Capital Bikeshare 2011-2012: an Analysis, Visualization, and Prediction

Samuel Joslin

12/15/2018

The Capital Bikeshare provides the means for renting bicycicles through a number of automated kiosks throughout the metro DC area. People are able to rent, use, and return bicycles at different locations at their convience.

I plan to explore data from the Washington DC Capital Bikeshare through multiple visualizations and estimators. Ultimately, I create a predictive model (Stochasitic Gradient Boosting and Linear Regression) to predict the number of bikes that will be rented in the future.

```
library(dplyr, quietly = T)
library(ggplot2, quietly = T)
library(caret, quietly = T)
library(lubridate, quietly = T)
library(sqldf, quietly = T)
library(Metrics, quietly = T)


#Read in the bikesharing data set

train <- read.csv("train.csv", header = T,
                  stringsAsFactors = F)
test <- read.csv("test.csv", header = T,
                  stringsAsFactors = F)
train %>% str
```

```
## 'data.frame':    10886 obs. of  12 variables:
##  $ datetime  : chr  "2011-01-01 00:00:00" "2011-01-01 01:00:00" "2011-01-01 02:00:00" "2011-01-01 03
##  $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
##  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
##  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
##  $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
##  $ windspeed : num  0 0 0 0 0 ...
##  $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
##  $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
##  $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
```

Understanding the data The "train" data.frame, as shown above, has columns that can be interpreted in this way:

datetime - hourly date + timestamp

season - 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday - whether the day is considered a holiday (0 for non-holiday, 1 for holiday)

workingday - whether the day is neither a weekend nor holiday weather - 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 - temperature in Celsius

atemp - "feels like" temperature in Celsius

humidity - relative humidity

windspeed - wind speed

casual - number of non-registered user rentals initiated

registered - number of registered user rentals initiated

count - number of total rentals

The "test" data.frame has the same coulmn data as "train", except without "casual", "registered", and "count"

Putting the data in a usable form My goal in the section is to put the data into a useable form such that my analysis portion runs smoothly and clearly. Fortunately, the data has no missing values and clearly labeled. However, I would like to sepretately add an "hour and"day of the week" coulmn and to transform the "temp" from Celsius to Ferenheit. In addition, I will turn four of the integer coulmns into factors for the model prediciton section, and remove the "causal" and "registered" coulmns becuse "count" = "casual" + "registered". Since we have the two data frames, "train" and "test", I generalize this process in two functions.

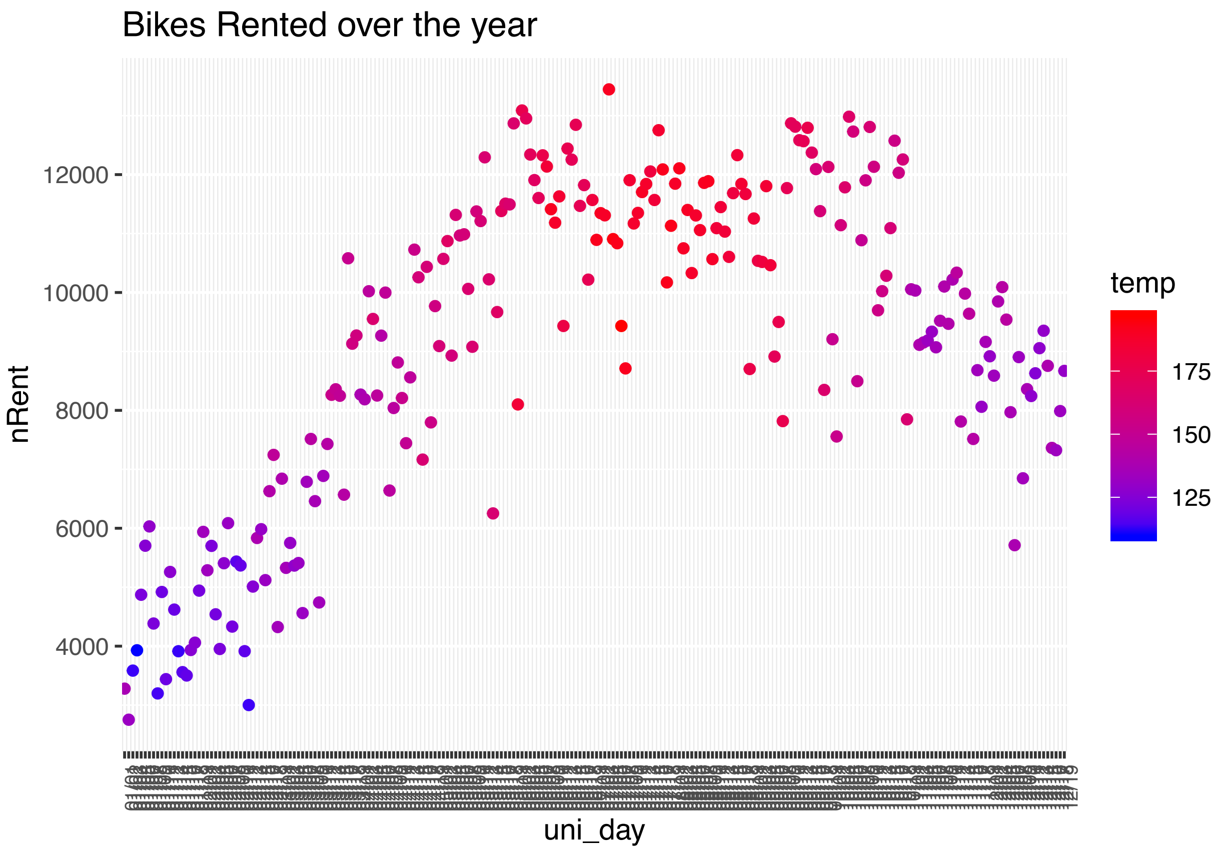
```
##### convert celcius to ferenheit #####
cTOf <- function(int){
  fer <- rep(NA,length(int))
  for (i in 1:length(int)){
    ci <- int[i]
    cel <- ci*(9/5)
    fer[i] <- cel+32
  }
  return(fer)
}

clean_data <- function(data_frame){

  #Create a hour coulmn
  data_frame$hour <- substr(data_frame$datetime, 12, 20) %>% as.factor

  #create day of week column
  data_frame$weekday <- data_frame$datetime %>% as.Date %>% weekdays %>% as.factor

  #turn temps in to ferenheit
  data_frame$temp <- cTOf(data_frame$temp)

  data_frame$season <- as.factor(data_frame$season)
  data_frame$holiday <- as.factor(data_frame$holiday)
  data_frame$workingday <- as.factor(data_frame$workingday)
  data_frame$weather <- as.factor(data_frame$weather)
  data_frame$casual <- NULL
  data_frame$registered <- NULL
```

```r
  return(data_frame)
}

train <- clean_data(train)
test <- clean_data(test)
```

Understanding the data through visualizations and estimators

I seek to answer the question: "How does bikeshare rentals vary over the course of the year?" I approach this question by plotting the "month/day" days on the x-axis and an aggregate count on the y-axis for all given years. Additionally, I highlight the temperature on the given day to show the relationship between the number bikes rented on the day and the temperature.

```r
plot_year_nRent <- function(unfactored){
  rows <- nrow(unfactored)
  mon_day <- rep(NA,rows)
  for(i in 1:rows){
    cd <- as.character(unfactored$datetime[i])
    month <- strsplit(cd,"\\- |\\-| ")[[1]][2]
    day <- strsplit(cd,"\\- |\\-| ")[[1]][3]
    mon_day[i] <- paste(month,day, sep = "/")
  }


  unfactored$mon_day <- mon_day

  cl <- length(unique(unfactored$mon_day))
  nRent <- rep(NA, cl)
  avg_temp <- rep(NA, cl)
  for (i in 1:cl){
    uni_day <-unique(unfactored$mon_day)
    cd <- uni_day[i]
    df_filt <- dplyr::filter(unfactored, mon_day == cd)
    nRent[i] <- sum(df_filt$count)
    avg_temp[i] <- df_filt$temp %>% as.numeric %>% mean
  }

  temp <- cTOf(avg_temp)
  uni_day <- unique(unfactored$mon_day)
  dayCount <- data.frame(uni_day,nRent,temp)

  p <- ggplot(aes(uni_day, nRent, color = temp), data = dayCount)
  p <- p + geom_point() + theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 7))+scale_colo
    ggtitle("Bikes Rented over the year")
  return (print(p))
}


plot_year_nRent(train)
```



The left most x-axis tick represents 01/01 and the right most is 12/31. It is no suprise that more bikes are rented in the summer months than the winter, but something that I found interesting is that it appears that there is more variability in the count on the warmer days than the colder days. To further investigate this idea, I plan, later on, the asses the standard deviation by season. I want to assess the validity of the statement "There is less variablity in the count when the temperature cold than when the temperature is hot."

Below is a function that accepts a data frame and season and returns the standard deviation of that season's count.

```
sd_seasons <- function(traindf, season){

  if(season == "Spring"){
    season2 <- 1
  }
  if(season == "Summer"){
    season2 <- 2
  }
  if(season == "Fall"){
    season2 <- 3
  }
  if(season == "Winter"){
    season2 <- 4
  }

  df_filt <- filter(traindf, season == season2)
  sd <- df_filt$count %>% sd
  return(sd)
```

```r
}

sd_seasons(train,"Summer")%>% print
```

```
## [1] 192.0078
```

```r
sd_seasons(train,"Fall")%>% print
```

```
## [1] 197.151
```

```r
sd_seasons(train,"Winter")%>% print
```

```
## [1] 177.6224
```

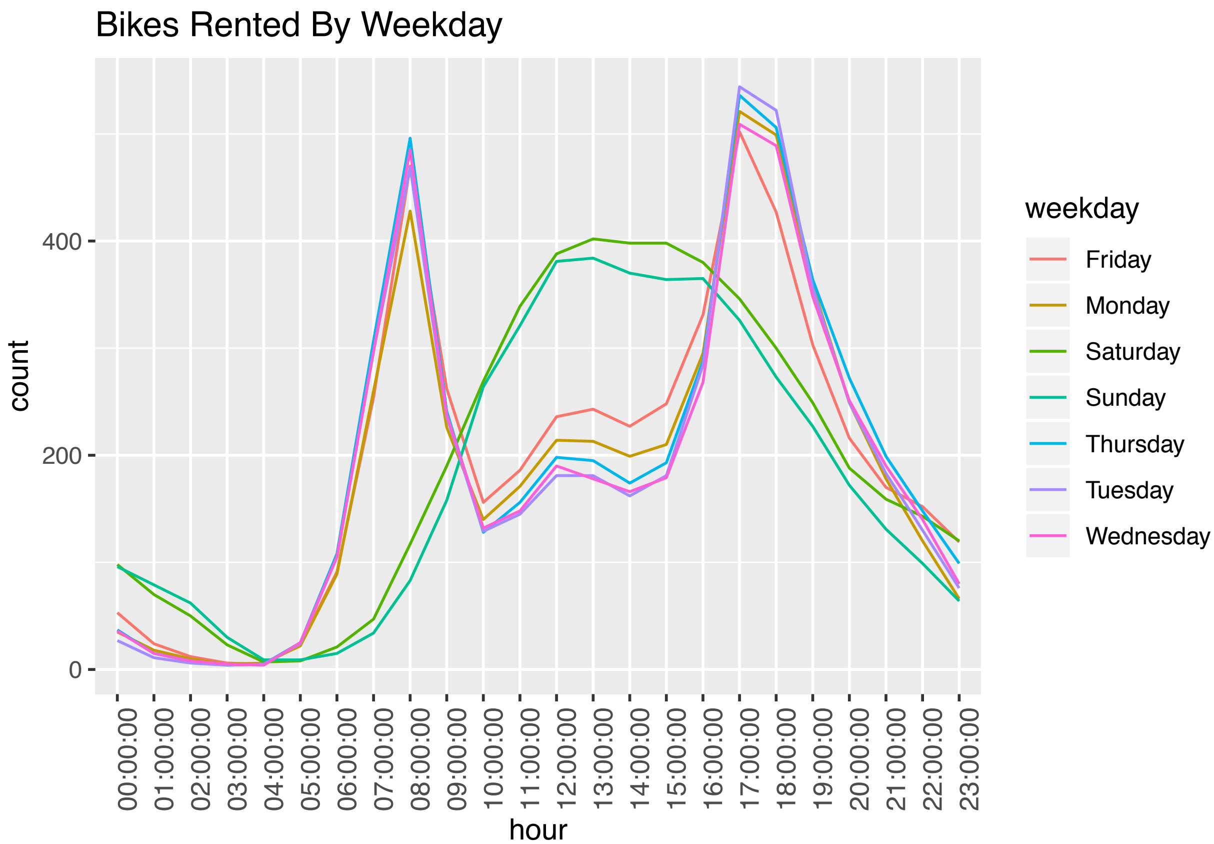
```r
sd_seasons(train,"Spring")%>% print
```

```
## [1] 125.274
```

It appears that there is less variation in the spring than in the other months, which suggests that people more consistently rent bikes in the spring.

I would also like to investigate the question: "How many bikes are rented per hour over on different days of the week". I will visualize this question by plotting the "hours" on the x-axis and the "count" on the y-axis and have a line represent the different days of the week.

```r
plot_Days_Count <- function(data_frame){
  day_hour <- sqldf::sqldf('select weekday, hour, avg(count) as count from data_frame group by weekday,

  p <-ggplot(train, aes(x=hour, y=count, color=weekday))+
    geom_line(data =day_hour, aes(group = weekday))+
    ggtitle("Bikes Rented By Weekday")+ theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 
  return(print(p))
}

plot_Days_Count(train)
```



It appears that on the weekdays people mostly use the bikes in the morning and in the afternoon, probably in accordace with a work schedule. On the weekends it apears that people steady increase the useage, peaking the afternoon.

Regresssion and count prediction

Given the relationships we have seen in the visualizations above, it would be interesting if we could provide some sort of prediction through a model on how many bikes are rented given certain time and weather conditions. I plan to create two models to assist the predict process, Stochastic Gradient Boosting and Linear Regression. Stochastic Gradient Boosting is a decision tree based model that consists of an ensemble of weak prediction to form one strong prediction model. Linear regression provides a simple scalar relationship between the dependent and independent variables. I will evaluate the model's accuracy using the Root Mean Squared Log Error.

```
#Partition the training and validation
inTrain <- caret::createDataPartition(y=train$count, p=0.75, list=F)
training <- train[inTrain, ]
validation <- train[-inTrain,]

Grid <-  expand.grid( n.trees = 500, interaction.depth = 10, shrinkage = 0.05,
                        n.minobsinnode = 10)

fitControl <- trainControl(
  method = "repeatedcv",
  number = 6,
  repeats = 6)

regress <- count ~ season + holiday + workingday + weather + temp + humidity + windspeed + hour + weekda
```

```r
t_out_gbm <- caret::train(regress, data=training,
                         tuneGrid = Grid, trControl = fitControl,
                         method = 'gbm', maximize = F)
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 8: weather4 has no variation.
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1    31059.6147             nan     0.0500 1258.4417
##      2    29805.9755             nan     0.0500 1235.0257
##      3    28709.4415             nan     0.0500 1031.9113
##      4    27642.3276             nan     0.0500 1129.9421
##      5    26676.8505             nan     0.0500  939.3728
##      6    25750.1999             nan     0.0500  914.9669
##      7    24932.0435             nan     0.0500  834.2625
##      8    24143.5202             nan     0.0500  763.3250
##      9    23370.2015             nan     0.0500  770.8170
##     10    22681.2949             nan     0.0500  690.3468
##     20    17307.2113             nan     0.0500  396.1854
##     40    11783.6438             nan     0.0500  198.9375
##     60     9304.2288             nan     0.0500   41.4277
##     80     8001.9804             nan     0.0500   37.2588
##    100     7173.1600             nan     0.0500   22.3287
##    120     6661.6692             nan     0.0500    8.8488
##    140     6268.7995             nan     0.0500    7.3683
##    160     5967.2323             nan     0.0500    3.8765
##    180     5758.4124             nan     0.0500   -4.4834
##    200     5563.3825             nan     0.0500    3.2273
##    220     5409.8655             nan     0.0500   -0.4730
##    240     5263.3595             nan     0.0500   -4.0573
##    260     5130.6885             nan     0.0500   -1.0004
##    280     5001.6212             nan     0.0500   -0.8368
##    300     4895.0530             nan     0.0500   -5.7753
##    320     4791.1030             nan     0.0500   -0.2971
##    340     4684.2769             nan     0.0500   -2.2857
##    360     4589.4750             nan     0.0500   -2.0791
##    380     4511.2551             nan     0.0500   -2.5979
##    400     4434.5989             nan     0.0500    1.3360
##    420     4356.4956             nan     0.0500   -0.7038
##    440     4286.2452             nan     0.0500   -0.4053
##    460     4212.5553             nan     0.0500   -2.6373
##    480     4149.0213             nan     0.0500   -1.5456
##    500     4095.5028             nan     0.0500   -3.0213
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1    31153.9895             nan     0.0500 1331.0569
##      2    29951.3644             nan     0.0500 1185.3219
##      3    28811.4353             nan     0.0500 1123.6020
##      4    27737.3592             nan     0.0500 1056.8432
##      5    26802.0427             nan     0.0500  942.8468
##      6    25932.0516             nan     0.0500  875.6669
##      7    24971.5997             nan     0.0500  843.6513
##      8    24209.5624             nan     0.0500  731.1394
##      9    23475.3540             nan     0.0500  678.4202
```

```r
t_out_gbm %>% print
```

```
## Stochastic Gradient Boosting
##
## 8166 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (6 fold, repeated 6 times)
## Summary of sample sizes: 6805, 6805, 6806, 6804, 6806, 6804, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   74.83737  0.8293206  53.74274
##
## Tuning parameter 'n.trees' was held constant at a value of 500
##  10
## Tuning parameter 'shrinkage' was held constant at a value of
##  0.05
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
```

```r
t_out_lm %>% print
```

```
## Linear Regression
##
## 8166 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (6 fold, repeated 6 times)
## Summary of sample sizes: 6805, 6804, 6805, 6804, 6806, 6806, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   110.3013  0.6293517  79.67259
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

A prediction is only valuable if it has a low error. Below I wrote a function that will show the Root Mean Squared Log Error.

```r
prediction_error <- function(model, validation){
  pred1 <- predict(model, validation)
  pred1[pred1<0] <- 0
  error <- Metrics::rmsle(validation$count,pred1)
  return(error)
}

prediction_error(t_out_gbm, validation)  %>% print
```

```
## [1] 0.756579
```

```r
prediction_error(t_out_lm, validation) %>% print
```

```
## [1] 1.045073
```

As expected the Stochastic Gradient Boosting model has a lower error than the Linear Regression. So, I will use the Stochastic Gradient Boosting for the prediction. Below is a fucntion that makes predictions.

```r
makePrediction <- function (model, test) {
  pred <- predict(model, test)
  pred <- round(pred)
  pred[pred<0] <- 0
  df <- data.frame(test$datetime, c(count=pred))
  names(df) <- c("datetime", "count")
  return(df)
}

makePrediction(t_out_gbm, test) %>% head %>% print
```

```
##                   datetime count
## count1 2011-01-20 00:00:00    52
## count2 2011-01-20 01:00:00     4
## count3 2011-01-20 02:00:00     0
## count4 2011-01-20 03:00:00     0
## count5 2011-01-20 04:00:00     0
## count6 2011-01-20 05:00:00     0
```

In the future, I would like to use higher number of repeated CV to improve the model's accuracy as well as comparing these models to other methods.

As an end note, I must cite Kaggle for providing the data set, the Caret Package introduction https://topepo.github.io/caret/index.html, and Stackoverflow for providing me with information on tuning models and how to use new r functions and packages.