202A Final Project Code

Stella Huang

12/14/2020

Load Libraries

```
library(tidyverse)
library(dplyr)
library(lubridate)
library(bigmemory)
library(biganalytics)
library(bigtabulate)
library(caret)
library(corrplot)
library(anytime)
library(klaR)
library(devtools)
library(xgboost)
library(gbm)
library(leaps)
```

1. Data Cleaning

```
# load data
data <- read.csv("full_listings.csv") # there are 30533 listings


######################
### data cleaning ###
######################


# remove unnecessary variables
colnames(data)
col_drops <- c("last_scraped", "name", "description","listing_url", "scrape_id", "neighborhood_overview
               "host_name", "host_url", "picture_url", "host_about", "host_thumbnail_url", "host_pictur
               "host_location","host_neighbourhood", "host_has_profile_pic", "neighbourhood_group_clean
               "calendar_last_scraped", "license", "has_availability", "host_response_time", "neighbour

data <- data[ , !(names(data) %in% col_drops)]


# remove listings wit no rating
data <- data[!is.na(data$review_scores_rating),]
```

```r
data


############################################
### text conversion / feature engineering ###
############################################


### PRICE PROCESSING
# change price to numeric
data$price <- as.numeric(substr(data$price,2,nchar(data$price)-3))


### HOST INFO
# change percentages to numeric
data$host_response_rate[data$host_response_rate=="N/A"] <- "0%"
data$host_acceptance_rate[data$host_acceptance_rate=="N/A"] <- "0%"
data$host_response_rate <- as.numeric(substr(data$host_response_rate,1,nchar(data$host_response_rate)-1)
data$host_acceptance_rate <- as.numeric(substr(data$host_acceptance_rate,1,nchar(data$host_acceptance_ra
# count number of host verification documents and amenities
data <- data %>% mutate(num_host_verif = str_count(host_verifications, "\'")/2, num_amenities=str_count


### BATHROOM COUNT CONVERSION
# add bathroom num
# get first element, which is the number of bathrooms
data$bathrooms <- unlist(lapply(str_split(data$bathrooms_text, " "), `[[`, 1))
# check which entries are half baths, and replace with 0
data[data$bathrooms_text %in% c("Half-bath", "Private half-bath", "Shared half-bath"),]$bathrooms <- 0.5
# create variable to indicate if private or not (0:no, 1:yes)
data$is_private_bath <- ifelse(str_detect(data$bathrooms_text, "private"), 1, 0)
# convert to numeric
data$bathrooms <- as.numeric(data$bathrooms)


### ROOM TYPE CONVERSION
# check room type
table(data$room_type)
# assign number encodings to room type
room_type2 <- data$room_type
room_type2[room_type2 == "Shared room"] <- 1
room_type2[room_type2 == "Private room"] <- 2
room_type2[room_type2 == "Hotel room"] <- 3
room_type2[room_type2 == "Entire home/apt"] <- 4
data$room_type2 <- as.numeric(room_type2)


### MISSING VALUES
data$bedrooms[is.na(data$bedrooms)] <- 0
data$beds[is.na(data$beds)] <- 0
data$reviews_per_month[is.na(data$reviews_per_month)] <- 0
```

```
### TRUE/FALSE CONVERSION
# find such col
binary_feat <- c("host_is_superhost", "host_identity_verified", "instant_bookable")
# replace with 0/1 and convert to numeric
binary_feat_data <- data[names(data) %in% binary_feat] %>% mutate_all(funs(str_replace(., "t", "1"))
                                                  ) %>% mutate_all(funs(str_replace(., "f", "0"))) %>% m
data[names(data) %in% binary_feat] <- binary_feat_data


### DATE CONVERSION
data %>% select_if(is.character)
date_feat <- c("host_since", "first_review", "last_review")
# convert to date format
for(i in date_feat){
  data[,i] <- as.Date.character(mdy(data[,i]))
}


# calculate durations
data$host_years <- interval(data$host_since,lubridate::ymd("2020-11-23")) / years(1)
data$last_review_date <- interval(data$last_review,lubridate::ymd("2020-11-23")) / years(1)


### MAP NEIGHBORHOOD TO REGION
regions <- read.csv("la_regions.csv")
table(regions$NAME)
# find unmapped neighborhoods and replace names
data[!data$neighbourhood_cleansed %in% regions$NAME,]
data$neighbourhood_cleansed <- str_replace(data$neighbourhood_cleansed, "La Canada Flintridge", "La Caña
data$neighbourhood_cleansed <- str_replace(data$neighbourhood_cleansed, "East Whittier", "Whittier")
sum(!data$neighbourhood_cleansed %in% regions$NAME)==0
# add regions
data$region <- regions[match(data$neighbourhood_cleansed, regions$NAME),]$REGION
# combine angeles forest w/ antelope valley b/c there's only 9 obs in angeles forest
data$region <- str_replace(data$region,  "Angeles Forest", "Antelope Valley")
table(data$region)
```

2. Remove Correlated Features

```
################################
### remove unneeded features ###
################################
var_to_remove <- c("host_verifications", "amenities", "first_review", "last_review", "host_since", "bat
data <- data[!names(data) %in% var_to_remove]


###########################
### feature correlation ###
###########################
feat_leaveout <- c("id", "host_id", "latitude", "longitude")
# find correlated feats
cor_matrix <- cor((data[!names(data) %in% feat_leaveout] %>% select_if(is.numeric)), use="complete.obs"
```

```r
cor_matrix[upper.tri(cor_matrix)] <- 0 #no symmetry => redundant pairs
diag(cor_matrix) <- 0
# create dataframe of cor var
cor_df <- as.data.frame(as.table(cor_matrix))
# filter with 0.75+ correlation
correlated_var <- cor_df %>% filter(Freq >= 0.75) %>% filter(Var1!=Var2)%>% mutate_if(is.factor, as.cha
# feat_to_keep <- cor_df %>% filter(Freq >= 0.75) %>% filter(Var1!=Var2) %>% select(Var2) %>% pull()

# selected some feat to keep
correlated_var
feat_to_keep <- c("review_scores_rating", "host_listings_count", "bedrooms", "beds",
                  "minimum_nights", "maximum_nights_avg_ntm", "availability_60",
                  "review_scores_rating", "reviews_per_month")
corfeat_to_remove <- unique(c(correlated_var$Var1, correlated_var$Var2)[!c(correlated_var$Var1, correla
# remove correlated var
corfeat_to_remove
data <- data[!names(data) %in% corfeat_to_remove]
```

2.5 Count listings nearby

```r
# use complete data only
data <- subset(data, select = -c(neighbourhood_cleansed, property_type, room_type)) # remove more var
data <- data[complete.cases(data),]


## DISTANCE BASED METRIC
# wanna find the number of listings within 2 mile radius
library(geosphere)
dt.haversine <- function(lat_from, lon_from, lat_to, lon_to, r = 6378137){
    radians <- pi/180
    lat_to <- lat_to * radians
    lat_from <- lat_from * radians
    lon_to <- lon_to * radians
    lon_from <- lon_from * radians
    dLat <- (lat_to - lat_from)
    dLon <- (lon_to - lon_from)
    a <- (sin(dLat/2)^2) + (cos(lat_from) * cos(lat_to)) * (sin(dLon/2)^2)
    return(2 * atan2(sqrt(a), sqrt(1 - a)) * r)
}

num_proximity <- c()

for(i in c(1:nrow(data))){
# for(i in c(1:10)){
  samp1 <- as.matrix(data[i,c("longitude", "latitude")])
  other_samp <- as.matrix(data[-i,c("longitude", "latitude")])
  pair_distances <- distHaversine(samp1, other_samp)/1609
  # keep samples within 3 miles
  close_listings <- sum(pair_distances<=3)
  num_proximity[i] <- close_listings

}
```

```
summary(num_proximity)
hist(num_proximity, breaks=100)
data$num_proximity <- num_proximity

# ## 2D KERNEL REGRESSION
library(splancs)
library(maps)
latlon_bw <- sqrt(bw.nrd0(data$longitude)^2+bw.nrd0(data$latitude)^2)
latlon <- as.points(data$longitude, data$latitude)


boundary <- matrix(c(range(data$longitude)[1]-0.1,range(data$latitude)[1]-0.1,
                     range(data$longitude)[2]+0.1,range(data$latitude)[1]-0.1,
                     range(data$longitude)[2]+0.1,range(data$latitude)[2]+0.1,
                     range(data$longitude)[1]-0.1,range(data$latitude)[2]+0.1,
                     range(data$longitude)[1]-0.1,range(data$latitude)[1]-0.1),ncol=2,byrow=T)

latlon_dens <- kernel2d(latlon, boundary, latlon_bw)
par(mfrow=c(1,2))
image(latlon_dens, col=gray((64:20)/64),xlab="Latitude",ylab="Longitude", main="2D Kernel Smoothing of 
points(latlon, col='red')
map('county', 'California', add=T)
```

3. Exploratory Analysis

```
data <- read.csv("cleaned_data.csv")
full_data <- read.csv("full_listings.csv", stringsAsFactors = FALSE)
full_data <- full_data[match(data$id,full_data$id),]
data$local_host <- ifelse(str_detect(full_data$host_location, "Los Angeles"), 1, 0)

# convert char to factor
data[sapply(data, is.character)] <- lapply(data[sapply(data, is.character)], as.factor)
data <- model.frame(price~., data=data, drop.unused.levels = TRUE)
# filter region
data <- data %>% filter(region %in% c("Santa Monica Mountains", "Westside", "Central L.A.", "South L.A.

data$num_amenities

ggplot(aes(x=as.factor(data$room_type2), y=log(price), group=as.factor(data$room_type2)), data=data) +
  geom_boxplot(aes(fill = as.factor(data$room_type2))) + facet_grid(.~region) + theme_light() +
  ggtitle("Room Type vs Price") + xlab("room type") +
  scale_fill_discrete(name = "Room Type", labels = c("Shared","Private","Hotel", "Entire apt/home"))

ggplot(aes(x=price), data=data[data$price<600,]) + geom_histogram(bins=100) + facet_grid(.~region) + th

histogram(log(data$price))
histogram(data$price[data$price<700])
```

3.5. Kernel Regression

C Code

```c
# include <R.h>
# include <Rmath.h>
void kernel_reg(int *n, double *x, double *y, double *b, int* m,
    double *g2, double *res2)
{
    int i,j;
    double c;
    double est_sum_y;
    double est_sum;


    // loop through each grid index
    for(i=0; i<*m; i++){
        est_sum = 0.0;
        est_sum_y = 0.0;

        // calculate density; loop through each data index
        for(j=0; j<*n; j++){
            c = dnorm(x[j]-g2[i], 0, *b, 0)/ *n; //compute kernel at x on grid
            est_sum += c; // denominator
            est_sum_y += y[j] * c; //multiply by y and add; numerator
        }

        res2[i] = est_sum_y / est_sum;
    }
}
```

```r
# let's examine kernel regression density of price vs longitude
system("R CMD SHLIB kernel.c")
dyn.load("kernel.so")

# declare var
x <- data$longitude
y <- log(data$price)
xgrid <- c(seq(from = range(x)[1], to = range(x)[2], length.out=300))
bw <- bw.nrd(x)


# load function
gaussian_kernel <- function(x, y, bw, xgrid){
  n <- length(x)
  m <- length(xgrid)
  a=.C("kernel_reg", as.integer(n), as.double(x), as.double(y),
       as.double(bw), as.integer(m), as.double(xgrid), y=double(m))
  a$y
}

kernel_est <- gaussian_kernel(x,y,bw,xgrid)
plot(kernel_est~xgrid, type='l')
kernel_est
```

```
# resampling to get estimate
est_matrix <- c()
for(i in c(1:200)){
  new_samples <- sample_n(data, 2000, replace=TRUE)
  new_y <- as.vector(log(new_samples$price))
  new_x <- as.vector(new_samples$longitude)

  new_kernel_est <- gaussian_kernel(new_x, new_y, bw, xgrid)
  est_matrix <- cbind(est_matrix, new_kernel_est)

}

# extract 5th largest and 195th largest: order each row -> get 5th & 195th col of matrix
dim(est_matrix)
ci95 <- t(apply(est_matrix, 1, sort))[, c(10,195)]
colnames(ci95) <-c("lower", "upper")

kernelest_df <- as.data.frame(cbind(xgrid, kernel_est, ci95))

# plot KDE with 95% CI
ggplot(aes(x=xgrid, y=kernel_est), data=kernelest_df) + geom_point() +
  geom_line() + geom_ribbon(data=kernelest_df,aes(ymin=lower,ymax=upper),alpha=0.3) +
  xlab("Longitude") + ylab("Estimated log(Price)") + ggtitle("KDE Price with 95% CI Band") + theme_mini
```

4. Train test split

```
# write.csv(data,"cleaned_data.csv")
data <- read.csv("cleaned_data.csv")
full_data <- read.csv("full_listings.csv", stringsAsFactors = FALSE)
full_data <- full_data[match(data$id,full_data$id),]
data$local_host <- ifelse(str_detect(full_data$host_location, "Los Angeles"), 1, 0)

# convert char to factor
data[sapply(data, is.character)] <- lapply(data[sapply(data, is.character)], as.factor)
data <- model.frame(price~., data=data, drop.unused.levels = TRUE)
# filter region
data <- data %>% filter(region %in% c("Santa Monica Mountains", "Westside", "Central L.A.", "South L.A.

# train test split
train_index <- createDataPartition(data$price, p = .7, list = FALSE, times = 1)
train_data <- data[train_index,]
train_y <- train_data$price
train_data <- subset(train_data, select = -c(price, id, host_id))

test_data <- data[-train_index,]
test_y <- test_data$price
test_data <- subset(test_data, select = -c(price, id, host_id))
```

Function to calculated adjusted r^2

```
adj_r2 <- function(r2, n, p){
  adj = 1-((1-r2)*(n-1)/(n-p-1))
  return (adj)
}
```

5. Linear Model

```
#################################
### BUILDING LINEAR MODEL ######
#################################
# drop unused levels in train data
train_data <- model.frame(train_price~., data=train_data, drop.unused.levels = TRUE)
linearmodel <- lm(train_y~., data=train_data) # original var
linearmodel <- lm(log(train_y)~., data=train_data) # transformed var
summary(linearmodel)
par(mfrow=c(2,2))
plot(linearmodel)
mtext(text=expression(bold("Residual Plot for Linear Model, Log(y)")), side = 3, line = -14, outer = TRU

# bc <- boxcox(linearmodel)
# lam1 <- bc$x[which.max(bc$y)]
# linearmodel <- lm(((train_price^lam1) - 1) / lam1~., data=train_data)

lm_predictions <- predict(linearmodel, test_data)
postResample(pred = lm_predictions, obs = log(test_y))


# USE CV
fit.control <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
set.seed(123)
# train linear model with CV
lm_caret <- train(log(train_y)~.,train_data, method = "lm", trControl = fit.control)
summary(lm_caret)
lm_predictions <- predict(lm_caret, test_data)
# calculate prediction accuracy
postResample(pred = lm_predictions, obs = log(test_y))
adj_r2(postResample(pred = lm_predictions, obs = log(test_y))[2], nrow(test_data),36)
```

6. Step-wise regression

```
# step wise selection with leaps library
sw_linearmod <- regsubsets(log(train_y)~., data=train_data, nvmax = 27, method = "seqrep")
reg_summary <- summary(sw_linearmod)
par(mfrow = c(2,2))
plot(reg_summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
plot(reg_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")

# We will now plot a red dot to indicate the model with the largest adjusted R^2 statistic.
```

```r
# The which.max() function can be used to identify the location of the maximum point of a vector
adj_r2_max = which.max(reg_summary$adjr2) # 11

# The points() command works like the plot() command, except that it puts points
# on a plot that has already been created instead of creating a new plot
points(adj_r2_max, reg_summary$adjr2[adj_r2_max], col ="red", cex = 2, pch = 20)

# We'll do the same for C_p and BIC, this time looking for the models with the SMALLEST statistic
plot(reg_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
cp_min = which.min(reg_summary$cp) # 10
points(cp_min, reg_summary$cp[cp_min], col = "red", cex = 2, pch = 20)

plot(reg_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
bic_min = which.min(reg_summary$bic) # 6
points(bic_min, reg_summary$bic[bic_min], col = "red", cex = 2, pch = 20)
mtext(text=expression(bold("Step-Wise Regression Performance")), side = 3, line = -1.5, outer = TRUE)

# choose best number of var to include
# find such coefficients
sw_bestvar <- names(coef(sw_linearmod, 23))


fit.control <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

set.seed(123)
# fit model with best subset selected
lm_caret <- train(log(train_y)~.,train_data[,names(train_data) %in% c(sw_bestvar, "region")],
                  method = "lm", trControl = fit.control)
lm_summary <- as.data.frame((exp(coef(summary(lm_caret)))-1)*100)
options(scipen=999)
lm_summary[order(lm_summary$Estimate),]

# calculate accuracy
lm_predictions <- predict(lm_caret, test_data)
postResample(pred = lm_predictions, obs = log(test_y))
adj_r2(postResample(pred = lm_predictions, obs = log(test_y))[2], nrow(test_data),36)
```

7. Recursive Feature Elimination

```r
################################
# ### RFE FOR LINEAR MODEL #######
# ################################
ctrl <- rfeControl(functions = lmFuncs,
                   method = "repeatedcv",
                   repeats = 5,
                   number = 3,
                   verbose = FALSE)


lmProfile <- rfe(x=train_data%>%select_if(is.numeric), y=log(train_y), sizes = c(1:20), rfeControl = ct

# rank of variables
```

```r
lmProfile$variables
best_rfe_lm <- lmProfile$variables %>% filter(Variables==32)

# sapply(best_rfe_lm, mean)

best_rfe_lm_var <- best_rfe_lm %>% filter(var %in% c("bedrooms", "room_type2", "longitude", "is_private_

best_rfe_lm_var$var <- str_replace(best_rfe_lm_var$var,
                                   "calculated_host_listings_count_shared_rooms","host_sharedRo
                                   oms_cnt")
best_rfe_lm_var$var <- str_replace(best_rfe_lm_var$var,"room_type2","room type")
ggplot(aes(x=reorder(var, Overall), y=Overall), data=best_rfe_lm_var) + geom_bar(stat="identity", fill=
+ coord_flip() + theme_minimal() + geom_text(aes(label=round(Overall,2)), hjust=1.1, size=3.5, col="whi
+ ggtitle("Variable Importance from LM RFE") + xlab("Importance") + ylab("Variable")
```

8. Random Forest

```r
#################################
### RANDOM FOREST REGRESSION ###
#################################
library(randomForest)
library(e1071)

mtry <- round(ncol(train_data)/3)
tunegrid <- expand.grid(.mtry=mtry)
train.control <- trainControl(method = "repeatedcv", number = 5, repeats=3)

# Train the model
rf_model <- train(log(train_y)~., data=train_data,
                  method = "rf", tuneGrid=tunegrid,
                  trControl = train.control, importance=T)

# RESULTS
print(rf_model)
#################################
# 8597 samples
#   32 predictor
#
# No pre-processing
# Resampling: Cross-Validated (5 fold, repeated 3 times)
# Summary of sample sizes: 6877, 6877, 6877, 6878, 6879, 6878, ...
# Resampling results
#
#   RMSE       Rsquared   RMSE SD      Rsquared SD
#   0.3498099  0.7528557  0.007383131  0.01263049
#
# Tuning parameter 'mtry' was held constant at a value of 11
#################################


rf_model$finalModel #output below
#################################
```

```r
# Call:
#  randomForest(x = x, y = y, mtry = param$mtry, importance = ..1)
#                Type of random forest: regression
#                      Number of trees: 500
# No. of variables tried at each split: 11
#
#           Mean of squared residuals: 0.1196127
#                     % Var explained: 75.54
################################

# make prediction
rf_predictions <- predict(rf_model, test_data, type="raw")
postResample(pred = rf_predictions, obs = log(test_y))
adj_r2(postResample(pred = rf_predictions, obs = log(test_y))[2], nrow(train_data), 32)


# VARIABLE IMPORTANCE
write.csv(varImp(rf_model, scale = TRUE)$importance, "price_rfImp.csv")
rf_imp <- read.csv("price_rfImp.csv")

# plot variable importance
ggplot(aes(x=Overall), data=varImp(rf_model, scale = TRUE)) + geom_bar(stat="identity", fill="steelblue
  coord_flip() + theme_minimal() + geom_text(aes(label=round(Importance,2)), hjust=1.1, size=3.5, col="
  ggtitle("Variable Importance from Random Forest") + xlab("Importance") + ylab("Variable")

ggplot(aes(x=reorder(X, Overall), y=Overall), data=(rf_imp %>% arrange(desc(Overall)))[1:7,]) +
  geom_bar(stat="identity", fill="steelblue") + coord_flip() + theme_minimal() +
  geom_text(aes(label=round(Overall,2)), hjust=1.1, size=3.5, col="white") + ggtitle("Variable Importanc
  xlab("Importance") + ylab("Variable")
```