STAT 652: Flight Delay Code Only

Vincent Chiu

12/05/2019

Contents

1 to knit or run, please put fltrain.csv.gz and fltest.csv.gz in the same folder as this file.

2 Please note that because some of the code chunks have been set to not evaluate, some tables will be blank in the document.

3 Code

The R version used was R version 3.6.1 (2019-07-05). The following packages were used:

- tidyverse
- nycflights13
- Hmisc
- lubridate
- imputeMissings
- dplyr
- gbm

Time to Run: Takes 2+ hours to knit the code, mostly because GBM takes a long time to train.

Table 1: A table of the first few rows of the nycflights13 data.

year.x	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	f
2013	11	7	600	600	0	826	825	1	WN	
2013	10	30	1252	1250	2	1356	1400	-4	AA	
2013	12	18	1723	1715	8	2008	2020	-12	DL	
2013	11	20	2029	2030	-1	2141	2205	-24	WN	
2013	10	21	1620	1625	-5	1818	1831	-13	DL	
2013	11	7	852	900	-8	1139	1157	-18	B6	

3.1 Preparing the programming environment

3.1.1 Loading Libraries

```
library(tidyverse)
```

3.2 Data Preprocessing

3.2.1 Loading the data

```
library(nycflights13)
library(Hmisc)
set.seed(42)
original_data <- read_csv("fltrain.csv.gz")
DF <- original_data
```

3.2.2 turning all columns with datatype characters to factors.

```
DF[sapply(DF, is.character)] <- lapply(DF[sapply(DF, is.character)],
                                       as.factor)
DF$flight <- as.factor(DF$flight)
```

3.2.3 parsing times from strings and calculating scheduled air time

```
library(lubridate)
DF$sched_arr_time_posix <- as.POSIXct(str_pad(as.character(DF$sched_arr_time), 4, pad="0"),format="%H%M
DF$sched_arr_time_hour <- hour(DF$sched_arr_time_posix)
DF$sched_arr_time_minute <- minute(DF$sched_arr_time_posix)

#num minute is number of minutes since start of day for scheduled arrival time
DF$sched_arr_time_num_minute <- 60*DF$sched_arr_time_hour + DF$sched_arr_time_minute

DF$sched_dep_time_posix <- as.POSIXct(str_pad(as.character(DF$sched_dep_time),4 , pad="0"),format="%H%M
DF$sched_dep_time_hour <- hour(DF$sched_dep_time_posix)
DF$sched_dep_time_minute <- minute(DF$sched_dep_time_posix)
```

```r
#num minute is number of minutes since start of day for scheduled depival time
DF$sched_dep_time_num_minute <- 60*DF$sched_dep_time_hour + DF$sched_dep_time_minute
```

```r
select(original_data, time_hour, sched_dep_time, sched_arr_time, tz, tzone)
select(DF, sched_arr_time, sched_arr_time_hour)
```

```r
DF$sched_air_time <- DF$sched_arr_time_posix - DF$sched_dep_time_posix
drops <- c('sched_arr_time_posix', 'sched_arr_time_hour', 'sched_dep_time_posix', 'sched_dep_time_hour'
DF <- DF[ , !(names(DF) %in% drops)]
```

```r
drops <- c("dep_time", "arr_time", "air_time", "arr_delay", "year.x", 'tailnum')
DF <- DF[ , !(names(DF) %in% drops)]
```

```r
## Remove columns with more than 50% NA
DF <- DF[, -which(colMeans(is.na(DF)) > 0.5)]
```

```r
DF$sched_air_time <- as.numeric(DF$sched_air_time)
```

3.2.4 Imputing Variables

```r
library(imputeMissings)
impute_model <- imputeMissings::compute(DF, method="median/mode")
DF <- impute(DF, object=impute_model, flag=TRUE)
DF <- DF[!duplicated(as.list(DF))]  #remove all redundant flag columns that are identical to each other
```

```r
numeric_only_df <- dplyr::select_if(DF, is.numeric)
library(corrplot)
```

3.3 Feature Scaling

```r
dep_delay_vec <- DF$dep_delay
DF$dep_delay <- NULL
head(DF)

library(dplyr)
DF <- DF %>% mutate_if(is.numeric, scale)
head(DF)
DF$dep_delay <- dep_delay_vec
```

3.4 Exploratory Data Analysis

```r
numeric_DF <- dplyr::select_if(DF, is.numeric) %>%  scale()
```

```r
prcomp_res <- prcomp(numeric_DF)
sdev <- prcomp_res$sdev
sdev
```

3.4.1 all four components at same time

proportion of variance explained by each component

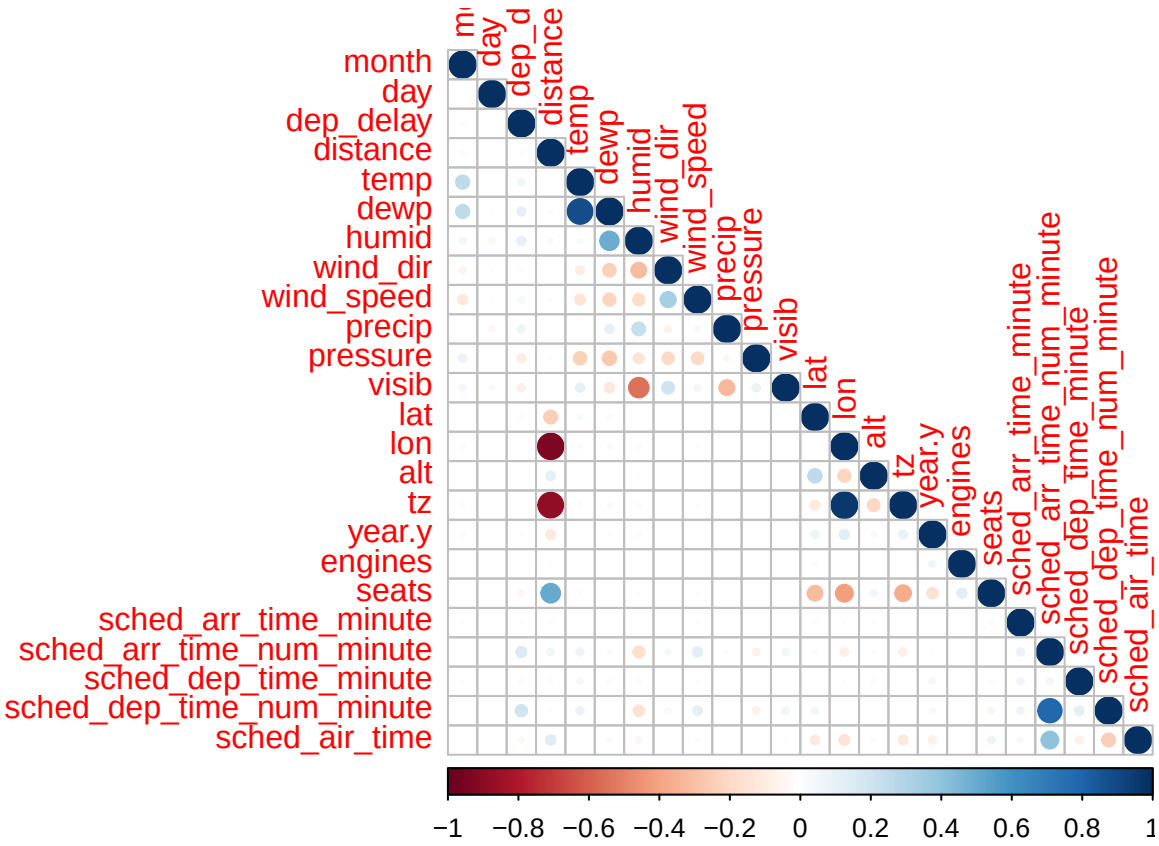

Figure 1: grid depicting correlation amongst all numerical variables

Table 2: proportion of variance explained by each principal component

	x
PC1	0.1352687
PC2	0.1061946
PC3	0.0862791
PC4	0.0688860
PC5	0.0605903
PC6	0.0582950

Table 3: coefficients for each variable on each principal component

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
lon	0.5342621	0.0082280	-0.0118128	-0.0071812	-0.1412554	0.0211945	0.0141719	-0.0238850
distance	-0.5303216	-0.0136727	0.0420910	0.0006060	-0.0429416	-0.0059427	-0.1227356	0.0437053
tz	0.5152321	0.0039180	-0.0038194	-0.0093898	-0.2074941	0.0304900	0.0014204	-0.0176097
seats	-0.3267975	-0.0218067	0.0923978	0.0075812	-0.2991549	-0.0095818	-0.2065632	-0.1443975
alt	-0.1291185	0.0012741	-0.0224310	0.0179065	0.4660781	-0.0623074	0.2417777	-0.0827608
sched_air_time	-0.1269319	-0.0592623	-0.0467563	-0.0535120	-0.2991127	0.1516353	0.6962022	-0.1792463

```
pve <- colSums(prcomp_res$x^2)/sum(numeric_DF^2)
```

```
rotation <- as.data.frame(prcomp_res$rotation)
rotation[order(-abs(rotation$PC1)),]
```

```
pca_rotation <- head(rotation[order(-abs(rotation$PC1)),])
```

3.4.2 take out extreme departure delays

```
DF<-DF[DF$dep_delay < 30,]
```

```
set.seed(42)
DF$flight <- NULL
train_index <- sample(1:nrow(DF),size=2*nrow(DF)/3,replace=FALSE)
train_df <- DF[train_index,]
test_df <- DF[-train_index,]
```

save entire dataset for later

```
train_and_validation_df <- DF
```

3.5 predicting 0

```
benchmark_df <- data.frame(model_description = character(), rmse = numeric(), stringsAsFactors = FALSE)
rmse = mean((test_df$dep_delay-0)^2) %>% sqrt()
model_description = "predicting 0"
benchmark_df <- rbind(benchmark_df, data.frame(model_description = model_description, rmse=rmse))
```

Table 4: RMSE on the validation set, benchmark comparing all the models that we tried

model_description	rmse
predicting 0	8.305710
predicting the mean	8.299767
predicting the median	8.469257
linear regression	7.989994
GBM (tuned)	7.890710

3.6 predicting the mean

```
rmse = mean((test_df$dep_delay-mean(train_df$dep_delay))^2)%>% sqrt()
rmse
model_description <- 'predicting the mean'
benchmark_df <- rbind(benchmark_df, data.frame(model_description = model_description, rmse=rmse))
benchmark_df
```

3.7 predicting the median

```
rmse = mean((test_df$dep_delay-median(train_df$dep_delay))^2)%>% sqrt()
rmse
model_description <- 'predicting the median'
benchmark_df <- rbind(benchmark_df, data.frame(model_description = model_description, rmse=rmse))
```

3.8 linear regression with dest

```
model <- lm(dep_delay ~ ., data=train_df)
model_without_dest <-  lm(dep_delay ~ .-dest, data=train_df)
anova(model, model_without_dest)
summary <- round(summary(model)$coefficients,6)
sorteddf <- summary[order(summary[,ncol(summary)]),]
head(sorteddf)
```

```
lm_test_df <- test_df

in_test_but_not_train <- setdiff(unique(lm_test_df$model), unique(train_df$model))
lm_test_df <- lm_test_df[ !lm_test_df$model %in% in_test_but_not_train, ]

in_test_but_not_train <- setdiff(unique(lm_test_df$dest), unique(train_df$dest))
lm_test_df <- lm_test_df[ !lm_test_df$dest %in% in_test_but_not_train, ]

preds = predict(model, newdata=lm_test_df)
rmse = sqrt(mean((lm_test_df$dep_delay - preds)^2))
rmse
model_description <- 'linear regression'
benchmark_df <- rbind(benchmark_df, data.frame(model_description = model_description, rmse=rmse))
```

```
benchmark_df <- rbind(benchmark_df, data.frame(model_description = 'GBM (tuned)', rmse=7.89071))
write_csv(benchmark_df, 'benchmark_df.csv')
```

3.9 GBM

```
library(gbm)

train_gbm <- function(filename){
num_trees <- 2^13
set.seed(42)
model <- gbm(dep_delay ~ ., data=train_df,
              n.trees=num_trees, shrinkage=0.01) # default shrinkage = 0.1
preds = predict(model, newdata=test_df, n.trees=num_trees)
rmse = sqrt(mean((test_df$dep_delay - preds)^2))
summary(model)
saveRDS(model, filename)
return(model)
}

destfile <- "gbm_shrinkage_0point01_ntrees_8192_v2.rds"
if (!file.exists(destfile)) {
   train_gbm(destfile)
 }
model <- readRDS(destfile)
```

```
gbmsummary <- summary(model)
```

Chunk below takes 2+ hours to run on an average laptop.

```
library(gbm)
train_gbm_rmse_vs_num_trees <- function(shrinkage, rerun, num_trees_2_exp=16) {
  #rerun <- TRUE
  set.seed(42)
  x <- 2 ^ seq(5, num_trees_2_exp, by = 1)
  rmse_vec <- numeric(length(x))
  count <- 1
  filename_vec1 <- c("models2/gbm_shrinkage_")
  filename_vec1 <-
    append(filename_vec1, gsub('\\.', 'point', toString(shrinkage)))
  filename_vec1 <- append(filename_vec1, "_ntrees_")
  filename_prefix1 <- paste(filename_vec1, collapse = '')

  for (val in x) {
    filename_vec2 <- append(filename_prefix1, val)
    filename_vec2 <- append(filename_vec2, "_v2.rds")
    filename <- paste(filename_vec2, collapse = '')
    if (!file.exists(filename) | rerun) {
      hboost <- gbm(
        dep_delay ~ .,
        data = train_df,
        n.trees = val,
        shrinkage = shrinkage
      ) # default shrinkage = 0.1
      saveRDS(hboost, filename)
      hboost <- readRDS(filename)
    } else {
      print("reading saved model")
```

```
      hboost <- readRDS(filename)
    }

    preds = predict(hboost, n.trees = val, newdata = test_df)
    mse = mean((test_df$dep_delay - preds) ^ 2)
    rmse <- sqrt(mse)
    rmse_vec[count] <- rmse
    print(val)
    print(rmse)
    count = count + 1
  }

  filename_vec1 <- c("performance2/gbm_shrinkage_")
  filename_vec1 <-
    append(filename_vec1, gsub('\\.', 'point', toString(shrinkage)))
  #filename_vec1 <- append(filename_vec1, "_ntrees_")
  #filename_prefix1 <- paste(filename_vec1, collapse = '')
  #filename_vec2 <- append(filename_prefix1, x[length(x)])

  #summary filename
  filename_vec_summary <- append(filename_vec1, "_v2_summary.csv")
  filename_summary <- paste(filename_vec_summary, collapse = '')

  #rmse_vs_num_trees filename
  filename_vec_rmse_vs_num_trees <- append(filename_vec1, "_v2_rmse_vs_num_trees.csv")
  filename_rmse_vs_num_trees <- paste(filename_vec_rmse_vs_num_trees, collapse = '')

  summary <- summary(hboost)
  write.csv(summary, filename_summary)

  num_trees_vs_rmse <-
    data.frame("num_trees" = x, "rmse" = rmse_vec)
  write.csv(
    num_trees_vs_rmse, filename_rmse_vs_num_trees
  )
}

train_gbm_rmse_vs_num_trees(shrinkage = 0.01, rerun = FALSE, num_trees_2_exp=18)
train_gbm_rmse_vs_num_trees(shrinkage = 0.001, rerun = FALSE, num_trees_2_exp=18)
```

```
shrinkage_0point01_bench <- read.csv('performance2/gbm_shrinkage_0point01_v2_rmse_vs_num_trees.csv')
shrinkage_0point01_bench$X <- NULL
shrinkage_0point01_bench <- shrinkage_0point01_bench %>%
  dplyr::rename(
    "shrinkage_0point01_rmse" = rmse
    )
shrinkage_0point01_bench
```

```
shrinkage_0point001_bench <- read.csv('performance2/gbm_shrinkage_0point001_v2_rmse_vs_num_trees.csv')
shrinkage_0point001_bench$X <- NULL
shrinkage_0point001_bench <- shrinkage_0point001_bench %>%
  dplyr::rename(
    "shrinkage_0point001_rmse" = rmse
    )
```

```
shrinkage_0point001_bench
```

```
gbm_merge_df <- merge(shrinkage_0point01_bench, shrinkage_0point001_bench, all.x = TRUE)
gbm_merge_df
```

Final training on entire dataset to prepare for test set.

```
library(gbm)
num_trees <- 2^13
train_final_gbm <- function(filename, num_trees){
set.seed(42)
model <- gbm(dep_delay ~ ., data=train_and_validation_df,
             n.trees=num_trees, shrinkage=0.01) # default shrinkage = 0.1
saveRDS(model, filename)
return(model)
}

destfile <- "models_ultimate/final_train_on_train_and_validation_gbm_v1.rds"
if (!file.exists(destfile)) {
   train_final_gbm(destfile)
}

final_model <- readRDS(destfile)
final_preds = predict(final_model, newdata=train_and_validation_df, n.trees=num_trees)
final_rmse = sqrt(mean((train_and_validation_df$dep_delay - final_preds)^2))
#summary(final_model)
```

```
rmse
```

3.10 Evaluate Error on Holdout Test Data

```
library(tidyverse)
library(nycflights13)
library(Hmisc)
library(lubridate)
library(imputeMissings)
library(dplyr)


preprocess_data <- function(filepath) {
  set.seed(42)
  original_data <- read_csv(filepath)
  DF <- original_data
  DF[sapply(DF, is.character)] <-
    lapply(DF[sapply(DF, is.character)],
           as.factor)
  DF$flight <- as.factor(DF$flight)
  DF$sched_arr_time_posix <-
    as.POSIXct(str_pad(as.character(DF$sched_arr_time), 4, pad = "0"), format =
                 "%H%M")
  DF$sched_arr_time_hour <- hour(DF$sched_arr_time_posix)
  DF$sched_arr_time_minute <- minute(DF$sched_arr_time_posix)
```

```r
#num minute is number of minutes since start of day for scheduled arrival time
DF$sched_arr_time_num_minute <-
  60 * DF$sched_arr_time_hour + DF$sched_arr_time_minute
DF$sched_dep_time_posix <-
  as.POSIXct(str_pad(as.character(DF$sched_dep_time), 4 , pad = "0"), format =
               "%H%M")
DF$sched_dep_time_hour <- hour(DF$sched_dep_time_posix)
DF$sched_dep_time_minute <- minute(DF$sched_dep_time_posix)

#num minute is number of minutes since start of day for scheduled depival time
DF$sched_dep_time_num_minute <-
  60 * DF$sched_dep_time_hour + DF$sched_dep_time_minute
DF$sched_air_time <-
  DF$sched_arr_time_posix - DF$sched_dep_time_posix
drops <-
  c(
    'sched_arr_time_posix',
    'sched_arr_time_hour',
    'sched_dep_time_posix',
    'sched_dep_time_hour',
    'sched_dep_time',
    'sched_arr_time',
    'hour',
    'time',
    'minute',
    'time_hour',
    "dep_time",
    "arr_time",
    "air_time",
    "arr_delay",
    "year.x",
    'tailnum'
  )
DF <- DF[, !(names(DF) %in% drops)]

## Remove columns with more than 50% NA
DF <- DF[, -which(colMeans(is.na(DF)) > 0.5)]

DF$sched_air_time <- as.numeric(DF$sched_air_time)

# impute
impute_model <- imputeMissings::compute(DF, method = "median/mode")
DF <- impute(DF, object = impute_model, flag = TRUE)
DF <-
  DF[!duplicated(as.list(DF))]  #remove all redundant flag columns that are identical to each other.


# scale all but dep_delay
dep_delay_vec <- DF$dep_delay
DF$dep_delay <- NULL
DF <- DF %>% mutate_if(is.numeric, scale)
DF$dep_delay <- dep_delay_vec
```

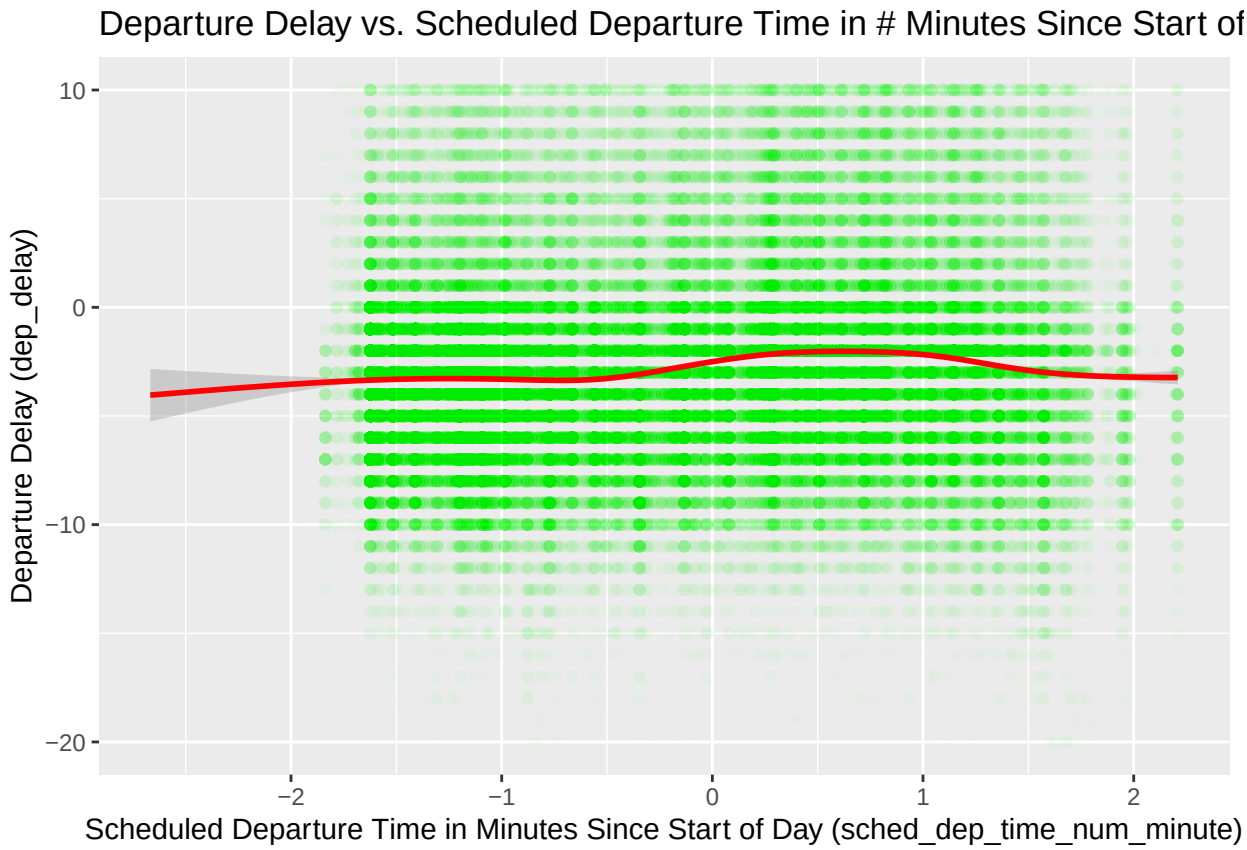

Figure 2: Departure Delay vs. Scheduled Departure Time in # Minutes Since Start of Day with Spline Plotted on Top Depicting Possible Non-Linear Relationship

```
  # exclude dep_delay >= 30
  DF <- DF[DF$dep_delay < 30,]
  DF$flight <- NULL

  return(DF)
}

final_test_df <- preprocess_data('fltest.csv.gz')
```

```
num_trees <- 2^13
set.seed(42)

destfile <- "models_ultimate/final_train_on_train_and_validation_gbm_v1.rds"
if (!file.exists(destfile)) {
   train_final_gbm(destfile)
}

final_model <- readRDS(destfile)
final_preds = predict(final_model, newdata=final_test_df, n.trees=num_trees)
final_rmse = sqrt(mean((final_test_df$dep_delay - final_preds)^2))
#summary(final_model)
print(final_rmse)
```

Table of destination sorted by mean delay

```
dest_ordered_by_mean_delay <- DF %>% group_by(dest) %>%dplyr::summarize(Mean = mean(dep_delay, na.rm=TRU
```

```
var(dest_ordered_by_mean_delay$Mean)
```

```
print("done")
```

4 References