# Statistics101C Final Report

[cyc00]

[TEAM ID:717009]

Yucong Chen(serefinachen@gmail.com)

Josh Oh(musicinkeys@gmail.com)

Victor Shih(kyleshihv@gmail.com)

Instructed by: Akram Almohalwas

Spring 2017

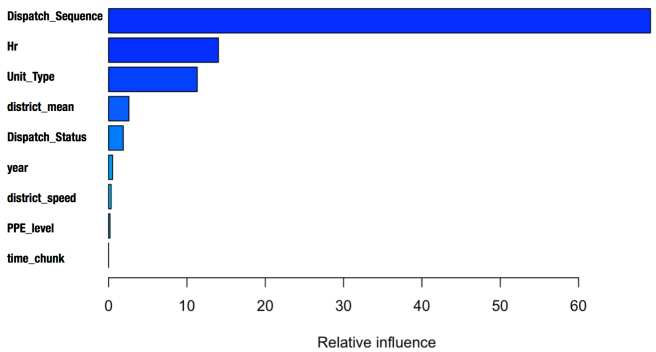
# 1. Data Cleaning and Transformations

- **Data Cleaning:** We deleted the variable “Emergency Dispatch Code” from the dataset, because it only had one level, Emergency. We also removed the variable “incident.ID”, since it was just a useless random ID number. There were missing values in both the training dataset and the testing dataset. Alongside, since most NAs in the training dataset were contained in the response variable, we simply deleted those observations. After removing NAs in the training data, there was only around 0.1% of data containing missing values for predictors. We removed those observations because of the low percentage. For the testing data, there were missing values in the predictor “Dispatch Sequence”. We used linear regression for training data to predict the dispatch sequence for these observations.

- **Data Transformation:** We created a new categorical variable called “hr” by extracting the hour of the incident from the predictor “Incident Creation Time (GMT)”. After grouping the average elapsed time by hour, we created a new predictor called “time_chunk”, which classifies the observations into one of the four levels: fastest, fast, slow or slowest. We also changed the predictor “year” and “First in District” into categorical variables. By grouping the median elapsed time by “First_in_District”(the fire station), we created “district_speed” which ranked each observation with a numerical score from 1 to 9, indicating the speed of that station. We took the log transformation on the predictor “Dispatch_sequence”, since it represented the “waiting time” of vehicles to go to incident places and may follow an exponential distribution. Since the categorical variable “First_in_District” have more than one hundred levels, we replaced it with the mean response value to avoid overfitting. For features “Dispatch Status”,“Unit Type” and “PPE Level” and “row.id”, we just leaved them as they are.

# 2. Our Model

```
bst3<-gbm(elapsed_time~.-row.id, interaction.depth = 3,
data=l[-idna,],distribution="gaussian",n.trees =300,verbose =F, shrinkage =0.15)
```




We used a boost model with the 9 predictors mentioned above. We tuned the parameters manually by creating a

| | var <fctr> | rel.inf <dbl> |
|---|---|---|
| Dispatch_Sequence | Dispatch_Sequence | 69.1774313 |
| hr | hr | 14.0257283 |
| Unit_Type | Unit_Type | 11.3173671 |
| district_mean | district_mean | 2.5945573 |
| Dispatch_Status | Dispatch_Status | 1.8727827 |
| year | year | 0.5073719 |
| district_speed | district_speed | 0.3274878 |
| PPE_Level | PPE_Level | 0.1772736 |
| time_chunk | time_chunk | 0.0000000 |

testing set of the actual size of the testing data and tried for almost 20 possible models. The model “bst3” with a shrinkage value of 0.15, 300 trees, and an interaction depth of 3. Together this usually produced the smallest MSE for the testing set we separated from the training data. After realizing bst3 was the best model, we used all of the training data to fit this model, thereby producing the final model shown above. The chart above is the relative importance of our predictors for the model.

# 3. Best MSE

Our best score on Kaggle is 1392287.13605.

# 4. Strengths and Weaknesses of the Model

- **Strengths:** Compared to other models that we tried(PCA, PLS, random forest, tree, Lasso, and ridge), the gradient boosting model had the highest predictive power, since it minimized the MSE. Furthermore, it is relatively computational inexpensive in comparison to random forest, making it possible for us to use all of the training data to build and tune our model. Our boosting model also avoids overfitting, since we tuned the parameter shrinkage, creating a penalty for less useful predictors. Unlike trees, boosting models allowed us to use categorical predictor “Unit_Type” which has around 40 levels. Therefore, we avoided losing information by reducing the number of levels in predictors (i.e. replacing levels of very low frequency with levels of high frequency).

- **Weaknesses**: We searched online for possible external data, such as the location of the fire station. But it turned out none of them was helpful in terms of decreasing MSE. If we could find any useful external information, the predictive power would have increased. Furthermore, we tuned the parameters manually by trying some possible combinations. (Because it would take a very long time to use package caret to find the best combination) Due to the limited number of combinations we tried, the combination we used for parameters may be a set of values that just produced a “local minimum” MSE. If we actually tuned the parameters using functions in caret, we may have gotten better results.

```
---
title: "Code for group Winning(cyc00 in Kaggle)"
author: "Yucong Chen"
date: "6/9/2017"
output: html_document
---

  ####Note: Only included the code for models and transformations that produced
the bset result.

  #change variables

```{r}
library(readr)
library(dplyr)
#skip Emergency Dispatch Code, because it only has one level; skip incident.ID
because it is not useful(just the ID of the incident)
train <- read_csv("~/Desktop/101C/final projects/lafdtraining updated.csv",
                  col_types = cols(`Emergency Dispatch Code` = col_skip(),
                                   incident.ID = col_skip()))

test <- read_csv("~/Desktop/101C/final projects/testing.without.response.txt",
                 col_types = cols(`Emergency Dispatch Code` = col_skip(),
                                  incident.ID = col_skip()))


#change the names of predictors
names(train)[3]<-"First_in_District"
names(train)[4]<-"Dispatch_Sequence"
names(train)[5]<-"Dispatch_Status"
names(train)[6]<-"Unit_Type"
names(train)[7]<-"PPE_Level"

names(test)[3]<-"First_in_District"
names(test)[4]<-"Dispatch_Sequence"
names(test)[5]<-"Dispatch_Status"
names(test)[6]<-"Unit_Type"
names(test)[7]<-"PPE_Level"

#remove NA based on y
train<-train[complete.cases(train$elapsed_time),]
train$elapsed_time<-as.numeric(train$elapsed_time)

#extract the hour of the incident
library(lubridate)
train$hr<-hour(train$`Incident Creation Time (GMT)`)
test$hr<-hour(test$`Incident Creation Time (GMT)`)

a<-train %>% group_by(hr) %>% summarise(mean_y=mean(elapsed_time),
median_y=median(elapsed_time))
train$fastest<-train$hr%in%c(2,1,0,3,23,4)
train$fast<-train$hr%in%c(20,22,21,5,16,19)
train$slow<-train$hr%in%c(15,17,18,6,14,7)
train$slowest<-train$hr%in%c(9,10,11,12,13,8)
library(tidyr)
```
```

```
tr<-gather(train, "time_chunk","yes",11:14)
train<-tr[tr$yes==TRUE,-12]

test$fastest<-test$hr%in%c(2,1,0,3,23,4)
test$fast<-test$hr%in%c(20,22,21,5,16,19)
test$slow<-test$hr%in%c(15,17,18,6,14,7)
test$slowest<-test$hr%in%c(9,10,11,12,13,8)
te<-gather(test, "time_chunk","yes",10:13)
test<-te[te$yes==TRUE,-11]
train<-train[,-8]
test<-test[,-8]

b<-train %>% group_by(First_in_District) %>% summarise(
median_y=median(elapsed_time))
train<-left_join(train, b, by="First_in_District")
train$s1<-
train$First_in_District%in%c(9,11,14,15,16,21,26,29,35,40,49,67,76,77,79,86,91,97,

train$s2<-train$First_in_District%in%c(2,3,6,10,12,13,20,27,73,87,104)
train$s3<-train$First_in_District%in%c(33,38,39,43,48,52,60,82,95)
train$s4<-train$First_in_District%in%c(17,36,46,55,64,75,93,94,100,102)
train$s5<-train$First_in_District%in%c(4,34,37,57,72,74,78,88,101,106)
train$s6<-train$First_in_District%in%c(7,25,59,61,66,68,70,81,92,98)
train$s7<-train$First_in_District%in%c(18,41,58,62,65,85,89,90,96,103)
train$s8<-train$First_in_District%in%c(1,42,47,50,63,69,71,83,84,105)
train$s9<-train$First_in_District%in%c(5,8,19,23,24,28,44,51,56,99)
tr<-gather(train, "district_speed","yes",12:20)
train<-tr[tr$yes==TRUE,-13]
library(stringr)
train$district_speed <- substr(train$district_speed, 2, 2)

test<-left_join(test, b, by="First_in_District")
test$s1<-
test$First_in_District%in%c(9,11,14,15,16,21,26,29,35,40,49,67,76,77,79,86,91,97,1

test$s2<-test$First_in_District%in%c(2,3,6,10,12,13,20,27,73,87,104)
test$s3<-test$First_in_District%in%c(33,38,39,43,48,52,60,82,95)
test$s4<-test$First_in_District%in%c(17,36,46,55,64,75,93,94,100,102)
test$s5<-test$First_in_District%in%c(4,34,37,57,72,74,78,88,101,106)
test$s6<-test$First_in_District%in%c(7,25,59,61,66,68,70,81,92,98)
test$s7<-test$First_in_District%in%c(18,41,58,62,65,85,89,90,96,103)
test$s8<-test$First_in_District%in%c(1,42,47,50,63,69,71,83,84,105)
test$s9<-test$First_in_District%in%c(5,8,19,23,24,28,44,51,56,99)
te<-gather(test, "district_speed","yes",11:19)
test<-te[te$yes==TRUE,-12]
test$district_speed <- substr(test$district_speed, 2, 2)
test$district_speed<-as.numeric(test$district_speed)
train$district_speed<-as.numeric(train$district_speed)
train<-train[,-11]
test<-test[,-10]

train<-na.omit(train)

#change the data type
train$year<-as.factor(train$year)
```

```
train$Dispatch_Status<-as.factor(train$Dispatch_Status)
train$Unit_Type<-as.factor(train$Unit_Type)
train$PPE_Level<-as.factor(train$PPE_Level)
train$First_in_District<-as.factor(train$First_in_District)
train$hr<-as.factor(train$hr)
train$time_chunk<-as.factor(train$time_chunk)

test$year<-as.factor(test$year)
test$Dispatch_Status<-as.factor(test$Dispatch_Status)
test$Unit_Type<-as.factor(test$Unit_Type)
test$PPE_Level<-as.factor(test$PPE_Level)
test$First_in_District<-as.factor(test$First_in_District)
test$hr<-as.factor(test$hr)
test$time_chunk<-as.factor(test$time_chunk)

#NA for testing
#There is still NA for Dispatch_Sequence
#predict dispatch sequence by a linear model
sum(is.na(test$Dispatch_Sequence))

#use linear model to predict from training data
na<-is.na(test$Dispatch_Sequence)
t<-train[,c(2:7,9,10,11)]
fit<-lm(Dispatch_Sequence~.,data=t)
summary(fit)

test[na,]$Dispatch_Sequence<-round(predict(fit, test[na,]),0)

sum(is.na(test$Dispatch_Sequence))
test$elapsed_time<-NA
```


#We tried for PLS, PCA, tree, random forest, boost, xgboost, and linear model.
Since the mse obtained from PLS, PCA, tree, random forest and xgboost are
higher, we did not include the code for these methods.

#the boost model

```{r}
#use the mean elasped time of each district"district_mean" to replace the
predictor "First_in_District"
train$First_in_District<-as.numeric(as.character(train$First_in_District))
test$First_in_District<-as.numeric(as.character(test$First_in_District))
ls<-train
%>%group_by(First_in_District)%>%summarise(district_mean=mean(elapsed_time) )
ls<-data.frame(ls)
tr<-left_join(train, ls, by="First_in_District")
train<-tr[,-3]
te<-left_join(test, ls, by="First_in_District")
test<-te[,-3]
#shuffle the training data(did not create a testing set because I have already
tuned the parameters to get the best model)
###################For the best result on kaggle, we forgot to set the seed when
shuffling the data. By default, boost will randomly select half of the parameter
```

```
of the next observation to propose the next tree in the expansion. Even though
we used all of the training data for the best model, the order of the data is
not the same. Therefore, the resulting model may vary a little bit. But the mse
should be farily similar.(We are so sorry about this, but we did not know this
randomness in bgm before submitting the result.)
set.seed(666)
id<-sample(nrow(train))
idte<-1:nrow(test)
idtr<-(nrow(test)+1):length(id)

#combine training and testing set
l<-rbind(train[id,],test)

#find the labels of testing data
idna<-which(is.na(l$elapsed_time))
#take log on dispatch sequence, because dispatch sequence is similar to a
waiting time, it may follow exponential distribution(and the result of using log
dispatch sequence is indeed better than that of using dispatch sequence
directly)
l$Dispatch_Sequence<-log(l$Dispatch_Sequence)
l$Dispatch_Status<-as.factor(l$Dispatch_Status)
l$hr<-as.factor(l$hr)
l$year<-as.factor(l$year)
l$time_chunk<-as.factor(l$time_chunk)
l$elapsed_time<-as.numeric(l$elapsed_time)
l$PPE_Level<-as.factor(l$PPE_Level)
l$Unit_Type<-as.factor(l$Unit_Type)
# t is the testing data set
t<-l[idte,]
library(gbm)
#We tuned the parameters manually(tried for about 20 combinations), the
following are some of the combination, and bst3 is the best one
set.seed(1)
bst1<-gbm(elapsed_time~.-row.id, interaction.depth =
2,data=l[idtr,],distribution="gaussian",n.trees =320,verbose =F,shrinkage =0.15)
summary(bst1)

set.seed(1)
bst2<-gbm(elapsed_time~.-row.id, interaction.depth =
3,data=l[idtr,],distribution="gaussian",n.trees =320,verbose =F,shrinkage =0.1,
n.minobsinnode = 15)
summary(bst2)

######best one
set.seed(1)
bst3<-gbm(elapsed_time~.-row.id, interaction.depth =
3,data=l[idtr,],distribution="gaussian",n.trees =320,verbose =F,shrinkage =0.15)
summary(bst3)


set.seed(1)
bst4<-gbm(elapsed_time~.-row.id, interaction.depth =
3,data=l[idtr,],distribution="gaussian",n.trees =320,verbose =F,shrinkage =0.15,
n.minobsinnode = 15)
summary(bst4)
```

```
t$pre<-predict(bst1,newdata=t, n.trees=300)
mean((t$pre-t$elapsed_time)^2)

t$pre<-predict(bst2,newdata=t, n.trees=300)
mean((t$pre-t$elapsed_time)^2)

t$pre<-predict(bst3,newdata=t, n.trees=300)
mean((t$pre-t$elapsed_time)^2)

t$pre<-predict(bst4,newdata=t, n.trees=320)
mean((t$pre-t$elapsed_time)^2)

#use all data for the model
set.seed(2)
bst3<-gbm(elapsed_time~.-row.id, interaction.depth = 3,data=l[-
idna,],distribution="gaussian",n.trees =400,verbose =F,shrinkage =0.15)
summary(bst3)

#try on true test data
re<-l[idna,]
re$prediction<-predict(bst3, l[idna,],n.trees=300)
#there are a few negative response time values, it doesn't make sense, so take
the absolute value of the result
re$prediction<-abs(re$prediction)
try<-re[,c(1,12)]
```