```
## Nankai University and Analytic Parnters Machine Learining Sample Code
## Hong Jin; Zihao Zhang;Zhilun Jiao;
## Data are public avaliable in Kaggle: https://www.kaggle.com/benhamner/sf-bay-area-bike-share
## we illustrate the whole process of data processing, feature engineering, modeling, and parameter tuning
## we illustrate models of OLS, Lasso, Ridge, Random Forest, XGboost


#################################################################
##                  enrivoment setting                         ##
#################################################################

detachAllPackages <- function() {

  basic.packages <-
c("package:stats","package:graphics","package:grDevices","package:utils","package:datasets","package:methods","package:base")


  package.list <- search()[ifelse(unlist(gregexpr("package:",search()))==1,TRUE,FALSE)]

  package.list <- setdiff(package.list,basic.packages)

  if (length(package.list)>0)  for (package in package.list) detach(package, character.only=TRUE)

}

detachAllPackages()
rm(list=ls(all=TRUE))


library("lubridate")
library("dplyr")
library("ggplot2")
library("timeDate")
library(dummies)
library(plotly)
library("ggmap")
library("ggrepel")
library("reshape2")
library(dplyr)
library(forecast)
library(knitr)
#install.packages("xgboost")
library("xgboost")
require(xgboost)
library(dummies)
library("randomForest")
library(glmnet)
#install.packages("h2o")
library(h2o)

#################################################################
##                    Reading data                             ##
#################################################################


trip = read.csv("P:/AnalyticPartnters _Machine Learning Sample Project/trip.csv")
station =  read.csv("P:/AnalyticPartnters _Machine Learning Sample Project/station.csv")
weather = read.csv("P:/AnalyticPartnters _Machine Learning Sample Project/weather.csv")



#################################################################
##                    Data Processing                          ##
#################################################################

trip$start_date = mdy_hm(as.character(trip$start_date))
trip$end_date = mdy_hm(as.character(trip$end_date))
trip$date = as.Date(trip$start_date)
trip$end_date = as.Date(trip$end_date)
trip_count = trip %>% group_by(date) %>% summarise(count = n())

##feature engineer, create month, day of week, hour of day.
trip$start_month = lubridate::month(trip$start_date)
trip$start_wday = lubridate::wday(as.Date(trip$start_date), label = TRUE)
trip$start_hour = lubridate::hour(trip$start_date)
trip$end_month = lubridate::month(trip$end_date)
trip$end_wday = lubridate::wday(as.Date(trip$end_date), label = TRUE)
trip$end_hour = lubridate::hour(trip$end_date)
##convert seconds to minutes
trip$duration = trip$duration/60
trip$is_weekend = ifelse(trip$start_wday %in% c("Sun", "Sat"), 1, 0)
trip$is_weekend = factor(trip$is_weekend, labels = c("weekday", "weekend"))
trip$is_weekend_v2 = ifelse(trip$end_wday %in% c("Sun", "Sat"), 1, 0)
trip$is_weekend_v2 = factor(trip$is_weekend_v2, labels = c("weekday", "weekend"))
###############insight getting##########
trip_date = trip %>% group_by(date) %>% summarise(trip_count = n())
ggplot(trip_date, aes(x = date, y = trip_count)) + geom_point() + geom_smooth(color = "#1A1A1A",method = 'loess') +
  labs(x = "Date", y = "# of Trips", title = "Daily # of Bicylcle Trips from 2013 - 2015") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
head(trip_date)
tail(trip_date)
summary(trip_date)

isweekend_date = trip %>% group_by(date, is_weekend) %>% summarise(count = n())
isweekend_date$is_weekend = factor(isweekend_date$is_weekend, labels = c("weekday", "weekend"))

ggplot(isweekend_date, aes(x = date, y=count)) + geom_point(aes(color = is_weekend), size = 3, alpha = 0.65) +
  labs(x = "Date", y = "Total # of Bicycle Trips") +
  theme(plot.title = element_text(hjust = 0.5))

ggplot(isweekend_date, aes(x = date, y=count)) +
  geom_point() +
  facet_wrap(~ is_weekend) +
  geom_smooth(se = F, method = 'loess') +
  labs(x = "Date", y = "Total # of Bicycle Trips") +
  theme(plot.title = element_text(hjust = 0.5))

hour_format <- c(paste(c(12,1:11),"AM"), paste(c(12,1:11),"PM"))

trip$start_wday <- factor(trip$start_wday)
trip$start_hour <- factor(trip$start_hour, level = 0:23, label = hour_format)

trip_hour_wday = trip %>% group_by(start_wday, start_hour) %>% summarise(count=n())

##time-diff and week-diff
ggplot(trip_hour_wday, aes(x = start_hour, y = start_wday, fill = count)) + geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.6), legend.title = element_blank(),
        legend.position = "top", legend.direction = "horizontal") +
  labs(x = "Hour of Day", y = "Day of Week of trips", title = "# of bicycle trips ") +
  scale_fill_gradient(low = "white", high = "black") +
  theme(plot.title = element_text(hjust = 0.5))

##subscriber and none
ggplot(trip, aes(x = date)) +
  geom_bar(aes(color=subscription_type), stat="count", position = "stack",fill="white") +
  facet_grid(~is_weekend) +
  labs(x = "Day of Week", y = "# of trips",
       title = "Customer Vs.Subscriber on Weekend and Weekdays") +
  theme(plot.title = element_text(hjust = 0.5), axis.text.x = element_blank())

station_trip = merge(trip, station, by.x = "start_station_id", by.y = "id")

##city-wise
ggplot(station_trip, aes(x = date)) +
  geom_bar(aes(color=subscription_type), stat="count", position = "stack") +
  facet_wrap(~city, scales = "free_y") +
  labs(x = "Day of Week", y = "# of trips",
       title = "Customer Vs.Subscriber by City") +
  theme(plot.title = element_text(hjust = 0.5), axis.text.x = element_blank())


city_wday = station_trip %>% group_by(start_wday, city) %>% summarise(count = n())
g = ggplot(city_wday, aes(y = city, x = start_wday)) +
  geom_point(aes(size = count, col = count)) +
  scale_size(range = c(1,10)) +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        legend.position = "none")

ggplotly(g, tooltip = c("x", "y", "colour"))


bbox = c(-122.4990, 37.31072, -121.7800, 37.88100)
sf = get_map(location = bbox, source = "stamen", maptype = "toner-lite")

map_trip = station_trip %>% group_by(long, lat, zip_code) %>% summarise(count = n())
ggmap(sf) + geom_point(data = map_trip, aes(x = long, y = lat, size = count, color = count)) +
  scale_size(name = "# Total Trips", range = c(3, 12))  + theme(legend.position = "none")

names(station_trip)[22] = "start_lat"
names(station_trip)[23] = "start_long"
end_station_trip = merge(trip, station, by.x = "end_station_id", by.y = "id")
names(end_station_trip)[22] = "end_lat"
names(end_station_trip)[23] = "end_long"

road_df = merge(station_trip, end_station_trip, by = "id") %>%
  select ("id","start_lat", "start_long", "end_lat", "end_long", "city.y", "city.x") %>%
  filter(city.y == "San Francisco" & city.x == "San Francisco")

road_map = road_df %>% group_by(start_lat, start_long, end_lat, end_long) %>% summarise(num_trips = n())
station_sf = station %>% filter(city=="San Francisco")


ggplot(road_map) + geom_segment(aes(x=start_long, xend = end_long, y = start_lat, yend=end_lat,
                                    size = num_trips, colour = num_trips, alpha = num_trips)) +
  geom_point(data = station_sf, aes(x=long, y=lat), size = 4) +
  geom_text_repel(data = station_sf, aes(x=long, y=lat, label=name), size = 4) +
  theme_light(base_size = 10) +
  scale_colour_gradient(low ="#132B43", high = "#56B1F7", limits=c(0, max(road_map$num_trips)), name="Number of Trips") +
```

```
  scale_alpha(limits=c(0, max(road_map$num_trips)), guide = F) +
  scale_size(limits=c(0, max(road_map$num_trips)), guide = F) +
  xlab("") + ylab("") + coord_fixed() +
  theme(axis.line = element_blank(),
        axis.text.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks=element_blank(),
        axis.title.x=element_blank(),
        axis.title.y=element_blank(),
        panel.grid=element_blank(),
        panel.border=element_blank())

## weather data processing ##
#############################


summary(weather)

summary(lubridate::mdy(as.character(weather$date)))

weather$date = lubridate::mdy(as.character(weather$date))
#set nan value as "Normal" in the event
levels(weather$events) = c(levels(weather$events), "Normal")
weather$events[weather$events==''] = "Normal"
weather$events[weather$events=='rain'] = "Rain"
weather$events = droplevels(weather$events, except = c("Normal", "Fog", "Rain", "Fog-Rain", "Rain-Thunderstorm"))
summary(weather)

weather$precipitation_inches = as.numeric(as.matrix(weather$precipitation_inches))
weather = weather %>% group_by(date) %>% mutate(precipitation_median = median(precipitation_inches, na.rm=T))

weather$precipitation_inches = ifelse(is.na(weather$precipitation_inches), weather$precipitation_median,
weather$precipitation_inches)

weather$precipitation_inches[is.na(weather$precipitation_inches)] = 0

summary(weather)

weather = weather %>% group_by(max_wind_Speed_mph) %>% mutate(gust_median = median(max_gust_speed_mph, na.rm=T))
weather$max_gust_speed_mph = ifelse(is.na(weather$max_gust_speed_mph), weather$gust_median, weather$max_gust_speed_mph)

##feature engineer, create month, day of week, dummay for weekend days, seasonal variable

weather$month = lubridate::month(weather$date)
weather$wday = wday(weather$date)
weather$year = lubridate::year(weather$date)
weather$day = lubridate::day(weather$date)

listHolidays()

Holiday = c(
  as.Date(USChristmasDay(2013)),
  as.Date(USColumbusDay(2013)),
  as.Date(USCPulaskisBirthday(2013)),
  as.Date(USDecorationMemorialDay(2013)),
  as.Date(USElectionDay(2013)),
  as.Date(USGoodFriday(2013)),
  as.Date(USInaugurationDay(2013)),
  as.Date(USIndependenceDay(2013)),
  as.Date(USLaborDay(2013)),
  as.Date(USLincolnsBirthday(2013)),
  as.Date(USMemorialDay(2013)),
  as.Date(USMLKingsBirthday(2013)),
  as.Date(USNewYearsDay(2013)),
  as.Date(USPresidentsDay(2013)),
  as.Date(USThanksgivingDay(2013)),
  as.Date(USVeteransDay(2013)),
  as.Date(USWashingtonsBirthday(2013)),
  as.Date(USChristmasDay(2014)),
  as.Date(USColumbusDay(2014)),
  as.Date(USCPulaskisBirthday(2014)),
  as.Date(USDecorationMemorialDay(2014)),
  as.Date(USElectionDay(2014)),
  as.Date(USGoodFriday(2014)),
  as.Date(USInaugurationDay(2014)),
  as.Date(USIndependenceDay(2014)),
  as.Date(USLaborDay(2014)),
  as.Date(USLincolnsBirthday(2014)),
  as.Date(USMemorialDay(2014)),
  as.Date(USMLKingsBirthday(2014)),
  as.Date(USNewYearsDay(2014)),
  as.Date(USPresidentsDay(2014)),
  as.Date(USThanksgivingDay(2014)),
  as.Date(USVeteransDay(2014)),
  as.Date(USWashingtonsBirthday(2014)),
  as.Date(USChristmasDay(2015)),
  as.Date(USColumbusDay(2015)),
  as.Date(USCPulaskisBirthday(2015)),
  as.Date(USDecorationMemorialDay(2015)),
  as.Date(USElectionDay(2015)),
```

```
  as.Date(USGoodFriday(2015)),
  as.Date(USInaugurationDay(2015)),
  as.Date(USIndependenceDay(2015)),
  as.Date(USLaborDay(2015)),
  as.Date(USLincolnsBirthday(2015)),
  as.Date(USMemorialDay(2015)),
  as.Date(USMLKingsBirthday(2015)),
  as.Date(USNewYearsDay(2015)),
  as.Date(USPresidentsDay(2015)),
  as.Date(USThanksgivingDay(2015)),
  as.Date(USVeteransDay(2015)),
  as.Date(USWashingtonsBirthday(2015))
)

weather$isholiday = ifelse(weather$date %in% Holiday, 1, 0)
weather$events = as.factor(as.character(weather$events))

indx = !(sapply(weather, is.factor) | sapply(weather,is.Date))

f=function(x){
  x<-as.numeric(as.character(x)) #first convert each column into numeric if it is from factor
  x[is.na(x)] =median(x, na.rm=TRUE) #convert the item with NA to median value from the column
  x #display the column
}
weather[,indx]=apply(weather[,indx],2,f)

summary(weather)

plot(weather$date,weather$max_sea_level_pressure_inches)


names(weather)

zip_city_match = data.frame(zip_code = unique(weather$zip_code), city= c("San Francisco","Redwood City",
                                                                         "Palo Alto","Mountain View","San Jose"))
weather = merge(weather,zip_city_match,by = "zip_code")


## station data processing ##
##############################

station
station$installation_date = mdy(as.character(station$installation_date))

#####add number of trips by day into weather data form for further analysis.

station_name = station[,c("name","city")]
names(station_name) = c("start_station_name","city")

trip_num = merge(trip,station_name,by=c("start_station_name"))
trip_num = trip_num %>% group_by(date,city,start_hour) %>% summarise(count = n())

dim(trip_num)
summary(trip_num)
df_v2 = merge(trip_num, weather, by = c("date","city"),x.all=TRUE)
hist(df_v2$count)

dim(weather)
dim(trip_num)
dim(df_v2)
summary(df_v2)

hist(df_v2$count)

df_v2$month <- as.factor(df_v2$month)
df_v2$year <- as.factor(df_v2$year)
df_v2$isholiday <- as.numeric(df_v2$isholiday)
df_v2$count <- as.numeric(df_v2$count)
#df_v2$day <- as.factor(df_v2$day)

summary(df_v2)
dim(df_v2)

df_v2$wday = as.factor(df_v2$wday)
df_v2 = df_v2[,!(names(df_v2) %in% c("zip_code"))]

dim(df_v2)

summary(df_v2)
dim(df_v2)



#####setting up trainning sample and test sample  ####
######################################################

dim(df_v2)
names(df_v2)

set.seed(0226)
```

```
#train_sample = sample(nrow(df_v2), nrow(df_v2)*0.8)

train_sample = c(1:nrow(df_v2))[df_v2$date<"2015-03-31"]
length(train_sample)/nrow(df_v2)


train_data = data.frame(df_v2[train_sample, -1])
test_data = data.frame(df_v2[-train_sample, -1])
train_data.y = df_v2$count[train_sample]
test_data.y =df_v2$count[-train_sample]

dim(train_data)
names(train_data)
dim(test_data)
length(train_data.y)
length(test_data.y)



require(caret)
flds <- createFolds(train_data$count, k = 10, list = TRUE, returnTrain = FALSE)



##################################################################
##                            OLS                               ##
##################################################################

lm1 <- lm(count~., data = train_data )
summary(lm1)

yhat <- predict(lm1,train_data)
r2insample <- (sum((yhat-mean(train_data.y))^2))/(sum((train_data.y-mean(train_data.y))^2))
r2insample

## Model fittin on training data
boost.a = data.frame(df_v2[train_sample, c('date','count')],yhat)
names(boost.a) = c("date", "Actual", "Prediction")
boost.AP = reshape2::melt(boost.a, id = "date")
boost.AP = boost.AP %>% group_by(date,variable) %>% summarise(value = sum(value))
boost.AP$variable= as.factor(boost.AP$variable)
ggplot(boost.AP, aes(x = date, y = value, colour = variable)) + geom_line() +
  labs(x = "Date", y = "number of trips", title = "Predicted Values vs Actual Values") +
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "transparent",colour = NA),
        panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        plot.background = element_rect(fill = "transparent",colour = NA),
        legend.position="top")



accuracy(lm1)
rmsein <- sqrt(mean((yhat-train_data.y)^2))
rmsein


yhat.boost <- predict(lm1,test_data)
r2outsample <- (sum((yhat.boost-mean(test_data.y))^2))/(sum((test_data.y-mean(test_data.y))^2))
r2outsample
lmerror <- sqrt(mean((yhat.boost-test_data.y)^2))
lmerror
Lmmae = mean(abs(yhat.boost-test_data.y))

hist(abs(yhat-train_data$count))
hist((yhat.boost-test_data$count)^2)

rownames(data.frame(train_data[,c("city","start_hour","month","wday","count")],predicted = yhat)[order(abs(yhat-
train_data$count),decreasing = T)[1:20],])



## check where and why model doesn'f fit well
a <- lm1$coefficients[-1]
lmcoef<- sort(a,decreasing = TRUE)
lmcoef
names(lmcoef)

boost.a =  data.frame(df_v2[-train_sample, c('date','month','wday','city','start_hour','count')],yhat.boost )
names(boost.a) = c("date",'month','wday','city','start_hour', "Actual", "Prediction")

summary(boost.a)
checksomething = boost.a[boost.a$month=="7",]
checksomething[order(abs(checksomething$Actual - checksomething$Prediction),decreasing = TRUE)[1:30],]
usecoef <- lm1$coefficients
usecoef[is.na(usecoef)]<-0
model.matrix(count~., data = df_v2[36232,-1]) %*% usecoef
model.matrix(count~., data = df_v2[36232,-1]) * usecoef

# the sea level pressure has very large coefficient, usually this is caused by multicollinearity,
```

```
# which can be solved by Lasso and rigid

## model prediction on new test data

boost.a = data.frame(df_v2[-train_sample, c('date','count')],yhat.boost )

boost.a = data.frame(df_v2[-train_sample, c('date','count')],yhat.boost )
names(boost.a) = c("date", "Actual", "Prediction")
boost.lm = reshape2::melt(boost.a, id = "date")
boost.lm = boost.lm %>% group_by(date,variable) %>% summarise(value = sum(value))
boost.lm$variable= as.factor(boost.lm$variable)


ggplot(boost.lm, aes(x = date, y = value, colour = variable)) + geom_line() +
  labs(x = "Date", y = "number of trips", title = "Predicted Values vs Actual Values") +
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "transparent",colour = NA),
        panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        plot.background = element_rect(fill = "transparent",colour = NA),
        legend.position="top")




#################################################################
##                           Lasso                             ##
#################################################################

#install.packages("glmnet")

x<-model.matrix(count~., data = train_data)
glmnetlasso<-cv.glmnet(x=x,y=train_data$count,type.measure='mse',nfolds=10,alpha=1)

## the key of Lasso and Ridgid are the choice regularization parameter Lambda
## here we use 10 folds cross validation to find the best regularization parameter for this dataset


plot(glmnetlasso,ylab = "MSE for Lasso")
names(glmnetlasso)

c<-coef(glmnetlasso,s='lambda.min',exact=TRUE)
c
inds<-which(c!=0)
variables<-row.names(c)[inds]
length(variables)
dim(x)


fitlasso <-glmnet(x = x, y = train_data$count, alpha = 1)
plot(fitlasso, xvar = "lambda")
abline(v = log(glmnetlasso$lambda.min))

## this plot shows how coefficient change with the choice of regularization parameter Lambda
## The far left are OLS and the far right are zero(too much regularization)
## We use Cross Validation to select the best Lambda
## howver, the selected Lambda is very close to left part
## this is saying that there are not much Multicollinearity in this problem, and Lasso cannot help much
## from this plot we will see the Lasso result would be very close to OLS




fit_lasso <-glmnet(x = x, y = train_data$count, alpha = 1, lambda = glmnetlasso$lambda.min)
summary(fit_lasso)
fit_lasso
Lassocoef <- fit_lasso$beta[match( names(lmcoef),rownames(fit_lasso$beta))]
Lassocoef

Lassosort<- sort(Lassocoef,decreasing = TRUE)[1:20]
Lassosort


newXin <- model.matrix(count~.,data=train_data)
yhat <- predict(fit_lasso,newx = newXin)
r2insample <- (sum((yhat-mean(train_data.y))^2))/(sum((train_data.y-mean(train_data.y))^2))
r2insample
rmsein <- sqrt(mean((yhat-train_data.y)^2))
rmsein


newX <- model.matrix(count~.,data=test_data)
yhat.boost <- predict(fit_lasso,newx = newX)


data.frame(train_data[,c("city","start_hour","month","count")],predicted = yhat)[order(abs(yhat-train_data$count),decreasing
= T)[1:20],]
data.frame(test_data[,c("city","start_hour","month","count")],predicted = yhat.boost)[order(abs(yhat.boost-
test_data$count),decreasing = T)[1:20],]
```

```
Lassoerror <- sqrt(mean((yhat.boost-test_data.y)^2))
Lassoerror
Lassomae = mean(abs(yhat.boost-test_data.y))

r2outsample <- (sum((yhat.boost-mean(test_data.y))^2))/(sum((test_data.y-mean(test_data.y))^2))
r2outsample

Lasso= data.frame(df_v2[-train_sample, c('date','city','start_hour','count')],yhat.boost )
names(Lasso) = c("date",'city','start_hour', "Actual", "Prediction")

Lasso[Lasso$Actual - Lasso$Prediction > 100,]

Lasso= data.frame(df_v2[-train_sample, c('date','count')],yhat.boost )
names(Lasso) = c("date", "Actual", "Prediction")



Lasso = melt(Lasso, id = "date")
Lasso$variable= as.factor(Lasso$variable)

Lasso = Lasso %>% group_by(variable,date) %>% summarize(value = sum(value))
hist(Lasso$value)

ggplot(Lasso, aes(x = date, y = value, colour = variable)) + geom_line() +
  labs(x = "Date", y = "number of trips", title = "LassoÃÂ¤Â²Ã¢ÃÂµÃÃ«ÃÃ¦ÃÂµÃÂµÂ±ÃÂ½Ã") +
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "transparent",colour = NA),
        panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        plot.background = element_rect(fill = "transparent",colour = NA),
        legend.position="top")



#################################################################
##                          Ridge                              ##
#################################################################

glmnetridge<-cv.glmnet(x=x,y=train_data$count,type.measure='mse',nfolds=10,alpha=0)

plot(glmnetridge,ylab= "MSE for Ridge")
names(glmnetridge)

c<-coef(glmnetridge,s='lambda.min',exact=TRUE)
inds<-which(c!=0)
variables<-row.names(c)[inds]

fit <-glmnet(x = x, y = train_data$count, alpha = 0)
plot(fit, xvar = "lambda")
abline(v = log(glmnetridge$lambda.min))


fit_ridge <-glmnet(x = x, y = train_data$count, alpha = 0, lambda = glmnetridge$lambda.min)
summary(fit_ridge)
ridgecoef <- fit_ridge$beta[match( names(lmcoef),rownames(fit_ridge$beta))]
ridgecoef

ridgesort<- sort(ridgecoef,decreasing = TRUE)[1:20]
ridgesort




newXin <- model.matrix(count~.,data=train_data)
yhat <- predict(fit_ridge,newx = newXin)


r2insample <- (sum((yhat-mean(train_data.y))^2))/(sum((train_data.y-mean(train_data.y))^2))

r2insample
rmsein <- sqrt(mean((yhat-train_data.y)^2))
rmsein

newX <- model.matrix(count~.,data=train_data)
yhat <- predict(fit_ridge,newx = newX)

newX <- model.matrix(count~.,data=test_data)
yhat.boost <- predict(fit_ridge,newx = newX)


data.frame(train_data[,c("city","start_hour","month","wday","count")],predicted = yhat)[order(abs(yhat-
train_data$count),decreasing = T)[1:20],]
data.frame(test_data[,c("city","start_hour","month","wday","count")],predicted = yhat.boost)[order(abs(yhat.boost-
test_data$count),decreasing = T)[1:20],]
```

```
Ridgeerror <- sqrt(mean((yhat.boost-test_data.y)^2))
Ridgeerror
Ridgemae = mean(abs(yhat.boost-test_data.y))

r2outsample <- (sum((yhat.boost-mean(test_data.y))^2))/(sum((test_data.y-mean(test_data.y))^2))
r2outsample

ridge = data.frame(df_v2[-train_sample, c('date','month','city','start_hour','count')],yhat.boost )
names(ridge) = c("date",'city', 'start_hour',"Actual", "Prediction")
ridge[ridge$Actual - ridge$Prediction>80,]

ridge = data.frame(df_v2[-train_sample, c('date','count')],yhat.boost )
names(ridge) = c("date", "Actual", "Prediction")

ridge = melt(ridge, id = "date")
ridge$variable= as.factor(ridge$variable)
dim(ridge)


ridge = ridge %>% group_by(date,variable) %>% summarise(value=sum(value))
hist(ridge$value)

ggplot(ridge, aes(x = date, y = value, colour = variable)) + geom_line() +
  labs(x = "Date", y = "number of trips", title = "Predicted Values vs Actual Values") +
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "transparent",colour = NA),
        panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        plot.background = element_rect(fill = "transparent",colour = NA),
        legend.position="top")



data.frame(Lmmae,Lassomae,Ridgemae)



##model comparision of OLS Lasso and Ridge

coefcomp1 = data.frame(VariableNames = names(lmcoef), OLS = as.numeric(lmcoef), Lasso = Lassocoef,Ridge = ridgecoef)
coefcomp1
coefcomp = coefcomp1[c(1:20),]
coefcomp

## cross validation RMSE on training data

cv_rmse_lm = cv_rmse_ridge = cv_rmse_lasso <-rep(0,10)
cv_mae_lm = cv_mae_ridge = cv_mae_lasso <-rep(0,10)

for(i in c(1:10))
{
  indx = c(unlist(flds[[i]]))
  cv_predict <- predict(lm1, train_data[indx,])
  cv_rmse_lm[i] <- sqrt(mean((cv_predict - train_data[indx,"count"])^2))
  cv_mae_lm[i] <- mean(abs(cv_predict - train_data[indx,"count"]))
  newXin <- model.matrix(count~.,data=train_data[indx,])
  cv_predict <- predict(fit_lasso, newXin)
  cv_rmse_lasso[i] <- sqrt(mean((cv_predict - train_data[indx,"count"])^2))
  cv_mae_lasso[i] <- mean(abs(cv_predict - train_data[indx,"count"]))
  cv_predict <- predict(fit_ridge, newXin)
  cv_rmse_ridge[i] <- sqrt(mean((cv_predict - train_data[indx,"count"])^2))
  cv_mae_ridge[i] <- mean(abs(cv_predict - train_data[indx,"count"]))

}

mean(cv_rmse_lm)
mean(cv_rmse_lasso)
mean(cv_rmse_ridge)
mean(cv_mae_lm)
mean(cv_mae_lasso)
mean(cv_mae_ridge)






################################################################
##                    Random Forest                           ##
################################################################



df_v2_dummy <- dummy.data.frame(df_v2, dummy.class="factor")

train_data = data.frame(df_v2_dummy[train_sample, -1])
test_data = data.frame(df_v2_dummy[-train_sample, -1])
train_data.y = df_v2_dummy$count[train_sample]
test_data.y = df_v2_dummy$count[-train_sample]
```

```
dim(train_data)
names(train_data)
dim(test_data)
length(train_data.y)
length(test_data.y)

library(h2o)
h2o.init(nthreads=-1,max_mem_size='6G')

trainHex<-as.h2o(train_data)
## Set up variable to use all features other than those specified here
features<-colnames(train_data)[!(colnames(train_data) %in% c("count"))]
## Train a random forest using all default parameters
Ntrees <- 50
j=1
boost.df <- NULL
for(i in c(10, 20, 30)){
rfHex <- h2o.randomForest(x=features,
                          y="count",
                          ntrees = 50,
                          max_depth = i,
                          training_frame=trainHex,
                          nfolds = 5)
if(j==1){boost.df <- data.frame(h2o.scoreHistory(rfHex)[-1,3:4])
names(boost.df) = c("number_of_trees",i)}
else{
  temp = data.frame(h2o.scoreHistory(rfHex)[-1,3:4])
  names(temp) = c("number_of_trees",i)
  boost.df = merge(boost.df, temp,by="number_of_trees",all=TRUE)}
j=j+1
summary(rfHex)
}


h2o.varimp_plot(rfHex)


error_rf = boost.df
error_rf = error_rf[complete.cases(error_rf), ]
names(error_rf) = c("ntree", "depth_10", "depth_20", "depth_30")
MSE_matrix_rf = melt(error_rf, id="ntree")

ggplot(MSE_matrix_rf, aes(x=ntree, y=value, colour = variable)) + geom_line() +
  labs(x="number of trees", y="Test MSE")


summary(rfHex)
## Load test data into cluster from R
testHex<-as.h2o(test_data)
## Get predictions out; predicts in H2O, as.data.frame gets them into R
Predicted_Value <-as.data.frame(h2o.predict(rfHex,testHex))


rderror <- sqrt(mean((Predicted_Value-test_data.y)^2))
rderror


a = data.frame(df_v2[-train_sample, c('date','count')], Predicted_Value)
names(a) = c("date", "Actual", "Prediction")

AP = melt(a, id = "date")
AP$variable= as.factor(AP$variabl)

AP = AP %>% group_by(date, variable) %>% summarise(value = sum(value))

hist(AP$value)

ggplot(AP, aes(x = date, y = value, colour = variable)) + geom_line() +
  labs(x = "Date", y = "number of trips", title = "RF Prediction Result") +
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "transparent",colour = NA),
        panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        plot.background = element_rect(fill = "transparent",colour = NA),
        legend.position="top")




##################################################################
##                    XGboost model                             ##
##################################################################


df_v2_dummy <- dummy.data.frame(df_v2, dummy.class="factor")

ix <- which( names(df_v2_dummy)  %in% c("date","count"))
eval_sample = sample(length(train_sample), length(df_v2)*0.2)

train_data_all = data.frame(df_v2_dummy[train_sample, -ix])
```

```
train_data = data.frame(train_data_all[-eval_sample,])
eval_data = data.frame(train_data[eval_sample,])
test_data = data.frame(df_v2_dummy[-train_sample, -ix])

train_data_all.y = df_v2_dummy$count[train_sample]
train_data.y = train_data_all.y[-eval_sample]
eval_data.y = train_data.y[eval_sample]
test_data.y = df_v2_dummy$count[-train_sample]


dtrain_all <- xgb.DMatrix(data = data.matrix(train_data_all), label=data.matrix(train_data_all.y))
dtrain <- xgb.DMatrix(data = data.matrix(train_data), label=data.matrix(train_data.y))
dtest <- xgb.DMatrix(data = data.matrix(test_data), label=data.matrix(test_data.y))
deval <- xgb.DMatrix(data = data.matrix(eval_data), label=data.matrix(eval_data.y))

watchlist <- list(eval = deval, train = dtrain)

Ntrees <- 50
boost.df <- data.frame(trees = c(1:Ntrees))
j=1

for(i in c(10,20,30)){
  bst <- xgb.train( data = dtrain,
                    max_depth=i,
                    eta = 0.15,
                    subsample = 0.7,
                    colsample_bytree = 0.7,
                    nthread = 8,
                    nrounds=Ntrees,
                    watchlist = watchlist,
                    eval_metric = "rmse",
                    objective = "reg:linear")
  j=j+1
  boost.df[,j] <-bst$evaluation_log$eval_rmse
}

names(bst)
bst$evaluation_log$eval_rmse

#model <- xgb.dump(bst , with.stats = T)
model <- xgb.dump(bst , with_stats = T)

model[1:10] #This statement prints top 10 nodes of the model

names <- dimnames(data.matrix(train_data))[[2]]
names

importance_matrix <- xgb.importance(names, model = bst)
importance_matrix

# Ã¥ÂÂ¶Ã¥ÂÂ¾
xgb.plot.importance(importance_matrix[1:20,])

error_boosting = boost.df
error_boosting

names(error_boosting) = c("ntree", "depth_10", "depth_20", "depth_30")
MSE_matrix = melt(error_boosting, id="ntree")

ggplot(MSE_matrix, aes(x=ntree, y=value, colour = variable)) + geom_line() +
  labs(x="number of trees", y="Validation MSE")


bst <- xgb.train( data = dtrain_all,
                  max_depth=30,
                  eta = 0.15,
                  subsample = 0.7,
                  colsample_bytree = 0.7,
                  nthread = 8,
                  nrounds=50,
                  watchlist = watchlist,
                  eval_metric = "rmse",
                  objective = "reg:linear")


yhat <- predict(bst,dtrain)
r2insample <- (sum((yhat-mean(train_data.y))^2))/(sum((train_data.y-mean(train_data.y))^2))
r2insample
rmsein <- sqrt(mean((yhat-train_data.y)^2))
rmsein


yhat.boost = predict(bst, dtest)
xgboosterror <- sqrt(mean((yhat.boost-test_data.y)^2))
xgboosterror
r2outsample <- (sum((yhat.boost-mean(test_data.y))^2))/(sum((test_data.y-mean(test_data.y))^2))
r2outsample

boost.a = data.frame(df_v2[-train_sample, c('date','count')], yhat.boost)
names(boost.a) = c("date", "Actual", "Prediction")
```

```
boost.AP = melt(boost.a, id = "date")
boost.AP$variable= as.factor(boost.AP$variable)


boost.AP = boost.AP %>% group_by(date,variable) %>% summarise(value = sum(value))

hist(boost.AP$value)

ggplot(boost.AP, aes(x = date, y = value, colour = variable)) + geom_line() +
  labs(x = "Date", y = "number of trips", title = "Predicted Values vs Actual Values") +
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "transparent",colour = NA),
        panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        plot.background = element_rect(fill = "transparent",colour = NA),
        legend.position="top")




check = data.frame(df_v2[-train_sample, c('date','count','city','start_hour')], yhat.boost)

data.frame(check %>% group_by(city,start_hour) %>% summarise(error= mean((count-yhat.boost)^2)))

check %>% group_by(city,date) %>% summarise(yhat.boost=sum(yhat.boost)) %>% ggplot(aes(date,yhat.boost,color = city)) +
geom_line()

##compare the RMSE of models

models <- c("OLS","Lasso","Ridge","Random Forest","XGboost")
models <- factor(models, as.character(models))
data.frame(Model =models ,
           Error = c(lmerror,Lassoerror,Ridgeerror,rderror,xgboosterror)) %>%
  ggplot( aes(Model, Error)) + geom_col()

#### discussion

## the reason of better prediction result of RF and GBDT is that this two tree-based model are able to
## capture the deep interaction between different variables. For example, we notice that from data visualization
## that the rush hour(8AM and 5PM) are high demend. However, the high demand only happens in weekday and
## only happens in San Francisco. So the interaction of time, city and weekday are important for prediction,
## which are hard to be captured in linear model(OLS, ridgid or Lasso).


data.frame(Model =models ,
           testError = c(lmerror,Lassoerror,Ridgeerror,rderror,xgboosterror))

dim(df_v2)
```