Regression Tree Interpolation

Alaina Stockdill

07/1/2021

Import libraries

```
library(tidyverse)
library(lubridate)
library(stringr)
library(caret)
library(rattle)
library(rpart)
library(kableExtra)
library(dplyr)
```

Predicting missing values in the LTRM data with regression trees

In order to get the LTRM data ready to use TDAmapper, we need to interpolate the missing values. While many methods have been considered, including linear/polynomial/multivariate regressions and inverse distance interpolation, we will also explore interpolation using regression trees. The benefit of the CART regression tree is that they will automatically choose the most important variables in predicting the target variables and are able to handle missing data very easily. With a data set that contains a large amount of incomplete rows, it seems that this method will be helpful in getting around this issue.

The data set that we are using has been cleaned to:

1. Replace values with bad QF codes with NA
2. Replace negative values for TP, TN, SS, TEMP, and with the minimum recorded value
3. Replace engative WDP values with 0
4. Filtered for surface samples (excluded mid and bottom samples)
5. Filtered for random samples (excluded fixed sites in river tributaries)
6. Combined rows that had the sample SHEETBAR values

```
setwd("/Users/alainastockdill/UMR-TDA-2021/InterpolationCode/CART interpolation")
water_data <- read.csv(file = "../../LTRM data/water_data_qfneg.csv")
```

Date and season are added in so that the model can use these values as predictors. Because these variables are included as predictors, we will not need to make separate trees from data that is filtered by these different values. Field number, stratum, and season are all made as categorical variables.

```
# Add in the date, year, and season
water_data <- water_data %>%
  mutate(nice_date = mdy(DATE),
         year = year(nice_date),
         season = quarter(nice_date, fiscal_start = 3)) %>%
  select(-SHEETBAR, -nice_date, -DATE, -LOCATCD)
```

```
# Make the FLDNUM and STRATUM categorical variables
water_data$FLDNUM <- as.character(water_data$FLDNUM)
water_data$STRATUM <- as.character(water_data$STRATUM)
water_data$season <- as.character(water_data$season)

water_vars <- list("TP", "TN", "VEL") #, "TURB", "COND","WDP", "SECCHI", "CHLcal", "SS", "TEMP", "DO")

predictor_vars <- c("TP", "TN", "TURB", "COND", "VEL", "WDP", "season", "STRATUM", "FLDNUM", "CHLcal",

rt_data <- water_data[predictor_vars]
```

Assumptions

While there are missing values in all continuous variables, we will be begin by interpolating for only TP, TN, and VEL, the variables with the most missingness. We will be usin TP, TN, TURB, COND, VEL, WDP, season, year, STRATUM, FLDNUM, CHLcal, SS, TEMP, SECCHI, and DO as our predictor variables. We will use an 80/20 train-test split to create the model and test for its accuracy. The train and test data for the tree for each interpolated variable will be made from a subset of the entire cleaned UMR data set that has no missing values for the current target variable.

After having created this models, we used the 1-SE rule to set the CP control method. Since TN is so poorly predicted, we will be only taking into account the CP value suggested for TP and VEL. The CP value for TP needed to be between .015 and .01 and for VEL it needed to be between .01 and .03. We will use CP = .011.

The MAE, RMSE, and summary stats for the regression tree models for interpolating TP, TN, VEL will be calculated using the test data.

Functions

- rmna_row: creates a subset of the entire data set that has a complete column of the variable we are currently estimating
- get_train_indices: creates a train/test split to test on our model
- make_trees: creates a regression tree to predict a target variable - saves the tree output (#https://stackoverflow.com/questions/7500219/reading-rpart-input-parameters-from-a-text-variable)
- rt_evalution: takes a target variable with a data and uses the three previous functions to output a testing data with a predicted column for that variable
- make_plots: makes an actual versus predicted scatter plot and box plots for each actual and predicted column of the current target variable - will save both plots
- get_model_stats: depending on the errorsBool, this function will either return a list containing the MAE and RMSE of actual versus predicted values or return a dataframe that contains the summary stats (Min, Q1, Mean, Q3, Max) for both the actual and predicted column

```
rmna_rows <- function(target_var, df) {
  df <- df %>% filter_at(vars(target_var), any_vars(!is.na(.)))
  return(df)
}

get_train_indices <- function(df) {
  sample_size = 0.80 * nrow(df)
  set.seed(571)
  train_indices <- sample(seq_len(nrow(df)), size = sample_size)
```

```
  return(train_indices)

}

make_trees <-function(target_var, df) {

  rpart_formula <- as.formula(paste(target_var, " ~ ."))

  model_controls <- rpart.control(cp = .011)
  tr_rpart <- rpart(rpart_formula,
                    data = df,
                    method = "anova",
                    control = model_controls )

  tree_title = paste("tree_",target_var, sep = "")
  png_name = paste(tree_title, ".png", sep = "")

  png(png_name)
  tree <- fancyRpartPlot(tr_rpart, main = tree_title)
  dev.off()

  return(tr_rpart)

}


rt_evaluation  <- function(target_var, df) {
  df <- rmna_rows(target_var, df)

  indices <- get_train_indices(df)

  data_train <- df[indices, ]
  data_test <- df[-indices, ]

  model <- make_trees(target_var, data_train)

  pred_col_name <- paste("PREDICTED_", target_var, sep = "") # New predicted column
  data_test[ , pred_col_name] <-  predict(model, data_test)

  #printcp(model)

  return(data_test)
}

make_plots <- function(data_test, target_actual, target_pred) {
    X <- data_test[ , target_actual]
    Y <- data_test[ , target_pred]

    # scatter plot
    png_name = paste(target_actual, "_predvactual.png", sep = "")
    ggplot(data_test, mapping = aes(x = X, y = Y)) +
      geom_point(alpha = .02) +
      coord_equal() +
```

```
      theme(aspect.ratio = 1) + xlim(0, 4) + ylim(0, 4) +
      labs( y = target_pred, x = target_actual)
    ggsave(png_name)

    # box plot -TODO
    #png_name = paste(target_actual, "_boxplot.png", sep = "")
    #ggplot(data_test) +
    #  geom_boxplot(aes(x = interaction(X,Y)))
    #ggsave(png_name)

    # historgram
    #png_name = paste(target_actual, "")

}


get_model_stats <- function(data_test, errorBool) {

  data_test <- data.frame(data_test) # make sure data_test is a dataframe

  # get the target variable
  target_pred  <- names(select(data_test, contains("PREDICTED_")))  # column name
  target_actual <- sub("PREDICTED_", "", target_pred)

  if( errorBool == TRUE )  {

    errors <- c(MAE(data_test[ , target_actual], data_test[ , target_pred]),
                RMSE(data_test[ , target_actual], data_test[ ,target_pred]))

    #errors <- errors %>% rename(errors = target_actual)

    return(errors)

  } else {

    actual_sum <- summary( data_test[ , target_actual] )
    predicted_sum <- summary( data_test[ , target_pred] )

    actual <- c(actual_sum[1], actual_sum[2], actual_sum[3],
                actual_sum[4], actual_sum[5], actual_sum[6])

    predicted <-c(predicted_sum[1], predicted_sum[2], predicted_sum[3],
                  predicted_sum[4], predicted_sum[5], predicted_sum[6])

    model_summary <- data.frame(actual, predicted)

    names(model_summary)[1] <- target_actual
    names(model_summary)[2] <- target_pred

    # actual v pred and boxplots
    make_plots(data_test, target_actual, target_pred)

    return(model_summary)
```

```
  }
}
```

```
water_var_preds <- lapply(water_vars, rt_evaluation, rt_data)
water_var_errors <- lapply( water_var_preds, get_model_stats, TRUE )
water_var_summary <- lapply( water_var_preds, get_model_stats, FALSE )

# combine columns
type <- c("MAE", "RMSE")
TN <- water_var_errors[[2]]
TP <- water_var_errors[[1]]
VEL <- water_var_errors[[3]]
model_errors <- data.frame( type, TP, TN, VEL)

one <- water_var_summary[[1]]
two <- water_var_summary[[2]]
three <- water_var_summary[[3]]
model_summary <- data.frame(one, two, three)
```

Scratch work

```
# ### COMPLETE TN COLUMN ###
# # Create a new data set that removes only the na TN values
# water_TN = model_data %>%
#   filter_at(vars(TN), any_vars(!is.na(.)))
#
#
# sample_size = 0.80 * nrow(water_TN)
# set.seed(571)
# train_indices <- sample(seq_len(nrow(water_TN)), size = sample_size)
#
# data_train <- water_TN[train_indices, ]
# data_test <- water_TN[-train_indices, ]
#
# set.seed(4321)
# model_controls <- rpart.control(max_depth = 20)
# tr.TN_rpart <- rpart(TN ~ .,
#                      data = data_train,
#                      method = "anova",
#                      control = model_controls )
# fancyRpartPlot(tr.TN_rpart, main = "TN prediction for all years")
#
# data_test$PREDICTED_TN <- predict(tr.TN_rpart, data_test)
# summary <- summary(data_test$TN)
# pred_summary <- summary(data_test$PREDICTED_TN)
#
# TN_sum <- c(summary[1], summary[2], summary[3], summary[4], summary[5], summary[6])
# TN_pred_sum <-c(pred_summary[1], pred_summary[2], pred_summary[3], pred_summary[4], pred_summary[5], p
#
# TN_sum2 <- list(unname(summary(data_test$TN)))
# TN_pred_sum2 <- list(unname(summary(data_test$PREDICTED_TN)))
#
# model_summary <- data.frame(TN_sum, TN_pred_sum)
```

```r
# model_summary2 <- data.frame(TN_sum2, TN_pred_sum2)
#
# TN <- c(MAE(data_test$TN, data_test$PREDICTED), RMSE(data_test$TN, data_test$PREDICTED), mean((data_t
#
```

```r
# ### RPART MODEL ###
# # Runs on the entire data set - even with na values in all columns
# set.seed(84726)
# temp_controls = rpart.control(cp = .01)
# tr.TN_rpart <- rpart(TN ~ .,
#                      data = water,
#                      method = "anova",
#                      control = temp_controls)
# fancyRpartPlot(tr.TN_rpart, main = "TN prediction: cp = .1")
# ```

# ```{r}
# # Tree evaluation summary
# plotcp(tr.TN_rpart)
# printcp(tr.TN_rpart)
# rsq.rpart(tr.TN_rpart)
# summary(tr.TN_rpart)
```