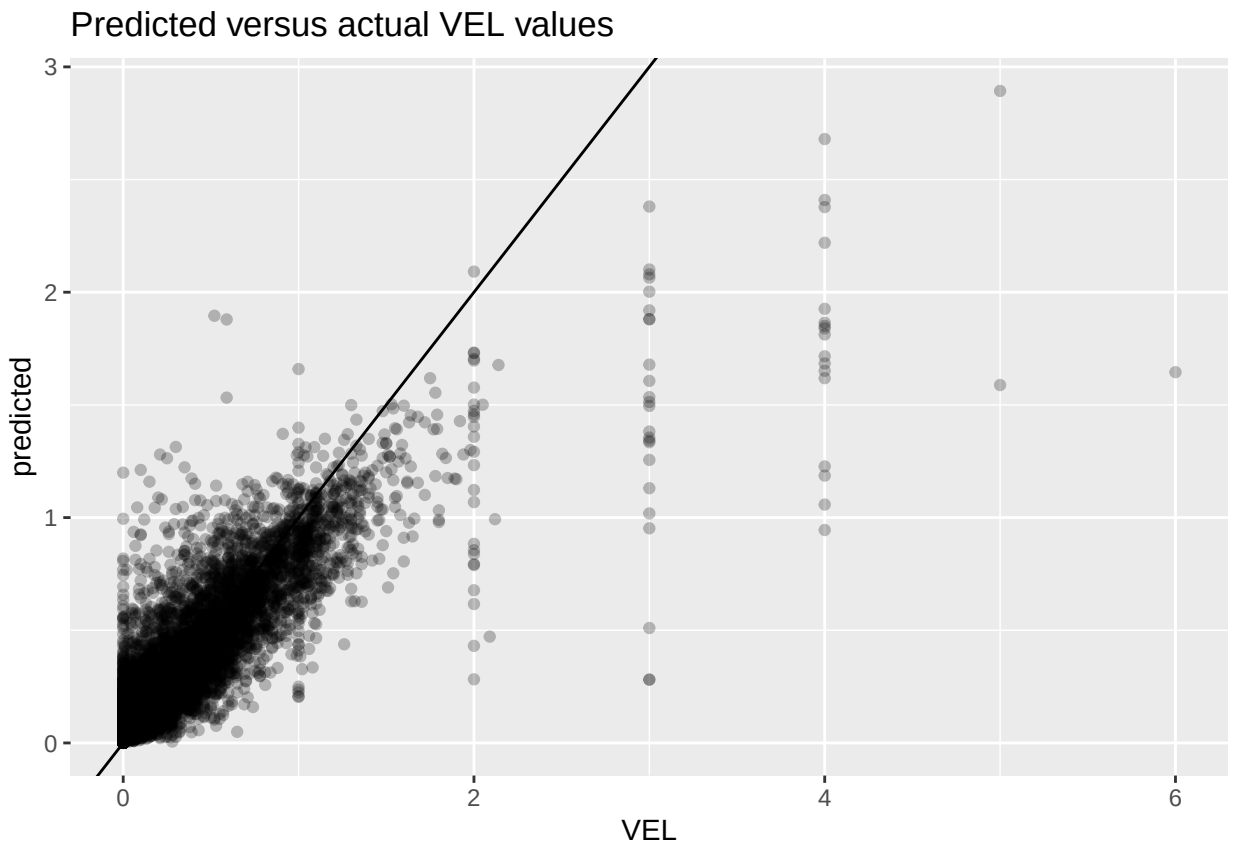
For `VEL`, a RF with trees minimizes the mean squared error.

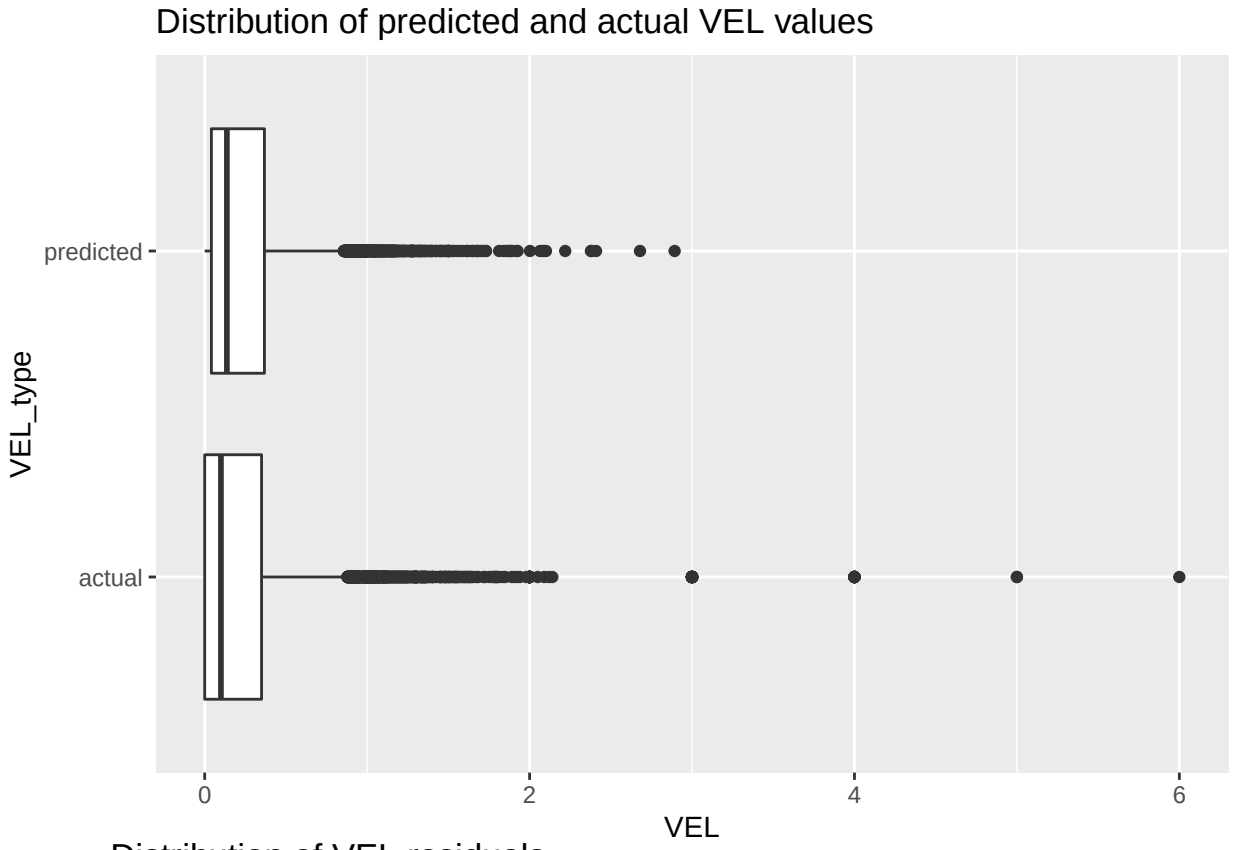
Evaluate

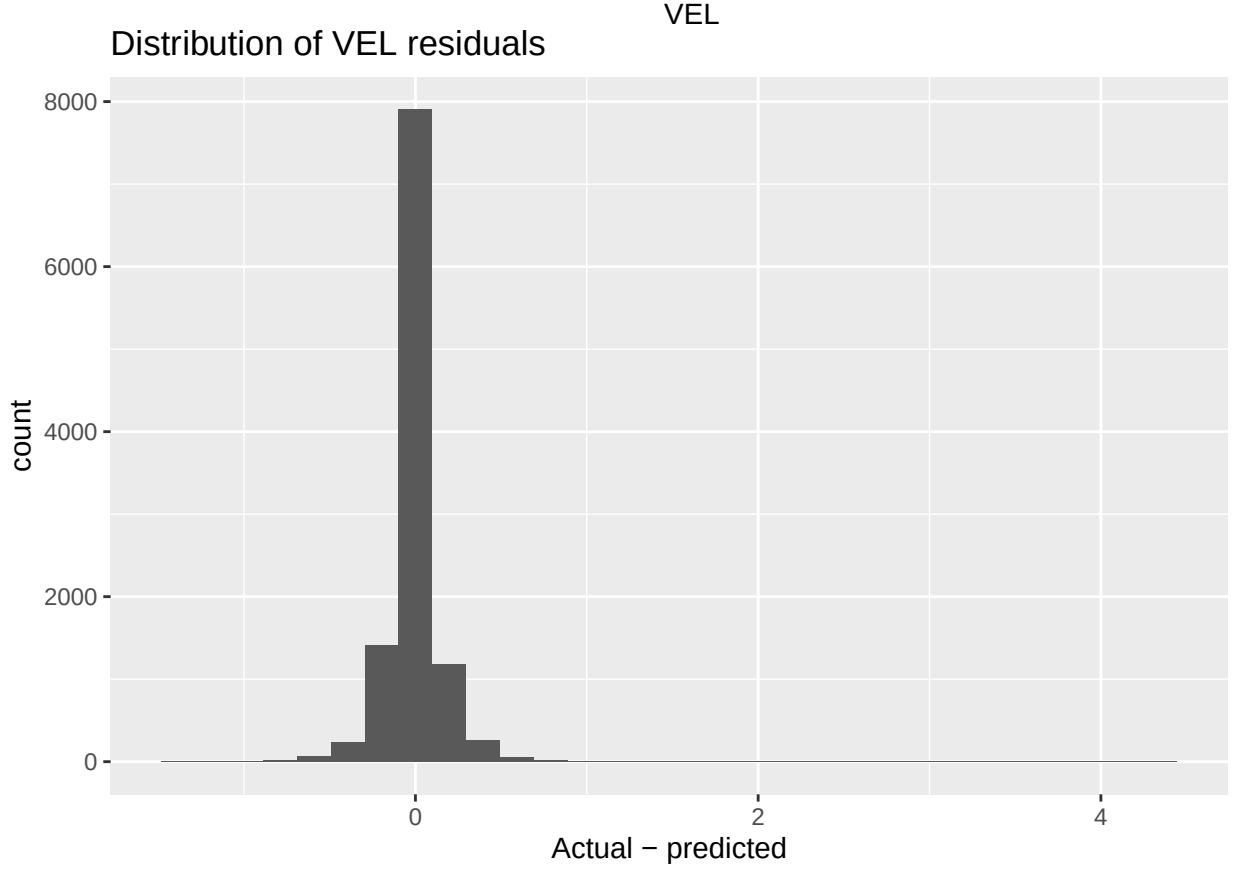
I run the same `VEL_rf` model on the test data.

```
VEL_test$predicted <- predict(VEL_rf, data = VEL_test)$predictions

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




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


Distribution of VEL residuals
8000
6000
4000
2000
0
count
0
2
4
Actual − predicted

Comparing results

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

Response	RSME	MAE	Test.data.size
TP	0.09311	0.03463	6290
TN	0.75663	0.38812	6437
VEL	0.20175	0.09989	11236

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

Response	Mean	Minimum	Quartile.1	Median	Quartile.3	Maximum
TP	0.20302	0.03509	0.11846	0.16926	0.16926	2.67701
TN	2.76707	0.72837	1.97261	2.57339	2.57339	7.76161
VEL	0.25516	0.00001	0.04121	0.13653	0.13653	2.89315