Small_Bank_forecasting-Prophet

Code

Alhad Pofali

Hide

Hide

```
library(prophet)
library(dplyr)
library(tidyr)
library(ggplot2)
library(forecast)
library(tibbletime)
library(tseries)
library(trend)
library(tidyverse)
library(rsample)
library(yardstick)
library(shiny)
```

Part 3

In continuation of earlier projects, this particular approach is to check how “Prophet” library from Facebook can be implemented to do forecasting. Previous project touched upon clustering of time series and forecasting using LSTM framework. For timeseries forecasting, it is known to be one of the best packages with ready made functions and ease of handling.

Load Data

Load data for deposits amounts. As observed before, we have some unusual spiky data for last 15 days in the time line. Hence restrict data this end of August only, ignoring September 2012 data.

Hide

Hide

```
#To remove outliers (for both deposit and withdrawa;s)
#function - to remove outliers for cash deposit counts
rem_out <- function(ID) {
  p <- filter(all_NB_SOL, all_NB_SOL$SOLID == as.integer(ID))
  q <- subset(p,!(p$CDA > quantile(p$CDA, probs=c(.02, .98))[2] | p$CDA < qu
antile(p$CDA, probs=c(.02, .98))[1]) )
  r <- subset(q,!(q$CWA > quantile(q$CWA, probs=c(.02, .98))[2] | q$CWA < qu
antile(q$CWA, probs=c(.02, .98))[1]) )
  r
}

#data frame to store the data
all_NB_SOL_out<- as.vector(0)

#Loop to store the data after outlier corrections
for (i in unique(all_NB_SOL$SOLID)) {
  x <-rem_out(i)
  all_NB_SOL_out <- if (all_NB_SOL_out == 0) x else rbind(all_NB_SOL_out,x)
}
```

Outliers

Create function to remove outliers from the data. Since there are 128 branches, we will need to check data at each branch level and address outliers at that level. For this create a function to ignore data beyond range of 0.2 to 0.98 quantile. This will retain few spikes but remove only exterme values. Loop this function across the branches and store data in new table. This will result is few datapoints getting ignored.

Hide

Hide

```r
#function - to fill in missing dates with 0 values to complete the matrix
test_fun_CA <- function(ID) {
  q <- filter(all_NB_SOL_out, all_NB_SOL_out$SOLID == as.integer(ID))
  q %>% complete(date = seq.Date(as.Date("2010-10-01"), as.Date("2012-08-31"
), by="day")) -> q
  q$SOLID[is.na(q$SOLID)] <- as.character(ID)
  q[is.na(q)] <- 0
  q
}
#create a new data with use of function for data interpolation
m <- as.vector(0)
for (i in unique(all_NB_SOL_out$SOLID)) {
  x <-test_fun_CA(i)
  m <- if (m == 0) x else rbind(m,x)
}

#get only required columns from outlier + expanded data
m <- subset(m, select = c("SOLID", "date", "CDA"))

head(m)
```

```
## # A tibble: 6 x 3
##   SOLID date             CDA
##   <chr> <date>         <dbl>
## 1 1     2010-10-01         0
## 2 1     2010-10-02         0
## 3 1     2010-10-03         0
## 4 1     2010-10-04  12470736
## 5 1     2010-10-05   9581359
## 6 1     2010-10-06   8055917
```

Missing data There are few missing days which are weekends, holidays and outliers. We have an option to create a continious time series by ignoring these dates and Prophet can handle this easily. As we want to retain the consistency across the input data from all other excersizes we will impute the values to create continious data.

For this we use function “complete” with start and end date (after checking input data). We use it in function and run it across all branches.

Above table shows the data with 0 values too. However, by doing this we have just introduced lot of outliers and spikes in data. This can be addressed using below approach.

Impute data Create a function to calculate moving average at the branch level. This would be used to fill the zeros that are generated due to inclusion of holidays including weekends. Moving average would remove this noise. 7 day moving average is taken to normalize data for a week.

First three records would be NA and would be required to be removed.

Hide

Hide

```
t<-as.vector(0)
for (i in unique(m$SOLID)) {
  q <-filter(m, m$SOLID == as.integer(i))
  q$ma7 <- as.integer(ma(q$CDA, order=7))
  for (i in 1:nrow(q)) {
    q$CDA[i] <- if(q$CDA[i] == 0) q$CDA[i]<-q$ma7[i] else q$CDA[i]
  }
  t <- if (t == 0) q else rbind(t,q)
}
t<-t[!is.na(t$CDA),]

head(t)
```

```
## # A tibble: 6 x 4
##   SOLID date            CDA     ma7
##   <chr> <date>        <dbl>   <int>
## 1 1     2010-10-04 12470736 5561320
## 2 1     2010-10-05  9581359 6780757
## 3 1     2010-10-06  8055917 7557228
## 4 1     2010-10-07  8821233 7557228
## 5 1     2010-10-08  8536057 7115722
## 6 1     2010-10-09  5435298 5746957
```

Tests for Trend and Seasonality

For all the branches, capture the p-value for ADF test(Augmented Dickey Fuller test) and CS test (Cox and Stuart test) to know trend and seasonality values. These can be used to segment branches appropriately and address data before submiting them to modelling in Prophet.

Loop across branches and capture same in a table against each branch.

Hide

Hide

```
tests_rslts <- data.frame(matrix(ncol = 3, nrow = length(unique(t$SOLID))))
colnames(tests_rslts) <- c("SOLID", "tst_seas", "tst_trnd" )

for (i in as.integer(unique(t$SOLID)[1:5])) {
  t_temp <- filter(t, t$SOLID == as.integer(i))
  t_temp <- subset(t_temp, select = c("date", "CDA"))
  t_temp<-t_temp[!(t_temp$CDA == 0),]
  tests_rslts$SOLID[i] <- as.integer(i)
  tests_rslts$tst_seas[i] <- adf.test(t_temp$CDA)$p.value
  tests_rslts$tst_trnd[i] <- cs.test(t_temp$CDA)$p.value
}

head(tests_rslts)
```

```
##   SOLID tst_seas     tst_trnd
## 1     1     0.01 1.842295e-05
## 2     2     0.01 7.314122e-02
## 3     3     0.01 4.574799e-01
## 4     4     0.01 2.053528e-02
## 5     5     0.01 6.732872e-03
## 6    NA       NA           NA
```

Prophet - forecasting

Before forecasting, we want to address the “0” that we introduced in the data. This can be done by calculating moving average data for 7 days(as we have a weekly cycle).
Use this data to interpolate with 0s in the deposit data for the missing days.
Also now evaluate outliers and check if they need be removed.
Since there many branches which have very less data we will ignore those branches which have less than 150 data points i.e. less than 6 months.
use 100 days of data for test and rest of the data as train.
Create a future data frame of 100 days for forecast.
Carry out forecast.
From forecast prophet forecast table create a table with only y hat values.
Create table with “actual” values of train and test set along with “predict” values for all the branches.

Hide

Hide

```
stforecast <-as.vector(0)
ret_all <- as.vector(0)
for(i in as.integer(unique(t$SOLID)[1:5])){

  t_temp <- filter(t, t$SOLID == as.integer(i)) #get data from original tabl
e
  t_temp <- subset(t_temp, select = c("date", "CDA")) # select only columns
  #t_temp$CDA <- t_temp$CDA
```

```r
  outliers <- boxplot(t_temp$CDA, plot = FALSE)$out
  t_temp <-if (is_empty(outliers) == TRUE) t_temp else t_temp[-which(t_temp$
CDA %in% outliers),]
  t_temp<-t_temp[!(t_temp$CDA == 0),]

  if (nrow(t_temp) > 150 ) {
    df_trn <- t_temp[1:(nrow(t_temp)-100),]
    df_tst <- t_temp[(nrow(t_temp)-99):nrow(t_temp),]
    df_trn_p <- subset(df_trn, select = c("date", "CDA"))
    df_ma7 <- filter(t, t$SOLID == as.integer(i))
    df_ma7 <- subset(df_ma7, select = c("date", "ma7"))
    df_ma7 <- df_ma7[(nrow(t_temp)-99):nrow(t_temp),]
    colnames(df_ma7) <- c("ds", "y")
    colnames(df_trn_p) <- c("ds", "y")

    # y <- prophet(growth = "linear", weekly.seasonality=FALSE)
    # y <- add_seasonality(y, name='yearly', period = 365, fourier.order = 9
)
    # y <- add_seasonality(y, name='monthly', period=30.5, fourier.order=11)
    # y <- add_seasonality(y, name='weekly', period = 7, fourier.order = 13)
    # y <- fit.prophet(y, df_trn_p)

    y <- prophet(df_trn_p)

    future <- make_future_dataframe(y, periods = 100)

    forecast <- predict(y, future)

    nam <- paste("y", as.integer(i), sep = "")
    assign(nam, y)

    forecast$SOLID <- rep(i, nrow(forecast))
    stforecast <- if(stforecast == 0) forecast else rbind(stforecast, foreca
st)
    for_p <- subset(filter(stforecast, stforecast$SOLID == i), select = c("d
s", "yhat"))
    for_p <- for_p[(nrow(t_temp)-99):nrow(filter(stforecast, stforecast$SOLI
D == i)),]

    tbl_1 <- df_trn_p %>%
      add_column(key = "actual")
    tbl_2 <- df_tst %>%
      add_column(key = "actual")
    tbl_3 <- for_p %>%
      add_column(key = "predict")
    tbl_4 <- df_ma7 %>%
      add_column(key = "MA7")
    colnames(tbl_2) <- c("ds","y","key")
    colnames(tbl_3) <- c("ds","y","key")
    ret<- rbind(tbl_1, tbl_2, tbl_3, tbl_4)
    ret$SOLID <- rep(i, nrow(ret))
    ret_all <- if (ret_all == 0) ret else rbind(ret_all, ret)
  } else {
    print(paste0(i, " Branch number has less than 100 historical number of r
ecords"))
  }

}
```

Details about model

y <- prophet(df_trn_p)
This is a base model where model itself determines the seasonalities parameters

y <- prophet(growth = "linear", weekly.seasonality=FALSE)
y <- add_seasonality(y, name='yearly', period = 365, fourier.order = 9)
y <- add_seasonality(y, name='monthly', period=30.5, fourier.order=11)
y <- add_seasonality(y, name='weekly', period = 7, fourier.order = 13)
y <- fit.prophet(y, df_trn_p)

This is a build up model with specific seasonalities defined to be considered. Based on data, it would be required to tune these parameters and check if they improve results.
For this data, base model was good enough for many branches. However, to optimize the results of each branch, clustering that was used can be implemented. For each cluster, we can use different model.

By default Prophet fits additive seasonalities, meaning the effect of the seasonality is added to the trend to get the forecast.

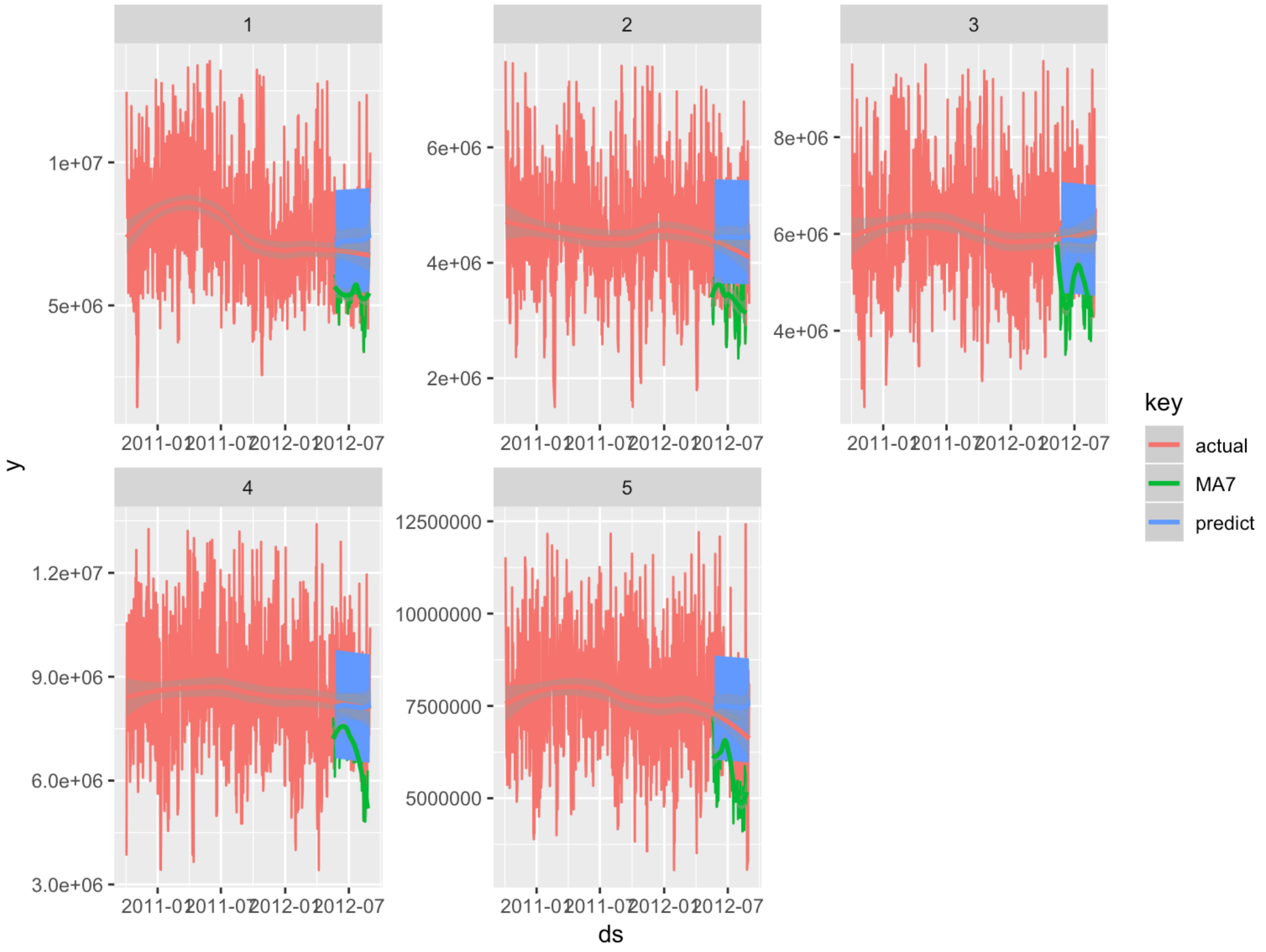
Plotting output for branch

From 128 branches, we will be plotting first 5 branches.

Hide

Hide

```
ggplot(ret_all, aes(ds, y, color = key)) + geom_line() + stat_smooth(method
= "loess") +
  facet_wrap(~SOLID, scales = "free")
```



Hide

Hide

```
#
# ggplot() +
#   geom_line(data = filter(ret_all, ret_all$key == 'actual'), aes(x = ds, y
= y), colour = "Red") +
#   geom_line(data = filter(ret_all, ret_all$key == 'predict'), aes(x = ds,
y = y), colour = "Blue") +
#   labs(title="Count of transaction for Deposits and W'drawals", x ="Years"
, y = "Amount")+
#   facet_wrap(~SOLID, scales = "free")
#   theme_bw()
```

Error Calculations

Once the predict table is created, we calculate RMSE for the forecast. For this create a function to calculate the RMSE so that it can be calculated for each branch in a loop. This function returns a single value for branch.

Hide

Hide

```
calc_rmse <- function(prediction_tbl) {

  rmse_calculation <- function(data) {
    data %>%
      spread(key = key, value = y) %>%
      select(-ds) %>%
      filter(!is.na(predict)) %>%
      rename(
        truth    = actual,
        estimate = predict
      ) %>%
      rmse(truth, estimate)
  }

  safe_rmse <- possibly(rmse_calculation, otherwise = NA)

  safe_rmse(prediction_tbl)
}
```

Hide

Hide

```
for (i in as.integer(unique(ret_all$SOLID))) {

  t<-t[!(t$CDA == 0),]

  if (nrow(t)> 150) {
    x <- calc_rmse(filter(ret_all, ret_all$SOLID == i))$.estimate
    tests_rslts$rmse[i] <- x
  }
}

head(tests_rslts)
```

```
##   SOLID tst_seas     tst_trnd      rmse
## 1     1     0.01 1.842295e-05 1582237.5
## 2     2     0.01 7.314122e-02  904611.9
## 3     3     0.01 4.574799e-01  992639.8
## 4     4     0.01 2.053528e-02 1496827.3
## 5     5     0.01 6.732872e-03 1646257.5
## 6    NA       NA           NA 1582237.5
```

Next steps

Above code and analysis provides a template for time series analysis using prophet. In the next project, we will approach optimization of the model and improvement of resutls.