ML2 Final Project Store and Item Forcasting

Team 15

4/19/2021

Store Item Demand Forecast

Problem Statement

The team objective for this project is to accurately predict the sales of 50 items sold at 10 different stores for a span of three months. We will use the sales data of these 50 items across all stores from 2013 to 2017. We will then use our model to predict the sales in the test dataset which spans the first three months of 2018. We plan to use the Root Mean Squared Error and Symmetric Mean Absolute Percentage Error to analyze the success of our models.

Libraries

```
library(tidyverse)
library(lubridate)
library(glue)
```

Loading Data from Kaggle

```
train <- read.csv("train.csv")
test <- read.csv("test.csv")
```

Exploring Training Dataset

```
min_date <- date(min(train$date))
max_date <- date(max(train$date))
periods <- as.integer(max_date - min_date) + 1
summary(train)
```

We can see the dataset only contains 3 features: the date, the store and the item. The dependent variable sales ranges from 0 to 231.We will have to engineer additional features from our current set to create an effective model

Feature engineering

```
train_full <- train %>%
  mutate(item = factor(item),
         store = factor(store),
         month = factor(month(date)),
         day = factor(day(date)),
         date = date(date),
         weekday = factor(wday(date))) %>%
```

```
  mutate(period = as.integer(date - min_date)) %>%
  mutate(sin_annual = sin(period*2*pi/365.25),
         cos_annual = cos(period*2*pi/365.25),
         sin_monthly = sin(period*2*pi/(365.25/12)),
         cos_monthly = cos(period*2*pi/(365.25/12)))

summary(train_full)
```

Using the dates we engineered features to model seasonal trends of the sales data. Item IDs and Store IDs were converted to categorical variables as appropriate. The dates were converted to integers. The annual trend was modeled by a month variable, the monthly trend was modeled by the day variable and the weekly trend was modeled by a weekday variable.
Categorical variables are not the only approach to modeling seasonal trends. If the period of cycle is fairly constant and the behavior of the within the period is well defined, we can model trend $\alpha \sin \frac{2\pi t}{p} + \beta \cos \frac{2\pi t}{p}$ where t is our current time period and p is the number of periods in the trend. We use sin and cos functions to create an annual and monthly seasonal trends.

Modeling Approach

To predict sales for the items and the stores 3 models will be developed:

- ARIMA models using auto.arima function
- Multiple Linear Regression Models
- XGBoost Boosting Decision Tree Models

To test the performance of the model, the data will be split into training and test sets. The test sets will contain the last 90 periods of data (Oct 2017 to Dec 2017) and should be effective in determining how the models should perform on the submission test data. The RMSE will be used to evaluate model performance and the MAPE will also be tracked.

Creating ARIMA Model

The dataset is essentially 500 different time series, one for each item sold for each store. A naive approach is to create an ARIMA model for every single time series and use the appropriate model for predictions.

```
library(forecast)
stores = 1:10
items = 1:50
arima_mse <- c()
arima_smape <- c()
itemID <- c()
storeID <- c()
start <- Sys.time()
for(i in items){
  for(s in stores){
    train_filtered <- train_full %>% filter(store == s, item == i)
    periods <- length(train_filtered$sales)
    sales <- ts(train_filtered$sales, frequency = 365.25, start = c(2013,1)) # time series object
    sales_train <- window(sales,end = c(2017,275))
    model <- auto.arima(sales_train)
    results <- forecast(model,90)$mean
    mse <- mean((sales[(periods-89):periods] - results)^2)
    smape <- 2*mean(abs((sales[(periods-89):periods] - results))/(abs(sales[(periods-89):periods]) + abs
    arima_mse <- c(arima_mse,mse)
    arima_smape <- c(arima_smape,smape)
    itemID <- c(itemID,i)
```

```
    storeID <- c(storeID,s)
  }
}
end <- Sys.time()
end - start
glue('ARIMA Model SMAPE is {mean(arima_smape)} RMSE is {sqrt(mean(arima_mse))}')
```

We can see that this approach is very resource and time intensive. For companies with many more items or stores this approach would be completely unfeasible. Using the individual models to predict 90 periods for each store and each item, we have a SMAPE of .22 and a RMSE of 14.8. This will provide a great benchmark for us to compare other models.

Visualizing ARIMA Model Performance

```
model_performance_summary = data.frame(itemID,storeID,arima_mse,arima_smape)

par(mfrow = c(1,2))
hist(sqrt(model_performance_summary$arima_mse), main = 'ARIMA RMSE Histogram', xlab = 'RMSE')
hist(model_performance_summary$arima_smape, main = 'ARIMA SMAPE Histogram', xlab = 'SMAPE')
```

Creating Linear Regression Model

Rather than treating the data as a time series, we can model the sales of items in stores as a regresion model. The date becomes a x-feature, incrementing by one as each day passes. The 'period' variable essentially represents the overall trend of sales. The item, store and other generated features become predictors in our model. Just like with any other multiple linear regression model, interaction terms can also be engineered to improve the model performance.

```
gc()
memory.limit(size = 35000) #Model will require extensive memory

train_filtered <- train_full %>% filter(period %in% 1:(periods-90))
train_validation <- train_full %>% filter(period %in% (periods - 89):periods)

linear_model <- lm(sales ~ period + item + store, data = train_filtered)
pred_lm <- predict(linear_model,train_validation)

linear_model_seasonal <- lm(sales ~ .-date-sin_annual-cos_annual - sin_monthly - cos_monthly, data = tra
pred_lms <- predict(linear_model_seasonal,train_validation)

linear_model_trig <- lm(sales ~ .-date-month-day, data = train_filtered)
pred_lmt <- predict(linear_model_trig,train_validation)

linear_model_seasonal_interactions <- lm(sales ~ period*item + period*store + item*store + day + month
pred_lmsi <- predict(linear_model_seasonal_interactions,train_validation)

linear_model_trig_interactions <- lm(sales ~  period*item + period*store + item*store + sin_annual + cos
pred_lmti <- predict(linear_model_trig_interactions,train_validation)

glue('Linear Regression Model SMAPE is {2*mean(abs(pred_lm - train_validation$sales)/(abs(train_validati
glue('Linear Regression Model with Seasonal Variables SMAPE is {2*mean(abs(pred_lms - train_validation$s
glue('Linear Regression Model with Trig Variables SMAPE is {2*mean(abs(pred_lmt - train_validation$sales
glue('Linear Regression Model with Seasonal Variables and Interactions SMAPE is {2*mean(abs(pred_lmsi -
glue('Linear Regression Model with Trig Variables and Interactions SMAPE is {2*mean(abs(pred_lmti - tra
```

5 models were created and the performance compared. The adjusted R-squared is calculated on the training data set while the RMSE and SMAPE is calculated on the test data set. The simple linear regression performed similarly to the 500 ARIMA models. Adding seasonality greatly improved model performance. It should be noted that adding interaction terms improved both the adjusted R squared and RMSE but slightly degraded the SMAPE metric. The models with day and month variables outperformed the models that used sin and cos variables.

Visualizing Seasonal Linear Model

```
test_results <- cbind(train_validation,pred_lms)
test_results_sum <- test_results %>% mutate(itemID = as.integer(item),storeID = as.integer(store)) %>%
  group_by(itemID,storeID) %>%
  summarise(lms_rmse = sqrt(mean((pred_lms - sales)^2)),
            lms_smape = 2*mean(abs(pred_lms - sales)/(abs(sales) + abs(pred_lms))))


model_performance_summary <- inner_join(model_performance_summary,test_results_sum)

par(mfrow = c(1,2))
hist(model_performance_summary$lms_rmse, main = 'Seasonal LM RMSE Histogram', xlab = 'RMSE')
hist(model_performance_summary$lms_smape, main = 'Seasonal LM SMAPE Histogram', xlab = 'SMAPE')
```

XGBoost Model

We will now attempt to create a boosting regression decision tree using XGBoost. We use the period variable and create 2 models, 1 using the month and day variables and the other using the trigonometry featues. Unlike with the linear regression, we do not need to create interactions. The decision tree is able to learn the conditional relationship between the variables based on the branch.

```
library(xgboost)

train_matrix_seasonal <- model.matrix(sales~.-date-sin_annual-cos_annual - sin_monthly - cos_monthly, da
test_matrix_seasonal <- model.matrix(sales~.-date-sin_annual-cos_annual - sin_monthly - cos_monthly, dat

train_matrix_trig <- model.matrix(sales~.-date-month-day, data = train_filtered)[,-1]
test_matrix_trig <- model.matrix(sales~.-date-month-day, data = train_validation)[,-1]

dtrain_seasonal <- xgb.DMatrix(data = train_matrix_seasonal,label = train_filtered$sales)
dtest_seasonal <- xgb.DMatrix(data = test_matrix_seasonal,label = train_validation$sales)

dtrain_trig <- xgb.DMatrix(data = train_matrix_trig,label = train_filtered$sales)
dtest_trig <- xgb.DMatrix(data = test_matrix_trig,label = train_validation$sales)


xgboost_model_seasonal <- xgboost(data = dtrain_seasonal, params = list(nthread = 5),
                          objective = 'reg:squarederror',
                          nrounds = 200,verbose = 0)

xgboost_model_trig <- xgboost(data = dtrain_trig, params = list(nthread = 5),
                          objective = 'reg:squarederror',
                          nrounds = 200,verbose = 0)

pred_xgb_seasonal <- predict(xgboost_model_seasonal,dtest_seasonal)
pred_xgb_trig <- predict(xgboost_model_trig,dtest_trig)
```

```
glue('The XGBoost Model with Seasonal Variables SMAPE is {2*mean(abs(pred_xgb_seasonal - train_validatio
glue('The XGBoost Model with Trig Variables SMAPE is {2*mean(abs(pred_xgb_trig - train_validation$sales)
```

We see that for the boosting model the trigonometric features performed the best. In fact, for both the RMSE and SMAPE it performed the best our of all considered models by a wide margin. The SMAPE is comparable to the Kaggle submission score as well, over a similar task.

```
test_results <- cbind(train_validation,pred_xgb_trig)
test_results_sum <- test_results %>% mutate(itemID = as.integer(item),storeID = as.integer(store)) %>%
  group_by(itemID,storeID) %>%
  summarise(xgb_trig_rmse = sqrt(mean((pred_xgb_trig - sales)^2)),
            xgb_trig_smape = 2*mean(abs(pred_xgb_trig - sales)/(abs(sales) + abs(pred_xgb_trig)))) %>%
  select(itemID,storeID,xgb_trig_rmse,xgb_trig_smape)


model_performance_summary <- inner_join(model_performance_summary,test_results_sum)

par(mfrow = c(1,2))
hist(model_performance_summary$xgb_trig_rmse, main = 'XGBoost RMSE Histogram', xlab='RMSE')
hist(model_performance_summary$xgb_trig_smape, main = 'XGBoost SMAPE Histogram',xlab='SMAPE')
```

Visualizing Trigonometric XGBoost Model

Creating Submision using Optimal XGBoost

```
test_full <- test %>%
  mutate(item = factor(item),
         store = factor(store),
         month = factor(month(date)),
         day = factor(day(date)),
         date = date(date),
         weekday = factor(wday(date))) %>%
  mutate(period = as.integer(date - min_date)) %>%
  mutate(sin_annual = sin(period*2*pi/365.25),
         cos_annual = cos(period*2*pi/365.25),
         sin_monthly = sin(period*2*pi/(365.25/12)),
         cos_monthly = cos(period*2*pi/(365.25/12))) %>%
  select(store,item,weekday,period,sin_annual,cos_annual,sin_monthly,cos_monthly)

train_matrix_trig <- model.matrix(sales~.-date-month-day, data = train_full)[,-1]
test_matrix_trig <- model.matrix(~., data = test_full)[,-1]

dtrain_trig <- xgb.DMatrix(data = train_matrix_trig,label = train_full$sales)
dtest_trig <- xgb.DMatrix(data = test_matrix_trig)

xgboost_model_trig <- xgboost(data = dtrain_trig, params = list(nthread = 5),
                          objective = 'reg:squarederror',
                          nrounds = 200,verbose = 0)

pred_xgb_trig <- predict(xgboost_model_trig,newdata = dtest_trig)

submission <- data.frame(id = test$id, sales = pred_xgb_trig)
```

```r
write.csv(submission,file = 'submission.csv',row.names = TRUE)
```