Project 5: Corporación Favorita Grocery Sales Forecasting - Group 1

Shiqi Duan, Peter Li

2017/12/4

Contents

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

```r
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)

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 togetehr 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.

```r
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
```

```r
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 25.9% of 1428019 rows
Read 59.5% of 1428019 rows
Read 91.0% of 1428019 rows
Read 1428019 rows and 19 (of 19) columns from 0.141 GB file in 00:00:05
```

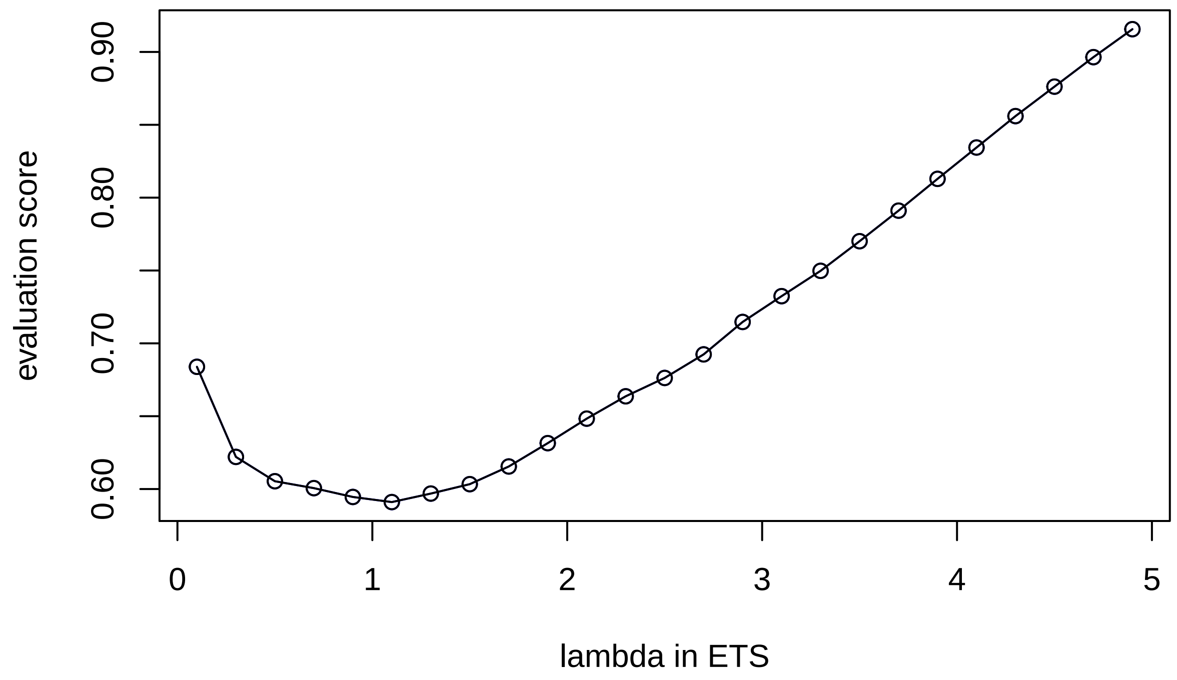
Step 2: The First Stack

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 usin

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")
```



```r
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"
```

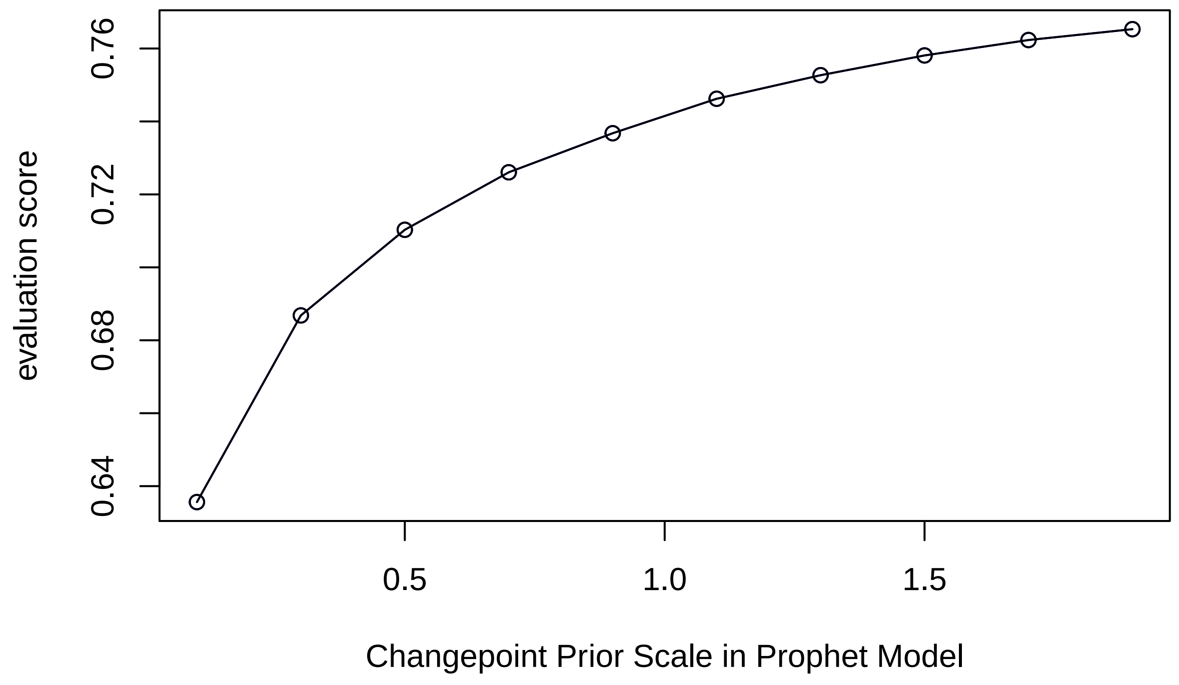
```r
# the features_valid corresponding to ets.par
load(paste0("../output/ets/ets_lambda_",ets.par,".RData"))
valid_ets <- train_save%>%select(V1, id, sales_pred)
colnames(valid_ets)[3] <- "ets_pred"

# 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")
}
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

Step 2.3 Prophet

In this model, we tune the parameter changepoint.prior.scale of the Prophet Modeling framework, as this dictates the flexilibty of the automatic changepoint selection in the model.

```
run.prophet = FALSE #if TRUE, run train-validation process on Prophet model and run the prediction on t

if(run.prophet){
  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
  changepoint.scale <- seq(0.1, 2, by = 0.2)
  valid_score <- rep(NA, length(lam.ranges))
  for (i in 1:length(lam.ranges)){
    lam <- changepoint.scale[j]
    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 = lam)
    future <- make_future_dataframe(m, periods = fcst_intv,freq = "day")

    fcst_matrix <- tail(predict(m, future)$yhat,fcst_intv)
    }
    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/prophet_",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/prophet_valid_score.RData")
}else{
  load("../output/prophet_valid_score.RData")
  lam.ranges <- seq(0.1, 2, by = 0.2)
}

# the best parameter lambda
plot(lam.ranges, valid_score, col="blue", xlab="Changepoint Prior Scale in Prophet Model", ylab="evaluat
points(lam.ranges,valid_score,type="o")
```



```r
prophet.par <- lam.ranges[which.min(valid_score)]
print(paste0("the best Changepoint Prior Scale in Prophet Model is ", prophet.par))
```

```
## [1] "the best Changepoint Prior Scale in Prophet Model is 0.1"
```

```r
load(paste0("../output/prophet_changepoint_",prophet.par,".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 = changepoint.scale[which.min(valid_score)])
    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

Step 2.5 Random Forest

```
run.rf = FALSE #if TRUE, run RF on the whole train set using 5-fold cross-validation
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 <- 5 #5-fold cross-validation
  n.fold <- floor(n_train/K)
  s <- sample(rep(1:K, c(rep(n.fold, K-1), n_train-(K-1)*n.fold)))

  cv_score.rf <- rep(NA, K)
  opt.mtry <- rep(NA, K) #optimal mtry for a certain fold

  n_trees <- seq(100, 10000, 100)
  cv.ntree.score <- rep(NA, length(n_trees)) # mean cv-score for certain n_tree value
  best.ntree.mtry <- rep(NA, length(n_trees)) # best mtry for certain n_tree value
  # run the cross-validation
  for (j in 1:length(n_trees)){
    for (i in 1:K){
      train.data <- train.rf[s != i,]
      #train.label <- y.train[s != i]
      test.data <- train.rf[s == i,]
      #test.label <- y.train[s == i]

      fit <- tuneRF(train.data[,-"unit_sales"], train.data$unit_sales,
                 ntreeTry = n_trees[j],
                  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)
      cv_score.rf[i] <- sqrt(sum(w * (log(pred + 1) - log(test.data$unit_sales + 1))^2)/sum(w))
    }

    # Get the lowest error rate of cross validation
    cv.ntree.score[j] <- mean(cv_score.rf)
    best.ntree.mtry[j] <- opt.mtry[which.min(cv_score.rf)]
  }


}
```

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