Bike-Sharing Data Analysis:
Prediction of Daily Bike Rental Counts
Based on Multiple Linear Regression

Final Project Report · MA 575 Fall 2021 · C3 · Team #2

Ali Taqi, Hsin-Chang Lin, Huiru Yang, Ryan Mahoney, Yulin Li

12/10/2021

Abstract

In this project, the following question is to be answered: If we have the past history of bike rental counts as well as records of environmental and seasonal conditions, how and how well could we predict the bike rental counts in the future? In this project, such questions are approached by predictive modeling of daily bike rental counts from a 2011-2012 Bike Sharing dataset [1]. The daily bike rental counts are predicted with models based on Multiple Linear Regression (MLS) using the environmental and seasonal variables as predictors. The initial goal of this project is to train the model using only the 2011 data, and then validate the prediction power of the model on the 2012 data. Given the limited time span of available training data, issues are found in the validation process using the 2012 data; the impact of user base on the future predictions is brought to our attention. The initial models are then revisited and corrected to account for the effect of user base. The refined models are expected to have better prediction powers than the initial MLS models, but a full validation would require further availability of bike rental data.

1 Introduction

2 a

3 a

4 aa

5 a

6 a

7 Appendix

Note: The source code can be found on Github in the repository found in the references. The primary files to consider are the source files found in the "final" directory, which will be refactored and tidied in an ongoing manner.

7.1 Data

- `instant`: record index
- `dteday`: date
- `season`: season (1:spring, 2:summer, 3:fall, 4:winter)
- `yr`: year (0:2011, 1:2012)
- `mnth`: month (1 to 12)
- `hr`: hour (0 to 23)
- `holiday`: weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- `weekday`: day of the week
- `workingday`: if day is neither weekend nor holiday is 1, otherwise is 0.
- `weather-sit`:
  - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- `temp`: Normalized temperature in Celsius. The values are divided to 41 (max)
- `atemp`: Normalized feeling temperature in Celsius. The values are divided to 50 (max)
- `hum`: Normalized humidity. The values are divided to 100 (max)
- `windspeed`: Normalized wind speed. The values are divided to 67 (max)
- `casual`: count of casual users
- `registered`: count of registered users
- `cnt`: count of total rental bikes including both casual and registered

7.2 Wrangling

```r
#===============================#
#       Wrangling Function     #
#================================#
# Initial wrangling wrapper function
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)
}
#=========================#
#        Subsetting      #
#=========================#
# Filter for the 2011 data
in_2011 <- function(data){ data[(data$dteday >= "2011-01-01" & data$dteday <= "2011-12-31"),] }
# Filter for the 2012 data
in_2012 <- function(data){ data[(data$dteday >= "2012-01-01" & data$dteday <= "2012-12-31"),] }
#======================#
#       Wrangling      #
#======================#
# Import and wrangle data
bike_day <- read.csv("bike-day.csv", header = T)
# Use the wrangling "wrapper" function to clean the data (fxn sourced from wrangle.R)
data <- wrangle_init(bike_day)
# Partition the 2011/2012 data
data2011 <- in_2011(data); data2012 <- in_2012(data)
```

7.3 Modeling

7.3.1 Model Building

```r
#=============================#
#       Helper Function       #
#=============================#
# 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)
}
#=======================================#
#       Model "Builder" Functions       #
#=======================================#
# Given a vector of variable names, return a linear model fitting cnt
lm.tot <- function(varset, train_data = data2011){
  lm(formula = .parseFormula(predictors = varset, response = "cnt"), data = train_data)
}
# Given a vector of variable names, return a linear model fitting cas
lm.cas <- function(varset, train_data = data2011){
  lm(formula = .parseFormula(predictors = varset, response = "casual"), data = train_data)
}
# Given a vector of variable names, return a linear model fitting reg
lm.reg <- function(varset, train_data = data2011){
  lm(formula = .parseFormula(predictors = varset, response = "registered"), data = train_data)
}
```

7.3.2 Model Formulas

```r
# Initial models
mod.cas.1.final <- lm.cas(c("workingday", "weathersit", "atemp"))
mod.reg.1.final <- lm.reg(c("workingday", "weathersit", "atemp", "I(atemp^2)"))
mod.tot.1.final <- lm.tot(c("workingday", "weathersit", "atemp", "I(atemp^2)"))
# Final models
mod.cas.2.final <- lm.cas(c("holiday", "season:weathersit", "season:workingday:atemp"))
mod.reg.2.final <- lm.reg(c("holiday", "season:weathersit",
                            "season:workingday:atemp", "season:workingday:I(atemp^2)"))
```

7.3.3 Growth-Adjusting Predictions

```
# Select some arbitrary model
model_example <- mod.cas.1.final
# Select a parameter estimate for g
g_hat <- 1.61
# Predictions without growth adjustment
preds <- predict(model_example, data2012)
# Predictions after growth adjustment
preds_adj <- preds * g_hat
```

7.4 Diagonostic Analysis

```
residual_QQ <- function(model, title_str = ""){
  # Plot parameters
  fsize0 <- 13; fsize1 <- 13; fsize2 <- 13
  # Plot
  df_res <- data.frame(ep = model$residuals)
  df_res %>%
    ggplot(aes(sample = ep)) +
    geom_qq(alpha = 0.8, color = "palegreen3") +
    stat_qq_line(color = "seagreen") +
    labs(x = "Theoretical Quantile", y = "Sample Quantiles",
         title = paste("Normal Q-Q Plot", title_str, sep = ""))
}
residual_fitted <- function(model, title_str = ""){
  # Plot parameters
  fsize0 <- 13; fsize1 <- 13; fsize2 <- 13
  # Plot
  df_res <- data.frame(fitted_value = model$fitted.values, residual = model$residuals)
  df_res %>%
    ggplot(aes(x = fitted_value, y = residual)) +
    geom_point(alpha = 0.7, color = "palegreen3") +
    stat_smooth(se = T, color = "forestgreen") +
    labs(x = "Fitted Value", y = "Residual",
         title = paste("Fitted vs Residual",title_str, sep = ""))
}
residual_histogram <- function(model, title_str = ""){
  # Plot parameters
  fsize0 <- 13; fsize1 <- 13; fsize2 <- 13
  epsilons <- unname(model$residuals)
  df_res <- data.frame(epsilon = epsilons)
  df_res %>%
    ggplot(aes(x = epsilon)) +
    geom_histogram(fill = "forestgreen", bins = 60) +
    labs(x = "Residual", title = paste("Residual Distribution",title_str, sep = ""))
}
residual_plots <- function(model, title_str){
  p1 <- model %>% residual_QQ(title_str)
  p2 <- model %>% residual_histogram(title_str)
  p3 <- model %>% residual_fitted(title_str)
  (p1 + p2 + p3)
}
```

7.5 Validation and Problemshooting

```
##########################################################################
#                               LOOCV
##########################################################################

get_loocv_rmse = function(model) {
  loocv_rmse <- sqrt(mean((resid(model) / (1 - hatvalues(model))) ^ 2))
  # Round the numbers
  digits <- 2
  loocv_rmse <- round(loocv_rmse, digits)
  # Return the dataframe row
  return(data.frame('CV_rmse' = loocv_rmse))
}

get_loocv_rmse_tot = function(cas, reg) {
  res_cas <- resid(cas) / (1 - hatvalues(cas))
  res_reg <- resid(reg) / (1 - hatvalues(reg))
  loocv_rmse <- sqrt(mean((res_cas + res_reg) ^ 2))
  # Round the numbers
  digits <- 2
  loocv_rmse <- round(loocv_rmse, digits)
  # Return the dataframe row
  return(data.frame('CV_rmse' = loocv_rmse))
}

##########################################################################
#                       Model Validation Functions
##########################################################################

get_rmse <- function(y, y_hat, name='none'){
  rmse <- sqrt(mean((y - y_hat)^2, na.rm = TRUE))
  normalized_rmse <- sqrt(mean(((y - y_hat)^2), na.rm = TRUE)) / sd(y)
  percentage_error <- mean(abs(y - y_hat) / y, na.rm = TRUE) * 100
  # Round the numbers
  digits <- 2
  rmse <- round(rmse, digits)
  normalized_rmse <- round(normalized_rmse, digits)
  percentage_error <- round(percentage_error, digits)
  # Return the dataframe row
  return(data.frame('name' = name, 'rmse' = rmse,
                    'nrmse' = normalized_rmse, 'prc_err' = percentage_error))
}

get_mod_eval_2011 <- function(cas, reg){
  ret1 <- rbind(get_rmse(data2011$casual,     predict(cas, data2011), '2011 cas'),
              get_rmse(data2011$registered, predict(reg, data2011), '2011 reg'),
              get_rmse(data2011$cnt,
                       predict(cas, data2011) + predict(reg, data2011), '2011 tot'))
  ret2 <- rbind(get_loocv_rmse(cas),
              get_loocv_rmse(reg),
              get_loocv_rmse_tot(cas, reg))
  return(cbind(ret1, ret2))
}
```

```r
get_mod_eval <- function(cas, reg, data2011, data2012, scale_2012 = TRUE, include_2011 = F){
  if(scale_2012){G_FACTOR <- 0.608} else{G_FACTOR <- 1}
  if(include_2011){
    return(rbind(
      get_rmse(data2011$casual,     predict(cas, data2011), '2011 cas'),
      get_rmse(data2011$registered, predict(reg, data2011), '2011 reg'),
      get_rmse(data2011$cnt,        predict(cas, data2011) + predict(reg, data2011), '2011 tot'),
      get_rmse(data2012$casual,     predict(cas, data2012)/G_FACTOR, '2012 cas'),
      get_rmse(data2012$registered, predict(reg, data2012)/G_FACTOR, '2012 reg'),
      get_rmse(data2012$cnt,
               predict(cas, data2012)/G_FACTOR + predict(reg, data2012)/G_FACTOR, '2012 tot')))
    }
  return(rbind(
    get_rmse(data2012$casual,     predict(cas, data2012)/G_FACTOR, '2012 cas'),
    get_rmse(data2012$registered, predict(reg, data2012)/G_FACTOR, '2012 reg'),
    get_rmse(data2012$cnt,
             predict(cas, data2012)/G_FACTOR + predict(reg, data2012)/G_FACTOR, '2012 tot')))
  }
get_mod_eval_tot_2011 <-function(tot){
  ret1 <- rbind(get_rmse(data2011$cnt, predict(tot, data2011), '2011 tot'))
  ret2 <- rbind(get_loocv_rmse(tot))
  return(cbind(ret1, ret2))
}
get_mod_eval_tot <-function(tot, data2011, data2012, scale_2012 = TRUE){
  if(scale_2012){G_FACTOR <- 0.608} else{G_FACTOR <- 1}
    return(rbind(get_rmse(data2011$cnt, predict(tot, data2011), '2011 tot'),
                 get_rmse(data2012$cnt, predict(tot, data2012)/G_FACTOR, '2012 tot'),
                 get_loocv_rmse(tot)))
}
```

7.6 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
  .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)
}
```

```r
#==============================================================================#
#==============================================================================#

#==========================#
#      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)
}


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

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

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")
}
```

```r
#============================================================================#
#============================================================================#

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

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,]
  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")
}
```

```r
#------------------------------------------------------------------------------#
#==============================================================================#
#                           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()

#=======================#
#    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)
```