predict\_stk <- function() {

library(quantmod)

library(liquidSVM)

library(caret)

#Tickers of companies

file = read.csv(file.choose())

comp = paste(file$Security.Id[1:10],'.BO',sep='')

disp = paste(1:10,". ",file$Security.Id[1:10],'.BO\n',sep='')

clear = function()

{

cat("\014") #Function to clear console

}

clear()

cat("\tWelcome to Stock Market Prediction","\n")

cat("Choose the dataset you wish to predict the stock for","\n\n")

cat(disp)

choice = as.numeric(readline("Enter your choice : "))

if(!(choice %in% (1:10)))

{

stop("Invalid input")

}

COMPANY = comp[choice]

#Default date range

START\_DATE = "2017-01-01"

END\_DATE = "2018-01-01"

choice = readline("Do you wish to use default date range? (y|n) ")

if(choice=="N" | choice=="n")

{

cat("Enter the stock data range to be fetched YYYY-MM-DD","\n")

START\_DATA = readline("START\_DATE of stock data to fetch")

END\_DATA = readline("END\_DATE of stock data to fetch")

}

cat("Fetching data for",COMPANY)

wantDate = function(dateAsString)

{

#Function to convert a string to date format

return(as.Date(dateAsString,format="%Y-%m-%d"))

}

getData = function(online)

{

#if ONLINE is set true, data is fetched online, otherwise, local file is used

data = NULL

if(online)

{

# data = Quandl(paste("EOD/",COMPANY,sep=""), api\_key="GZxFKM3hsT6xmBGBqnZP", start\_date=START\_DATE, end\_date=END\_DATE, type="raw",column\_index='4')

data = getSymbols.yahoo(COMPANY,env=globalenv(), from = START\_DATE, to = END\_DATE)

data = eval(expr = parse(text=data))

data = data.frame(Date = index(data), Close = data[,4], stringsAsFactors = F)

colnames(data)[2]='Close'

} else

{

data = read.csv("KO.csv")[,c('Date','Close')]

}

#Convert string to dates

data['Date'] = wantDate(data$Date)

#Arrange data chronologically

data = data[order(data['Date'],decreasing = FALSE),]

#Setting rownames to null resets the numbering of rows

rownames(data) = NULL

return(na.omit(data))

}

plotData = function(xval,yval=NULL,color)

{

points.default(xval, yval,col=color,type="l",lwd="2")

}

getAcc = function(predicted,actual)

{

#Returns the R-Squared value

return(round(R2(predicted,actual),3))

}

perc = function(val,justappend=FALSE)

{

#Converts value into percentage rounded to 2 decimals

#If justappend is true, it just appends % symbol.

if(!justappend)

return(paste(round(val\*100,2),"%",sep=""))

else

return(paste(round(val,2),"%",sep=""))

}

slidingWindow = function(data)

{

#Sliding-window method

#Refer Methodology, IEEE Paper

#Forward forecast of stock price using sliding-window metaheuristic -optimized Machine Learning Regression

y = data[-(1:LAG),]

rownames(y) = NULL

x= data[1:(DATA\_SIZE-LAG),]

train = data.frame(x[,2],y[,2])

colnames(train) = c("x","y")

return(train)

}

#Setting colors for the graph

#predict color

color3 = '#FF5722'

#color3 = paste("#","9C27B0",sep="")

#train color

color2 = paste("#","03A9F4",sep="")

#act color

color1 = paste("#","606060",sep="")

#Reading data from the file

data = getData(TRUE)

#===========================Constants=========================#

#Keep lag always as one

LAG = 1

DATA\_SIZE = nrow(data)

#CUR represents the current stock data. All prediction is done from this day forward. All data from 1:CUR is the known stock value upto today.

#Let ratio of TEST:TRAIN be 60:40% split

#So set CUR 60% of DATA\_SIZE

CUR = round(0.6 \* DATA\_SIZE)

#Window size. It has been observed that if we wish to predict stock for X days in advance, the window size must have proportionally equal or greater than X days of past data [Window size starts from 1]

W\_SIZE = CUR - 1

#The training set of the data is of size CUR starting from 1

#Can be modified for flexibility

TRN\_RANGE = (CUR - W\_SIZE):CUR

#Test data size, or the size of data to be predicted. Prediction days CANNOT EXCEED the value of REM. So you can only predict upto REM days of stock in advance

REM = (DATA\_SIZE - CUR) -1

DELAY = as.numeric(readline(paste("Predict stock for how many days in advance? (1 - ",REM,") : ")))

# DELAY = REM

#=============================================================#

#Plotting all data

plot(ts(data$Close),type="l",xlab = paste(substr(START\_DATE,0,4)," to ",substr(END\_DATE,0,4)),ylab = "Closing Price",main=paste(COMPANY," Stock Prediction",sep=""),col=color1,lwd="2")

par(font=1) #Setting font style to BOLD

#Adding legend

legend("topright", c("Actual Stock","Training Data","Predicted Future Stock"), fill=c(color1,color2,color3))

#Phase Space Reconstruction using LAG

train = slidingWindow(data)

#Obtain all the dates and store it so it can be plotted.

#Example x\_axis = [2017-04-11, 2017-04-10...]

x\_axis = as.numeric(rownames(train))

#Plot only training data

plotData(x\_axis[TRN\_RANGE],train$y[TRN\_RANGE],color=color2)

#WIND is the window. This window moves as the number of days advance

#The logic has been explained in the Phase Space Reconstruction and paper

wind = train[TRN\_RANGE,]

#Create empty vectors where values are stored when the window is moved.

#Once the window has reached the end, all values inside these vectors are used to plot the graph. Basically, all results and details are stored in these

r\_day = vector(mode="numeric",length = 0L) #Vector of Dates [2017-04-11]

r\_pprice = vector(mode="numeric",length = 0L) #Predicted Prices

r\_aprice = vector(mode="numeric",length = 0L) #Actual Prices

results = data.frame(r\_day,r\_aprice,r\_pprice) #Data frame of the above

accuracy = vector(mode="numeric",length = 0L) #Accuracy of each prediction

#This vector is used in the end of obtain the overall accuracy or mean accuracy

avg\_perform = vector(mode="numeric",length = 0L)

space = c(1,10,30,60,90)

range = list(c(1:1),c(2:10),c(11:30),c(31:60),c(61:90))

i = 1

#Print the header

cat("DAY","\t","DATE","\t\t","PREDICTED PRICE","\n")

cat("----------------------------------------","\n")

#For each day, train the data and predict the result.

#The predicted value of each day is fed as training data to the next day

start = Sys.time()

for (day in 1:DELAY) {

#Prepare and store the training model

model = lsSVM(y ~ x,wind,max\_gamma=3125,min\_lambda=8e-06,GPUs = 1)

X = train$x[CUR + day] #Get training data within WINDOW

X\_axisname = x\_axis[CUR + day] #Get all dates in string

A = train$y[CUR + day] #Get actual prices

#Predict prices given X

pred = as.numeric(test(model,X))

#Store results in future dataframe

results[nrow(results) + 1,] = c(X\_axisname,A,pred)

#Once we have predicted stock for DAY 1, we add it to the training set. When we do this, the oldest value of the training set is removed. Basically, we are adding a row and removing the oldest. This is the "Sliding-Window". It acts like a QUEUE. Last in, First Out

#Remove first row from the window

wind = wind[-1,]

#Add predicted price to last row

wind[length(wind[,1]) +1,] = c(X,pred)

#Rownames represet the DATE, so we need to append the appropriate date to each row.

#Get first row number

first = as.numeric(rownames(wind[1,]))

#Last row number will be.... (first + W\_SIZE)

last = first + W\_SIZE

#rownames start from first:last

#Example 20:200 represents stock from 20th to 200th day in the stock data

rownames(wind) = c(first:last)

#Convert the date from string to DATE format

date = as.character(data[X\_axisname,1])

#Output the results on console, (DAY, DATE, PRED\_PRICE)

cat("Day",day,"\t",date,"\t",pred,"\n")

#Calculate accuracy by dividing actual and predicted and multiply by 100 to show in percentage.

acc = A/pred

if(acc>1)

acc = (pred/A) \* 100

else

acc = acc \* 100

#Store accuracy in the empty vector we defined earlier

accuracy[length(accuracy) +1] = acc

if(day %in% space)

{

n = length(accuracy)

#Calculate percentage decline

avg\_perform[length(accuracy) +1] = mean(accuracy[range[[i]]])

i = i+1

}

plotData(results$r\_day,results$r\_pprice,color = color3)

}

end = Sys.time()

#Plot the predicted results

#plotData(results$r\_day,results$r\_pprice,color = color3)

A = results$r\_aprice

P = results$r\_pprice

#Calculate average of all accuracy

avg\_perform = na.omit(avg\_perform)

mean\_acc = perc(mean(avg\_perform),TRUE)

#Calculate R-squared

r\_squared = getAcc(P,A)

#Display results on graph as a legend

par(font=2)

legend('bottomright',c(paste("Accuracy ",mean\_acc)))

#The steps that follow are basically for those who wish to see summary of results after the prediction is done

date = as.character(data[results$r\_day,1])

results = data.frame(date,A,P)

colnames(results) = c("Date","Actual","Predicted")

plotAcc = function()

{

plot(accuracy,type='l',col='gray',lwd=2,main="Daywise Accuracy",xlab = "No. of days ahead",ylab = "Accuracy %")

ac = sort(accuracy,decreasing = TRUE)

lines(ac,type='l',col='red',lwd=2)

legend("bottomleft", c("Accuracy","Net Accuracy "), fill=c('gray','red'))

}

plotPredAcc = function()

{

plot(avg\_perform,type='l',col='red',lwd=2,main="Predictive Performance",xlab = "No. of days ahead",ylab = "Accuracy %",ylim = c((min(avg\_perform)-5),100))

}

plotBoth = function()

{

par(mfrow=c(1,2))

plotAcc()

plotPredAcc()

par(mfrow=c(1,1))

}

my\_summary = function()

{

cat("\n","--------------------------------Prediction Summary------------------------------","\n\n")

cat("BASIC STATS","\n")

res = end - start

cat("\t ")

print(res)

cat("\t","No. of days predicted","\t",DELAY,"\n")

cat("\t","Training data (days)","\t",length(TRN\_RANGE),"\n")

cat("\t","Max. pred. accuracy","\t",perc(max(accuracy),TRUE),"\n")

cat("\t","Min. pred. accuracy","\t",perc(min(accuracy),TRUE),"\n")

cat("\t","Mean pred. accuracy","\t",perc(mean(accuracy),TRUE),"\n")

cat("\t","Predictive Accuracy","\t",mean\_acc,"\n\n")

cat("FUNCTIONS","\n")

cat("\t","plotBoth()","\t","To plot both performance curves","\n")

cat("\t","plotAcc()","\t","To plot the accuracy curve","\n")

cat("\t","plotPredAcc()","\t","To plot the predictive accuracy curve","\n\n")

cat("VECTORS","\n")

cat("\t","results","\t\t","To print entire results: day,actual,predicted","\n")

cat("\t","accuracy","\t\t","To print daywise accuracy","\n")

cat("\t","avg\_perform","\t\t","To print predictive accuracy","\n")

}

my\_summary()

#Started: 14th April 2018 Completed: 19th April 2018#

}

predict\_stk()