

# Project 5: Corporación Favorita Grocery Sales Forecasting - Group 1

*Shiqi Duan, Peter Li*

*2017/12/4*

## Contents

<b>Project 5: Corporación Favorita Grocery Sales Forecasting - Group 1</b>	<b>1</b>
Step 0: Load the packages, specify directories . . . . .	2
Step 1: Load and process the data . . . . .	2
Step 2: The First Stack . . . . .	4
Step 2.1 ETS (Exponential Smoothing State Space Model) . . . . .	4
Step 2.2 ARIMA . . . . .	7
Step 2.3 Prophet . . . . .	8
Step 2.4 XgBoost . . . . .	9
Step 2.5 Random Forest . . . . .	11
Step 2.6 Combine Features . . . . .	14
Step 3: The Second Stack . . . . .	14

## Project 5: Corporación Favorita Grocery Sales Forecasting - Group 1

Background: Brick-and-mortar grocery stores are always in a delicate dance with purchasing and sales forecasting. Predict a little over, and grocers are stuck with overstocked, perishable goods. Guess a little under, and popular items quickly sell out, leaving money on the table and customers fuming.

The problem becomes more complex as retailers add new locations with unique needs, new products, ever transitioning seasonal tastes, and unpredictable product marketing. Corporación Favorita, a large Ecuadorian-based grocery retailer, knows this all too well. They operate hundreds of supermarkets, with over 200,000 different products on their shelves.

Corporación Favorita has challenged the Kaggle community to build a model that more accurately forecasts product sales. They currently rely on subjective forecasting methods with very little data to back them up and very little automation to execute plans. They're excited to see how machine learning could better ensure they please customers by having just enough of the right products at the right time.

Reference: [Kaggle Website](#)

Data: As the datasets on Kaggle are too large, to the nature of the method, we only implement our method on a subset of the data. We work on the training data to find the top 10 stores and top 200 items in terms of observation frequency, and choose the subset of data related to these items and stores to use. We split our data into a training set(20140816-20160815) and a test set(20160816-20170815), and split the training set into a subtrain set (20140816-20160415) and a validation set(20160416-20160815).

Method:

Overall we use Stacking method. That is, we build two layers. In the first layer, we obtain the prediction results from each first-layer model and use these results as features in the second layer to get the final prediction values.

In the first layer, we take use of both Time Series models as well as supervised machine learning models. For Time Series models, we first train them on the subtrain set with different parameters and use the models on the validation set to compare the performance. Then we get the best model and use that to predict the sales on validation set and test set, and call these results as `features_valid` and `features_test` respectively. For machine learning models, we train the models using cross-validation on the whole training set and get the best model. Then we also use the best model to do the same thing on validation set and test set.

In the second layer, we use machine learning models (Random Forest) on the `features_valid` by cross-validation to find the best model. Then use the best model on `feature_test` to get the final predicted sales.

Evaluation: We use the same evaluation formula as on Kaggle [evaluation formula](#).

## Step 0: Load the packages, specify directories

```
packages.used=c("data.table", "lubridate", "dplyr", "date", "reshape2", "forecast", "doMC", "foreach", "xgboost")
packages.needed=setdiff(packages.used, intersect(installed.packages()[,1], packages.used))
if(length(packages.needed)>0){
  install.packages(packages.needed, dependencies = TRUE)
}

library(data.table)
library(lubridate)
library(date)
library(reshape2)
library(forecast)
library(doMC)
library(foreach)
library(dplyr)
library(xgboost)
library(readr)
library(randomForest)
library(prophet)
library(caret)
library(mlr)
library(tseries)
library(doParallel)

setwd("~/Documents/GitHub/fall2017-project5-group1/doc")
```

## Step 1: Load and process the data

As the `train.csv` on Kaggle is too large to upload on GitHub, you can download it from [Kaggle\\_data](#), and save it together with `items.csv`, `oil.csv`, and `holidays_events.csv` on local GitHub `data/original data` folder. Then you can use the following code to preprocess the data.

```
run.pro = FALSE #if TRUE, preprocess the original data
if(run.pro){
```

```

whole <- fread("../data/original data/train.csv")
train <- whole[whole$date < "2016-08-16"&whole$date>="2014-08-15",]
test <- whole[whole$date > "2016-08-15",]
rm(whole)

# chose top 10 stores and top 200 items with respect to observations
store_tbl <- table(train$store_nbr)
store_max <- store_tbl[order(store_tbl, decreasing = T)][1:10]
item_tbl <- table(train$item_nbr)
item_max <- item_tbl[order(item_tbl, decreasing = T)][1:200]

store_test <- test$store_nbr[test$store_nbr %in% names(store_max)]
item_test <- test$item_nbr[test$item_nbr %in% names(item_max)]
store_test_perfm <- table(store_test)
item_test_perfm <- table(item_test)

train <- train[train$store_nbr %in% names(store_max) & train$item_nbr %in% names(item_max),]
test <- test[test$store_nbr %in% names(store_max) & test$item_nbr %in% names(item_max),]

train$date <- as.Date(parse_date_time(train$date, '%y-%m-%d'))
train$store_item_nbr <- paste(train$store_nbr, train$item_nbr, sep="_")
test$date <- as.Date(parse_date_time(test$date, '%y-%m-%d'))
test$store_item_nbr <- paste(test$store_nbr, test$item_nbr, sep="_")

write.csv(train, "../data/our_train.csv")
write.csv(test, "../data/our_test.csv")

## combine with other useful variables
items<-read.csv("../data/original data/items.csv",header=TRUE)
holidays <- read.csv("../data/original data/holidays_events.csv",header = TRUE)
oil <- read.csv("../data/original data/oil.csv",header=TRUE)

train1 <- train %>%
  mutate(year = year(ymd(date))) %>%
  mutate(month = month(ymd(date))) %>%
  mutate(dayOfWeek = wday(date)) %>%
  mutate(day = day(ymd(date)))
train1 <- merge(train1, items, by.x = "item_nbr", by.y = "item_nbr")

holidaysNational = holidays %>%
  filter(type != "Work Day") %>%
  filter(locale == "National")
holidaysNational <- holidaysNational%>%select(date,type,transferred)
holidaysNational$celebrated <- ifelse(holidaysNational$transferred == "True", FALSE, TRUE)
holidaysNational$date <- as.Date(parse_date_time(holidaysNational$date, '%y-%m-%d'))
train_comb = left_join(train1,holidaysNational,by='date')

oil$date <- as.Date(parse_date_time(oil$date, '%y-%m-%d'))
train_comb <- left_join(train_comb, oil, by='date')

write.csv(train_comb, "../output/combined_train.csv")

test1 <- test %>%

```

```

mutate(year = year(ymd(date))) %>%
mutate(month = month(ymd(date))) %>%
mutate(dayOfWeek = wday(date)) %>%
mutate(day = day(ymd(date)))
test1 <- merge(test1, items, by.x = "item_nbr", by.y = "item_nbr")

test_comb = left_join(test1, holidaysNational, by='date')
test_comb <- left_join(test_comb, oil, by='date')

write.csv(test_comb, "../output/combined_test.csv")

## split train into subtrain and validation sets
sub_train <- train%>%filter(date < "2016-04-16")
valid_train <- train%>%filter(date > "2016-04-15")

write.csv(sub_train, "../output/subtrain.csv")
write.csv(valid_train, "../output/validation.csv")
}else{
train <- fread("../output/combined_train.csv")
test <- fread("../output/combined_test.csv")
sub_train <- train%>%filter(date < "2016-04-16")
valid_train <- train%>%filter(date > "2016-04-15")
}

```

```

##
Read 0.0% of 1428019 rows
Read 51.1% of 1428019 rows
Read 79.8% of 1428019 rows
Read 1428019 rows and 19 (of 19) columns from 0.141 GB file in 00:00:05

```

## Step 2: The First Stack

In the first stack, we use three Time Series models: ETS, ARIMA, and Prophet, as well as two machine learning models: XgBoost, and Random Forest.

### Step 2.1 ETS (Exponential Smoothing State Space Model)

In this model, we tune the parameter lambda of [Box-Cox Transformation](#), as other parameters in this model can be automatically estimated by the `ets()` function in R.

```

run.ets = FALSE #if TRUE, run train-validation process on ETS model and run the prediction on test using

if(run.ets){
train_valid <- valid_train
train_sub <- sub_train[, c('date', 'store_item_nbr', 'unit_sales')]
train_sub_wide <- dcast(train_sub, store_item_nbr ~ date, mean, value.var = "unit_sales", fill = 0)
train_ts <- ts(train_sub_wide, frequency = 7)

fcst_intv = length(unique(train_valid$date)) # number of days of forecast in the validation set
fcst_matrix <- matrix(NA, nrow=nrow(train_ts), ncol=fcst_intv)

```

```

# train the models by forecasting sales in validation set
lam.ranges <- seq(0.1, 5, by = 0.2)
valid_score <- rep(NA, length(lam.ranges))
for (i in 1:length(lam.ranges)){
  lam <- lam.ranges[i]
  registerDoMC(detectCores()-1)
  fcst_matrix <- foreach(i=1:nrow(train_ts),.combine=rbind, .packages=c("forecast")) %dopar% {
    fcst_matrix <- forecast(ets(train_ts[i,], lambda = lam),h=fcst_intv)$mean
  }
  colnames(fcst_matrix) <- as.character(seq(from = as.Date("2016-04-16"),
                                          to = as.Date("2016-08-15"),
                                          by = 'day'))
  fcst_df <- as.data.frame(cbind(train_sub_wide[, 1], fcst_matrix))
  colnames(fcst_df)[1] <- "store_item_nbr"

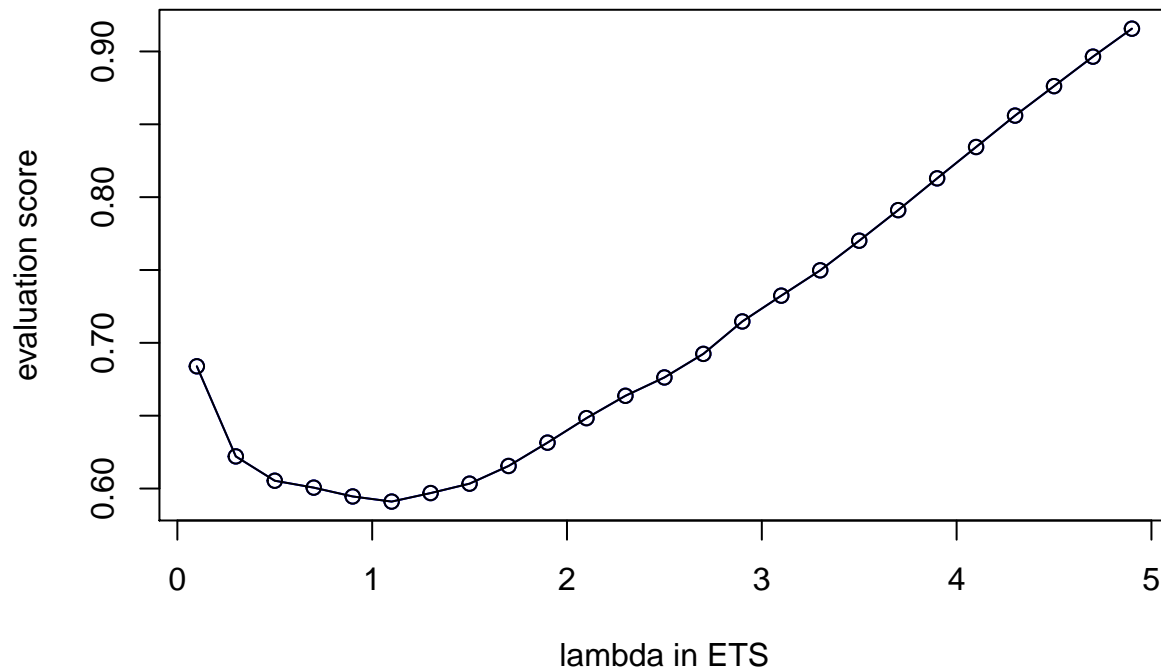
  fcst_df_long <- melt(fcst_df, id = 'store_item_nbr',
                      variable.name = "fcst_date",
                      value.name = 'unit_sales')
  fcst_df_long$store_item_nbr <- as.character(fcst_df_long$store_item_nbr)
  fcst_df_long$fcst_date <- as.Date(parse_date_time(fcst_df_long$fcst_date, '%y-%m-%d'))
  fcst_df_long$unit_sales <- as.numeric(fcst_df_long$unit_sales)
  colnames(fcst_df_long)[3] <- "sales_pred"

  train_valid$date <- as.Date(parse_date_time(train_valid$date, '%y-%m-%d'))
  train_comb <- left_join(train_valid, fcst_df_long,
                        c("store_item_nbr" = "store_item_nbr", 'date' = 'fcst_date'))
  train_comb$sales_pred[train_comb$sales_pred < 0] <- 0
  train_comb$unit_sales[train_comb$unit_sales < 0] <- 0

  train_save <- train_comb[,c(1:6,8,ncol(train_comb))]
  save(train_save, file = paste0("../output/ets/ets_lambda_",lam.ranges[i],".RData"))
  # calculate the score of accuracy prediction on validation set
  w <- ifelse(train_comb$perishable == 0, 1, 1.25)
  valid_score[i] <- sqrt(sum(w * (log(train_comb$sales_pred + 1) - log(train_comb$unit_sales + 1))^2))
}
save(valid_score, file = "../output/valid_score.RData")
}else{
  load("../output/ets/valid_score.RData")
  lam.ranges <- seq(0.1, 5, by = 0.2)
}

# the best parameter lambda
plot(lam.ranges, valid_score, col="blue", xlab="lambda in ETS", ylab="evaluation score",type="o")
points(lam.ranges,valid_score,type="o")

```



```
ets.par <- lam.ranges[which.min(valid_score)]
print(paste0("the best lambda in ETS is ", ets.par))
```

```
## [1] "the best lambda in ETS is 1.1"
```

```
# the features_valid corresponding to ets.par
load(paste0("../output/ets/ets_lambda_", ets.par, ".RData"))
valid_ets <- train_save%>%select(V1, id, sales_pred)
colnames(valid_ets)[3] <- "ets_pred"

# the features_test corresponding to ets.par
if(run.ets){
  # performance on test dataset
  test$date <- as.Date(parse_date_time(test$date, '%y-%m-%d'))

  train_sub_wide <- dcast(train, store_item_nbr ~ date, mean, value.var = "unit_sales", fill = 0)
  train_ts <- ts(train_sub_wide, frequency = 7)

  fcst_intv = 365 # number of days of forecast in the test set
  fcst_matrix <- matrix(NA, nrow=nrow(train_ts), ncol=fcst_intv)

  # forecast the sales in test set use the best ets.par
  registerDoMC(detectCores()-1)
  fcst_matrix <- foreach(i=1:nrow(train_ts), .combine=rbind, .packages=c("forecast")) %dopar% {
    fcst_matrix <- forecast(ets(train_ts[i,], lambda = ets.par), h=fcst_intv)$mean
  }
  colnames(fcst_matrix) <- as.character(seq(from = as.Date("2016-08-16"),
                                             to = as.Date("2017-08-15"),
                                             by = 'day'))
  fcst_df <- as.data.frame(cbind(train_sub_wide[, 1], fcst_matrix))
  colnames(fcst_df)[1] <- "store_item_nbr"
```

```

fcst_df_long <- melt(fcst_df, id = 'store_item_nbr',
                    variable.name = "fcst_date",
                    value.name = 'unit_sales')
fcst_df_long$store_item_nbr <- as.character(fcst_df_long$store_item_nbr)
fcst_df_long$fcst_date <- as.Date(parse_date_time(fcst_df_long$fcst_date, '%y-%m-%d'))
fcst_df_long$unit_sales <- as.numeric(fcst_df_long$unit_sales)
colnames(fcst_df_long)[3] <- "sales_pred"

test_comb <- left_join(test, fcst_df_long,
                      c("store_item_nbr" = "store_item_nbr", 'date' = 'fcst_date'))
test_comb$sales_pred[test_comb$sales_pred < 0] <- 0
test_comb$unit_sales[test_comb$unit_sales < 0] <- 0

save(test_comb, file = "../output/ets/test_comb.RData")
}else{
  load("../output/ets/test_comb.RData")
}
test_ets <- test_comb%>%select(V1, id, sales_pred)
colnames(test_ets)[3] <- "ets_pred"

```

## Step 2.2 ARIMA

```

run.arima = FALSE #if TRUE, train ARIMA model and predict using ARIMA model on main.Rmd
if(run.arima){
  subtrain1 = subtrain[ ,c("date","store_item_nbr","unit_sales")]
  subtrain1$Date = as.Date(subtrain$date)
  validation = valid_train[ ,c("date","store_item_nbr","unit_sales","perishable")]

  train_sub = dcast(subtrain1, store_item_nbr ~ Date, mean, value.var = "unit_sales", fill = 0)
  a = seq(from=as.Date("2016-04-16"), to=as.Date("2016-08-15"),by = 'day')
  wide = length(a)

  #Predict ARIMA(Try several times here is the best parameters)
  fc.wiki = foreach(i=1:nrow(train_sub), .combine=rbind, .packages="forecast") %dopar% {
    y = tsclean(as.ts(unlist(train_sub[i, -1])))
    forecast(auto.arima(y, max.p=2, max.d=2, max.q=1), h=wide)$mean
  }

  colnames(fc.wiki) = as.character(seq(from = as.Date("2016-04-16"),
                                       to = as.Date("2016-08-15"),
                                       by = 'day'))

  train_result = cbind(train_sub[, 1], fc.wiki)
  rownames(train_result) = rownames(train_sub)
  colnames(train_result)[1] = "store_item_nbr"
  subtrain_result = as.data.frame(train_result)

  #Reshaped the predict dataset again
  train_result_long = melt(subtrain_result, id = "store_item_nbr", variable.name = "Date", value.name =
                           "sales_pred")
  train_result_long$Date = as.Date(train_result_long$Date)

```

```

train_result_long$sales_pred = as.numeric(train_result_long$sales_pred)
#Get the final results dataframe and save as RData
train_comb = left_join(validation, train_result_long,
                        c("store_item_nbr" = "store_item_nbr", 'date' = 'Date'))
train_comb$sales_pred[train_comb$sales_pred < 0] = 0
train_comb$unit_sales[train_comb$unit_sales < 0] = 0
save(train_comb, file = "ARIMA Feature validate.RData")

#Used the training result to the test set
wide = length(unique(test$date))
fc.wiki = foreach(i=1:nrow(train_sub), .combine=rbind,
                  .packages="forecast") %dopar% {
  y = tsclean(as.ts(unlist(train_sub[i, -1])))
  forecast(auto.arima(y, max.p=2, max.d=2, max.q=1), h=wide)$mean
}
colnames(fc.wiki) = as.character(unique(test$date))
test_result = cbind(train_sub[, 1], fc.wiki)
colnames(test_result) [1] = "store_item_nbr"
test_result = as.data.frame(test_result)

test_result_long = melt(test_result, id = "store_item_nbr", variable.name = "Date", value.name =
                        "sales_pred")
test_result_long$Date = as.Date(test_result_long$Date)

test_result_long$sales_pred = as.numeric(test_result_long$sales_pred)
test$date = as.Date(test $ date)
test_comb = left_join(test, test_result_long,
                      c("store_item_nbr" =
                        "store_item_nbr", 'date' = 'Date'))

save(test_comb, file = "ARIMA Feature test.RData")
}else{
  load("../output/ARIMA Feature validate.RData")
  load("../output/ARIMA Feature Test.RData")
}
valid_arima <- train_comb[,c(2,4,21)]
colnames(valid_arima)[3] <- "arima_pred"
test_arima <- test_comb[,c(1:3, 8, 9)]
colnames(test_arima)[5] <- "arima_pred"

```

### Step 2.3 Prophet

Prophet is an open-source modeling framework developed at Facebook used to provide automated forecasting specifically aimed at solving problems regarding efficient allocation of resources over time for capacity planning. Due to the nature of how the automated forecasting is structured, the function tunes the parameters, so no parameter selection is needed from us.

```

run.prophet = FALSE #if TRUE, run train process on Prophet model and run the prediction on test using b

load(paste0("../output/prophet_changepoint_0.1.RData"))
valid_prophet <- train_save%>%select(V1, id, sales_pred)
colnames(valid_prophet)[3] <- "prophet_pred"

```



```

# the features_test corresponding to ets.par
if(run.prophet){
  # performance on test dataset
  test$date <- as.Date(parse_date_time(test$date, '%y-%m-%d'))

  train_sub_wide <- dcast(train, store_item_nbr ~ date, mean, value.var = "unit_sales", fill = 0)
  train_ts <- ts(train_sub_wide, frequency = 7)

  fcst_intv = 365 # number of days of forecast in the test set
  fcst_matrix <- matrix(NA, nrow=nrow(train_ts), ncol=fcst_intv)

  # forecast the sales in test set use the best ets.par
  registerDoMC(detectCores()-1)
  fcst_matrix <- foreach(i=1:nrow(train_ts), .combine=rbind, .packages=c("forecast")) %dopar% {
tmp = data.frame(ds = colnames(train_ts),
                  y = train_ts[i,])
    m <- prophet(tmp, changepoint.prior.scale = 0.1)
    future <- make_future_dataframe(m, periods = fcst_intv, freq = "day")

    fcst_matrix1 <- tail(predict(m, future)$yhat, fcst_intv)
  }
  colnames(fcst_matrix) <- as.character(seq(from = as.Date("2016-08-16"),
                                             to = as.Date("2017-08-15"),
                                             by = 'day'))
  fcst_df <- as.data.frame(cbind(train_sub_wide[, 1], fcst_matrix))
  colnames(fcst_df)[1] <- "store_item_nbr"

  fcst_df_long <- melt(fcst_df, id = 'store_item_nbr',
                      variable.name = "fcst_date",
                      value.name = 'unit_sales')
  fcst_df_long$store_item_nbr <- as.character(fcst_df_long$store_item_nbr)
  fcst_df_long$fcst_date <- as.Date(parse_date_time(fcst_df_long$fcst_date, '%y-%m-%d'))
  fcst_df_long$unit_sales <- as.numeric(fcst_df_long$unit_sales)
  colnames(fcst_df_long)[3] <- "sales_pred"

  test_comb <- left_join(test, fcst_df_long,
                        c("store_item_nbr" = "store_item_nbr", 'date' = 'fcst_date'))
  test_comb$sales_pred[test_comb$sales_pred < 0] <- 0
  test_comb$unit_sales[test_comb$unit_sales < 0] <- 0

  save(test_comb, file = "../output/prophet_test_comb.RData")
}
load("../output/prophet_test_comb.RData")
test_prophet <- test_comb%>%select(V1, id, sales_pred)
colnames(test_prophet)[3] <- "prophet_pred"

```

## Step 2.4 XgBoost

```

run.xgboost = FALSE #if TRUE, run all the processing, cross-validation, and prediction process on XgBoost

# process data function
df_format <- function(df){

```

```

df$onpromotion[df$onpromotion == FALSE] <- 0
df$onpromotion[df$onpromotion == TRUE] <- 1

df$unit_sales[df$unit_sales < 0] <- 0
df$log_sales <- log(df$unit_sales + 1)
return(df)
}

if(run.xgboost){
  # set core number and seed
  registerDoMC(cores=4)
  set.seed(6)
  # process data for XgBoost
  combined_train <- df_format(train)
  combined_test <- df_format(test)
  combined_valid <- df_format(valid_train)

  features <- c('item_nbr', 'id', 'store_nbr', 'onpromotion', 'year', 'month', 'dayOfWeek', 'class',
                'perishable', 'dcoilwtico')

  dtrain <- xgb.DMatrix(data = as.matrix(combined_train[, features]), label = combined_train$log_sales)
  dtest <- xgb.DMatrix(data = as.matrix(combined_test[, features]), label = combined_test$log_sales)
  dvalid <- xgb.DMatrix(data = as.matrix(combined_valid[, features]), label = combined_valid$log_sales)

  # watchlist, when training we can see the performance on both train and test sets
  watchlist <- list(train=dtrain, test = dtest)

  # gridsearch parameters, when tuning you can specify a list of values to test for
  # each parameter. The current parameters in this grid represent the best values
  # for our training set
  searchGridSubCol <- expand.grid(subsample = c(0.7),
                                colsample_bytree = c(0.7),
                                max_depth = c(8),
                                # min_child = c(1, 5, 10),
                                min_child = c(10),
                                eta = c(0.7),
                                # gamma = c(0.1, 1, 10)
                                gamma = c(1))

  ntrees <- 40

  rmseErrorsHyperparameters <- apply(searchGridSubCol, 1, function(parameterList){

    #Extract Parameters to test
    currentGamma <- parameterList[["gamma"]]
    currentSubsampleRate <- parameterList[["subsample"]]
    currentColsampleRate <- parameterList[["colsample_bytree"]]
    currentDepth <- parameterList[["max_depth"]]
    currentEta <- parameterList[["eta"]]
    currentMinChild <- parameterList[["min_child"]]

    xgboostModelCV <- xgb.cv(data = dtrain, nrounds = ntrees, nfold = 2, showsd = TRUE,
                             metrics = "rmse", verbose = TRUE, "eval_metric" = "rmse",

```

```

        "objective" = "reg:linear", "max.depth" = currentDepth, "eta" = currentEta,
        "subsample" = currentSubsampleRate, "colsample_bytree" = currentColsampleRate,
        , print_every_n = 10, "min_child_weight" = currentMinChild, "gamma" =
          currentGamma,
        booster = "gbtree",
        early_stopping_rounds = 10)

xvalidationScores <- as.data.frame(xgboostModelCV$evaluation_log)
rmse <- tail(xvalidationScores$test_rmse_mean, 1)
trmse <- tail(xvalidationScores$train_rmse_mean, 1)
output <- return(c(rmse, trmse, currentSubsampleRate, currentColsampleRate, currentDepth, currentEta,
                  currentMinChild, currentGamma))

})

#
output <- as.data.frame(t(rmseErrorsHyperparameters))
varnames <- c("TestRMSE", "TrainRMSE", "SubSampRate", "ColSampRate", "Depth", "eta", "currentMinChild",
              "Gamma")
names(output) <- varnames
save(output, file = "../output/xgboost/cverror.RData")

bst <- xgb.train(data=dtrain, eta=0.7, gamma = 1, max_depth = 8,
                min_child_weight = 10, subsample = 0.7, colsample_bytree = 0.7,
                nrounds=40, watchlist=watchlist, nthread = 4, objective = "reg:linear")

features_test <- predict(bst, dtest)
save(features_test, file = "../output/xgboost/features_test.RData")

features_valid <- predict(bst, dvalid)
save(features_valid, file = "../output/xgboost/features_valid.RData")
}else{
  load("../output/xgboost/features_test.RData")
  load("../output/xgboost/features_valid.RData")
}

valid_xgboost <- features_valid
test_xgboost <- features_test

```

## Step 2.5 Random Forest

As the dataset is quite large, it takes over several hours to train Random Forest on the train set for one cross-validation iteration. So we used resampling method here: randomly sampling the data in K folds. Every time, we choose two folds, one for training Random Forest model, and one for evaluation. After doing this for all the values of ntree, we can get the best Random Forest model. Although this method may have some bias when training the models, it can help obtain a reasonable model in a short time.

```

run.rf = FALSE #if TRUE, run RF on the samples from the whole train set and predict sales on validation

n_trees <- seq(500, 1000, 100)
if(run.rf){
  # process the whole train data so they can be used in RF smoothly

```

```

train$onpromotion <- ifelse(train$onpromotion == TRUE, 1, 0)
train$unit_sales[train$unit_sales < 0] <- 0

train.rf <- train[,-c(1,3,4,8,16,17,19)]
train.rf$celebrated[is.na(train.rf$celebrated)] <- FALSE
train.rf$family <- factor(train.rf$family)
train.rf$celebrated <- factor(train.rf$celebrated)

set.seed(2)
n_train <- nrow(train.rf)
K <- 100 #sample the data into 100 folds
n.fold <- floor(n_train/K)
s <- sample(rep(1:K, c(rep(n.fold, K-1), n_train-(K-1)*n.fold)))

score.rf <- rep(NA, length(n_trees))
opt.mtry <- rep(NA, length(n_trees)) #optimal mtry for a certain n_tree value

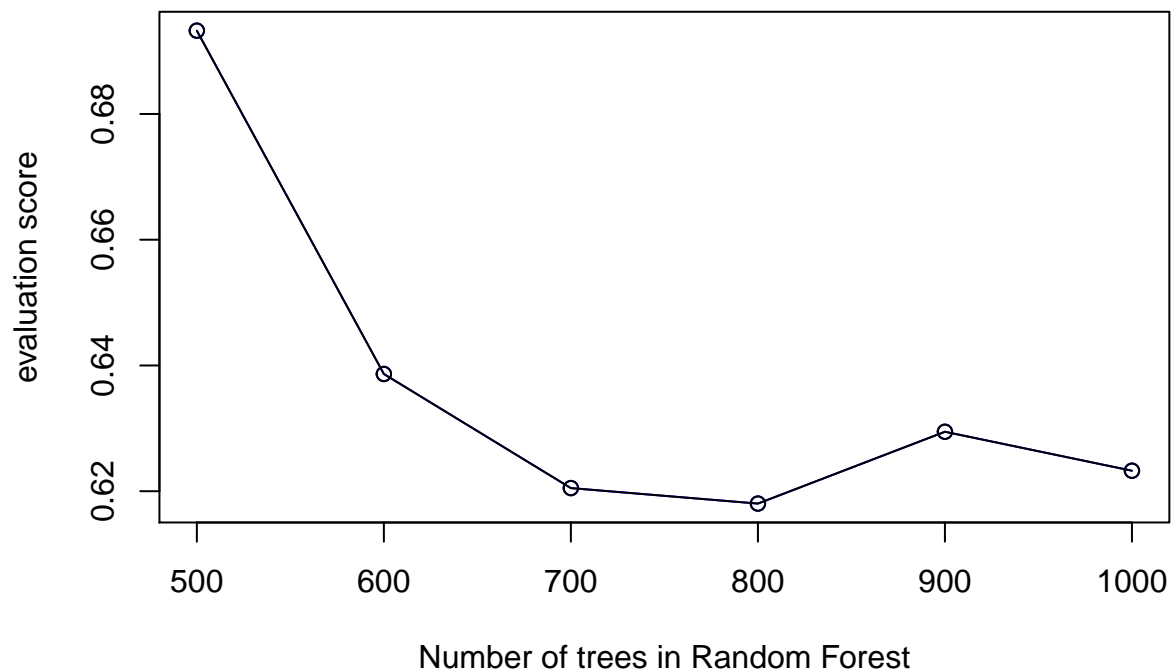
# run the RF on random sampled data
for (i in 1:length(n_trees)){
  train.data <- train.rf[s == i,]
  test.data <- train.rf[s == 100-i,]

  fit <- tuneRF(train.data[,-"unit_sales"], train.data$unit_sales,
               ntreeTry = n_trees[i],
               doBest = TRUE)

  # Get the 'mtry' for trained model
  opt.mtry[i] <- fit$mtry
  pred <- predict(fit, test.data[, -c("unit_sales")])
  w <- ifelse(test.data$perishable == 0, 1, 1.25)
  score.rf[i] <- sqrt(sum(w * (log(pred + 1) - log(test.data$unit_sales + 1))^2)/sum(w))
  save(fit, file=paste0("../output/RF_ntree",n_trees[i],".RData"))
}
save(score.rf, file="../output/RF_score100.RData")
save(opt.mtry, file="../output/RF_mtry.RData")
}else{
  load("../output/RF_score100.RData")
  #load("../output/RF_mtry.RData")
}

# the best parameter ntree
plot(n_trees, score.rf, col="blue", xlab="Number of trees in Random Forest", ylab="evaluation score",type="o")
points(n_trees,score.rf,type="o")

```



```
rf.par <- n_trees[which.min(score.rf)]
print(paste0("the best ntree in Random Forest is ", rf.par))
```

```
## [1] "the best ntree in Random Forest is 800"
```

```
load(paste0("../output/RF_ntree", rf.par, ".RData"))
if(run.rf){
  # predict on validation set
  valid_rf <- valid_train
  valid_rf$onpromotion <- ifelse(valid_rf$onpromotion == TRUE, 1, 0)
  valid_rf$unit_sales[valid_rf$unit_sales < 0] <- 0

  valid_rf <- valid_rf[,-c(1,3,4,8,16,17,19)]
  valid_rf$celebrated[is.na(valid_rf$celebrated)] <- FALSE
  valid_rf$family <- factor(valid_rf$family)
  valid_rf$celebrated <- factor(valid_rf$celebrated)

  rf_pred <- predict(fit, valid_rf[, -which(colnames(valid_rf) == "unit_sales")])
  save(rf_pred, file=paste0("../output/RF_validation.RData"))
  # predict on test set
  test_rf <- test
  test_rf$onpromotion <- ifelse(test_rf$onpromotion == TRUE, 1, 0)
  test_rf$unit_sales[test_rf$unit_sales < 0] <- 0

  test_rf <- test_rf[,-c(1,3,4,8,16,17,19)]
  test_rf$celebrated[is.na(test_rf$celebrated)] <- FALSE
  test_rf$family <- factor(test_rf$family)
  test_rf$celebrated <- factor(test_rf$celebrated)

  del.row <- which(colnames(test_rf) == "unit_sales") # that is 3 here
  rf_pred1 <- predict(fit, test_rf[, -3])
```

```

    save(rf_pred1, file=paste0("../output/RF_test.RData"))
  }else{
    load("../output/RF_validation.RData")
    load("../output/RF_test.RData")
  }
  valid_rf <- rf_pred
  test_rf <- rf_pred1

```

## Step 2.6 Combine Features

```

run.merge = FALSE #if TRUE, combine the features from the first Stack in main.Rmd
if(run.merge){
  # Combine features on validation set
  comb_valid <- merge(valid_ets, valid_prophet, by.x = c("V1","id"), by.y = c("V1", "id"))
  valid_train$rf_pred <- valid_rf
  valid_train$хgboost_pred <- valid_xgboost
  comb_valid <- merge(valid_train[,c(1:6, 8, 15, 20, 21)], comb_valid, by.x = c("V1", "id"), by.y = c("V1", "id"))

  comb_valid <- merge(comb_valid, valid_arima, by.x=c("V1", "id"), by.y = c("V1", "id"))
  save(comb_valid, file="../output/comb_validFeatures.RData")

  # Combine feature on test set
  comb_test <- merge(test_ets, test_prophet, by.x = c("V1","id"), by.y = c("V1", "id"))
  test$rf_pred <- test_rf
  test$хgboost_pred <- test_xgboost
  comb_test <- merge(test[,c(1:6, 8, 15, 20, 21)], comb_test, by.x = c("V1", "id"), by.y = c("V1", "id"))
  comb_test <- merge(comb_test, test_arima, by.x = "id", by.y = "id")
  comb_test <- comb_test[, -c(13:15)]
  save(comb_test, file="../output/comb_testFeatures.RData")
}else{
  load("../output/comb_validFeatures.RData")
  load("../output/comb_testFeatures.RData")
}

```

## Step 3: The Second Stack

In the second Stack, we train Random Forest Models on the validation set use the features from Step 2 by a 5-fold cross-validation