STAT 652: Predicting Flight Delays Project

Vincent Chiu

11/26/2019

Contents

1 Introduction

The goal of this project is to predict the response variable, departure delays for a particular flight given the explanatory variables.
* Please note that you can click on figure and table numbers to jump to that figure or table.

2 Data

The dataset consists of information about all the flights leaving from New York City in 2013. The dataset contains 43 variables in total. The dataset is an amalgamation of several datasets including datasets containing information on weather, the airports, the flights, and the models of airplanes. The training dataset provided to us contains 200,000 observations. Please see table 2 for what the data looks like.

3 Methods

We will now outline the various methods used to clean and perform prediction on the data. We will discuss our techniques for data preprocessing, and cross validation, and the different models that we tried.

3.1 Data Preprocessing

We performed data preprocessing on the nycflights13 dataset. Our data preprocessing steps include the following:

- loading the data from a csv
- setting the random seed for reproducibility of results
- casting all the columns with the character data type into the factor data type
- converting the 'shed_arr_time' and 'sched_dep_time' columns into the POSIX time format so that we could accurately take the difference of them.
- Dropping columns that contain data from after the flights' departure which may leak information about the response variable 'dep_delay'. We drop columns 'dep_time', 'arr_time', 'air_time', and 'arr_delay'.
- We drop column 'year.x' because all the values are 2013
- We also drop 'talinum' because it produces too many dummy variable columns for one hot encoding.
- Dropping columns which consists of over 50% NAs which include the speed column. However, it should be noted that a rule of thumb suggested by Professor McNeney is to drop any columns with over 5% NAs. We use a different threshold for dropping columns leading to us keeping columns such as model instead of dropping it.
- Afterwards, we impute the missing values. However, there are limitations to this approach of imputing the missing values. It is possible, that the missingness of the plane model variable is related to 'dep_delay'. In this scenario, we may be creating an inferior feature set by keeping the variable 'model' and imputing it. For example, say a highly unreliable plane model that frequently causes long delays has a high probability of being labeled as NA and represents the majority of NAs in the dataset. We would not be able to capture the relationship between this plane 'model' and 'dep_delay' if we imputed the model with the mode. To counteract this affect, we imputed NAs for the remaining columns using the 'imputeMissings' library, adding a boolean flag which indicates 0 if the associated value was present and 1 if the associated value was NA. For example, the 'model_flag' for a given row is 1 if the 'model' value was NA for that given row. Hence, no information is lost from our imputation.
- Normalizing the data, to work well with methods like lasso regression.
- Only kept data which had a departure delay of less than 30 minutes late, which reduced the dataset from 200,000 rows to approximately 170,000. This is because we consider extreme delays of over 30 minutes late to be freak accidents which cannot be accurately predicted by the available explanatory variables.

3.2 Exploratory Analysis

3.2.1 Correlations

First, we created a correlation plot for the numeric variables to see if there any correlations between the variables.

We see that there is very little correlation between the response variable dep_delay and any of the other variables. Some of the strongest correlations include the correlation between distance and longitude and time zone and a smaller correlation between distance and latitude. This makes sense as most of the planes are inter US flights from west to east or vice versa, there is not as much distance flown in the north south direction. Please see figure 1 for the correlation plot. (Please click on the figure number to jump to the plot).

3.3 Principal Component Analysis (PCA)

Next we performed Principal Component Analysis (PCA) on only the numeric variables as techniques to perform PCA on mixed datasets (numerical and categorical) were not covered in class. When looking at the contribution of each variable to the first principal component, we notice that the variables 'lon', 'distance', 'tz', 'seats', 'alt', 'sched_air_time' have the greatest absolute coefficients for the first principal component. The fact that the aforementioned variables have large coefficients in the first principal component suggests that they are highly correlated with each other. The fact that 'dep_delay' has a small coefficient in the first principal component suggests that 'dep_delay' is not highly correlated with any of the above variables.

As expected, it turns out that variables like 'lon', 'distance' and 'tz' are not important for predicting 'dep_delay' according to the GBM model. This maybe be because although variables like 'lon', 'distance' and 'tz' help explain most of the variance in the dataset, they have a weak relationship with 'dep_delay'.

Please see table 3 for the proportion of variance explained by each principal component. Please see table 4 for the variable coefficients for each principal component ordered by the magnitude of the variable coefficient for the first principal component.

3.4 Validation

Initially, we used the most basic validation technique where we have a training dataset and a validation dataset. We reserved 2/3 of the original dataset's observations for the training dataset and 1/3 of the observations for the validation dataset. There is an additional data which would be provided by the professor at a later date which we will use as the holdout test set. We believe that 2/3 of the data gives enough data for the models to train on while 1/3 is enough data for us to get an accurate assessment of the error. After the data preprocessing steps, which included removing all rows where 'dep_delay' was greater than 30 minutes, we were left with 170,645 observations. Of these 170,645 observations, 113,763 was used in the training dataset and 56,882 was used in the validation set. The training dataset and the validation dataset do not have any common observations. k-folds cross validation was not initially used in order to save on compute time as we were initially only exploring the models. k-folds cross validation would increase training time for the models by a factor of k. However, k-folds cross validation would lead to a more stable estimate of holdout test set error.

3.5 Models

We first explored some basic models to establish a baseline performance and compared it to our most sophisticated model, the Generalized Boosted Regression Model (GBM).

3.5.1 Basic Models

'dep_delay' is the number of minutes that the plane either departs early or late. Negative numbers are for early departures and positive numbers are for the number of minutes the plane is late. First, we used a basic model of simply predicting the 'dep_delay' to always be 0. This was done to establish baseline performance. This model had a root mean squared error (RMSE) of 8.30571. The model in which we predicted the median for all the predictions had an RMSE of 8.46926.

3.5.2 Linear Regression

A linear regression model assumes a linear relationship between the explanatory variables and the response variable. The model minimizes the squared loss function. This minimization process generates coefficients which are used for a linear combination of the explanatory variables. The linear combination is the prediction. Then we tried linear regression with 'dep_delay' as the response variables and all the other remaining variables as the explanatory variables. This model was better than predicting the mean with an RMSE of 7.98999. This suggests that there is some relationship between the 'dep_delay' and the explanatory variables.

3.5.3 Generalized Boosted Regression Model (GBM)

3.5.3.1 Explanation of Model

How boosted regression models work is we first start off with a baseline prediction. For example, we can use the mean as our baseline prediction. Then we use a base classifier to iteratively predict on the residuals multiplied by the shrinkage hyperparameter. Then this base classifier is added to the ensemble of base classifiers trained so far. A lower shrinkage effectively means a lower learning rate and therefore you need more iterations to reduce the train set residuals by the same amount. The benefit of a smaller shrinkage (with sufficient trees) is that you end up with a larger ensemble of trees that can reach a lower cross validation loss. This iterative prediction process is called boosting. In our case, our base classifiers are regressor trees. Each tree decides on its splitting criterion greedily by picking the split which results in the lowest mean squared error or some other splitting heuristic. We continue for n number of trees where n is specified by the user. Each iteration produces one tree, so the number of iterations is equal to the number of trees.

3.5.3.2 Tuning

Afterwards, we tried a Generalized Boosted Regression Model (GBM). This model with our best tuned hyperparameters had the lowest RMSE of 7.91635 on the validation set after it was tuned to have a shrinkage of 0.01 and 8,192 trees. Shrinkage is proportional to the learning rate. 8,192 is the number of trees used in the model. Each iteration uses 1 tree, so 8,192 trees also refers to the number of iterations. According to the vignette provided by the 'GBM' package, the RMSE can always be improved by decreasing shrinkage, but this provides diminishing returns. A good strategy would be to pick a small shrinkage that balances performance and compute time. Then with this fixed shrinkage value, increase the number of trees until you get diminishing returns. We decided to follow the aforementioned strategy. See figure 2 for a summary of our tuning experiments.

3.5.3.3 Influence

Please see table 5 for a table of the relative influence of each explanatory variable. Here, you can see the relative influence for each variable for GBM.
For a GBM, the improvement in the splitting criterion (which is mean squared error for regression) for a given variable is calculated at each step. The relative influence for a given variable is the average of these improvements over all the trees where the aforementioned variable is used.

4 Results

In regression and GBM, we found different features to be important.
Please see the table below for all the models and their root mean squared error (RMSE) on the validation set.

Table 1: benchmark comparing all the models that we tried

model_description	rmse
predicting 0	8.30571
predicting the mean	8.29977
predicting the median	8.46926
linear regression	7.98999
GBM (tuned)	7.89071

4.1 GBM

For the best GBM model, 'dest', which refers to which airport a given plane was flying to, was one of the most important features. However, the one hot encoding versions of the variable 'carrier' were some of the most important features for regression. Based on the relative influence scores provided by the best GBM, some of the most important feature variables include 'dest', 'model', and 'sched_dep_time_num_minute'. The 'dest' column contains the airport code for where a given flight is flying to. Based on my run of GBM with a shrinkage of 0.01 and 16834 trees, 'dest' was the most important feature with 49.56 relative influence. ("Gradient Boosting Machines · UC Business Analytics R Programming Guide" 2019). 'dest' is the destination airport code. 'sched_dep_time_num_minute' is the number of minutes since the beginning of a given day for that flight. 'model' is the plane model.

4.2 Linear Regression

On the other hand, 'dest' does appear as an important feature in linear regression as well but it is not the most important feature. If we can somehow sum up all the contributions from each of the one-hot-encoded variables derived from 'dest', then, it might appear as the most important feature for linear regression as well. We use ANOVA to measure the statistical significance of 'dest'. Performing ANOVA to compare linear regression models with and without 'dest' yielded a low p-value of 0.0001863 associated with having 'dest' meaning that keeping at least one of the one-hot-encoded variables derived from 'dest' is beneficial for the linear regression model with high statistical significance.

4.3 Comparison

'dest' was most one of the most important features in GBM. It is possible that 'dest' is important in combination with other variables which is something that the linear regression model without interaction terms cannot capture the relationship of, whereas GBM can discover these non-linear relationships.

5 Conclusion and Discussion

5.1 Discussion

We considered removing outliers in terms of 'dep_delay' in train but not in validation, then use k-folds cross validation on test to determine how many outliers we should remove to boost performance on the cross-validation set. We considered removing highly influential points in order to train a better model. In this case, we consider highly influential points to be points with high cook's distances. However, this was infeasible as we did not have enough computational resources available and it took too long.

5.2 Conclusion

None of the models that we tried performed particularly well. We surmise that this may be due to the explanatory variables having a weak relationship with the 'dep_delay' variable. There is a lack of information about the condition of a particular flight before it reaches New York City. Instead, we get information about where the flight is going next, which would reveal less information about the current condition of the plane and what kind of maintenance it would need, and therefore what 'dep_delay' it would have. Out of all the models, that we tried, gradient boosted regression models yielded the best performance based on having the lowest root mean squared error on the hold out test set. We believe that this makes sense because GBMs are able to capture non-linear relationships between the explanatory variables and 'dep_delay' whereas linear regression cannot.

6 Code

6.1 Preparing the programming environment

6.1.1 Loading Libraries

```r
library(tidyverse)
```

6.2 Data Preprocessing

6.2.1 Loading the data

```r
library(nycflights13)
library(Hmisc)
```

```
## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```r
set.seed(42)
original_data <- read_csv("fltrain.csv.gz")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   carrier = col_character(),
##   tailnum = col_character(),
##   origin = col_character(),
```

Table 2: 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	

```
##   dest = col_character(),
##   time_hour = col_datetime(format = ""),
##   name = col_character(),
##   dst = col_character(),
##   tzone = col_character(),
##   type = col_character(),
##   manufacturer = col_character(),
##   model = col_character(),
##   engine = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
DF <- original_data
```

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)
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
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)
#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
```

```
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)
library(imputeMissings)
```

```
##
## Attaching package: 'imputeMissings'

## The following object is masked from 'package:Hmisc':
##
##     impute

## The following object is masked from 'package:dplyr':
##
##     compute
```

```r
impute_model <- imputeMissings::compute(DF, method="median/mode")
impute_model
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)
```

```
## corrplot 0.84 loaded
```

6.3 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
```

6.4 Exploratory Data Analysis

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

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

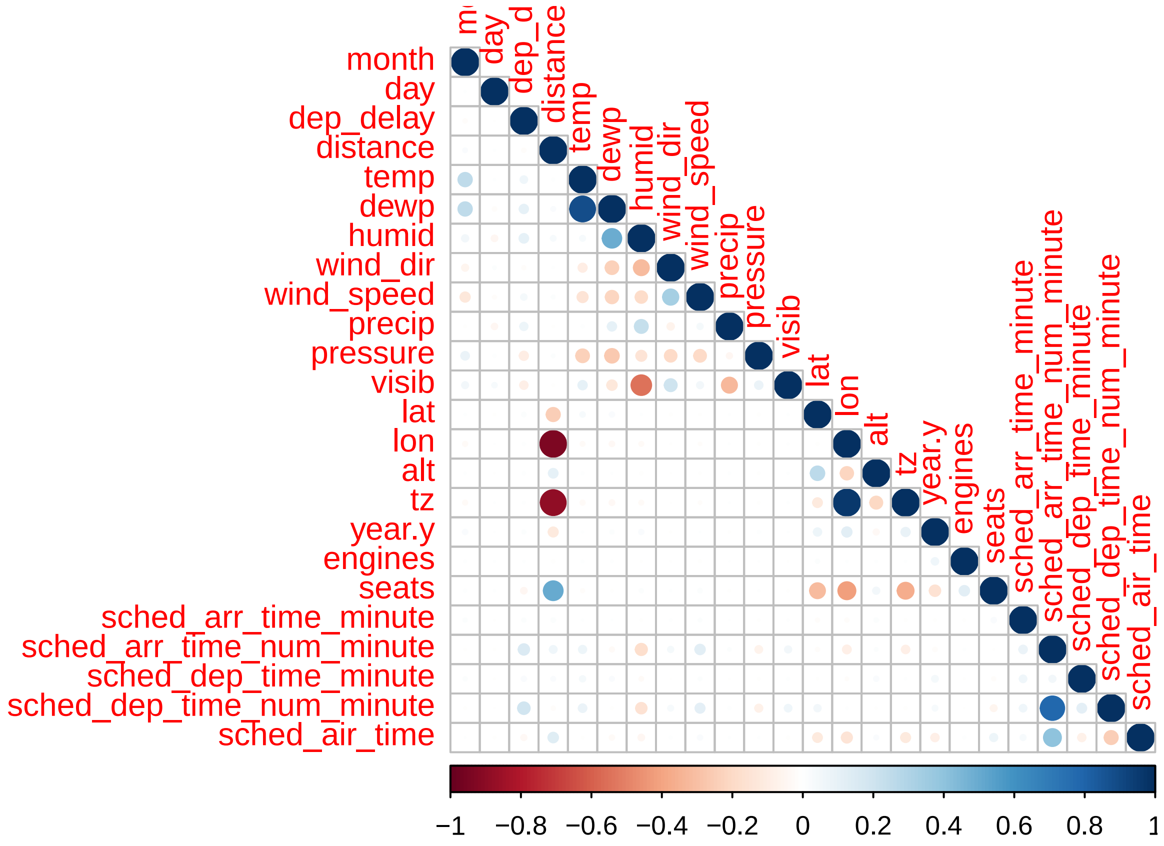


Figure 1: grid depicting correlation amongst all numerical variables

Table 3: proportion of variance explained by each principal component

	x
PC1	0.135269
PC2	0.106195
PC3	0.086279
PC4	0.068886
PC5	0.060590
PC6	0.058295

Table 4: coefficients for each variable on each principal component

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	P
lon	0.534262	0.008228	-0.011813	-0.007181	-0.141255	0.021195	0.014172	-0.023885	-0.0224
distance	-0.530322	-0.013673	0.042091	0.000606	-0.042942	-0.005943	-0.122736	0.043705	0.0200
tz	0.515232	0.003918	-0.003819	-0.009390	-0.207494	0.030490	0.001420	-0.017610	-0.0442
seats	-0.326797	-0.021807	0.092398	0.007581	-0.299155	-0.009582	-0.206563	-0.144397	0.0156
alt	-0.129118	0.001274	-0.022431	0.017906	0.466078	-0.062307	0.241778	-0.082761	-0.1384
sched_air_time	-0.126932	-0.059262	-0.046756	-0.053512	-0.299113	0.151635	0.696202	-0.179246	-0.1491

6.4.1 all four components at same time

proportion of variance explained by each component

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

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

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

6.4.2 take out extreme departure delays

```r

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

```r
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,]
```

```r
# pre-allocate space
preallocate_df <- function(n){
  df <- data.frame(model_description = character(n), rmse = numeric(n), stringsAsFactors = FALSE)
  for(i in 1:n){
    df$model_description[i] <- i
    df$rmse[i] <- toString(i)
  }
  df
}
```

6.5 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))
```

6.6 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
```

6.7 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))
```

6.8 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)
```

```
## Warning in predict.lm(model, newdata = lm_test_df): prediction from a rank-
## deficient fit may be misleading
```

```
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))
```

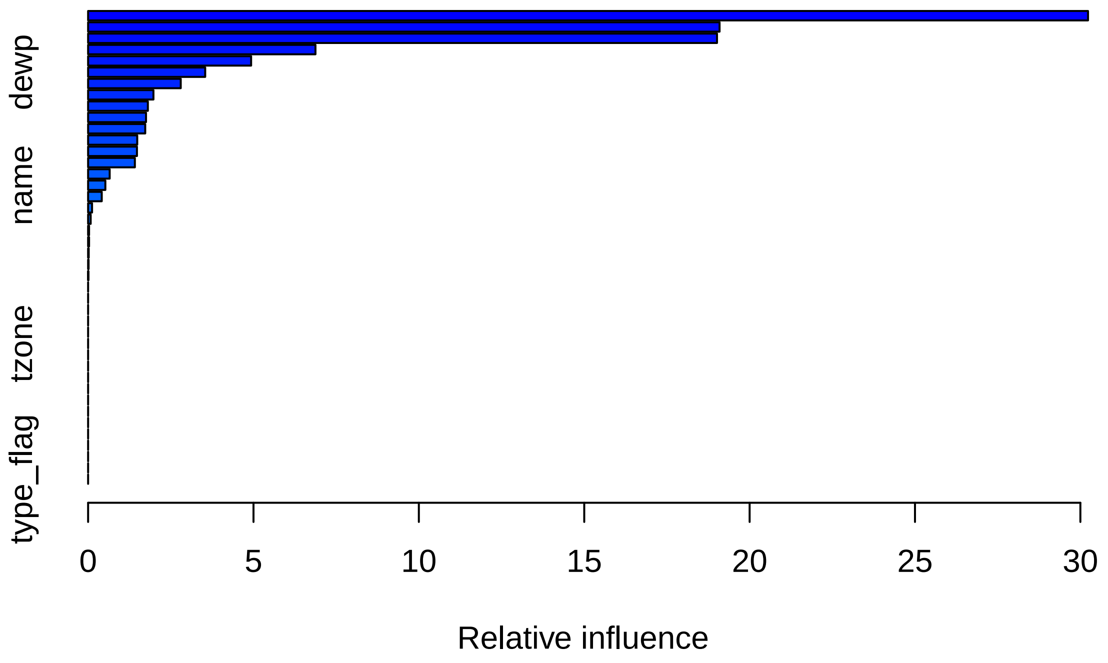
6.9 GBM

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
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 <- "models2/gbm_shrinkage_0point01_ntrees_8192_v2.rds"
if (!file.exists(destfile)) {
   train_gbm(destfile)
 }
model <- readRDS(destfile)
```



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

```
library(gbm)
train_gbm_rmse_vs_num_trees <- function(shrinkage, rerun) {
  #rerun <- TRUE
  set.seed(42)
  x <- 2 ^ seq(5, 15, by = 1)
  rmse_vec <- numeric(length(x))
  count <- 1
  filename_vec1 <- c("models2/gbm_shrinkage_")
```

Table 5: gbm relative influence

	var	rel.inf
dest	dest	30.23130
model	model	19.09165
sched_dep_time_num_minute	sched_dep_time_num_minute	19.01377
month	month	6.87392
carrier	carrier	4.92863
dewp	dewp	3.54021

```r
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_vec2, "_v2_summary.csv")
```

Table 6: benchmark comparing all the models that we tried

model_description	rmse
predicting 0	8.30571
predicting the mean	8.29977
predicting the median	8.46926
linear regression	7.98999
GBM (tuned)	7.89071

```
  filename_summary <- paste(filename_vec_summary, collapse = '')

  #rmse_vs_num_trees filename
  filename_vec_rmse_vs_num_trees <- append(filename_vec2, "_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)
train_gbm_rmse_vs_num_trees(shrinkage = 0.001, rerun = FALSE)
```

```
shrinkage_0point01_bench <- read.csv('performance2/gbm_shrinkage_0point01_ntrees_32768_v2_rmse_vs_num_t
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_ntrees_32768_v2_rmse_vs_num
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)
gbm_merge_df
```

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

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

```
## [1] "done"
```

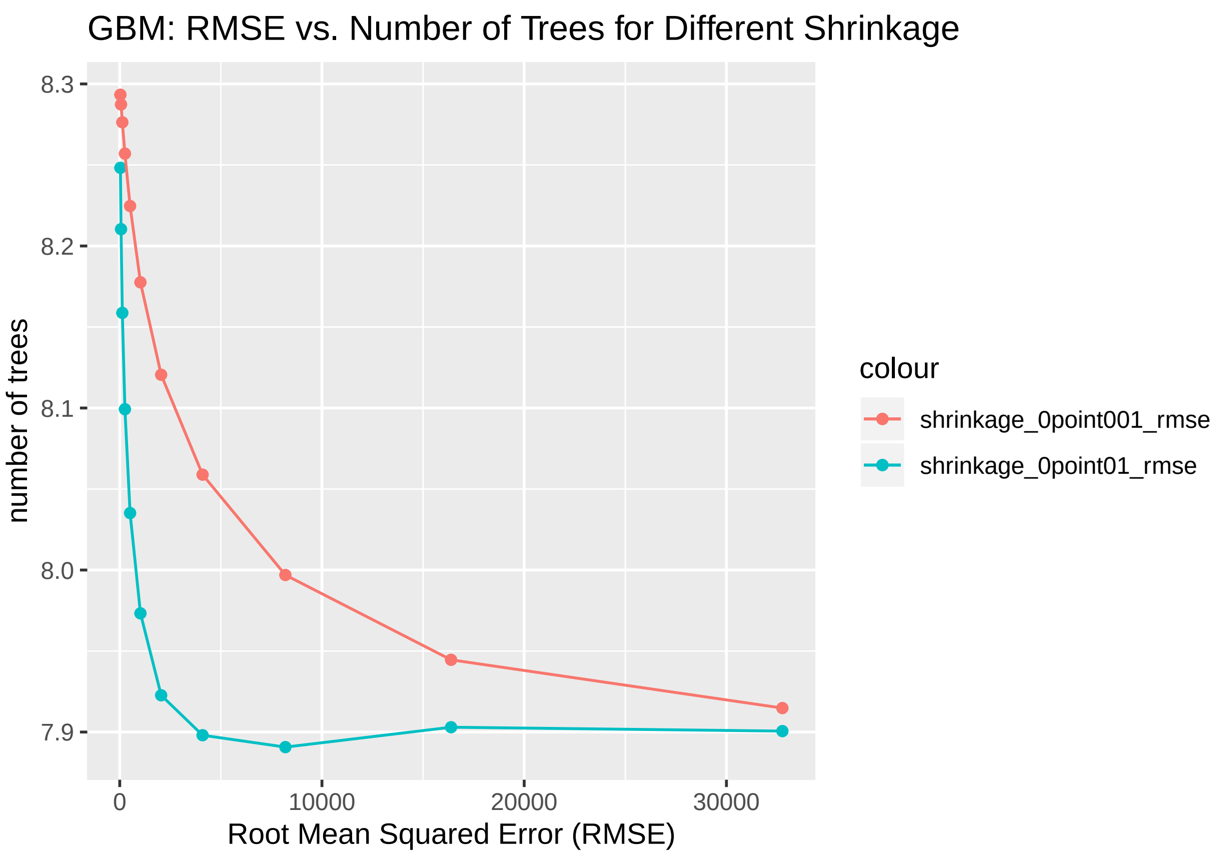


Figure 2: plot of RMSE vs number of trees for shrinkage = 0.01 and shrinkage = 0.001

References

"Gradient Boosting Machines · UC Business Analytics R Programming Guide." 2019. http://uc-r.github.io/gbm_regression#h2o.