```
#################################################################
## Major Research Project
## Prepared by : Kisha Taylor
## Date: 09-August-2018
##
## Stock Market Prediction- Artificial Trader
## Automatically generates signal for trading stock at the
#  open price next day and exucutes trades.
#  Accuracy, Precision, Recall and Return Metrics computed in backtesting.



## Compares performance of technical analysis & ML models that use
## hybridized inputs (i.e. inputs that are techical indicators and raw prices)


##
## Basic steps:
## 1) Find Top Technical indicator
## 2) Case 1 (Baseline model) : Tune ML models (ANN & SVM) using only closing price as input
## 3) Case 2.1 : Tune ML models (ANN & SVM) using only closing price & top tech. indicator "No buffer" as inputs
## 4) Case 2.2 : Tune ML models (ANN & SVM) using only closing price & top tech. indicator "buffer" as inputs


## What is a "Buffer/Threshold" ??
# Buffer is a threshold used to derive technical indicator trading signal based on # historical days & margin rate.
# Historical days are the number of days in recent history that the short tend indicator supassed the longer trend
# indicator when the next-closing price was higher than the  same-day closing price.
# Margin is the % of this difference actually used in the derivation of the trading signal.
```

Example if for the last 30days when the next-day closing price was above the same-day closing price

and the shorter trend was higher than the longer trend indicator, the average was $10.

##If the margin is 50% then we would only trigger a buy signal if the short trend tech. indicator was

>$5 (50% of $10) above the long-trend tech. indicator.

##5) Select winning model across all three cases (i.e model with highest Accuracy)

```
###install.packages("quantmod")
###install.packages("PerformanceAnalytics")
###install.packages("TTR")

#Artificial Neural Network
###install.packages("neuralnet")
###install.packages("xlsx")

###install.packages("stringi" )


library(neuralnet)
library("xlsx")

library(quantmod)
library(PerformanceAnalytics)
library(TTR)

library("stringi" )
# Rcode Reference--> : https://gist.github.com/geoquant/3118985
```

```
#####################################################################
### User Input ######################################################
#####################################################################
#####################################################################

####### Data Collection            ##########################

#####################################################################

getSymbols("SPY", from = "2015-01-01", to="2018-06-08")
close <- SPY$SPY.Close

open <- SPY$SPY.Open
high <- SPY$SPY.High
low <- SPY$SPY.Low

min(index(close))
max(index(close))
equity <-10000
riskFreeRate<-.04/252 # Annual Trading days = 252

#####################################################################

####### Start of EXPLORATORY  Analysis ##########################

#####################################################################
```

```
head(close)

ma10<-SMA( close ,n=10)
ma20<-SMA(close ,n=20)
ma100<-SMA(close ,n=100)
ma200 <- SMA(close ,n=200)
expma <- EMA(close ,n=100)


DsetNew <- as.data.frame(na.omit(cbind(close,ma10,ma20,ma100,ma200,expma)))
head(DsetNew)
names(DsetNew) <- c("ClosingPr","ma10","ma20","ma100","ma200","expma")
myDates= rownames(DsetNew)
rownames(DsetNew) = NULL
DsetNew= cbind(myDates,DsetNew)


summary(DsetNew)
str(DsetNew)

myMin =round(min(na.omit(cbind(close,ma10,ma20,ma100,ma200,expma)),2))
myMax =round(max(na.omit(cbind(close,ma10,ma20,ma100,ma200,expma)),2))
myMin
myMax

graphics.off()
par("mar")
par(mar=c(1,1,1,1))
```

```
#par(reset=TRUE)
dev.off()
plot(x=DsetNew$myDates,y=DsetNew$ClosingPr,ylim=c(myMin,myMax),rgb(1, 0, 0))

par(new=T)
plot(x=DsetNew$myDates,y=DsetNew$ma10,ylim=c(myMin,myMax),col="blue")

par(new=T)
plot(x=DsetNew$myDates,y=DsetNew$ma20,ylim=c(myMin,myMax),col="red")

par(new=T)
plot(x=DsetNew$myDates,y=DsetNew$ma100,ylim=c(myMin,myMax),col="green")

par(new=T)
plot(x=DsetNew$myDates,y=DsetNew$ma200,ylim=c(myMin,myMax),col="purple")

par(new=T)
plot(x=DsetNew$myDates,y=DsetNew$expma,ylim=c(myMin,myMax),col="pink")


par(mgp = c(0, 1, 0))
Closing_and_movingAvgs <-  na.omit(merge(close,ma10,ma20,ma100,ma200,expma))
plot(Closing_and_movingAvgs ,col=c("black","blue","red","purple","green","pink"))
legend("topleft", legend=c("closing Pr.", "ma10","ma20","ma100","ma200","expma"),
      col=c("black","blue","red","purple","green","pink"), lty=c(1,1,1,1,1,1), cex=0.6,
      title="Line types",horiz = F,bty="n")


par(mar = c(2.5,2.5,2.0,2.0))
```

```
head(DsetNew)
# No missing values in the columns

sapply(DsetNew[,-1], function(x) sum(is.na(x)))



# Calculating mean & plotting
sapply(DsetNew[,-1], function(x) mean(x))
meanvals <-sapply(DsetNew[,-1], function(x) mean(x))
bp1 <- barplot(meanvals,ylim=c(0,300),main="Mean values")
text(x=bp1,y=meanvals,labels=round(meanvals,4), pos=1,cex=1.0)

# Calculating median & plotting
bp2 <-barplot(sapply(DsetNew[,-1], function(x) median(x)),ylim=c(0,340),main="Median values")
text(x=bp2,y=meanvals,labels=round(meanvals,4), pos=1,cex=0.8)

# Calculating & plotting variance
max(sapply(DsetNew[,-1], function(x) var(x)))
bp3 <- barplot(sapply(DsetNew[,-1], function(x) var(x)),ylim=c(0,700),main="Variance values")
text(x=bp3,y=meanvals,labels=round(meanvals,4), pos=1,cex=0.8)

head(DsetNew)

#boxplot
par(mar = c(2.5,2.5,2.0,2.0))
```

```
boxplot(DsetNew[,-1],cex.lab=3,cex.axis=2,col=rainbow(6))



head(DsetNew[,-1])
mycorExplore <- cor(DsetNew[,-1])

#remove.packages("caret")

###install.packages('Rcpp', dependencies = TRUE)
###install.packages('caret', dependencies = TRUE)
library(caret)

#remove.packages("ggplot2")
###install.packages("ggplot2")
library(ggplot2)

method = "circle"
###install.packages("ggcorrplot")
library(ggcorrplot)



# method = "circle"
ggcorrplot(mycorExplore, method = "circle")
print(mycorExplore)




#####################################################
```

```
####### End of EXPLORATORY  ########################
##########################################

# We buy and sell at the open if we get the signal to do so.
#Signal is based on an expeced increase or decrease in the closing
# price for the next day.

# Indicator Rules: n-Day Moving Average or n-Day Expo Moving Average

GetPosBH <- function(close){
MarginRate = 0
HistdaysCalc = 0
pos <- close
pos[1:length(pos),] <- 1 # buy/hold for the entire investment period.
Strategyname <- "Buy_Hold"
names(pos)<- Strategyname
return(list(pos,MarginRate,HistdaysCalc))
}

# tests to confirm if start and end dates are within index date range.
IsvalidRtrange <- function(indexA,startdt,enddt){

  startdt = as.Date(startdt)
  enddt = as.Date(enddt)

  if (class(indexA)[1]=="list"){
    index = indexA[[1]]
```

```
  } else
  {
    index= indexA
  }


  indxstart = min(index(index))
  indxend = max(index(index))
  validrange = "FALSE"
  if (startdt < enddt){
    if (startdt >= indxstart && startdt <= indxend){
      if (enddt <= indxend && enddt >= indxstart){
        validrange = "TRUE"
      } else
      { print("End date entered not valid- must be between index range.")
        print(paste("Index st. date :",indxstart," and Index end date",indxend,"Vs. End date
entered:",enddt))
      }
    } else
    { print("start date entered not valid- must be between index range.")
      print(paste("Index st. date :",indxstart," and Index end date",indxend,"Vs. start date
entered:",startdt))
    }
  } else {print("not a valid date range, start date later than end date")}
  return(validrange)
}


#########################################################################
```

```
# Takes any list of positions or indices or a single
# pos. or indexand returns the same subset by start and end date entered.
TruncBasedOnDates <- function(indexA,startdt,enddt){
 startdt = as.Date(startdt)
 enddt = as.Date(enddt)

 if (class(indexA)[1]=="list"){
  validrange = IsvalidRtrange(indexA,startdt,enddt)
  index = indexA[[1]]
 } else
 { validrange = IsvalidRtrange(list(indexA),startdt,enddt)
 index= indexA
 }


 #datesExist = DodatesExist(indexA,startdt,enddt)
 #print(validrange)
 if (validrange=="TRUE"){
  diststDt = index(index)-(startdt)
  distEndDt = index(index)-(enddt)
  MinDistTostDt = min(abs(diststDt))
  MinDistToenddt = min(abs(distEndDt))

  # Accounting for direction of closest date
  # Determining if the closest date is below or above then findng the row number
  if (sum(which(diststDt==(-1*min(abs(diststDt))))) !=0){
   rnumSt= which(diststDt==(-1*min(abs(diststDt))))
  } else {rnumSt= which(diststDt==(min(abs(diststDt))))}
```

```
  if (sum(which(distEndDt==(-1*min(abs(distEndDt))))) !=0){
    rnumEnd = which(distEndDt==(-1*min(abs(distEndDt))))
  } else {rnumEnd = which(distEndDt==(min(abs(distEndDt))))}

  newstartdate = index(index)[rnumSt]
  newenddate = index(index)[rnumEnd]

  subindex= index[rnumSt:rnumEnd]
  if (newstartdate != startdt){
    print(paste(("Nearest start date:"),newstartdate))
  }
  if (newenddate != enddt){
    print(paste(("Nearest end date"),newenddate))
  }

  Trunced <- list(subindex)
  return(list(Trunced,as.Date(newstartdate),as.Date(newenddate)))
} else
{
  print("Please re-enter with valid date range.")
  print("Target range must be within existing Index range")
  startrangeDt = min(index(index))
  endrangeDt = max(index(index))
  print(paste("Index Start date range:",startrangeDt))
  print(paste("Index End date range:",endrangeDt))
  print("Versus")
  print("Target range entered:")
  print(paste("Target start date:",startdt))
  print(paste("Target End date range:",enddt))
```

```
   return()
  }
}
```

```
# "Cross" means if n1 MAvg. is greater than the n2 MAvg. then we buy, otherwise we sell
GetInputValues <- function(close,MAtype="SMA",n1,n2=0,Crosstype="SMA"){
  IsValidInputEntry <- function(MAtype,n1,n2,Crosstype="SMA"){
    if ( ((MAtype =="SMA") && (n1 != 0) && (n2 ==0) && Crosstype=="SMA") | ((MAtype =="EMA") && (n1 != 0) && (n2 ==0) && Crosstype=="SMA")  | ((MAtype =="Cross") && (n1 != 0) && (n2 !=0) && ((Crosstype=="SMA") | (Crosstype=="EMA"))) ){
      ValidEntry = "TRUE"
    } else  ValidEntry = "FALSE"
    return(ValidEntry)
  }

  SMAstr = "SMA"
  EMAstr = "EMA"
  Crossstr= "Cross"

  if (IsValidInputEntry(MAtype,n1,n2)=="TRUE"){

    if (MAtype ==SMAstr) { # Simple/ Exponential moving average NOT Cross
      RawValues = (SMA(close,Matype=SMAstr,n1))
      names(RawValues) = gsub(" ","",paste(MAtype,n1))
```

```
  } else # Incorrect SMA entry
  { if (MAtype==EMAstr){
    RawValues = (EMA(close,Matype=EMAstr,n1))
    names(RawValues) = gsub(" ","",paste(MAtype,n1))
  } else
  { if (MAtype==Crossstr){
    if (Crosstype==SMAstr){
      n1Values = SMA(close,n1)
      n2Values = SMA(close,n2)
      RawValues = ifelse(n1Values > n2Values,1,-1)
      names(RawValues) = gsub(" ","",paste(MAtype,Crosstype,n1,"_",n2))
    } else
    {
      n1Values = EMA(close,n1)
      n2Values = EMA(close,n2)
      RawValues = ifelse(n1Values > n2Values,1,-1)
      index(RawValues) <- as.Date(index(RawValues))
      names(RawValues) = gsub(" ","",paste(MAtype,Crosstype,n1,"_",n2))
    }
  }
  }
  }
  ResultVals = na.omit(RawValues)
  return(ResultVals)
 } else
 { print("Invalid entry-kindly re-enter")
  return()
 }
}
```

```
GetNxtDyClose <- function(myclose){
  myNxtDyClose <-
xts(as.numeric(myclose[2:length(myclose)]),order.by=index(myclose)[(1:(length(myclose)-1))])
  names(myNxtDyClose) = "NxtDyClosePr"
  return(myNxtDyClose)
}



TruncNMergeToDf <- function(inputlist,startdt,enddt){
  lstsize = length(inputlist)
  namesls= c()

  for (i in 1:lstsize){
    ##must Trunc then match
    inputlist[[i]] = TruncBasedOnDates(indexA=inputlist[[i]],startdt,enddt)[[1]][[1]]
    namesls[i] = names(inputlist[[i]])
  }
  Dfinputs= as.data.frame(myMerge(inputlist))
  Date = rownames(Dfinputs)
  rownames(Dfinputs)= NULL
  NewDf= cbind(Date=as.Date(Date),Dfinputs)
  colnames(NewDf)[-1] = namesls
  return(NewDf)
}
```

```
AvgDstMAvgRule<- function(MAtype,n,HistdaysCalc=90){
  # Calculates the average amt. that the closing price is above the MAvg. Price
  # When rule holds (i.e. when the closing moves up next day.)
  nx = GetNxtDyClose(close)
  cl= close[1:(length(close)-1),]

  myMA = GetInputValues(close,MAtype,n)



  m = myMerge(list(myMA,nx,cl))
  names(m)= c(names(myMA),names(nx),names(cl))
  Ma = m[,1]
  nx = m[,2]
  cl = m[,3]

  #diff_cl_Ma = cl-Ma
  #diff_nx_cl = nx-cl
  diff_cl_Ma = (cl-Ma)/Ma
  diff_nx_cl = nx-cl
  dir_cl_Ma= ifelse((diff_cl_Ma)>0,1,0)
  dir_nx_cl = ifelse((diff_nx_cl)>0,1,0)

  AmtAbove = ifelse(dir_cl_Ma==1,ifelse(dir_nx_cl==1,diff_cl_Ma,0),0)
  names(AmtAbove) = "Diff_cl_maPosChgPr"
  L1 = length(index(AmtAbove))
```

```
 maxpastdys = L1-1
 if (HistdaysCalc<=maxpastdys){
  StopLen = HistdaysCalc
 } else
 {
  StopLen = maxpastdys
  #print(paste("Number of History days changed from",HistdaysCalc,"to:",StopLen,"days","which
reflects total history days"))
 }
 ActualHistDaysMAX = StopLen
 zero = mat.or.vec(L1,1)

 TrackAvg = xts(zero,order.by=index(AmtAbove))
 names(TrackAvg) = gsub(" ","",paste("MaxHistdys_",ActualHistDaysMAX))

 for (j in 2:L1){
  mysum = 0
  Cnt1 = 0
  dysleft = L1-j+1
  #tester=c()
  # Starts at the most recent date and goes back StopLen #of days ( eg. 90-days from today)
  if (dysleft < StopLen){
   StopLen = dysleft
  }
  for (i in 1:StopLen){
   if (AmtAbove[(L1-i+1-j+1),1]!=0){
    mysum= mysum + as.numeric(AmtAbove[(L1-i+1-j+1),1])
    Cnt1 = Cnt1 +1
   }
```

```
    }

    # keeps track of the rolling avg. of the amt by which the closing exceeded the moving avg.
    # N.B. only counts the days when this occurs (i.e. closing is above MAvg.)
    if (Cnt1 !=0){
      TrackAvg[dysleft+1]=mysum/Cnt1
    } else TrackAvg[dysleft+1] = 0
  }

  return(list(TrackAvg,ActualHistDaysMAX))
}



GetPosMovAvg <- function(close,MAtype,n,MarginRate=0,HistdaysCalc=90){
  # Indicator Rules: n-Day Moving Average
  # - buy when daily price is > n-day moving average
  # - sell and move to cash when daily price < n-day moving average
  Buffer= 1+(MarginRate * AvgDstMAvgRule(MAtype,n,HistdaysCalc)[[1]])
  head(Buffer)
  if (MAtype=="SMA"){
     a <- SMA(close,n)
  } else if (MAtype=="EMA"){
     a <- EMA(close,n)
  }
  range <- na.omit(a)
  position <- ifelse(close[n:length(close)] > (range*Buffer),1,-1) #  1 buy or hold,-1 sell
  index(position) <- as.Date(index(position))
  Strategyname <- gsub(" ","",paste(MAtype,n,".MRt",MarginRate,".HDys",HistdaysCalc))
```

```
  names(position)<- Strategyname
  return(list(position,MarginRate,HistdaysCalc))
}




AvgDstCrossRule<- function(MAtype="SMA",n1,n2,HistdaysCalc=90){
  # Calculates the average amt. that the closing price is above the MAvg. Price
  # When rule holds (i.e. when the closing moves up next day.)

  nx = GetNxtDyClose(close)
  cl= close[1:(length(close)-1),]

  myMAn1 = GetInputValues(close,MAtype,n1)
  myMAn2 = GetInputValues(close,MAtype,n2)

  m = myMerge(list(myMAn1,myMAn2,nx,cl))
  names(m)= c(names(myMAn1),names(myMAn2),names(nx),names(cl))
  Ma1 = m[,1]
  Ma2 = m[,2]
  nx = m[,3]
  cl = m[,4]



  diff_Ma1_Ma2 = (Ma1-Ma2)/Ma2
  diff_nx_cl = nx-cl
  dir_Ma1_Ma2 = ifelse((diff_Ma1_Ma2)>0,1,0)
  dir_nx_cl = ifelse((diff_nx_cl)>0,1,0)
```

```
AmtAbove = ifelse(dir_Ma1_Ma2==1,ifelse(dir_nx_cl==1,diff_Ma1_Ma2,0),0)
names(AmtAbove) = "Diff_Ma1_Ma2ChgPr"
L1 = length(index(AmtAbove))



maxpastdys = L1-1
if (HistdaysCalc<=maxpastdys){
  StopLen = HistdaysCalc
} else
{
  StopLen = maxpastdys
  #print(paste("Number of History days changed from",HistdaysCalc,"to:",StopLen,"days","which reflects total history days"))
}
ActualHistDaysMAX = StopLen
zero = mat.or.vec(L1,1)

TrackAvg = xts(zero,order.by=index(AmtAbove))
names(TrackAvg) = gsub(" ","",paste("MaxHistdys_",ActualHistDaysMAX))

for (j in 2:L1){
  mysum = 0
  Cnt1 = 0
  dysleft = L1-j+1
  #tester=c()
  # Starts at the most recent date and goes back StopLen #of days ( eg. 90-days from today)
  if (dysleft < StopLen){
    StopLen = dysleft
```

```
  }
  for (i in 1:StopLen){
    if (AmtAbove[(L1-i+1-j+1),1]!=0){
      mysum= mysum + as.numeric(AmtAbove[(L1-i+1-j+1),1])
      Cnt1 = Cnt1 +1
    }
  }

  # keeps track of the rolling avg. of the amt by which the closing exceeded the moving avg.
  # N.B. only counts the days when this occurs (i.e. closing is above MAvg.)
  if (Cnt1 !=0){
    TrackAvg[dysleft+1]=mysum/Cnt1
  } else TrackAvg[dysleft+1] = 0
 }

 return(list(TrackAvg,ActualHistDaysMAX))
}



GetCrossPos <- function(close,MAtype,n1,n2,MarginRate=0,HistdaysCalc=90){
 # Indicator Rules: n1,n2 Crossover
 # - buy when n1-day moving average > n2-day moving average
 # - sell and move to cash when n1-day moving average < n2-day moving average

 if (n1<n2){
  Buffer= 1 +(MarginRate * AvgDstCrossRule(MAtype,n1,n2,HistdaysCalc)[[1]])

  if (MAtype=="SMA"){
    a1 <- SMA(close,n1)
```

```
    a2 <- SMA(close,n2)
  } else if (MAtype=="EMA"){
    a1 <- EMA(close,n1)
    a2 <- EMA(close,n2)
  }
  mergen1_n2 <-na.omit(merge(a1[n2:length(a1)],a2))
  colnames(mergen1_n2) <- c(gsub(" ","",paste(MAtype,n1)),gsub(" ","",paste(MAtype,n2)))
  positionn1_n2<- ifelse(mergen1_n2[,1] > (mergen1_n2[,2]*Buffer),1,-1) # 1 buy/hold, -1 sell
(otherwise)
  index(positionn1_n2) <- as.Date(index(positionn1_n2))
  Strategyname <- gsub(" ","",paste(MAtype,n1,"_",n2,".MRt",MarginRate,".HDys",HistdaysCalc))
  names(positionn1_n2)<- Strategyname
  return(list(positionn1_n2,MarginRate,HistdaysCalc))
 } else (print("Error: n1 must be less than n2"))
}



GetreturnIndex <- function(positionA,close,open){
 # Blank data frame (Df) to store closing price, position, number of shares, market value
 position= positionA[[1]]

 newl = MatchDates(list(position,close,open))
 position= newl[[1]]
 close = newl[[2]]
 open= newl[[3]]

 zero = mat.or.vec(length(position),1)
 blankShares <- zoo(zero, order.by=index(position))
 blankMV<- zoo(zero,order.by=index(position))
```

```
 blankCashPos<- zoo(zero,order.by=index(position))
 blankTotMktVCashPos<- zoo(zero,order.by=index(position))



 # Let start date for closing prices align with that of position
 #startdate= index(position)[1]
 #startindex=which(index(close)[]==startdate)


 Df <- data.frame(na.omit(merge(close,open,position, blankShares,
blankMV,blankCashPos,blankTotMktVCashPos)))


 #Df <-
data.frame(na.omit(merge(close[startindex:length(close)],open[startindex:length(open)],position,
blankShares, blankMV,blankCashPos,blankTotMktVCashPos)))


 names(Df) <- c("close","open","position","Shares","MarketValue","CashPos","TotMVPlusCash")


 # Assign first number of shares and cash position; No trading on day0 only cash count.
 Df[1,"CashPos"] <- equity # My initial cash is my cash starting position
 Df[1,"MarketValue"] = Df[1,"Shares"] * Df[1,"close"]
 Df[1,"TotMVPlusCash"] <- Df[1,"MarketValue"] + Df[1,"CashPos"] # Total MarketValue is always MV
plus Cash



#  if(Df[1,3] == 1){
#    if (Df[1,"CashPos"] >= Df[1,"open"]){
#       Df[1,"Shares"] = floor(Df[1,"CashPos"]/Df[1,"open"])
#    }
#  }
#  Df[1,"MarketValue"] = Df[1,"Shares"] * Df[1,"close"]
```

```
# Number of Shares and Market Value:
# a for loop iterates through the vector specified: in this example, we're
# looping sequentially through a vector that starts at 2 and goes to the
# Next day trading strategy is dataed on the same day.
# That is, Trading strategy for Tuesday is dated on Monday. So, position value
# rep. trading staretegy must be checked at time t-1 for action at time t.

#head(Df)
# end of the data frame
#i=2
nrowsDf <- nrow(Df)

for(i in 2:nrowsDf){

 # blank Shares: three conditions:
 # if the decision is to buy, go ahead if cash is available (otherwise hold).

 # check next-day trading strategy given previous day (hence t-1).
 if(Df[i-1,"position"] == 1){
  sharesTraded = floor(Df[i-1,"CashPos"]/Df[i,"open"])
  Df[i,"Shares"] = Df[i-1,"Shares"] + sharesTraded
 }else if(Df[i-1,"position"]==-1){ # if decision is to sell, go ahead if you have shares (otherwise hold)
  sharesTraded = Df[i-1,"Shares"]
  Df[i,"Shares"] = Df[i-1,"Shares"] - sharesTraded
 }
 Df[i,"MarketValue"] = Df[i,"Shares"] * Df[i,"close"]
 Df[i,"CashPos"] = Df[i-1,"CashPos"] -(Df[i,"open"]*Df[i-1,"position"]*sharesTraded)
 Df[i,"TotMVPlusCash"] = Df[i,"MarketValue"] + Df[i,"CashPos"]
```

```
  }
  # index
  indexn = Df[,"TotMVPlusCash"]/Df[1,"TotMVPlusCash"]
  indexXTS = xts(indexn[1:length(index(position))],order.by=index(position))
  names(indexXTS) = names(position)
  return(list(indexXTS,Df))
}



#####################################################################


ReturnNRiskMetrics <- function(index){
  index = index[[1]]
  daily <- dailyReturn(index)
  annualizedReturn <- 100*Return.annualized(daily, scale = 252, geometric = TRUE)
  cumulativeReturn <- 100*Return.cumulative(daily)

  # Risk Statistics
  annualizedStd <-sd.annualized(daily,scale=252)
  maxDrawdown <- maxDrawdown(daily)
  duration <- sortDrawdowns(findDrawdowns(daily))

  # Sharpe Ratio
  excess <- daily-riskFreeRate
  dailystd <- sd(excess)
  avgExcess <- mean(excess)
  if (dailystd !=0){
```

```
   sharpe <- 10*(avgExcess/dailystd*sqrt(252))
  } else
   {
     sharpe <- 0
   }
  MetricsRR <- matrix(c(annualizedReturn,cumulativeReturn,sharpe),nrow=3,ncol=1,byrow=TRUE)
  rownames(MetricsRR) <- c("annualizedReturn","cumulativeReturn","sharpe")
  colnames(MetricsRR) <- names(index)
  MetricsRR <- as.data.frame(MetricsRR)
  return(MetricsRR)
}



############################################################


RMetricsBasedOnDates <- function(IndxListA,startdt,enddt){
  # Takes a list of indices or a list of a list
  if (class(IndxListA[[1]])[1]!="list"){
    IndxList = list(IndxListA)
  } else IndxList = IndxListA
  if (length(IndxList)>1){
    NewIndxList =MatchDates(IndxList)
  } else  NewIndxList= IndxList
  LCnt = length(NewIndxList)
  RetList =list()
  if (IsvalidRtrange(NewIndxList[[1]],startdt,enddt)=="FALSE"){
    print("Please re-enter target range")
```

```
    return()
  } else
  {
    for (i in 1:LCnt){
      sublist = TruncBasedOnDates(NewIndxList[[i]],startdt,enddt)[[1]]
      newstartdate = TruncBasedOnDates(NewIndxList[[i]],startdt,enddt)[[2]]
      newenddate = TruncBasedOnDates(NewIndxList[[i]],startdt,enddt)[[3]]
      RetList[[i]] = ReturnNRiskMetrics(sublist)
    }
    if ((newstartdate !=startdt) |  (newenddate !=enddt)){
      print("change to date range")
    }

    if (newstartdate !=startdt) {
      print(paste("New start date:",newstartdate))
    }
    if (newenddate !=enddt){
      print(paste("New end date:",newenddate))
    }
    #print(paste("Actual Start date:",newstartdate,"Actual End date:",newenddate))
    return(list(RetList,newstartdate,newenddate))
  }
}
```

```
############################################################
```

```
CreateMatxOfReturns <- function(myRlistA){
  Cnt = length(myRlistA[[1]])
  stdate= myRlistA[[2]]
  enddate= myRlistA[[3]]

  myRlist = myRlistA[[1]]

  Allbind <- t(myRlist[[1]][[1]])

  rownames(Allbind)[1] = colnames(myRlist[[1]])[1]
  colnames(Allbind) = rownames(myRlist[[1]])
  if (Cnt>=2){
    for (i in 2:Cnt){
      Allbind <- rbind(Allbind,t(myRlist[[i]][[1]]))
      rownames(Allbind)[i] = colnames(myRlist[[i]])[1]
    }
  }
  return(Allbind)
}



############################################################
```

```
Unmerge <- function(mergedxts){
  numcol = dim(mergedxts)[2]
  Unmergedz = list()
  for (i in 1:numcol){
    Unmergedz[[i]] = mergedxts[,i]
  }
  return(Unmergedz)
}



myMerge <- function(mylist){
  CntLs <- length(mylist)
  if (class(mylist[[1]])[1]=="list"){
    Indx = mylist[[1]][[1]]
  } else  Indx = mylist[[1]]
  newz = Indx
  if (CntLs >1){
    for (i  in 2:CntLs){
      #test if object is list rep. index or zoo rep. positiion
      if (class(mylist[[i]])[1]=="list"){
        Indx = mylist[[i]][[1]]
      } else  Indx = mylist[[i]]
      newz = merge(newz,Indx,join="inner")
    }
  }
  return(na.omit(newz))  # returns a merged list
}
```

```
MatchDates <- function(mylist){
  newz = myMerge(mylist)
  if (length(newz) >1 ){
    newmylist= Unmerge(newz)
  }
  return(newmylist)
}



ActualDirPrChange <- function(close){
  Dates= index(close)
  N= length(close)
  CurrClosePr = xts(as.numeric(close[1:(N-1)]),order.by=Dates[1:N-1])
  NextDayClosePr = xts(as.numeric(close[2:N]),order.by=Dates[1:N-1])
  ChgPrice = xts((as.numeric(close[2:N]) - as.numeric(close[1:(N-1)])),order.by=Dates[1:N-1])
  DirChange = ifelse(ChgPrice>0,1,ifelse(ChgPrice==0,0,-1))
  names(DirChange) = "Act_DirPrice"
  #head(DirChange)
  return(DirChange)
}



SignalAccMetricsBasedOnDates <- function(mylistSig,close,startdt,enddt){
  SignalAccMetrics <- function(Signal,ActualNextDyPrDir){
    DirChng =ActualNextDyPrDir
    LenDir = length(DirChng)
    Acc = c()
```

```
TP = 0

FP = 0

TN = 0

FN = 0



for (i in 1:LenDir){
 ## Signal is correct if either conditions hold
 ## (i.e. if signal atches the direction or signal is negative when direction is either negative or zero)
 ## Recall our rule is that we only buy when the price is expected to rise otherwise
 # (if the expected price doesn't reflect a change or it falls, we sell)

 if ((Signal[i,1] == DirChng[i,1]) | (Signal[i,1] == -1 && DirChng[i,1] == 0)){
  Acc[i]= 1
 } else (Acc[i]=0)

 #True Positive
 if ((Signal[i,1] ==1) && (DirChng[i,1]==1)){
  TP = TP + 1
 }

 # False Positive
 if ((Signal[i,1] ==1) && ((DirChng[i,1]==0) | (DirChng[i,1]==-1))) {
  FP = FP +1
 }

  #True Negative
 if ( ((Signal[i,1] == -1) && (DirChng[i,1] == 0)) | ((Signal[i,1] == -1) && (DirChng[i,1]==-1) )) {
  TN = TN + 1
```

```
    }

    # False Negative
    if ((Signal[i,1] ==-1) | (DirChng[i,1]==1) ){
      FN=FN +1
    }

  }
  Acc = xts(Acc,order.by=index(DirChng))
  AllFinal = na.omit(merge(Signal,DirChng,Acc))

  names(AllFinal) = c("Signal","Dir_Change","Signal_Acc")
  Tot = length(Acc[,1])
  AccCnt = sum(Acc[,1])
  AccPerc = round((AccCnt/Tot),4)*100

  ErrorCnt = Tot - AccCnt
  ErrorPerc = round((ErrorCnt/Tot),4)*100

  if  ((TP+FP)==0){
   Precision=0
  } else  Precision= round((TP/(TP+FP)),4)*100

  if  ((TP+FN)==0){
   Recall=0
  } else  Recall = round((TP/(TP+FN)),4)*100

  return(list(AccPerc,Precision,Recall))
}
```

```
myActualDirPrChg = ActualDirPrChange(close)
LenSig = length(mylistSig)
mylistSig2 = mylistSig
mylistSig2[[LenSig +1]]=myActualDirPrChg
HistdaysCalc = c()
MarginRate = c()
Ls = length(mylistSig2)
for (i in 1:(Ls-1)){
  MarginRate[i] = mylistSig2[[i]][[2]]
  HistdaysCalc[i] = mylistSig2[[i]][[3]]
}

mylist = MatchDates(mylistSig2)
newActDirChg = mylist[[length(mylist)]]

if (IsvalidRtrange(newActDirChg,startdt,enddt)=="TRUE"){
  TruncActDirChg = TruncBasedOnDates(newActDirChg,startdt,enddt)[[1]][[1]]
  Cnt = length(mylist)-1
  SignalAcc = c()
  SignalPrec = c()
  SignalRecall = c()
  strategyNames = c()
  #BasicStrategyNames =c()
  #MovingAvgType = c()
  #MyisCross=c()
  Actstartdt = c()
  ActEnddt = c()
```

```
   for (i in 1:Cnt){
     myPos =mylist[[i]]
     TruncPos = TruncBasedOnDates(myPos,startdt,enddt)[[1]][[1]]
     SignalAcc[i] = SignalAccMetrics(TruncPos,TruncActDirChg)[[1]]
     SignalPrec [i] = SignalAccMetrics(TruncPos,TruncActDirChg)[[2]]
     SignalRecall[i] = SignalAccMetrics(TruncPos,TruncActDirChg)[[3]]
     strategyNames[i] = c(names(myPos)[1])
     #BasicStrategyNames[i] = GetBstrgy(strategyNames[i])
     #MovingAvgType[i] = GetMAType(strategyNames[i])
     #MyisCross[i] = isCross(strategyNames[i])
     Actstartdt[i]=as.Date(TruncBasedOnDates(myPos,startdt,enddt)[[2]]) #tester
     ActEnddt[i]=as.Date(TruncBasedOnDates(myPos,startdt,enddt)[[3]])  #tester
   }

   mymat =
matrix(c(SignalAcc,SignalPrec,SignalRecall,MarginRate,HistdaysCalc),nrow=Cnt,ncol=5,byrow=F)
   rownames(mymat) <- strategyNames
   colnames(mymat) <- c("Signal.Accuracy","Signal.Precision","Signal.Recall","Margin.Rate","Hist.days")
  } else
   { print("Invalid range")
     return()
   }
  #print(paste("Signal Accuracy Start date: ",as.Date(Actstartdt[i]),"End Date:",as.Date(ActEnddt[i])))
#tester

  return(mymat)
}

#Sigtester2 = SignalAccMetricsBasedOnDates(mylistSig=myposlist2,close,startdt="2018-01-
02",enddt="2018-06-06")
```

```
#head(Sigtester2)



CombineMat <- function(MatA,MatB){
  Matls = list(MatA,MatB)
  if (nrow(Matls[[1]]) !=   nrow(Matls[[2]])){

    lenList = length( Matls)
    maxMatInx = 1

    if (nrow(Matls[[i+1]]) >  maxMatInx )
    for (i in 1:(lenList-1)){

      if (nrow(Matls[[i+1]]) >  maxMatInx ){
        maxMatInx = i+1
        maxrvalue = nrow(Matls[[maxMatInx]])
      }
    }



    maxrvalue = nrow(Matls[[maxMatInx]])
    minMatInx = lenList - maxMatInx
    minrvalue = nrow(Matls[[minMatInx]])

    Diffrows = maxrvalue - minrvalue
    # Add "missing" # of rows
    AddRows = matrix(rep(999999999,Diffrows),nrow=Diffrows,ncol=ncol(Matls[[minMatInx]]),byrow=T)
    rownames(AddRows)= c(rownames(Matls[[minMatInx]]))
```

```
   newMat = rbind(Matls[[minMatInx]],AddsRows)


   FinalMatrix = cbind(Matls[[maxMatInx]],newMat)
  } else {FinalMatrix = cbind(Matls[[1]],Matls[[2]])}


  return(FinalMatrix)
}




GetAllReturnsFrDates  <- function(PoslistA,startdt,enddt){
  if (class(PoslistA[[1]])[[1]]==  "list"){
    Poslist = PoslistA
  } else
  { if (class(PoslistA[[1]])[[1]]== "xts"){
      Poslist = PoslistA[[1]]
      myRIndex =GetreturnIndex(myPos,startdt,enddt)
      MyRMetrics = list(RMetricsBasedOnDates(myRIndex,startdt,enddt))
      newstartdate= MyRMetrics [[2]]
      newenddate= MyRMetrics [[3]]
      return(list(MyRMetrics,newstartdate,newenddate))
    }
  }
  Lenls = length(Poslist)
  myR = list()
  for (i in 1:Lenls){
   myPos = Poslist[[i]]
   myR[[i]] = GetreturnIndex(myPos,close,open)
   }
```

```
  AllmyRMetrics = RMetricsBasedOnDates(myR,startdt,enddt)
  FinalRMat = CreateMatxOfReturns(AllmyRMetrics)
  return(FinalRMat)
}

#####################################################################
#####    MAIN MODULE        ###################################
#####################################################################

#############################################################################################
#####################
#############################################################################################
#####################

##       MAIN EVALUATION FOR TECH ANALYSIS

#############################################################################################
#####################
#############################################################################################
#####################
##install.packages('parallel')
```

```
##install.packages('foreach')
##install.packages('doParallel')
##install.packages('psych')
##install.packages('magrittr')
##install.packages('PerformanceAnalytics')
library(parallel)
library(foreach)
library(doParallel)
library(psych)
library(magrittr)
library(xts)
library(TTR)
library('PerformanceAnalytics')



lsStDates = c("2018-01-02","2017-07-28","2015-01-02")
lsEndDates = c("2018-06-06","2017-12-29","2017-07-27")

firstRow= which(index(close)=="2015-10-16") # start of training priod used fo ML later
lastRow = which(index(close)=="2017-12-29") # end of validation period used for ML later

widthK=round(((lastRow-firstRow)/10),0) #56



LRnum= length(seq(firstRow,lastRow,widthK))
lastbatch= seq(firstRow,lastRow,widthK)[LRnum]

# No Buffer case
myMarginRate=0
```

```
myHdys=0

library(TTR)

posBH <- GetPosBH(close)

position50 <- GetPosMovAvg(close,"SMA",50,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
position100 <- GetPosMovAvg(close,"SMA",100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
position200 <- GetPosMovAvg(close,"SMA",200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
position50EMA <- GetPosMovAvg(close,"EMA",50,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
position100EMA <- GetPosMovAvg(close,"EMA",100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
position200EMA <- GetPosMovAvg(close,"EMA",200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos100_200 <- GetCrossPos(close,"SMA",100,200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos100_200EMA <- GetCrossPos(close,"EMA",100,200,MarginRate=myMarginRate
,HistdaysCalc=myHdys)
pos50_200 <- GetCrossPos(close,"SMA",50,200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos50_200EMA <- GetCrossPos(close,"EMA",50,200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos10_100 <- GetCrossPos(close,"SMA",10,100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos10_100EMA <- GetCrossPos(close,"EMA",10,100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos50_100 <- GetCrossPos(close,"SMA",50,100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
pos50_100EMA <- GetCrossPos(close,"EMA",50,100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)

myposlist=
list(posBH,position50,position100,position200,position50EMA,position100EMA,position200EMA,pos100
_200,pos100_200EMA,pos50_200,pos50_200EMA,pos10_100,pos10_100EMA)
myposlist=list(posBH,position50)

no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
registerDoParallel(cl)
```

```
##No Buffer case ##

DateGrpsIndiv <- foreach
(Rnum=seq(firstRow,lastRow,widthK),.packages=c("foreach","xts","quantmod","PerformanceAnalytics",
"TTR"))%dopar%{


  startdt = index(close)[Rnum]

  if (Rnum==lastbatch){

    enddt = index(close)[lastRow]

  } else enddt = index(close)[(Rnum+widthK-1)]


  myReturnsMetrics = GetAllReturnsFrDates(myposlist,startdt,enddt) # Validation period

  #mySigMetrics=
SignalAccMetricsBasedOnDates(mylistSig=myposlist,close,startdt=Mystartdt,enddt=Myenddt)

  mySigMetrics= SignalAccMetricsBasedOnDates(mylistSig=myposlist,close,startdt ,enddt)

  AllSignNReturnsMetrics  = CombineMat(myReturnsMetrics,mySigMetrics)

}

stopCluster(cl)


sumR = 0


for (i in 1:LRnum){

  sumR= sumR + DateGrpsIndiv[[i]]

}


meanDateGroups= sumR/LRnum


#### Top Performing Technical Indicator ###

meanDateGroups[order(-meanDateGroups[,"Signal.Accuracy"]),]
```

Winner is : EMA50_200 for assessment period : training period used fo ML

annualizedReturn cumulativeReturn sharpe Signal.Accuracy Signal.Precision

#Buy_Hold 11.656503 1.8015304 11.5438198 52.228 52.228

#EMA50_200.MRt0.HDys0 9.364486 1.4980451 5.8363260 52.228 41.783

#EMA100_200.MRt0.HDys0 9.870187 1.5951893 10.2857134 52.227 37.497

Signal.Recall Margin.Rate Hist.days

#Buy_Hold 50.000 0 0

#EMA50_200.MRt0.HDys0 35.909 0 0

#EMA100_200.MRt0.HDys0 33.265 0 0

write.table(meanDateGroups,"TechsInd_NoBuffer_RevisedExpResults16July.xls",sep="\t",row.names=TRUE)

#---

Winner for assessment period: training & validation used for ML

EMA100_200 & EMA50_200

annualizedReturn cumulativeReturn sharpe Signal.Accuracy Signal.Precision Signal.Recall

#Buy_Hold 15.318390 3.0731839 12.515761 53.983 53.983 50.000

#EMA100_200.MRt0.HDys0 12.483089 2.5200057 10.024581 53.983 43.983 36.429

#EMA50_200.MRt0.HDys0 12.407799 2.5016893 10.992716 53.983 43.983 38.750

#SMA50_200.MRt0.HDys0 11.664934 2.3322165 7.960776 53.805 53.092 39.570 0 0

#SMA100_200.MRt0.HDys0 12.494993 2.5228812 6.873200 53.448 40.983 35.189 0 0

##########

```
#Bufer Case :
################################################################################################



no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
registerDoParallel(cl)


time1 <- system.time(
DateGrpsBuffer <- foreach
(Rnum=seq(firstRow,lastRow,widthK),.packages=c("foreach","xts","quantmod","PerformanceAnalytics",
"TTR"))%dopar%{


  startdt = index(close)[Rnum]
  if (Rnum==lastbatch){
    enddt = index(close)[lastRow]
  } else enddt = index(close)[(Rnum+widthK-1)]


  MasterLs <- foreach (myHdys=  c(30,60,90,999999),.combine='rbind')%dopar%{


      subMetrics <-  foreach (myMarginRate= c(0.3,0.5,0.8,1.0,1.2,1.5),.combine='rbind')%dopar%{


      position50 <- GetPosMovAvg(close,"SMA",50,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
      position100 <- GetPosMovAvg(close,"SMA",100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
      position200 <- GetPosMovAvg(close,"SMA",200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)
      position50EMA <- GetPosMovAvg(close,"EMA",50,MarginRate=myMarginRate
,HistdaysCalc=myHdys)
      position100EMA <- GetPosMovAvg(close,"EMA",100,MarginRate=myMarginRate
,HistdaysCalc=myHdys)
```

```
    position200EMA <- GetPosMovAvg(close,"EMA",200,MarginRate=myMarginRate
,HistdaysCalc=myHdys)

    pos100_200 <- GetCrossPos(close,"SMA",100,200,MarginRate=myMarginRate
,HistdaysCalc=myHdys)

    pos100_200EMA <- GetCrossPos(close,"EMA",100,200,MarginRate=myMarginRate
,HistdaysCalc=myHdys)

    pos50_200 <- GetCrossPos(close,"SMA",50,200,MarginRate=myMarginRate ,HistdaysCalc=myHdys)

    pos50_200EMA <- GetCrossPos(close,"EMA",50,200,MarginRate=myMarginRate
,HistdaysCalc=myHdys)

    pos10_100 <- GetCrossPos(close,"SMA",10,100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)

    pos10_100EMA <- GetCrossPos(close,"EMA",10,100,MarginRate=myMarginRate
,HistdaysCalc=myHdys)

    pos50_100 <- GetCrossPos(close,"SMA",50,100,MarginRate=myMarginRate ,HistdaysCalc=myHdys)

    pos50_100EMA <- GetCrossPos(close,"EMA",50,100,MarginRate=myMarginRate
,HistdaysCalc=myHdys)



    myposlist=
list(posBH,position50,position100,position200,position50EMA,position100EMA,position200EMA,pos100
_200,pos100_200EMA,pos50_200,pos50_200EMA,pos10_100,pos10_100EMA)

    myReturnsMetrics = GetAllReturnsFrDates(myposlist,startdt,enddt) # Validation period

    mySigMetrics= SignalAccMetricsBasedOnDates(mylistSig=myposlist,close,startdt,enddt)

    AllSignNReturnsMetrics  = CombineMat(myReturnsMetrics,mySigMetrics)

 }


 }

}

)

stopCluster(cl)


sumR = 0
```

```
for (i in 1:LRnum){
  sumR= sumR + DateGrpsBuffer[[i]]
}

meanDateGrpsBuffer= sumR/LRnum

#### Top Performing Technical Indicator ###
meanDateGrpsBuffer[order(-meanDateGrpsBuffer[,"Signal.Accuracy"]),]
# Top Performer :EMA50_200.MRt0.8.HDys30   with Acc. = 54.162% versus Buy/Hold Strategy Acc. = 53.983%

#                        annualizedReturn cumulativeReturn       sharpe Signal.Accuracy Signal.Precision Signal.Recall Margin.Rate Hist.days
#EMA50_200.MRt0.8.HDys30         11.53330742      2.30689370  10.06719517         54.162          44.302      37.193        0.8       30
#Buy_Hold                        15.31838953      3.07318394  12.51576142         53.983          53.983      50.000        0.0        0
#EMA100_200.MRt0.3.HDys30        12.48308880      2.52000574  10.02458102         53.983          43.983      36.429        0.3       30
#EMA50_200.MRt0.3.HDys30         12.40779936      2.50168935  10.99271563         53.983          43.983      38.750        0.3       30
#Buy_Hold                        15.31838953      3.07318394  12.51576142         53.983          53.983      50.000        0.0        0
#EMA100_200.MRt0.5.HDys30        12.48308880      2.52000574  10.02458102         53.983          43.983      36.429        0.5       30
#EMA50_200.MRt0.5.HDys30         12.40779936      2.50168935  10.99271563         53.983          43.983      38.750        0.5       30
#Buy_Hold                        15.31838953      3.07318394  12.51576142         53.983          53.983      50.000        0.0        0
#EMA100_200.MRt0.8.HDys30        12.48308880      2.52000574  10.02458102         53.983          43.983      36.429        0.8       30
```

```
colnames(meanDateGrpsBuffer)
meanDateGrpsBuffer[order(-meanDateGrpsBuffer[,4]),4] # Top performers over-all
write.table(meanDateGrpsBuffer,"Tech_ExpResults_Mean_Tues.xls",sep="\t",row.names=TRUE)
head(meanDateGrpsBuffer)
```

```
##############################################################################################
#####################
##############################################################################################
#####################
```

```
#### MACHINE LEARNING
##############################################################################################
```

```
##############################################################################################
#####################
##############################################################################################
#####################
```

Artificial Neural Networks (ANN) - Feedforward Multilayer Neural Network

##

Scale data for neural network

```
myScale <- function(x) {
  xLen = length(x)
  xnew = c()

  # if all the values are the same then assign value to either 10 or 11
  if ( length(unique(x))==1){
    if (x[1]==-1){
      xnew[which(x==-1)] = 0
    }
    if (x[1]==1){
      xnew[which(x ==1)] = 1
    }
  } else
  {
    xnew = ((x - min(x)) / (max(x) - min(x)))
  }

  return(xnew)
}
```

```
UnScale <- function(xnew,xOrig) {
```

```
  xLen = length(xnew)
  UnscaledResult = c()
  # if all the values match then assign value to either 10 or 11
  if (length(unique(xnew))==1){
    if (xnew[1]==0){
      UnscaledResult[which(xnew==0)] = -1
    }
    if (xnew[1]==1){
      UnscaledResult[which(xnew==1)]= 1
    }
  } else
  {
    UnscaledResult = ((xnew)*(max(xOrig)-min(xOrig)))+min(xOrig)
  }
  return(UnscaledResult)
}




GetDatasetML <- function(open,high,low,close){



  lastLabelrownum = (length(close)-1)


  Open = as.numeric(open[1:lastLabelrownum])
  High = as.numeric(high[1:lastLabelrownum])
  Low = as.numeric(low[1:lastLabelrownum])
  Close = as.numeric(close[1:lastLabelrownum])
```

```
  Next.Close = as.numeric(close[2:(length(close))])

  dset <- data.frame(Date=index(close)[1:lastLabelrownum],Open,High,Low,Close,Next.Close)
  #dset[nrow(dset),]
  return(dset)
}

ScaleDset <- function(dset){
  numcols = ncol(dset)
  dsetA = cbind(Date=as.Date(dset[,1]),as.data.frame(apply(dset[,2:numcols],2,myScale)))
  return(dsetA)
}


GetPosML <- function(SamedayPrice,PredNextDyPr,type){
  l = MatchDates(list(SamedayPrice,PredNextDyPr))
  SamedayPrice = l[[1]]
  PredNextDyPr = l[[2]]
  NextDayPos = ifelse((PredNextDyPr > SamedayPrice),1,-1)
  index(NextDayPos) = index(PredNextDyPr)
  #DfML <- as.data.frame(merge(NextDayPos,SamedayPrice,PredNextDyPr))

  names(NextDayPos) <- c(type)

  return(list(NextDayPos,c(0),c(0)))
}
```

```
#########################################################################
##          MAIN MODULE - ML

## Ref.  RE: CRoss validation for Time Series analysis

## https://stats.stackexchange.com/questions/14099/using-k-fold-cross-validation-for-time-series-model-selection

## Blog on "How To Backtest Machine Learning Models for Time Series Forecasting" by by Jason Brownlee on December 19, 2016 in Time Series

#https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

#

#########################################################################

TrainStartDt="2015-10-16" # changed to facilitate ech. indicators as inputs

TrainEndDt= "2017-07-27"


ValidTestStartDt= "2017-07-28"

ValidTestEndDt= "2017-12-29"


Train2StartDt="2015-10-16"

Train2EndDt= ValidTestEndDt



TestStartDt= "2018-01-02"

TestEndDt= "2018-06-06"

#####################################################

## GET ALL INPUTS and Create Dataframe for ML models

#####################################################

## Use top perfomers from Techical Indicators ( No Buffer & Buffer cases)


#Buffer -Top performer
```

```
posEMA50_200MRt0.8Hdys30 <- GetCrossPos(close,"EMA",50,200,MarginRate=0.8 ,HistdaysCalc=30)
head(posEMA50_200MRt0.8Hdys30)

# No Bufer Tech. indicators
SMA_50_200  =   GetInputValues(close,MAtype="Cross",n1=50,n2=200,Crosstype="SMA")
SMA_100_200 =   GetInputValues(close,MAtype="Cross",n1=100,n2=200,Crosstype="SMA")

# Top Performer without Buffer
EMA_50_200  =   GetInputValues(close,MAtype="Cross",n1=50,n2=200,Crosstype="EMA")


EMA_100_200 =   GetInputValues(close,MAtype="Cross",n1=100,n2=200,Crosstype="EMA")


NextCl = xts(as.numeric(close[2:(length(close))]),order.by=index(close)[1:(length(close)-1)])
names(NextCl) = "Next.Close"
head(NextCl)
Modclose = close[1:(length(close)-1)]
# Add signals
CorrectSignal = ifelse(NextCl>Modclose,1,-1)
names(CorrectSignal) = "Signal"
MatchList =
list(posEMA50_200MRt0.8Hdys30,SMA_50_200,SMA_100_200,EMA_50_200,EMA_100_200,open,high,l
ow,close,CorrectSignal,NextCl)
t =myMerge(MatchList)
head(t)

OrigDsetML = cbind(Date = index(t),(as.data.frame(t)))
OrigDsetML = cbind(Date = as.Date(index(t)),(as.data.frame(t)))
```

```
rownames(OrigDsetML)=NULL

head(OrigDsetML)

colnames(OrigDsetML)[c(7,8,9,10)]=c("Open","High","Low","Close")

head(OrigDsetML)




min(OrigDsetML$Date)

max(OrigDsetML$Date)


# Split into 3 parts for training, validation and testing

trainOrigDset <-
OrigDsetML[(which(OrigDsetML$Date==TrainStartDt):which(OrigDsetML$Date==TrainEndDt)),]

train_ <- ScaleDset(trainOrigDset)


ValidOrigDset <-
OrigDsetML[(which(OrigDsetML$Date==ValidTestStartDt):which(OrigDsetML$Date==ValidTestEndDt)),]

Validtest_ <- ScaleDset(ValidOrigDset )


train2OrigDset <-
OrigDsetML[(which(OrigDsetML$Date==Train2StartDt):which(OrigDsetML$Date==Train2EndDt)),]

train2_ <- ScaleDset(train2OrigDset)
```

```
head(ValidOrigDset)

min(ValidOrigDset$Date)

max(ValidOrigDset$Date)

length(ValidOrigDset$Date)

min(Validtest_$Date)

max(Validtest_$Date)

length(Validtest_$Date)


testOrigDset <-
OrigDsetML[(which(OrigDsetML$Date==TestStartDt):which(OrigDsetML$Date==TestEndDt)),]

test_ <- ScaleDset(testOrigDset)


min(testOrigDset$Date)

max(testOrigDset$Date)




####

PredValidate <-
function(myInputs,ann_,myNHL,myNHuL1,myNHuL2=0,myNHuL3=0,myNHuL4=0,myNHuL5=0,ClassifyPr
oblem="False",vDset){

 vDset_ = ScaleDset(vDset)

 predicted.nn <- compute(ann_, vDset_[,myInputs])

 predicted.nn_ <- predicted.nn$net.result


 if (ClassifyProblem=="True"){

  #PosANN  <- xts(UnScale(x=predicted.nn_,xOrig=ValidOrigDset$Signal),order.by=ValidOrigDset$Date)

  prob  <- xts(predicted.nn_,order.by= vDset$Date)

  #converting prob. into signals
```

```
  q= ifelse((round(prob,0)==1),1,-1)
  index(q)= as.Date(index(q))
  PosANN = list(q,c(0),c(0))
 } else
 {
  PredNextDyPr_Vt  <- xts(UnScale(x=predicted.nn_,xOrig= vDset$Next.Close),order.by= vDset$Date)
  a = na.omit(merge(NextCl,PredNextDyPr_Vt,join="inner"))
  # Get MSSE Metric
  MSSE_ANN_Vt  = sum((a[,1]-a[,2])^2)/length(a[,1])
  PosANN <- GetPosML(SamedayPrice=close,PredNextDyPr_Vt,type="ANN")
 }
 indexANN <- GetreturnIndex(PosANN,close,open)
 myReturnsMetrics = GetAllReturnsFrDates(list(PosANN),startdt=
min(vDset$Date),enddt=max(vDset$Date))# Added
 SignalAcc_ANN_Vt <-SignalAccMetricsBasedOnDates(list(PosANN),close,startdt=
min(vDset$Date),enddt=max(vDset$Date))

 if (ClassifyProblem=="True"){
  myR = matrix(c(
myReturnsMetrics[,1:3],SignalAcc_ANN_Vt[,1:3],myNHL,myNHuL1,myNHuL2,myNHuL3,myNHuL4,myNHuL5),ncol=12,byrow=T)
  rownames(myR) = gsub("
","",paste("ANN.HL",myNHL,"[",myNHuL1,",",myNHuL2,",",myNHuL3,",",myNHuL4,",",myNHuL5,"]"))
  colnames(myR) = c("Annualized Return","Cummulative Return","Sharpe
Ratio","Signal.Accuracy","Signal.Precision","Signal.Recall","NHiddenLayers","NHunitsLayer1","NHunitsLayer2","NHunitsLayer3","NHunitsLayer4","NHunitsLayer5")
  return(myR)
 } else
 myR = matrix(c(
myReturnsMetrics[,1:3],SignalAcc_ANN_Vt[,1:3],MSSE_ANN_Vt,myNHL,myNHuL1,myNHuL2,myNHuL3,myNHuL4,myNHuL5),ncol=13,byrow=T)
```

```
  rownames(myR) = gsub("
","",paste("ANN.HL",myNHL,"[",myNHuL1,",",myNHuL2,",",myNHuL3,",",myNHuL4,",",myNHuL5,"]"))

  colnames(myR) = c("Annualized Return","Cummulative Return","Sharpe
Ratio","Signal.Accuracy","Signal.Precision","Signal.Recall","MSSE","NHiddenLayers","NHunitsLayer1","N
HunitsLayer2","NHunitsLayer3","NHunitsLayer4","NHunitsLayer5")

  return(myR)

}

##########################################################################################
################

##########################################################################################
################

##########################################################################################
################

#########################################################################3

## Hybrid Case - Classification (Baseline)

## Case 1

## Inputs : Close only

## Output:  Signal

##


colnames( train_)

myInputs =c(10)  #Inputs :  Close

f = Signal ~ Close
```

```
threshold = 0.005

NHunits = 1:2
maxHlayers = 1




IndexTrStart = which(OrigDsetML$Date==TrainStartDt)
IndexTrEnd = which(OrigDsetML$Date==TrainEndDt)



IndexVStart = which(OrigDsetML$Date==ValidTestStartDt)
IndexVEnd = which(OrigDsetML$Date==ValidTestEndDt)

Nchunks =10
rangeIndex= IndexVEnd - IndexVStart +1
chunksize =  round((rangeIndex/Nchunks),0)
lastchunksize = chunksize -(chunksize*Nchunks - rangeIndex)



Rnum=c(rep(chunksize,(Nchunks-1)),lastchunksize)
LenR = length(Rnum)
```

```
set.seed(3)



no_cores <- detectCores() - 1

cl <- makeCluster(no_cores)

registerDoParallel(cl)



AllR <-
foreach(Cnt=1:LenR,.packages=c("foreach","xts","quantmod","PerformanceAnalytics","TTR"))%dopar%{

  if (Cnt==1){

    stepsize = 0

    IndexVEnd =IndexVStart+ Rnum[1] -1

  } else stepsize = Rnum[Cnt]

  IndexTrStart = IndexTrStart+ stepsize

  IndexTrEnd = IndexTrEnd + stepsize

  IndexVStart= IndexVStart + stepsize

  IndexVEnd= IndexVEnd + stepsize


  mytrain= OrigDsetML[IndexTrStart: IndexTrEnd,]

  mytrain_ = ScaleDset(mytrain)

  MyvDset = OrigDsetML[IndexVStart :IndexVEnd,]



  Results=
foreach(NHLayers=1:maxHlayers,.combine='rbind',.packages=c("foreach","xts","quantmod","Performan
ceAnalytics","TTR"))%dopar%{
```

```
   if (NHLayers==1){

     matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

       ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1), threshold = 0.005, linear.output = FALSE)

       TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=mytrain)

       ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=MyvDset)

       TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

       rownames(TRValR) = rownames(TrR)

       TRValR

     }

   } else if (NHLayers==2){

     matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

       foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

         ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2), threshold = 0.005, linear.output =
FALSE)

         TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,ClassifyProblem="True",vDset=mytra
in)

         ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,ClassifyProblem="True",vDset=MyvD
set)

         TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

         rownames(TRValR) = rownames(TrR)

         TRValR

       }

     }

   } else if (NHLayers==3){
```

```
    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

        foreach(NHunits3=NHunits,.combine='rbind')%dopar%{

          ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3), threshold = 0.005,
linear.output = FALSE)

          TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,ClassifyProblem="True",vD
set=mytrain)

          ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,ClassifyProblem="True",vD
set=MyvDset)

          TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

          rownames(TRValR) = rownames(TrR)

          TRValR

        }

      }

    }

  } else if (NHLayers==4){

    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

        foreach(NHunits3=NHunits,.combine='rbind')%dopar%{

          foreach(NHunits4=NHunits,.combine='rbind')%dopar%{

            ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3,NHunits4), threshold =
0.005, linear.output = FALSE)

            TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,ClassifyProblem=
"True",vDset=mytrain)

            ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,ClassifyProblem=
"True",vDset=MyvDset)
```

```
        TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

        rownames(TRValR) = rownames(TrR)

        TRValR

        }

      }

     }

    }

  } else if (NHLayers==5){

    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

       foreach(NHunits3=NHunits,.combine='rbind')%dopar%{

         foreach(NHunits4=NHunits,.combine='rbind')%dopar%{

          foreach(NHunits5=NHunits,.combine='rbind')%dopar%{

            ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3,NHunits4,NHunits5),
threshold = 0.005, linear.output = FALSE)

            TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,NHunits5,Classif
yProblem="True",vDset=mytrain)

            ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,NHunits5,Classif
yProblem="True",vDset=MyvDset)

            TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

            rownames(TRValR) = rownames(TrR)

            TRValR

            }

         }

       }

      }

    }
```

```
   }
 }
}
stopCluster(cl) # free resources


VTrAvgR =AllR



LenRDts= length(VTrAvgR)
#PredValidate <-
function(myInputs,ann_,myNHL,myNHuL1,myNHuL2=0,myNHuL3=0,myNHuL4=0,myNHuL5=0,ClassifyPr
oblem="True",vDset=MyvDset){
#Case 1 -Baseline using:
#       Input:  close &
#       Output: next-day close price


sumR = 0


for (i in 1:LenRDts){
  sumR= sumR +
    VTrAvgR[[i]]
}



VTrAvgR= sumR/LenRDts



colnames(VTrAvgR) =
c("AnnualizedReturn.Train","CummulativeReturn.Train","SharpeRatio.Train","Accuracy.Train","Precision
.Train","Recall.Train","AnnualizedReturn.Validation","CummulativeReturn.Validation","SharpeRatio.Vali
```

```
dation","Accuracy.Validation","Precision.Validation","Recall.Validation","NHiddenLayers","NHunitsLayer
1","NHunitsLayer2","NHunitsLayer3","NHunitsLayer4","NHunitsLayer5")



cn=
c("AnnualizedReturn.Train","CummulativeReturn.Train","SharpeRatio.Train","Accuracy.Train","Precision
.Train","Recall.Train","AnnualizedReturn.Validation","CummulativeReturn.Validation","SharpeRatio.Vali
dation","Accuracy.Validation","Precision.Validation","Recall.Validation","NHiddenLayers","NHunitsLayer
1","NHunitsLayer2","NHunitsLayer3","NHunitsLayer4","NHunitsLayer5")

length(cn)

cn[1:12]



VTrAvgR[order(-VTrAvgR[,"Accuracy.Validation"]),]



#####################################################################

########################## RESULTS - TUNING ##############################

## Hybrid Case - Classification (Baseline)

##

## Inputs:  Close only

## Output:  Signal

##


#### Top Performing Technical Indicator ###



ResultsClassyBaseCase_CrossVal2_ANN= VTrAvgR
```

```
nr = nrow(ResultsClassyBaseCase_CrossVal2_ANN)

CaseN= rep(0,nr)

ResultsClassyBaseCase_CrossVal2_ANN =cbind(ResultsClassyBaseCase_CrossVal2_ANN,CaseN)



ResultsClassyBaseCase_CrossVal2_ANN[order(-
ResultsClassyBaseCase_CrossVal2_ANN[,"Accuracy.Validation"]),]



#> ResultsClassyBaseCase_CrossVal2_ANN[order(-
ResultsClassyBaseCase_CrossVal2_ANN[,"Accuracy.Validation"]),]

#                    AnnualizedReturn.Train CummulativeReturn.Train SharpeRatio.Train Accuracy.Train
Precision.Train Recall.Train AnnualizedReturn.Validation CummulativeReturn.Validation

#ANN.HL1[2,0,0,0,0]          9.355076003          17.23699772          4.860233330          52.614          52.615
49.373          23.57964676          10.57999744

#ANN.HL2[1,1,0,0,0]          9.428278379          17.38088865          4.908699601          52.659          52.617
49.778          23.57964676          10.57999744

#ANN.HL1[1,0,0,0,0]          9.764755249          18.02125732          5.196190263          52.768          52.742
48.736          23.08454849          10.62879684

#ANN.HL2[2,1,0,0,0]          9.421057215          17.36579817          4.900816407          52.569          52.558
49.989          22.83014968          10.53839768

#ANN.HL2[2,2,0,0,0]          9.556751231          17.62778751          5.046688449          52.681          52.632
49.793          22.45843733          10.40467763

#ANN.HL2[1,2,0,0,0]          9.850710683          18.18702495          5.326352511          52.836          52.730
49.422          23.98848679          10.95597005

#                    SharpeRatio.Validation Accuracy.Validation Precision.Validation Recall.Validation
NHiddenLayers NHunitsLayer1 NHunitsLayer2 NHunitsLayer3 NHunitsLayer4 NHunitsLayer5 CaseN

#ANN.HL1[2,0,0,0,0]          27.07946027          62.465          62.067          49.579          1          2
0          0          0          0    0

#ANN.HL2[1,1,0,0,0]          27.07946027          62.372          62.008          49.615          2          1
1          0          0          0    0
```

#ANN.HL1[1,0,0,0,0] 26.45393532 61.928 61.820 49.679 1 1
0 0 0 0 0

#ANN.HL2[2,1,0,0,0] 25.65588555 61.556 61.522 49.964 2 2
1 0 0 0 0

#ANN.HL2[2,2,0,0,0] 25.24071657 60.908 61.313 49.538 2 2
2 0 0 0 0

#ANN.HL2[1,2,0,0,0] 28.64784313 60.817 61.557 48.685 2 1
2 0 0 0 0

```
#########################################################################
#### ANN - CASE-2.1 Classy : Hybrid WITH Signals only from Top Tech Indicators
#                Output : Signal (not price)
## NO Buffer case
# Top based on Accuracy Acc:
# Only one instance being shown

colnames( train_)
myInputs = c(5,10) #Inputs : "CrossEMA50_200" & "Close"
f = Signal ~ CrossEMA50_200 + Close
threshold = 0.005



NHunits = 1:2
maxHlayers = 1



IndexTrStart = which(OrigDsetML$Date==TrainStartDt)
IndexTrEnd = which(OrigDsetML$Date==TrainEndDt)
```

```
IndexVStart = which(OrigDsetML$Date==ValidTestStartDt)

IndexVEnd = which(OrigDsetML$Date==ValidTestEndDt)


Nchunks =10

rangeIndex= IndexVEnd - IndexVStart +1

chunksize =  round((rangeIndex/Nchunks),0)

lastchunksize = chunksize -(chunksize*Nchunks - rangeIndex)




Rnum=c(rep(chunksize,(Nchunks-1)),lastchunksize)

LenR = length(Rnum)




set.seed(3)



no_cores <- detectCores() - 1

cl <- makeCluster(no_cores)

registerDoParallel(cl)



AllR <-
foreach(Cnt=1:LenR,.packages=c("foreach","xts","quantmod","PerformanceAnalytics","TTR"))%dopar%{

  if (Cnt==1){
```

```
   stepsize = 0
   IndexVEnd =IndexVStart+ Rnum[1] -1
  } else stepsize = Rnum[Cnt]
  IndexTrStart = IndexTrStart+ stepsize
  IndexTrEnd = IndexTrEnd + stepsize
  IndexVStart= IndexVStart + stepsize
  IndexVEnd= IndexVEnd + stepsize


  mytrain= OrigDsetML[IndexTrStart: IndexTrEnd,]
  mytrain_ = ScaleDset(mytrain)
  MyvDset = OrigDsetML[IndexVStart :IndexVEnd,]



  Results=
foreach(NHLayers=1:maxHlayers,.combine='rbind',.packages=c("foreach","xts","quantmod","Performan
ceAnalytics","TTR"))%dopar%{


   if (NHLayers==1){
     matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{
      ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1), threshold = 0.005, linear.output = FALSE)
      TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=mytrain)
      ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=MyvDset)
      TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)
      rownames(TRValR) = rownames(TrR)
      TRValR
     }
   } else if (NHLayers==2){
```

```
    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

       ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2), threshold = 0.005, linear.output =
FALSE)

       TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,ClassifyProblem="True",vDset=mytra
in)

       ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,ClassifyProblem="True",vDset=MyvD
set)

       TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

       rownames(TRValR) = rownames(TrR)

       TRValR

      }

    }

  } else if (NHLayers==3){

    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

       foreach(NHunits3=NHunits,.combine='rbind')%dopar%{

        ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3), threshold = 0.005,
linear.output = FALSE)

        TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,ClassifyProblem="True",vD
set=mytrain)

        ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,ClassifyProblem="True",vD
set=MyvDset)

        TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

        rownames(TRValR) = rownames(TrR)

        TRValR
```

```
        }
       }
      }
    } else if (NHLayers==4){
      matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{
       foreach(NHunits2=NHunits,.combine='rbind')%dopar%{
        foreach(NHunits3=NHunits,.combine='rbind')%dopar%{
          foreach(NHunits4=NHunits,.combine='rbind')%dopar%{
            ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3,NHunits4), threshold =
0.005, linear.output = FALSE)
            TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,ClassifyProblem=
"True",vDset=mytrain)
            ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,ClassifyProblem=
"True",vDset=MyvDset)
            TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)
            rownames(TRValR) = rownames(TrR)
            TRValR
          }
        }
       }
      }
    } else if (NHLayers==5){
      matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{
       foreach(NHunits2=NHunits,.combine='rbind')%dopar%{
        foreach(NHunits3=NHunits,.combine='rbind')%dopar%{
          foreach(NHunits4=NHunits,.combine='rbind')%dopar%{
```

```
          foreach(NHunits5=NHunits,.combine='rbind')%dopar%{

            ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3,NHunits4,NHunits5),
threshold = 0.005, linear.output = FALSE)

            TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,NHunits5,Classif
yProblem="True",vDset=mytrain)

            ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,NHunits5,Classif
yProblem="True",vDset=MyvDset)

            TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

            rownames(TRValR) = rownames(TrR)

            TRValR

          }

         }

        }

      }

    }

  }

 }

}

stopCluster(cl) # free resources


VTrAvgR =AllR



LenRDts= length(VTrAvgR)

#PredValidate <-
function(myInputs,ann_,myNHL,myNHuL1,myNHuL2=0,myNHuL3=0,myNHuL4=0,myNHuL5=0,ClassifyPr
oblem="True",vDset=MyvDset){

#Case 1 -Baseline using:

#       Input:  close &
```

```
#       Output: next-day close price


sumR = 0


for (i in 1:LenRDts){
  sumR= sumR +
    VTrAvgR[[i]]
}



VTrAvgR= sumR/LenRDts



colnames(VTrAvgR) =
c("AnnualizedReturn.Train","CummulativeReturn.Train","SharpeRatio.Train","Accuracy.Train","Precision
.Train","Recall.Train","AnnualizedReturn.Validation","CummulativeReturn.Validation","SharpeRatio.Vali
dation","Accuracy.Validation","Precision.Validation","Recall.Validation","NHiddenLayers","NHunitsLayer
1","NHunitsLayer2","NHunitsLayer3","NHunitsLayer4","NHunitsLayer5")




VTrAvgR[order(-VTrAvgR[,"Accuracy.Validation"]),]



#************
ResultsClassy2.1_CrossVal2_ANN = VTrAvgR
```

```
ResultsClassy2.1_CrossVal2_ANN[which(ResultsClassy2.1_CrossVal2_ANN[,"Accuracy.Validation"]==max
(ResultsClassy2.1_CrossVal2_ANN[,"Accuracy.Validation"])),-c(2,3)]


Rnum=
which(ResultsClassy2.1_CrossVal2_ANN[,"Accuracy.Validation"]==max(ResultsClassy2.1_CrossVal2_ANN
[,"Accuracy.Validation"]))

length(Rnum) # total of 1 winner only

nrow(ResultsClassy2.1_CrossVal2_ANN)

ResultsClassy2.1_CrossVal2_ANN[Rnum[1:2],]




nr = nrow(ResultsClassy2.1_CrossVal2_ANN)

CaseN= rep(2.1,nr)

ResultsClassy2.1_CrossVal2_ANN=cbind(ResultsClassy2.1_CrossVal2_ANN,CaseN)

head(ResultsClassy2.1_CrossVal2_ANN)


ResultsClassy2.1_CrossVal2_ANN[order(-ResultsClassy2.1_CrossVal2_ANN[,"Accuracy.Validation"]),]




#> ResultsClassy2.1_CrossVal2_ANN[order(-ResultsClassy2.1_CrossVal2_ANN[,"Accuracy.Validation"]),]

#                AnnualizedReturn.Train CummulativeReturn.Train SharpeRatio.Train Accuracy.Train
Precision.Train Recall.Train AnnualizedReturn.Validation

#ANN.HL1[1,0,0,0,0]        14.77864987          27.83816082       9.740631096        55.178        54.709
41.487          26.33296646

#ANN.HL1[2,0,0,0,0]        14.94753761          28.12375299       9.856510808        55.222        54.465
44.080          24.16811578
```

#ANN.HL1[3,0,0,0,0] 15.52883089 29.28314160 10.825927322 55.758 55.164 41.143 26.54607716

CummulativeReturn.Validation SharpeRatio.Validation Accuracy.Validation Precision.Validation Recall.Validation NHiddenLayers NHunitsLayer1

#ANN.HL1[1,0,0,0,0] 10.65127001 31.83830350 63.929 63.314 48.026 1 1

#ANN.HL1[2,0,0,0,0] 10.78919788 28.22836224 62.836 62.335 49.364 1 2

#ANN.HL1[3,0,0,0,0] 10.48743910 31.73231215 62.836 63.124 46.306 1 3

NHunitsLayer2 NHunitsLayer3 NHunitsLayer4 NHunitsLayer5 CaseN

#ANN.HL1[1,0,0,0,0] 0 0 0 0 2.1

#ANN.HL1[2,0,0,0,0] 0 0 0 0 2.1

#ANN.HL1[3,0,0,0,0] 0 0 0 0 2.1

ANN - CASE-2.2 Classy : Hybrid WITH Signals only from Top Tech Indicators

Output : Signal (not price)

#

Buffer case

colnames(train_)

myInputs = c(2,10) #Inputs : "EMA50_200.MRt0.8.HDys30" & "Close"

f = Signal ~ EMA50_200.MRt0.8.HDys30 + Close

```
NHunits = 1:2
maxHlayers = 1



IndexTrStart = which(OrigDsetML$Date==TrainStartDt)
IndexTrEnd = which(OrigDsetML$Date==TrainEndDt)



IndexVStart = which(OrigDsetML$Date==ValidTestStartDt)
IndexVEnd = which(OrigDsetML$Date==ValidTestEndDt)

Nchunks =10
rangeIndex= IndexVEnd - IndexVStart +1
chunksize =  round((rangeIndex/Nchunks),0)
lastchunksize = chunksize -(chunksize*Nchunks - rangeIndex)



Rnum=c(rep(chunksize,(Nchunks-1)),lastchunksize)
LenR = length(Rnum)



set.seed(3)
```

```
no_cores <- detectCores() - 1

cl <- makeCluster(no_cores)

registerDoParallel(cl)



AllR <-
foreach(Cnt=1:LenR,.packages=c("foreach","xts","quantmod","PerformanceAnalytics","TTR"))%dopar%{

 if (Cnt==1){

  stepsize = 0

  IndexVEnd =IndexVStart+ Rnum[1] -1

 } else stepsize = Rnum[Cnt]

 IndexTrStart = IndexTrStart+ stepsize

 IndexTrEnd = IndexTrEnd + stepsize

 IndexVStart= IndexVStart + stepsize

 IndexVEnd= IndexVEnd + stepsize


 mytrain= OrigDsetML[IndexTrStart: IndexTrEnd,]

 mytrain_ = ScaleDset(mytrain)

 MyvDset = OrigDsetML[IndexVStart :IndexVEnd,]



 Results=
foreach(NHLayers=1:maxHlayers,.combine='rbind',.packages=c("foreach","xts","quantmod","Performan
ceAnalytics","TTR"))%dopar%{


  if (NHLayers==1){
```

```
    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1), threshold = 0.005, linear.output = FALSE)

      TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=mytrain)

      ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=MyvDset)

      TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

      rownames(TRValR) = rownames(TrR)

      TRValR

    }

  } else if (NHLayers==2){

    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

        ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2), threshold = 0.005, linear.output =
FALSE)

        TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,ClassifyProblem="True",vDset=mytra
in)

        ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,ClassifyProblem="True",vDset=MyvD
set)

        TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

        rownames(TRValR) = rownames(TrR)

        TRValR

      }

    }

  } else if (NHLayers==3){
```

```
    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

        foreach(NHunits3=NHunits,.combine='rbind')%dopar%{

          ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3), threshold = 0.005,
linear.output = FALSE)

          TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,ClassifyProblem="True",vD
set=mytrain)

          ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,ClassifyProblem="True",vD
set=MyvDset)

          TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)

          rownames(TRValR) = rownames(TrR)

          TRValR

        }

      }

    }

  } else if (NHLayers==4){

    matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{

      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{

        foreach(NHunits3=NHunits,.combine='rbind')%dopar%{

          foreach(NHunits4=NHunits,.combine='rbind')%dopar%{

            ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3,NHunits4), threshold =
0.005, linear.output = FALSE)

            TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,ClassifyProblem=
"True",vDset=mytrain)

            ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,ClassifyProblem=
"True",vDset=MyvDset)
```

```
          TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)
          rownames(TRValR) = rownames(TrR)
          TRValR
        }
       }
      }
    } else if (NHLayers==5){
     matx
=foreach(NHunits1=NHunits,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmo
d","PerformanceAnalytics","TTR"))%dopar%{
      foreach(NHunits2=NHunits,.combine='rbind')%dopar%{
       foreach(NHunits3=NHunits,.combine='rbind')%dopar%{
        foreach(NHunits4=NHunits,.combine='rbind')%dopar%{
         foreach(NHunits5=NHunits,.combine='rbind')%dopar%{
          ann_ <- neuralnet(f, mytrain_, hidden= c(NHunits1,NHunits2,NHunits3,NHunits4,NHunits5),
threshold = 0.005, linear.output = FALSE)
          TrR <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,NHunits5,Classif
yProblem="True",vDset=mytrain)
          ValR  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,NHunits2,NHunits3,NHunits4,NHunits5,Classif
yProblem="True",vDset=MyvDset)
          TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:6)]),nrow=1,byrow = TRUE)
          rownames(TRValR) = rownames(TrR)
          TRValR
         }
        }
       }
      }
     }
```

```
   }
 }
}
stopCluster(cl) # free resources



VTrAvgR =AllR



LenRDts= length(VTrAvgR)
#PredValidate <-
function(myInputs,ann_,myNHL,myNHuL1,myNHuL2=0,myNHuL3=0,myNHuL4=0,myNHuL5=0,ClassifyPr
oblem="True",vDset=MyvDset){
#Case 1 -Baseline using:
#       Input:  close &
#       Output: next-day close price


sumR = 0


for (i in 1:LenRDts){
  sumR= sumR +
    VTrAvgR[[i]]
}



VTrAvgR= sumR/LenRDts



colnames(VTrAvgR) =
c("AnnualizedReturn.Train","CummulativeReturn.Train","SharpeRatio.Train","Accuracy.Train","Precision
```

.Train","Recall.Train","AnnualizedReturn.Validation","CummulativeReturn.Validation","SharpeRatio.Validation","Accuracy.Validation","Precision.Validation","Recall.Validation","NHiddenLayers","NHunitsLayer1","NHunitsLayer2","NHunitsLayer3","NHunitsLayer4","NHunitsLayer5")

VTrAvgR[order(-VTrAvgR[,"Accuracy.Validation"]),]

ResultsClassy2.2_CrossVal2_ANN = VTrAvgR

########################## RESULTS - TUNING ##############################

ANN - CASE-2B : Hybrid WITH Signals only from Top Tech Indicators

Output : Signal (not price)

#

Buffer case, f = Signal ~ EMA50_200.MRt0.8.HDys30 + Close

Top based on Accuracy Acc:

Only one instance being shown

```
nr = nrow(ResultsClassy2.2_CrossVal2_ANN)

CaseN= rep(2.2,nr)

ResultsClassy2.2_CrossVal2_ANN =cbind(ResultsClassy2.2_CrossVal2_ANN,CaseN)


ResultsClassy2.2_CrossVal2_ANN[order(-ResultsClassy2.2_CrossVal2_ANN[,"Accuracy.Validation"]),]

#> ResultsClassy2.2_CrossVal2_ANN[order(-ResultsClassy2.2_CrossVal2_ANN[,"Accuracy.Validation"]),]

#AnnualizedReturn.Train CummulativeReturn.Train SharpeRatio.Train Accuracy.Train Precision.Train
Recall.Train AnnualizedReturn.Validation

#ANN.HL1[2,0,0,0,0]      14.64735071      27.52046288      9.923430963      55.916      55.297
40.748          27.30018429

#ANN.HL1[1,0,0,0,0]      13.82732600      25.93203292      9.191836968      55.758      55.395
39.771          27.13432022

#ANN.HL1[3,0,0,0,0]      13.38582648      25.03891152      8.973412171      55.446      55.033
40.533          23.83517931

#                CummulativeReturn.Validation SharpeRatio.Validation Accuracy.Validation
Precision.Validation Recall.Validation NHiddenLayers NHunitsLayer1

#ANN.HL1[2,0,0,0,0]      10.99851731      32.17937002      64.210      63.175
48.799          1       2

#ANN.HL1[1,0,0,0,0]      10.94486989      32.26550799      64.207      63.287
48.510          1       1

#ANN.HL1[3,0,0,0,0]      10.67199692      27.93087482      63.022      62.510
49.149          1       3

#NHunitsLayer2 NHunitsLayer3 NHunitsLayer4 NHunitsLayer5 CaseN

#ANN.HL1[2,0,0,0,0]      0       0       0       0  2.2

#ANN.HL1[1,0,0,0,0]      0       0       0       0  2.2

#ANN.HL1[3,0,0,0,0]      0       0       0       0  2.2
```

```
#############################################################################################



###install.packages("readxl")
#library(readxl)









#########################################################################
###  SVM - Machine Learning Algo. #2  ---ORIGINAL RCODE BY KISHA
####      Best: kernel="radial"  <---------
#########################################################

##install.packages('e1071')
library(e1071)
# For Buffer Case with inputs = closing prices  & EMA50_200.MRt0.8.HDys30
#                    & output = Signal




PredValidateSVM  <- function(tunedM,gm,c,ktype,caseN,respVar="Next.Close",testset){
  if (ktype =="linear"){
```

```
  ktype=1
 } else if (ktype =="radial"){
  ktype=2
 } else if (ktype =="polynomial"){
  ktype=3
 } else if (ktype =="sigmoid"){
  ktype=4
 }

 testset_= ScaleDset(testset)
 #svm_=baseM
 bestfit =tunedM
 #predResults1_SVM <- predict(svm_, Validtest_[,-1])
 predResults2_SVM <- predict(bestfit, testset_ [,-1])


 #PredNextDyPr_SVM1  <-
xts(UnScale(x=predResults1_SVM,xOrig=ValidOrigDset$Next.Close),order.by=ValidOrigDset$Date)

 if (respVar=="Next.Close"){ #Regression Problem
  # Unscaled Results, transforming back to the original form)
  PredNextDyPr_SVM2_tuned <-
xts(UnScale(x=predResults2_SVM,xOrig=testset[,respVar]),order.by=testset$Date)

  PosSVM <- GetPosML(SamedayPrice=close ,PredNextDyPr_SVM2_tuned,type="SVM")

  ######################################################
  # Get MSSE Metric
  a = na.omit(merge(testset$Next.Close,PredNextDyPr_SVM2_tuned,join="inner"))
```

```
  MSSE   = sum((a[,1]-a[,2])^2)/length(a[,1])
 } else  # For Classification Problem
 {

  MSSE =0
  predResults2_SVM = xts(predResults2_SVM,order.by=testset$Date)
  PredNextDyPr_SVM2_tuned = ifelse(round(predResults2_SVM,0)==1,1,-1)
  PosSVM=list(PredNextDyPr_SVM2_tuned,c(0),c(0))
 }
 index(PosSVM[[1]])= as.Date(index(PosSVM[[1]]))

 myposlist = list(PosSVM)
 myReturnsMetrics =
GetAllReturnsFrDates(myposlist,startdt=min(testset$Date),enddt=max(testset$Date))

 #Performance evaluation
 mySigMetrics=
SignalAccMetricsBasedOnDates(mylistSig=myposlist,close,startdt=min(testset$Date),enddt=max(testset
$Date))
 All = matrix(c((CombineMat(myReturnsMetrics,mySigMetrics)[,-
c(7,8)]),MSSE,gm,c,ktype,caseN),nrow=1,byrow=TRUE)
 return(All)
}

#####################################################################

#############################################################################

############### Trial Classification task- SVM    ###############################

#############################################################################
```

```
# Input  : close
# Output : Signal (Up/down)



colnames(train_)
f = Signal ~ Close

kerneltypes= c("linear","radial","polynomial","sigmoid")

myGamma= seq(0.1,1.0,0.1)

costls= seq(100,1000,50)



IndexTrStart = which(OrigDsetML$Date==TrainStartDt)
IndexTrEnd = which(OrigDsetML$Date==TrainEndDt)



IndexVStart = which(OrigDsetML$Date==ValidTestStartDt)
IndexVEnd = which(OrigDsetML$Date==ValidTestEndDt)

Nchunks =10
rangeIndex= IndexVEnd - IndexVStart +1
chunksize =  round((rangeIndex/Nchunks),0)
lastchunksize = chunksize -(chunksize*Nchunks - rangeIndex)



Rnum=c(rep(chunksize,(Nchunks-1)),lastchunksize)
```

```
LenR = length(Rnum)

##------------------------------------------------------------
no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
registerDoParallel(cl)

AllR <-
foreach(Cnt=1:LenR,.packages=c("foreach","xts","quantmod","PerformanceAnalytics","TTR"))%dopar%{
  if (Cnt==1){
    stepsize = 0
    IndexVEnd =IndexVStart+ Rnum[1] -1
  } else stepsize = Rnum[Cnt]
  IndexTrStart = IndexTrStart+ stepsize
  IndexTrEnd = IndexTrEnd + stepsize
  IndexVStart= IndexVStart + stepsize
  IndexVEnd= IndexVEnd + stepsize

  mytrain= OrigDsetML[IndexTrStart: IndexTrEnd,]
  mytrain_ = ScaleDset(mytrain)
  MyvDset = OrigDsetML[IndexVStart :IndexVEnd,]
```

```
 All4Kernels =foreach(ktype =
kerneltypes,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmod","Performance
Analytics","TTR","e1071"))%do%{

  if (ktype=="radial"){

    All=foreach(gm=myGamma,.combine='rbind')%dopar%{

      foreach(c=costls,.combine='rbind')%dopar%{

      svm_ <- svm(f, mytrain_,kernel=ktype,cost=c,gamma=gm,method = 'C-classification')

      ValR =PredValidateSVM(svm_,gm,c,ktype,caseN=0,respVar="Signal",testset=MyvDset)

      TrR= PredValidateSVM(svm_,gm,c,ktype,caseN=0,respVar="Signal",testset=mytrain)

      TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:7)]),nrow=1,byrow = TRUE)

      rownames(TRValR) = rownames(TrR)

      TRValR

    }

  }


  } else

  {


    foreach(c=costls,.combine='rbind')%dopar%{

      svm_ <- svm(f, mytrain_,kernel=ktype,cost=c,method = 'C-classification')

      ValR =PredValidateSVM(svm_,gm=0,c,ktype,caseN=0,respVar="Signal",testset=MyvDset)

      TrR= PredValidateSVM(svm_,gm=0,c,ktype,caseN=0,respVar="Signal",testset=mytrain)

      TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:7)]),nrow=1,byrow = TRUE)

      rownames(TRValR) = rownames(TrR)

      TRValR

    }
```

```
   }


 }
}
stopCluster(cl)


#####################################################################




lenDts=length(AllR)
MasterAll4KClassy0_CrossVal2_SVM =0
for (i in 1:lenDts){

 MasterAll4KClassy0_CrossVal2_SVM  = MasterAll4KClassy0_CrossVal2_SVM  + AllR[[i]]
}


MasterAll4KClassy0_CrossVal2_SVM = MasterAll4KClassy0_CrossVal2_SVM/lenDts


colnames(MasterAll4KClassy0_CrossVal2_SVM
)=c(cn[1:12],c("gamma","cost","kernel_type:N","caseN"))


MasterAll4KClassy0_CrossVal2_SVM [order(-
MasterAll4KClassy0_CrossVal2_SVM[,"Accuracy.Validation"]),][1:3,]
```

```
#####################################################################
#####################################################################
# Case2.1 : Input: close & Tech Indicator
#         Output: Signal
#####################################################################
#####################################################################

colnames(train_)
f = Signal ~ CrossEMA50_200 + Close
kerneltypes= c("linear","radial","polynomial","sigmoid")

myGamma= seq(0.1,1.0,0.1)

costls= seq(100,1000,50)



IndexTrStart = which(OrigDsetML$Date==TrainStartDt)
IndexTrEnd = which(OrigDsetML$Date==TrainEndDt)



IndexVStart = which(OrigDsetML$Date==ValidTestStartDt)
IndexVEnd = which(OrigDsetML$Date==ValidTestEndDt)

Nchunks =10
rangeIndex= IndexVEnd - IndexVStart +1
chunksize =  round((rangeIndex/Nchunks),0)
lastchunksize = chunksize -(chunksize*Nchunks - rangeIndex)
```

```
Rnum=c(rep(chunksize,(Nchunks-1)),lastchunksize)
LenR = length(Rnum)




##-------------------------------------------------------
no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
registerDoParallel(cl)




AllR <-
foreach(Cnt=1:LenR,.packages=c("foreach","xts","quantmod","PerformanceAnalytics","TTR"))%dopar%{
  if (Cnt==1){
    stepsize = 0
    IndexVEnd =IndexVStart+ Rnum[1] -1
  } else stepsize = Rnum[Cnt]
  IndexTrStart = IndexTrStart+ stepsize
  IndexTrEnd = IndexTrEnd + stepsize
  IndexVStart= IndexVStart + stepsize
  IndexVEnd= IndexVEnd + stepsize
```

```
 mytrain= OrigDsetML[IndexTrStart: IndexTrEnd,]
 mytrain_ = ScaleDset(mytrain)
 MyvDset = OrigDsetML[IndexVStart :IndexVEnd,]




 All4Kernels =foreach(ktype =
kerneltypes,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmod","Performance
Analytics","TTR","e1071"))%do%{
   if (ktype=="radial"){
    All=foreach(gm=myGamma,.combine='rbind')%dopar%{
      foreach(c=costls,.combine='rbind')%dopar%{
        svm_ <- svm(f, mytrain_,kernel=ktype,cost=c,gamma=gm,method = 'C-classification')
        ValR =PredValidateSVM(svm_,gm,c,ktype,caseN=2.1,respVar="Signal",testset=MyvDset)
        TrR= PredValidateSVM(svm_,gm,c,ktype,caseN=2.1,respVar="Signal",testset=mytrain)
        TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:7)]),nrow=1,byrow = TRUE)
        rownames(TRValR) = rownames(TrR)
        TRValR
      }
    }

   } else
   {

    foreach(c=costls,.combine='rbind')%dopar%{
      svm_ <- svm(f, mytrain_,kernel=ktype,cost=c,method = 'C-classification')
      ValR =PredValidateSVM(svm_,gm=0,c,ktype,caseN=2.1,respVar="Signal",testset=MyvDset)
      TrR= PredValidateSVM(svm_,gm=0,c,ktype,caseN=2.1,respVar="Signal",testset=mytrain)
```

```
      TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:7)]),nrow=1,byrow = TRUE)
      rownames(TRValR) = rownames(TrR)
      TRValR
    }


  }


 }
}
stopCluster(cl)

#####################################################################
# Tuning results Case 2.1 : Signal ~ CrossSMA50_200 + Close


lenDts=length(AllR)
MasterAll4KClassy2.1_CrossVal_SVM =0
for (i in 1:lenDts){

 MasterAll4KClassy2.1_CrossVal_SVM  = MasterAll4KClassy2.1_CrossVal_SVM  + AllR[[i]]
}

MasterAll4KClassy2.1_CrossVal_SVM = MasterAll4KClassy2.1_CrossVal_SVM/lenDts
colnames(MasterAll4KClassy2.1_CrossVal_SVM
)=c(cn[1:12],c("gamma","cost","kernel_type:N","caseN"))
```

```
MasterAll4KClassy2.1_CrossVal_SVM  [order(-MasterAll4KClassy2.1_CrossVal_SVM
[,"Accuracy.Validation"]),][1:2,]

#########################################################################

# Case2.2 : Input: close & Tech Indicator (with Buffer)

#         Output: Signal

#########################################################################

#########################################################################

colnames(train_)

f = Signal ~ EMA50_200.MRt0.8.HDys30 + Close

kerneltypes= c("linear","radial","polynomial","sigmoid")

myGamma= seq(0.1,1.0,0.1)

costls= seq(100,1000,50)

myGamma= seq(0.1,0.1,0.1)

costls= seq(100,100,50)

IndexTrStart = which(OrigDsetML$Date==TrainStartDt)

IndexTrEnd = which(OrigDsetML$Date==TrainEndDt)

IndexVStart = which(OrigDsetML$Date==ValidTestStartDt)
```

```
IndexVEnd = which(OrigDsetML$Date==ValidTestEndDt)


Nchunks =10
rangeIndex= IndexVEnd - IndexVStart +1
chunksize =  round((rangeIndex/Nchunks),0)
lastchunksize = chunksize -(chunksize*Nchunks - rangeIndex)




Rnum=c(rep(chunksize,(Nchunks-1)),lastchunksize)
LenR = length(Rnum)




##------------------------------------------------------------
no_cores <- detectCores() - 1
cl <- makeCluster(no_cores)
registerDoParallel(cl)




AllR <-
foreach(Cnt=1:LenR,.packages=c("foreach","xts","quantmod","PerformanceAnalytics","TTR"))%dopar%{
  if (Cnt==1){
    stepsize = 0
```

```
  IndexVEnd =IndexVStart+ Rnum[1] -1
 } else stepsize = Rnum[Cnt]
 IndexTrStart = IndexTrStart+ stepsize
 IndexTrEnd = IndexTrEnd + stepsize
 IndexVStart= IndexVStart + stepsize
 IndexVEnd= IndexVEnd + stepsize

 mytrain= OrigDsetML[IndexTrStart: IndexTrEnd,]
 mytrain_ = ScaleDset(mytrain)
 MyvDset = OrigDsetML[IndexVStart :IndexVEnd,]
```

```
  All4Kernels =foreach(ktype =
kerneltypes,.combine='rbind',.packages=c("foreach","xts","neuralnet","xlsx","quantmod","Performance
Analytics","TTR","e1071"))%do%{
   if (ktype=="radial"){
     All=foreach(gm=myGamma,.combine='rbind')%dopar%{
       foreach(c=costls,.combine='rbind')%dopar%{
         svm_ <- svm(f, mytrain_,kernel=ktype,cost=c,gamma=gm,method = 'C-classification')
         ValR =PredValidateSVM(svm_,gm,c,ktype,caseN=2.2,respVar="Signal",testset=MyvDset)
         TrR= PredValidateSVM(svm_,gm,c,ktype,caseN=2.2,respVar="Signal",testset=mytrain)
         TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:7)]),nrow=1,byrow = TRUE)
         rownames(TRValR) = rownames(TrR)
         TRValR
       }
     }
```

```
   } else
   {

     foreach(c=costls,.combine='rbind')%dopar%{
       svm_ <- svm(f, mytrain_,kernel=ktype,cost=c,method = 'C-classification')
       ValR =PredValidateSVM(svm_,gm=0,c,ktype,caseN=2.2,respVar="Signal",testset=MyvDset)
       TrR= PredValidateSVM(svm_,gm=0,c,ktype,caseN=2.2,respVar="Signal",testset=mytrain)
       TRValR <- matrix(c(TrR[,1:6],ValR[,1:6],TrR[,-c(1:7)]),nrow=1,byrow = TRUE)
       rownames(TRValR) = rownames(TrR)
       TRValR
     }


   }


 }
}
stopCluster(cl)



lenDts=length(AllR)
MasterAll4KClassy2.2_CrossVal_SVM =0
for (i in 1:lenDts){

  MasterAll4KClassy2.2_CrossVal_SVM  = MasterAll4KClassy2.2_CrossVal_SVM  + AllR[[i]]
}
```

```
MasterAll4KClassy2.2_CrossVal_SVM = MasterAll4KClassy2.2_CrossVal_SVM/lenDts

colnames(MasterAll4KClassy2.2_CrossVal_SVM
)=c(cn[1:12],c("gamma","cost","kernel_type:N","caseN"))


MasterAll4KClassy2.2_CrossVal_SVM  [order(-MasterAll4KClassy2.2_CrossVal_SVM
[,"Accuracy.Validation"  ]),][1:2,]




#####################################################################

#####################################################################




#############################################################################################
############################

#############################################################################################
############################


#### Test Winner - ANN #######

NHunits1=1

NHLayers=1



myInputs = c(2,10) #Inputs : "EMA50_200.MRt0.8.HDys30" & "Close"

f = Signal ~ CrossEMA50_200 + Close

MyvDset =testOrigDset



ann_ <- neuralnet(f, train2_, hidden= c(2), threshold = 0.005, linear.output = FALSE)
```

```
ResultOntestsetANN  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=MyvDset)

ResultOntestsetANN



MyvDset = train2OrigDset

ResultOnNewtrainsetANN  <-
PredValidate(myInputs,ann_,myNHL=NHLayers,NHunits1,ClassifyProblem="True",vDset=MyvDset)


ResultOnNewtrainsetANN



#> ResultOntestsetANN

#                Annualized Return Cummulative Return Sharpe Ratio Signal.Accuracy Signal.Precision

#ANN.HL1[1,0,0,0,0]      8.433253374         3.5308162   3.29225675          56.48           56.25

#                        Signal.Recall NHiddenLayers NHunitsLayer1 NHunitsLayer2 NHunitsLayer3

#ANN.HL1[1,0,0,0,0]           45           1           1           0           0

#                    NHunitsLayer4 NHunitsLayer5

#ANN.HL1[1,0,0,0,0]           0           0


#> ResultOnNewtrainsetANN

#                    Annualized Return Cummulative Return Sharpe Ratio Signal.Accuracy Signal.Precision

#ANN.HL1[1,0,0,0,0]      16.15782972        39.1616862  11.58912013          56.29           55.8

#                    Signal.Recall NHiddenLayers NHunitsLayer1 NHunitsLayer2 NHunitsLayer3

#ANN.HL1[1,0,0,0,0]          44.7           1           1           0           0

#                   NHunitsLayer4 NHunitsLayer5

#ANN.HL1[1,0,0,0,0]           0           0

#>
```

```
#############################################################################################
############################

#############################################################################################
############################

#### Test Winner - SVM (Over-all Winner) #######




colnames( train2_)

myInputs = c(5,10) #Inputs : "CrossEMA50_200" & "Close"

f = Signal ~ CrossEMA50_200 + Close

## Configuration

gm = 0.6

c = 300

ktype = "radial"




 MyvDset = testOrigDset


 svm_ <- svm(f, train2_,kernel="radial",cost=c,Gamma=gm,method = 'C-classification')

 testResultFinal = PredValidateSVM(svm_,gm,c,ktype,caseN=2.2,respVar="Signal",testset=MyvDset)


 testResultFinal

 testResultFinal = testResultFinal[,c(1:6,8:9,11)]
```

```
names(testResultFinal)=c(c(colnames(ResultOntestsetANN )[1:6]), c("Gamma","cost","caseN"))

testResultFinal


MyvDset = train2OrigDset


testOntrainDsetResultFinal =
PredValidateSVM(svm_,gm,c,ktype,caseN=2.2,respVar="Signal",testset=MyvDset)


testOntrainDsetResultFinal

testOntrainDsetResultFinal= testOntrainDsetResultFinal[,c(1:6,8:9,11)]


names(testOntrainDsetResultFinal)=c(c(colnames(ResultOnNewtrainsetANN )[1:6]),
c("Gamma","cost","caseN"))



testOntrainDsetResultFinal


#> testResultFinal
#Annualized Return Cummulative Return      Sharpe Ratio    Signal.Accuracy   Signal.Precision
#8.653300839       3.620806380       3.427834739       55.560000000       56.040000000
#Signal.Recall            Gamma             cost            caseN
#42.860000000       0.100000000      100.000000000       2.200000000
#> testOntrainDsetResultFinal
#Annualized Return Cummulative Return      Sharpe Ratio
#17.24046209       42.03949996       12.04662394
#Signal.Accuracy   Signal.Precision      Signal.Recall
# 56.47000000       55.75000000       45.77000000
# Gamma             cost            caseN
```

```
 #0.60000000       300.00000000        2.20000000




#####################################################################

#############################################################

## Visualizations of Results

#############################################################



##install.packages('ggplot2')


library(ggplot2)

library(reshape2)

# Ref. Code : https://stats.stackexchange.com/questions/3842/how-to-create-a-barplot-diagram-where-bars-are-side-by-side-in-r

#           https://stackoverflow.com/questions/12018499/how-to-put-labels-over-geom-bar-for-each-bar-in-r-with-ggplot2

df = melt(data.frame(ANN=c(62.65, 63.18,65.14), SVM=c(64.88,65.07,64.88),

                Cases=c("a/Base Case", "b/Hybrid_No-Buffer","c/Hybrid_Buffer")),

        variable.name="models")



p1 <- ggplot(df , aes(Cases, value, fill=models)) +

  geom_bar(position="dodge",stat="identity") +

  coord_flip() +

  geom_text(aes(label=value), color="white", position=position_dodge(width=0.9), hjust=1.10) +

  ggtitle("Results of Experiments for Classification \n - Accuracy) % \n All Cases ") +
```

```
  scale_fill_manual(values=c("blue","coral"))
p2 <- p1 +    theme(axis.text=element_text(size=12),
                axis.title=element_text(size=16,face="bold"))
p2


## Summary Table




df2 <- data.frame(tests=c("a/OrigTrainTest", "b/ValidationTest", "c/NewTrainTest","d/Test"),
                Accuracy=c(55.51,65.07, 56.47,55.56))
head(df2)



# Basic barplot
p<-ggplot(data=df2, aes(x=tests, y=Accuracy)) +
  geom_bar(stat="identity",fill="coral")
p1 <-p +   geom_text(aes(label=Accuracy), hjust=-0.01) + coord_flip()
p2 <- p1 +  ggtitle("SVM for Classification on various test data sets\n -Accuracy %")
p3 <- p2 +    theme(axis.text=element_text(size=12),
                axis.title=element_text(size=13,face="bold"))
p3



###################
## Results compared to Buy/Hold Strategy
```

Winner is : EMA50_200 for assessment period : training period used fo ML

annualizedReturn cumulativeReturn sharpe Signal.Accuracy Signal.Precision

#Buy_Hold 11.656503 1.8015304 11.5438198 52.228 52.228

#EMA50_200.MRt0.HDys0 9.364486 1.4980451 5.8363260 52.228 41.783

#EMA100_200.MRt0.HDys0 9.870187 1.5951893 10.2857134 52.227 37.497

Signal.Recall Margin.Rate Hist.days

#Buy_Hold 50.000 0 0

#EMA50_200.MRt0.HDys0 35.909 0 0

#EMA100_200.MRt0.HDys0 33.265 0 0

Test results for Buy/Hold Strategy

myposlist=list(posBH)

startdt=min(testOrigDset$Date)

enddt=max(testOrigDset$Date)

myReturnsMetrics = GetAllReturnsFrDates(myposlist,startdt,enddt) # test period

mySigMetrics= SignalAccMetricsBasedOnDates(mylistSig=myposlist,close,startdt ,enddt)

AllSignNReturnsMetrics = CombineMat(myReturnsMetrics,mySigMetrics)

AllSignNReturnsMetrics

AllSignNReturnsMetrics - Buy/Hold Strategy

annualizedReturn cumulativeReturn sharpe Signal.Accuracy Signal.Precision Signal.Recall Margin.Rate Hist.days

#Buy_Hold 7.652995094 3.210879482 2.805037376 54.63 54.63 50 0 0

```
##########################################

## Champion MOdel vs. Buy/Hold for test set


df = data.frame(Cases=c("ChampionSVM","BuyHold"),

            Testresults=c(55.56,54.63))


# Change the width of bars

p = ggplot(data=df, aes(x=Cases, y=Testresults)) +

  geom_bar(stat="identity",  color="blue", fill=c("black","coral"), width=0.9) +

  coord_flip() +

  geom_text(aes(label=Testresults), size=5.5, color="white",position=position_dodge(width=0.2),
hjust=1.25) +

  ggtitle("Experiment Cases for Classification \n -Accuracy \n Champion Model Versus Buy/Hold
Strategy%") +

  theme(axis.text=element_text(size=12),

       axis.title=element_text(size=14,face="bold"))

p

#############################################################################################

## All Metrics for Winning Models after tuning

#############################################################################################
#




dfWinners = melt(data.frame(ANN=c(62.47,62.07,49.58,23.58,10.58,27.08), SVM=c(64.88,63.92,47.90,
           25.89,   8.31,     25.97),
```

```
                    Metrics=c("f/Accuracy", "e/Precision","d/Recall","c/Annualized \n
Return","b/Cummulative \n Return","a/Sharpe Ratio")),

              variable.name="models")


p1 <- ggplot(dfWinners , aes(Metrics, value, fill=models)) +

  geom_bar(position="dodge",stat="identity") +

  coord_flip() +

  geom_text(aes(label=value), color="white", position=position_dodge(width=0.9), hjust=1.10) +

  ggtitle("Results of Experiments for Classification \n -All Metrics (Main: Accuracy) % \n Baseline Model ")
+

  scale_fill_manual(values=c("blue","coral"))

p2 <- p1 +    theme(axis.text=element_text(size=12),

                axis.title=element_text(size=14,face="bold"))

p2


######## Hybrid Case2.1 - No Buffer


dfWinners2.1 = melt(data.frame(ANN=c(63.93,   63.31,    48.03,26.33,10.65,31.84), SVM=c(65.07,64.04,
           47.82,    25.86,    8.29,      25.93),

                      Metrics=c("f/Accuracy", "e/Precision","d/Recall","c/Annualized \n
Return","b/Cummulative \n Return","a/Sharpe Ratio")),

                variable.name="models")


p1 <- ggplot(dfWinners2.1 , aes(Metrics, value, fill=models)) +

  geom_bar(position="dodge",stat="identity") +

  coord_flip() +

  geom_text(aes(label=value), color="white", position=position_dodge(width=0.9), hjust=1.10) +

  ggtitle("Results of Experiments for Classification \n -All Metrics (Main: Accuracy) % \n Hybrid Case2.1-
'No-Buffer' Model") +

  scale_fill_manual(values=c("blue","coral"))
```

```
p2 <- p1 +    theme(axis.text=element_text(size=12),
                axis.title=element_text(size=14,face="bold"))
p2


## Summary Table


dfWinners2.1


######## Hybrid Case2.2 - Buffer


dfWinners2.2 = melt(data.frame(ANN=c(           64.21,   63.18,   48.80,   27.30,   11.00,   32.18), SVM=c(
          64.88,   63.91,   47.90,   25.89,   8.31,     25.97),
                      Metrics=c("f/Accuracy", "e/Precision","d/Recall","c/Annualized \n
Return","b/Cummulative \n Return","a/Sharpe Ratio")),
                variable.name="models")
p1 <- ggplot(dfWinners2.2 , aes(Metrics, value, fill=models)) +
  geom_bar(position="dodge",stat="identity") +
  coord_flip() +
  geom_text(aes(label=value), color="white", position=position_dodge(width=0.9), hjust=1.10) +
  ggtitle("Results of Experiments for Classification \n -All Metrics (Main: Accuracy) % \n Hybrid Case2.2-
'Buffer' Model") +
  scale_fill_manual(values=c("blue","coral"))
p2 <- p1 +    theme(axis.text=element_text(size=12),
                axis.title=element_text(size=14,face="bold"))
p2


## Summary Table


dfWinners2.2
```

##

###

#Results - Selection of Technical Indicators Metrics (%)

#Cases Technical Analysis Indicators Accuracy Precision Recall Annualized Return Cummulative Return Sharpe Ratio

#Buy/Hold strategy - 53.983 53.983 50.00 15.318 3.073 12.516

#No Buffer Case EMA50_200 & 53.983 43.983 38.75 12.408 2.50 10.993 # Winner no Buffer case

#EMA100_200 53.983 43.983 36.429 12.483 2.52 10.025

#Buffer Case EMA50_200.MRt0.8.HDys30 54.162 44.302 37.193 11.533 2.31 10.067 # Winner Buffer case

#EMA100_200.MRt0.3.HDys30 53.983 43.983 36.429 12.483 2.52 10.025

```
dfTech = melt(data.frame(Tech_NoBuffer=c(53.98,  43.98,  38.75,  12.41,  2.50,  10.99),
                         Tech_Buffer=c(  54.16,  44.30,  37.19,  11.53,  2.31,  10.07),
                         Buy_Hold=c(53.98,  53.98,  50.00,  15.32,  3.07,  12.52),
```

```
                    Metrics=c("f/Accuracy", "e/Precision","d/Recall","c/Annualized \n
Return","b/Cummulative \n Return","a/Sharpe Ratio")),

            variable.name="Tech_Indicators_Vs_Buy_Hold")


p1 <- ggplot(dfTech , aes(Metrics, value, fill=Tech_Indicators_Vs_Buy_Hold)) +

  geom_bar(position="dodge",stat="identity") +

  coord_flip() +

  scale_fill_manual(values=c("purple","red","black")) +

  geom_text(aes(label=value),color="white",position=position_dodge(width=0.9), hjust=1.00) +

  ggtitle("Results of Experiments -Technical Analysis \n -All Metrics (Main: Accuracy) % \n 'No Buffer &
Buffer Technical Indicators")

p2 <- p1 +    theme(axis.text=element_text(size=12),

                axis.title=element_text(size=14,face="bold"))

p2

## Summary Table


dfTech




#################################################################

## The END

#################################################################
```