Random Forest Fires ~ Stats 101C Final

Zachary Meeks. Kaylin Dee

Development of Models:

Since the name of our team is "Random Forest Fires" we decided to make our first model with Random Forest. Running scatter plots of features 3 to 10 against the response variable elapsed_time we saw that features 6 -- 9 appeared the most promising, while features 4 and 10 seemed by definition that they should be important (yet appeared more like random noise in the scatter plots). Our first random forests only used variables 6,7,8 and 6,7,9 and scored around 142xxxx. It took many, many more random forests and the introduction of feature 10 before we saw a slight improvement with Random Forests. We decided to then switch efforts over to XGBoosting.

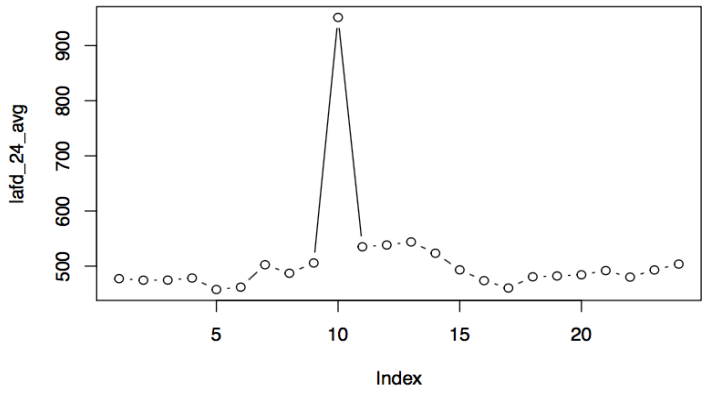
At first we were getting horrible scores with xgb with max tree depths 10 to 75 deep. We found a kaggle article that suggested xgb performs best between 1 to 6 branches deep. So we then tried max-depth at 1-5 and found best results with max-depth set at 3 and then built our first xgb model that could beat our best random forest model.

We were curious why feature 10 (incident creation time) affected both the random forest and the xgb and yet looked ambiguous from the original exploratory analysis. We tested transforming feature 10 into categorical variables. We tested 48, 24, 18, and 12, and found that 18 performed best. At the time we assumed that this was due to excess of traffic during rush hours and lack of traffic during sleep hours.

We still weren't getting scores near the best of the class so we went searching for outliers. The first things that we noticed were that dispatch sequence in kaggle maxed out at 155, while the training set had values above 400, so we deleted all rows with values above 156. Looking back over all of the scatter plots we noticed that the small number of elapsed_time values above 60000 or 70000 seemed to be possible outliers. Cross validating values between 60000 and 85000 we found that deleting elapsed_time values above 79250 from the model led to improvements.

We thought the faults of the current model related to overfitting. To combat this we looked up how to make use of the other parameters in xgb to combat overfitting. In particular we played with eta and gamma so that we could increase tree count without over-fitting.

Later, after the competition was over and we were searching for a good graph to include in our upcoming report we thought that that we should have an analytic graph to show why the 18 categories of incident creation time (feature 10) improved our model so we plotted the average elapsed_time for the 24 hours of the day, expecting spikes around rush hour. We were shocked at what we saw instead:

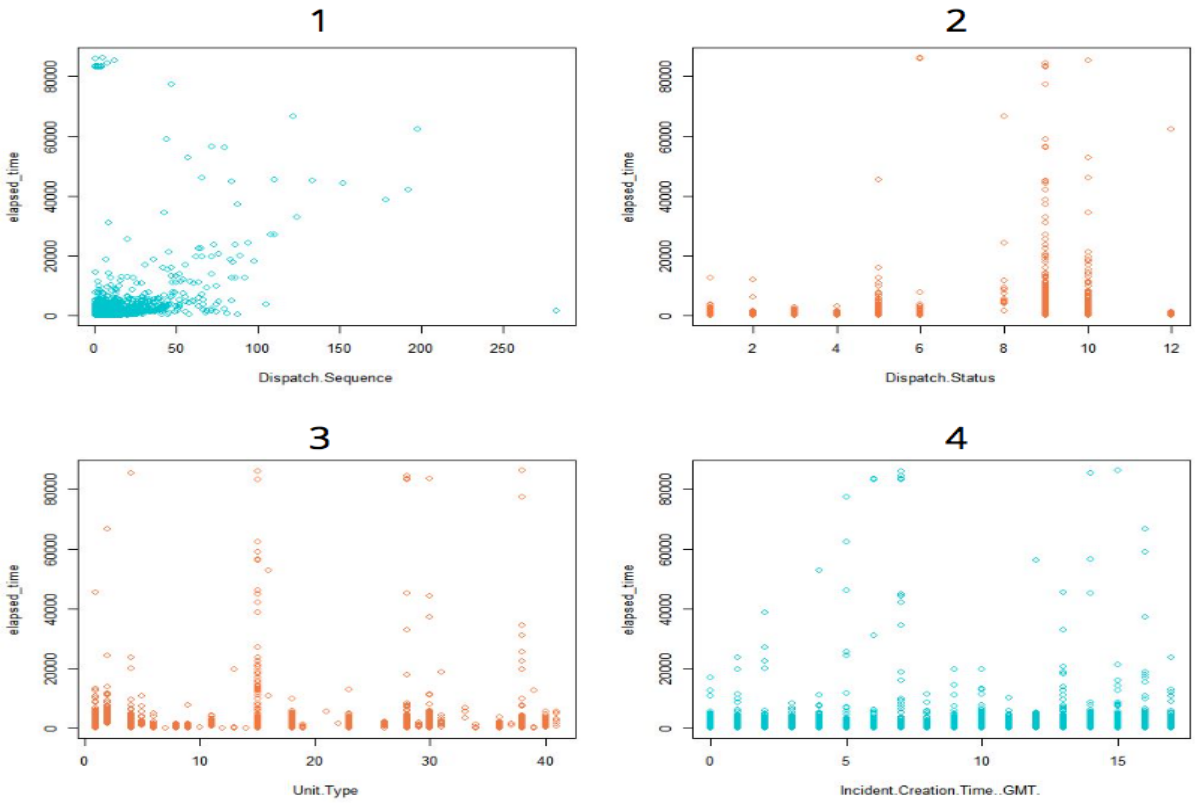


We made totally wrong assumptions!!! In fact, evening rush hour had one of the lowest elapsed_times. The biggest shock was that 9:00 -- 9:59 had a huge spike. (note: the above graph is off by 0.5 due to

ceiling function). We tried running a few models taking this new information into account but didn't empirically find any particular signal beneath the noise (we only gained a marginally better mse score by turning the eighteen categories into two). This might suggest that the spike around 9:30am is an error in the data. In any case, if we had more time, we would focus our efforts on figuring out how to make use of the discovery that the elapsed_times around 9.30am have an average elapsed_time almost twice as large as all other hours of the day.

Summary:

EXPLORATORY DATA ANALYSIS PLOTS



Used Variables

Dispatch Sequence, Dispatch Status, Unit Type, Incident Creation Time

Data Cleaning and Feature Engineering

Deleted all rows with dispatch sequence > 156; Deleted all rows with elapsed_time > 79250; Transformed "Incident.Creation.Time..GMT." into a categorical variable with 18 categories. Deleted all NAs

Description of the model

The final model utilizes the XGBoost algorithm in order to predict the response time for LAFD.

Why final model works

We address over-fitting in parameter settings and feature selection.
xgboost is a great boost package which ensembles many small decision trees with high bias and low variance, making gradual improvements and aggregating the total number of trees to significantly decrease the bias and further decrease the variance.

Best MSE
1400735.87

Code:

```
library(xgboost)
library(randomForest)
set.seed(14307)
lafd <- read.csv("enter_applicable_file_path_here/lafd")
kaggle_data <- read.csv("enter_applicable_file_path_here/kaggle")

lafd_sub <- na.omit(lafd[complete.cases(lafd[,c(6,7,8,10,11)]),c(6,7,8,10,11)])

lafd_sub_2 <- lafd_sub
lafd_sub_2[,4] <- floor(as.numeric(lafd_sub[,4])/4800)
lafd_sub_2[,4] <- as.character(lafd_sub[,4])

lafd_sub_3 <- lafd_sub[which(lafd$Dispatch.Sequence < 157), ]

s <- length(lafd_sub$elapsed_time)
s

#           this chunk for running 70/30 validation
#train <- sample(c(1:s),s*.70, replace = FALSE)
#lafd_xgb_train0 <- lafd_sub[train,]
#lafd_xgb_test <- lafd_sub[-train,]
#lafd_xgb_train <- lafd_xgb_train0[which(lafd_xgb_train0$elapsed_time < 79250), ]
#train_matrix <- sparse.model.matrix(elapsed_time ~ .-1, data = lafd_xgb_train)
#test_matrix <- sparse.model.matrix(elapsed_time ~ .-1, data = lafd_xgb_test)
#labels <- lafd_xgb_train$elapsed_time

lafd_xgb_full <- lafd_sub_3[which(lafd_sub_3$elapsed_time < 79250), ]
full_matrix <- sparse.model.matrix(elapsed_time ~ .-1, data = lafd_xgb_full)
labels_f <- lafd_xgb_full$elapsed_time

xgb <- xgboost(data = full_matrix,
 label = labels_f,
 eta = 0.0173,
 gamma = 1.43,
 max_depth = 3,
 nround=307,
 subsample = 0.5115,
 colsample_bytree = 0.5,
 seed = 17,
 eval_metric = "rmse",
 nthread = 2
)

#xgb.pred <- predict(xgb, test_matrix)
#xgb.mse <- mean((xgb.pred - lafd_xgb_test$elapsed_time)^2)
#xgb.mse
```

```
kag_sub <- kaggle_data[,c(6,7,8,10)]
kag_sub[,4] <- floor(as.numeric(kag_sub[,4])/4800)
kag_sub[,4] <- as.character(kag_sub[,4])

kag_matrix <- sparse.model.matrix(~ .-1, data = kag_sub)

kag_pred <- predict(xgb, kag_matrix)
kag_rows <- na.omit(kaggle_data[complete.cases(kaggle_data[,c(6,7,8,10)]),1])
full_rows <- kaggle_data[,1]
missing_rows <- setdiff(full_rows, kag_rows)
avg_pred <- mean(kag_pred)
ss <- length(missing_rows)
pred_rep <- rep(avg_pred, ss)
kag_xx <- cbind(missing_rows, pred_rep)
prediction <- kag_pred
row.id <- kag_rows
kag_x0 <- cbind(row.id, prediction)
kag_xgb_mod3x <- rbind(kag_x0, kag_xx)

write.csv(kag_xgb_mod3x, file = "enter_valid_file_path_here.csv", row.names = FALSE, quote=FALSE)
# note: will need to manually convert 3e5 into 300000 and 3e6 into 3000000 in the resulting .csv file
```

Note

The code above results in a slightly different score than our final score for our best model -- we played around with the parameters in the model afterward and forgot the exact original setting -- however model above is very, very close to best scoring model.