Code Appendix

Taqi

12/10/2021

Appendix

Preprocessing

```
#======================#
#       Wrangling      #
#======================#
# Initial wrangling
wrangle_init <- function(data, omit_NA = TRUE, omit_idx = TRUE){
  # Boolean variables (from int to logical type)
  data$holiday <- as.logical(data$holiday)          # 0 or 1
  data$workingday <- as.logical(data$workingday)    # 0 or 1
  # Other categorical variables (from int to factor type)
  data$season <- as.factor(data$season)             # 1 to 4
  data$yr <- as.factor(data$yr)                     # 0 to 1
  data$mnth <- as.factor(data$mnth)                 # 1 to 12
  data$weekday <- as.factor(data$weekday)           # 0 to 6
  data$weathersit <- as.factor(data$weathersit)     # 1 to 4
  # Re-scale the normalized measurements
  data$temp <- data$temp * 41
  data$atemp <- data$atemp * 50
  data$hum <- data$hum * 100
  data$windspeed <- data$windspeed * 67
  # Change type of Dates (from char to Date type)
  data$dteday <- as.Date(data$dteday)
  # Remove NAs (if prompted) default value is TRUE
  if(omit_NA) { data <- na.omit(data) }
  # Remove instance column (if prompted) default value is TRUE
  if(omit_idx) { data <- data %>% select(-c("instant")) }
  # Observe christmas
  data$holiday[359] <- T; data$holiday[725] <- T
  # Return the wrangled dataset
  return(data)
}
```

```
#==========================#
#      Weekly Averages     #
#==========================#
# Compute the (1-week lagged) weekly averages of a given variable
weekly_avgs <- function(data, var){
  # Compute the averages of the variable by week
  weekly_cnts <-
```

```r
    data %>%
    group_by(week) %>%
    summarize(wavg = mean({{ var }}))
  # Lag week by 1
  weekly_cnts$week <- weekly_cnts$week + 1
  # Remove excess weeks
  return(weekly_cnts %>% filter(week <= 53))
}

# Returns a dataset with an added column of weekly averages of a given variable
add_weekly_avg_var <- function(data, var, var_name){
  # Obtain the weekly averages of the desired variable
  var_avgs <- data %>% weekly_avgs({{ var }})
  # Rename the weekly average the desired variable name
  colnames(var_avgs)[2] <- var_name
  # Join the week column in the dataset by the weekly averages in the var_avgs dataframe
  return(data %>% left_join(var_avgs))
}

# Given the bike dataset, returns the dataset with week
# column and weekly averages for the three response variables
add_weekly_averages <- function(data){
  # Add the week variable to the dataset
  data <- data %>% mutate(week = ceiling(1:nrow(data)/7))
  # Add the cnt, reg, and cas weekly averages to the data
  data <- data %>% add_weekly_avg_var(cnt, "wavg_cnt")
  data <- data %>% add_weekly_avg_var(registered, "wavg_reg")
  data <- data %>% add_weekly_avg_var(casual, "wavg_cas")
  return(data)
}

#=========================#
#        Subsetting       #
#=========================#
# Filter for the 2011 data
in_2011 <- function(data){ return(data[(data$dteday >= "2011-01-01" & data$dteday <= "2011-12-31"),]) }
# Filter for the 2012 data
in_2012 <- function(data){ return(data[(data$dteday >= "2012-01-01" & data$dteday <= "2012-12-31"),]) }
```

Variable Selection

Predictors Selection

Predictors Selection

Response Transformation

Initial Modeling

```r
# A helper function that returns a formula in the "lm" syntax
# Takes predictors, a vector of variable name strings as an input
.parseFormula <- function(predictors, response = "cnt"){
  f <- as.formula(
    paste(response,
```

```
          paste(predictors, collapse = " + "),
          sep = " ~ "))
  return(f)
}
```

```
lm.tot <- function(varset, train_data = data2011){
  model <- lm(formula = .parseFormula(predictors = varset, response = "cnt"), data = train_data)
  model
}
lm.cas <- function(varset, train_data = data2011){
  model <- lm(formula = .parseFormula(predictors = varset, response = "casual"), data = train_data)
  model
}
lm.reg <- function(varset, train_data = data2011){
  model <- lm(formula = .parseFormula(predictors = varset, response = "registered"), data = train_data)
  model
}
```

Beginning Model

Final Model

Diagonostic Analysis

Validation and Problemshooting

Refined Model

Prediction of the Yearly Growth Ratio

```
#==========================#
#      Miscellaneous       #
#==========================#

# Enumerate all the pairs in the lower-triangular matrix scheme
# In other words, (i < j or i - j < 0) so that day_i preceeds day_j
.unique_pairs_lower <- function(N){
  is <- do.call("c", purrr::map(1:N, function(i){rep(i,N)}))
  js <- rep(1:N, N)
  # Helper function: selects elements only if they are upper triangular
  .LowerTri <- function(i, j){if(i > j) { c(i = i, j = j) }}
  pairs <- do.call("rbind", purrr::map2(is, js, .f = .LowerTri))
  data.frame(pairs)
}

#===================================#
#     Dataframe Wrapper Functions   #
#===================================#

# Given a dataframe of loss values between all possible day pairs
# and a set of idx and loss bounds, compute the g estimates
get_df_param <- function(df_loss, idx_bds, loss_bds){
  # Compute the index pairs
  MIP <- .unique_pairs_lower(length(idx_bds))
  # Use helper function to compute the g estimates for each set of bounds
```

```r
  .helper <- function(k, df_loss = df_loss){
    idx <- MIP$i[k]; loss <- MIP$j[k]
    results <- g_estimate(df_loss, idx, loss)
    data.frame(idx_bd = idx_bds[idx], loss_bd = loss_bds[loss], g = results[[1]], n = results[[2]])
  }
  # Run helper over 1,...,n where n is the number of pairs surviving bound cutoff
  df_param <- map_dfr(1:nrow(MIP), .helper, df_loss)
  # Return the estimates and number of pairs used for each bound pair
  return(df_param)
}

get_df_loss <- function(data2011){
  # Obtain the continous variables and holiday
  data <- data2011 %>% select(dteday, holiday, atemp, hum, windspeed, "cnt")
  # Normalize the continous variables
  data <- data %>% mutate_at(c("atemp", "hum", "windspeed"), ~(scale(.) %>% as.vector))
  # Take out holidays (to avoid needing to account for its effect, omits only few days of data)
  data <- data %>% filter(!holiday)
  # Enumerate all the possible index pairs (for use in purrr::map2_dfr)
  pairs <- .unique_pairs_lower(nrow(data))
  # Helper function that takes two indices and returns the loss between the days at those indices
  .helper <- function(i,j){data.frame(i = i, j = j, loss_ij = loss_ij(i,j))}
  # Compute the loss function exhaustively for every possible (lower-triangle) pair
  df_loss <- map2_dfr(pairs[["i"]], pairs[["j"]], .f = .helper)
  # Add the index difference column (useful for determining space between days)
  df_loss <- df_loss %>% mutate(idx_diff = i - j)
  # Return the exhaustive dataset of loss values for every unique day ordered pair
  return(df_loss)
}

#==============================================================================#
#==============================================================================#

#==========================#
#      Loss Function       #
#==========================#

# Given two rows (days) from the data, compute the loss function
loss <- function(day0, day1){
  atemp_diff <- day0[["atemp"]] - day1[["atemp"]]
  wind_diff <- day0[["windspeed"]] -  day1[["windspeed"]]
  hum_diff <- day0[["hum"]] - day1[["hum"]]
  norm(as.matrix(c(4*atemp_diff, wind_diff, hum_diff)))
}

# Given two row indices, compute the loss function between those two days
loss_ij <- function(i, j){
  day0 <- data[i,]; day1 <- data[j,]
  loss(day0, day1)
}

g_estimate <- function(df_loss, idx_bound, loss_bound){
  # Compute the g-estimate after filtering through bounds
```

```r
  df_loss <- df_loss %>%
    filter(idx_diff > idx_bound) %>%
    filter(loss_ij < loss_bound) %>%
    mutate(g = growth_ratio(i, j))
  # Return the mean (g^) and number of pairs used (sample size n)
  list(mean(df_loss$g), nrow(df_loss))
}

#==================================#
#     Loss Fxn Technique Plots     #
#==================================#

triangle_plot <- function(df_param){
  df_param %>%
    ggplot() +
    geom_point(aes(x = idx_bd, y = loss_bd, color = g)) +
    labs(x = "Index (Difference) Lower Bound", y = "Loss Upper Bound", title = "Growth Factor Estimates
}

g_plot <- function(df_param){
  # Filter bounds to obtain a good window frame
  df_param <- df_param %>% filter(idx_bd > 10, loss_bd < 5)
  # Create the proportion of maximum pairs variable (for alpha)
  df_param <- df_param %>% mutate(perc_pairs_omitted = 1 - n/max(df_param$n))
  # Scatterplot
  df_param %>%
    ggplot() +
    geom_point(aes(color = idx_bd, x = loss_bd, y = g, alpha = perc_pairs_omitted)) +
    geom_abline(slope = 0, intercept = 3.35) +
    theme(legend.position = "bottom") +
    labs(x = "Loss Upper Bound", y = "g", title = "Growth Factor Esimates by Loss Upper Bound") +
    #scale_alpha(guide = "none") +
    scale_color_gradient(low = "palevioletred2", high = "seagreen3")
}

#============================================================================#
#============================================================================#

#==========================#
#     Window Technique     #
#==========================#

growth_ratio <- function(i, j){ data2011$cnt[i]/data2011$cnt[j] }

#get_rel_ratio <- function(row){ growth_ratio(i = env_diffs_sset[row, 1], j = env_diffs_sset[row, 2]) }


window_g <- function(w, no_outlier = T){
  # Compute the indices of the first and last w days
  lower_idx <- 1:w
  upper_idx <- 366 - (w:1)
  # Obtain the first and last w days
  last_w <- data2011[upper_idx,]
```

```r
  first_w <- data2011[lower_idx,]
  # Since Jan-3 is environmentally different (by loss function value), we remove it
  if(no_outlier){ first_w <- first_w %>% filter(dteday != "2011-1-3") }
  # Compute and return the difference in means between the first and last w days
  return(mean(last_w$cnt)/mean(first_w$cnt))
}

#================================#
#     Window Technique Plots    #
#================================#

plot_window <- function(tbl_window){
  # Plot parameters
  col0 <- "steelblue3"
  # Scatterplot
  tbl_window %>%
    ggplot(aes(w, g_w)) +
    geom_point(color = col0) +
    geom_line(color = col0) +
    geom_vline(xintercept = 6, color = "red") +
    labs(title = "Growth Factor Estimates by Window Size")
}

#---------------------------------------------------------------------------#
#===========================================================================#
#                           Primary Code Script
#===========================================================================#
#---------------------------------------------------------------------------#

# Control flow for avoiding computational/time waste
recompute <- F
if(recompute){
  # Obtain the exhaustive dataset of loss values for every unique day ordered pair
  df_loss <- data_2011 %>% get_df_loss
  # Write CSV to avoid future recomputation
  write.csv(df_loss, "df_loss.csv", row.names = F)
} else{
  df_loss <- read.csv("df_loss.csv")
}

#=====================================================#
#   Estimate Performance over Bound Paramater Space
#=====================================================#

# Set a sequence of index bounds
idx_bds <- seq(122, 365, 8)
# Set a sequence of loss bounds: more critical
loss_bds <- seq(1, 4, 0.10)
# Obtain the parameter dataframe of g estimates
df_param <- df_loss %>% get_df_param(idx_bds, loss_bds)
# Plot the estimates over bound parameter space
plot_g <- df_param %>% g_plot()
```

```r
#========================#
#    Window Technique    #
#========================#

# Obtain the table of g esimates by window paramater w
tbl_window <- purrr::map_dfr(1:20, function(w){data.frame(w = w, g_w = window_g(w))})
# Plot the values
plot_w <- plot_window(tbl_window)

#============================================================================#
#----------------------------------------------------------------------------#
#============================================================================#
```

Prediction without the Yearly Growth Ratio