Predicting Stock Market Returns

DIVYA NAIDU

Load Libraries

```
library(DMwR)
```

```
## Loading required package: lattice
```

```
## Loading required package: grid
```

```
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(tseries)
```

Import Data

```
GSPC <- as.xts(get.hist.quote("^GSPC",start="1970-01-02",
          quote=c("Open", "High", "Low", "Close","Volume","Adjusted")))
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

```
## time series ends   2018-11-23
```

```
head(GSPC)
```

```
##             Open  High   Low Close   Volume Adjusted
## 1970-01-02 92.06 93.54 91.79 93.00  8050000    93.00
## 1970-01-05 93.00 94.25 92.53 93.46 11490000    93.46
## 1970-01-06 93.46 93.81 92.13 92.82 11460000    92.82
## 1970-01-07 92.82 93.38 91.93 92.63 10010000    92.63
## 1970-01-08 92.63 93.47 91.99 92.68 10670000    92.68
## 1970-01-09 92.68 93.25 91.82 92.40  9380000    92.40
```

```
GSPC <- as.xts(get.hist.quote("^GSPC",
          start="1970-01-02",end='2009-09-15',
          quote=c("Open", "High", "Low", "Close","Volume","Adjusted")))
```

```
## time series ends   2009-09-14
```

```
#install.packages("quantmod")
library(quantmod)
```

```
## Loading required package: TTR
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
getSymbols('^GSPC')
```

```
## [1] "GSPC"
```

```
getSymbols('^GSPC',from='1970-01-01',to='2009-09-15')
```

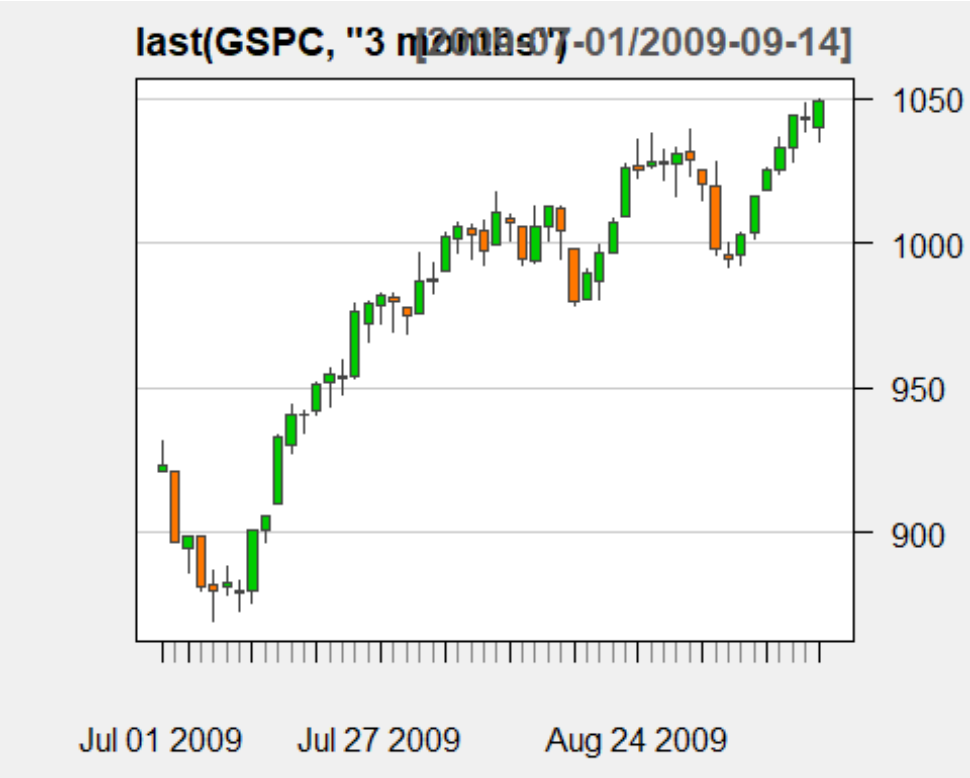
```
## [1] "GSPC"
```

```
colnames(GSPC) <- c("Open", "High", "Low", "Close","Volume","Adjusted")
```

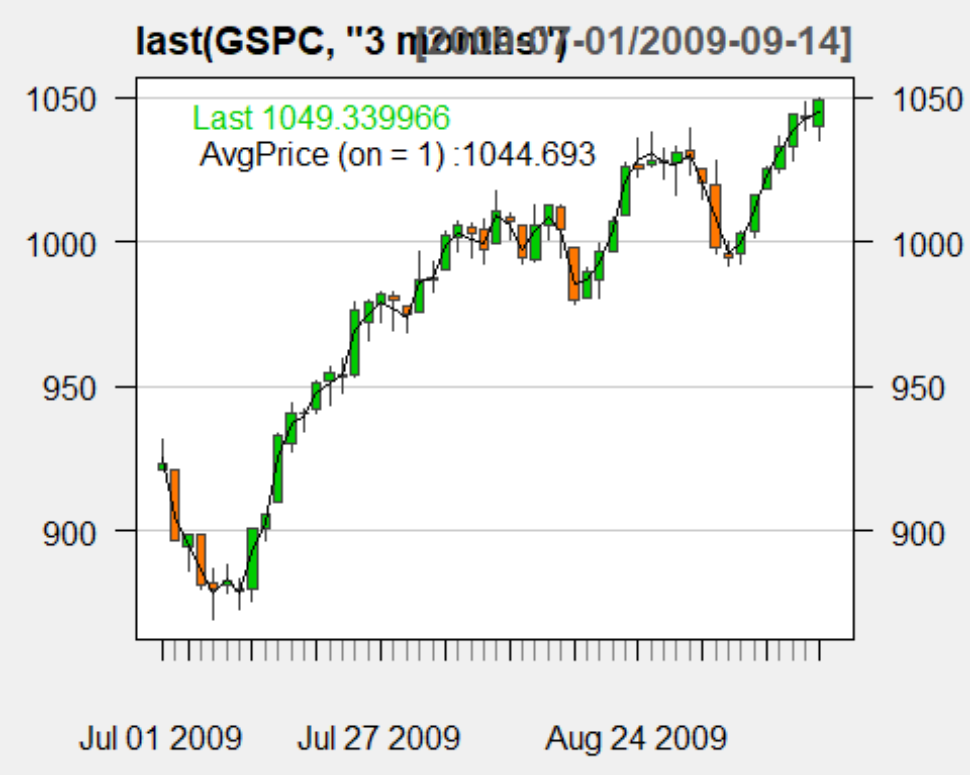
Defining the Predictive Tasks

```
# function to calculate the indicator value of the stock price
T.ind <- function(quotes,tgt.margin=0.025,n.days=10) {
  # returns a vector, with the mean stock price for every day.
  v <- apply(HLC(quotes),1,mean)

  r <- matrix(NA,ncol=n.days,nrow=NROW(quotes))
  # fills up the matrix,Next(),Delt() functions are of quantmod package
  for(x in 1:n.days) r[,x] <- Next(Delt(Cl(quotes),v,k=x),x)

  x <- apply(r,1,function(x) sum(x[x > tgt.margin | x < -tgt.margin]))
  if (is.xts(quotes)) xts(x,time(quotes)) else x
}

# draws a candle chart, the box indicates , the opening and closing values,
the tails indicate the day's highest
# and lowest values, orange indicates decrease from previous day closing,
green indicates reverse.
```

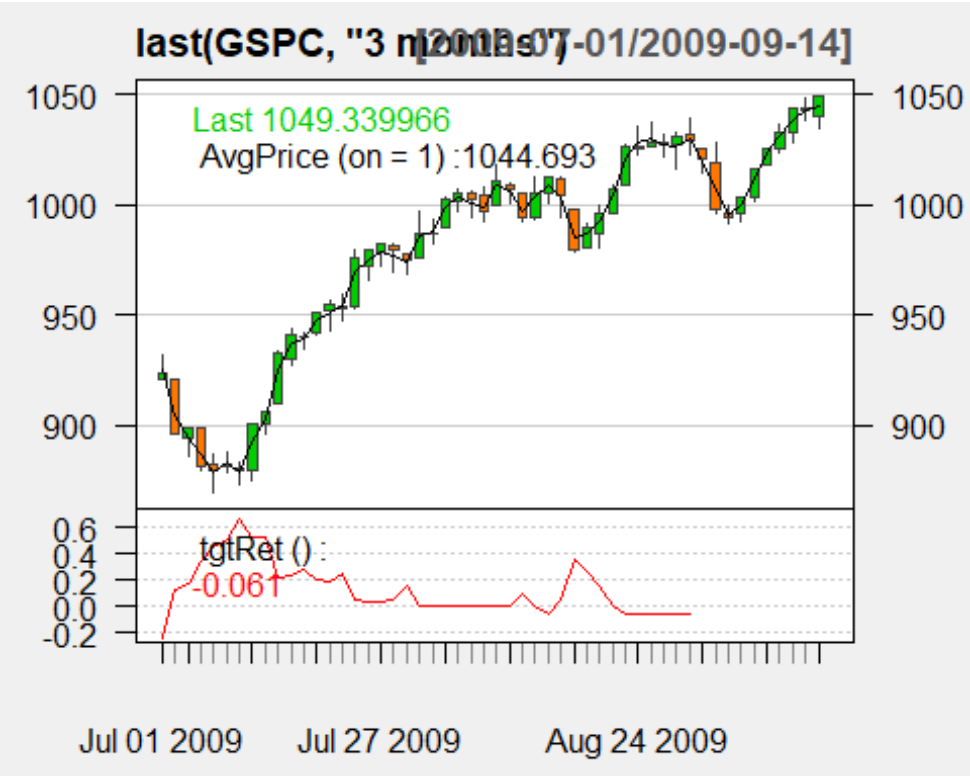
```
candleChart(last(GSPC,'3 months'),theme='white',TA=NULL)
```




```
# newTA is used to create plotting funtions for indicators, which are to be
included in the candle stick chart
avgPrice <- function(p) apply(HLC(p),1,mean)
addAvgPrice <- newTA(FUN=avgPrice,col=1,legend='AvgPrice')
# on value indiactes the same plot as the candle chart plot.
addT.ind <- newTA(FUN=T.ind,col='red',legend='tgtRet')
addAvgPrice(on=1)
```


last(GSPC, "3 months")[2009-07-01/2009-09-14]
Last 1049.339966
AvgPrice (on = 1) :1044.693
1050
1000
950
900
Jul 01 2009
Jul 27 2009
Aug 24 2009

```
addT.ind()
```

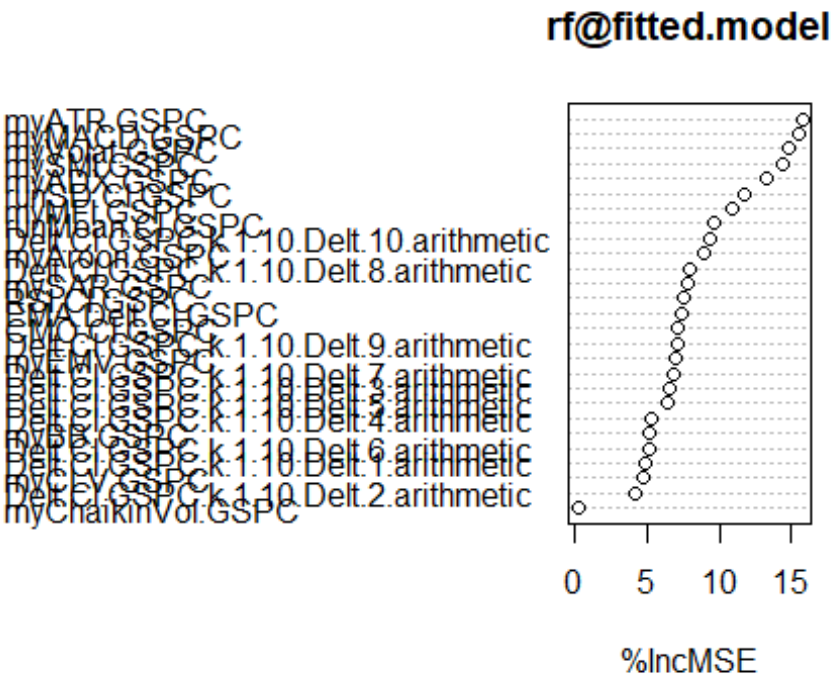

last(GSPC, "3 months")[2009-07-01/2009-09-14]
Last 1049.339966
AvgPrice (on = 1) :1044.693
tgtRet () :
-0.061
1050
1000
950
900
0.6
0.4
0.2
0.0
-0.2
Jul 01 2009
Jul 27 2009
Aug 24 2009

Choosing the Predictors

```
#various indicators from the TTR package :
myATR <- function(x) ATR(HLC(x))[,'atr']
mySMI <- function(x) SMI(HLC(x))[,'SMI']
myADX <- function(x) ADX(HLC(x))[,'ADX']
myAroon <- function(x) aroon(x[,c('High','Low')])$oscillator
myBB <- function(x) BBands(HLC(x))[,'pctB']
myChaikinVol <- function(x) Delt(chaikinVolatility(x[,c("High","Low")]))[,1]
myCLV <- function(x) EMA(CLV(HLC(x)))[,1]
myEMV <- function(x) EMV(x[,c('High','Low')],x[,'Volume'])[,2]
myMACD <- function(x) MACD(Cl(x))[,2]
myMFI <- function(x) MFI(x[,c("High","Low","Close")], x[,"Volume"])
mySAR <- function(x) SAR(x[,c('High','Close')]) [,1]
myVolat <- function(x) volatility(OHLC(x),calc="garman")[,1]
```

Random Forest

```
# elimination of unimportant variables from the initial set using random
forest

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

data.model <- specifyModel(T.ind(GSPC) ~ Delt(Cl(GSPC),k=1:10) +
       myATR(GSPC) + mySMI(GSPC) + myADX(GSPC) + myAroon(GSPC) +
       myBB(GSPC)  + myChaikinVol(GSPC) + myCLV(GSPC) +
       CMO(Cl(GSPC)) + EMA(Delt(Cl(GSPC))) + myEMV(GSPC) +
       myVolat(GSPC)  + myMACD(GSPC) + myMFI(GSPC) + RSI(Cl(GSPC)) +
       mySAR(GSPC) + runMean(Cl(GSPC)) + runSD(Cl(GSPC)))
set.seed(1234)
rf <- buildModel(data.model,method='randomForest',
             training.per=c(start(GSPC),index(GSPC["1999-12-31"])),
             ntree=50, importance=T)

varImpPlot(rf@fitted.model,type=1)
```




```
imp <- importance(rf@fitted.model,type=1)
rownames(imp)[which(imp > 10)]

## [1] "myATR.GSPC"    "mySMI.GSPC"    "myADX.GSPC"    "myVolat.GSPC"
## [5] "myMACD.GSPC"   "myMFI.GSPC"    "runSD.Cl.GSPC"

#new abstract model created with only select varibles identified by random
forest.

data.model <- specifyModel(T.ind(GSPC) ~ Delt(Cl(GSPC),k=1) + myATR(GSPC) +
                             myADX(GSPC) +    myEMV(GSPC) + myVolat(GSPC)  +
myMACD(GSPC) +
                             mySAR(GSPC) + runMean(Cl(GSPC)) )
```

The prediction tasks

```
Tdata.train <- as.data.frame(modelData(data.model,
                       data.window=c('1970-01-02','1999-12-31')))
Tdata.eval <- na.omit(as.data.frame(modelData(data.model,
                       data.window=c('2000-01-01','2009-09-15'))))
Tform <- as.formula('T.ind.GSPC ~ .')
```

Artificial Neural Network

```
set.seed(1234)
library(nnet)
```

```r
# Obtaining the neural network model
norm.data <- scale(Tdata.train)
nn <-
nnet(Tform,norm.data[1:1000,],size=10,decay=0.01,maxit=1000,linout=T,trace=F)
# obtaining prediction from the ANN model
norm.preds <- predict(nn,norm.data[1001:2000,])
preds <- unscale(norm.preds,norm.data)

# Transforming numeric values in to trading signals
sigs.nn <- trading.signals(preds,0.1,-0.1)
true.sigs <- trading.signals(Tdata.train[1001:2000,'T.ind.GSPC'],0.1,-0.1)
sigs.PR(sigs.nn,true.sigs)
```

```
##     precision    recall
## s   0.1950549 0.3944444
## b   0.3070866 0.2452830
## s+b 0.2240326 0.3244838
```

```r
# ANN for classification problems
set.seed(1234)
library(nnet)
signals <- trading.signals(Tdata.train[,'T.ind.GSPC'],0.1,-0.1)
norm.data <- data.frame(signals=signals,scale(Tdata.train[,-1]))
nn <- nnet(signals ~
.,norm.data[1:1000,],size=10,decay=0.01,maxit=1000,trace=F)
preds <- predict(nn,norm.data[1001:2000,],type='class')

sigs.PR(preds,norm.data[1001:2000,1])
```

```
##     precision    recall
## s   0.2794118 0.2111111
## b   0.3016760 0.3396226
## s+b 0.2952381 0.2743363
```

Support Vector Machine

```r
library(e1071)
sv <- svm(Tform,Tdata.train[1:1000,],gamma=0.001,cost=100)

s.preds <- predict(sv,Tdata.train[1001:2000,])
sigs.svm <- trading.signals(s.preds,0.1,-0.1)
true.sigs <- trading.signals(Tdata.train[1001:2000,'T.ind.GSPC'],0.1,-0.1)
sigs.PR(sigs.svm,true.sigs)
```

```
##     precision      recall
## s         0.3 0.016666667
## b         NaN 0.000000000
## s+b       0.3 0.008849558
```

```
#  Support Vector Machines for classfication task using kernlab package
library(kernlab)
data <- cbind(signals=signals,Tdata.train[,-1])
ksv <- ksvm(signals ~ .,data[1:1000,],C=10)
# predicting the values and finding out precision and recall
ks.preds <- predict(ksv,data[1001:2000,])
sigs.PR(ks.preds,data[1001:2000,1])
```

```
##     precision    recall
## s   0.1587983 0.2055556
## b   0.2808989 0.1572327
## s+b 0.1925466 0.1828909
```

Multivariate Adaptive Regression Splines[MARS]

```
library(earth)
```

```
## Loading required package: plotmo
```

```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

```
e <- earth(Tform,Tdata.train[1:1000,])
e.preds <- predict(e,Tdata.train[1001:2000,])
sigs.e <- trading.signals(e.preds,0.1,-0.1)
true.sigs <- trading.signals(Tdata.train[1001:2000,'T.ind.GSPC'],0.1,-0.1)
sigs.PR(sigs.e,true.sigs)
```

```
##     precision    recall
## s   0.2585034 0.2111111
## b   0.4098361 0.1572327
## s+b 0.3028846 0.1858407
```

Putting Everything Together: A Simulated Trader

From Predictions into Actions

```
policy.1 <- function(signals,market,opened.pos,money,
                     bet=0.2,hold.time=10,
                     exp.prof=0.025, max.loss= 0.05
                     )
  {
    d <- NROW(market) # this is the ID of today
    orders <- NULL
    nOs <- NROW(opened.pos)
    # nothing to do!
    if (!nOs && signals[d] == 'h') return(orders)

    # First lets check if we can open new positions
```

```
    # i) long positions
    if (signals[d] == 'b' && !nOs) {
      quant <- round(bet*money/market[d,'Close'],0)
      if (quant > 0)
        orders <- rbind(orders,
              data.frame(order=c(1,-1,-1),order.type=c(1,2,3),
                         val = c(quant,
                                 market[d,'Close']*(1+exp.prof),
                                 market[d,'Close']*(1-max.loss)
                                ),
                         action = c('open','close','close'),
                         posID = c(NA,NA,NA)
                        )
                       )

    # ii) short positions
    } else if (signals[d] == 's' && !nOs) {
      # this is the nr of stocks we already need to buy
      # because of currently opened short positions
      need2buy <- sum(opened.pos[opened.pos[,'pos.type']==-1,
                                 "N.stocks"])*market[d,'Close']
      quant <- round(bet*(money-need2buy)/market[d,'Close'],0)
      if (quant > 0)
        orders <- rbind(orders,
              data.frame(order=c(-1,1,1),order.type=c(1,2,3),
                         val = c(quant,
                                 market[d,'Close']*(1-exp.prof),
                                 market[d,'Close']*(1+max.loss)
                                ),
                         action = c('open','close','close'),
                         posID = c(NA,NA,NA)
                        )
                       )
    }

    # Now lets check if we need to close positions
    # because their holding time is over
    if (nOs)
      for(i in 1:nOs) {
        if (d - opened.pos[i,'Odate'] >= hold.time)
          orders <- rbind(orders,
                data.frame(order=-opened.pos[i,'pos.type'],
                           order.type=1,
                           val = NA,
                           action = 'close',
                           posID = rownames(opened.pos)[i]
                          )
                         )
      }
```

```
    orders
  }

policy.2 <- function(signals,market,opened.pos,money,
                     bet=0.2,exp.prof=0.025, max.loss= 0.05
                    )
  {
    d <- NROW(market) # this is the ID of today
    orders <- NULL
    nOs <- NROW(opened.pos)
    # nothing to do!
    if (!nOs && signals[d] == 'h') return(orders)

    # First lets check if we can open new positions
    # i) long positions
    if (signals[d] == 'b') {
      quant <- round(bet*money/market[d,'Close'],0)
      if (quant > 0)
        orders <- rbind(orders,
              data.frame(order=c(1,-1,-1),order.type=c(1,2,3),
                         val = c(quant,
                                 market[d,'Close']*(1+exp.prof),
                                 market[d,'Close']*(1-max.loss)
                                ),
                         action = c('open','close','close'),
                         posID = c(NA,NA,NA)
                        )
                       )

    # ii) short positions
    } else if (signals[d] == 's') {
      # this is the money already committed to buy stocks
      # because of currently opened short positions
      need2buy <- sum(opened.pos[opened.pos[,'pos.type']==-1,
                                 "N.stocks"])*market[d,'Close']
      quant <- round(bet*(money-need2buy)/market[d,'Close'],0)
      if (quant > 0)
        orders <- rbind(orders,
              data.frame(order=c(-1,1,1),order.type=c(1,2,3),
                         val = c(quant,
                                 market[d,'Close']*(1-exp.prof),
                                 market[d,'Close']*(1+max.loss)
                                ),
                         action = c('open','close','close'),
                         posID = c(NA,NA,NA)
                        )
                       )
    }
```

```
    orders
  }

# Train and test periods
start <- 1
len.tr <- 1000
len.ts <- 500
tr <- start:(start+len.tr-1)
ts <- (start+len.tr):(start+len.tr+len.ts-1)

# getting the quotes for the testing period
data(GSPC)
date <- rownames(Tdata.train[start+len.tr,])
market <- GSPC[paste(date,'/',sep='')][1:len.ts]

# learning the model and obtaining its signal predictions
library(e1071)
s <- svm(Tform,Tdata.train[tr,],cost=10,gamma=0.01)
p <- predict(s,Tdata.train[ts,])
sig <- trading.signals(p,0.1,-0.1)

# now using the simulated trader
t1 <-
trading.simulator(market,sig,'policy.1',list(exp.prof=0.05,bet=0.2,hold.time=
30))
t1

##
## Object of class tradeRecord with slots:
##
##   trading: <xts object with a numeric  500 x 5  matrix>
##   positions: <numeric  7 x 7  matrix>
##   init.cap :  1e+06
##   trans.cost :  5
##   policy.func :  policy.1
##   policy.pars : <list with  3  elements>

summary(t1)

##
## == Summary of a Trading Simulation with  500  days ==
##
## Trading policy function :  policy.1
## Policy function parameters:
##   exp.prof  =  0.05
##   bet  =  0.2
##   hold.time  =  30
##
## Transaction costs :  5
## Initial Equity    :  1e+06
## Final Equity      :  989632   Return :  -1.04 %
```

```
## Number of trading positions:  7
##
## Use function "tradingEvaluation()" for further stats on this simulation.
```

```
#The function tradingEvaluation() is used to obtain a series of economic
indicators of the performance during this simulation period
tradingEvaluation(t1)
```

```
##     NTrades       NProf    PercProf          PL         Ret   RetOverBH
##        7.00        3.00       42.86    -10368.05       -1.04       -7.89
##       MaxDD SharpeRatio     AvgProf     AvgLoss       AvgPL     MaxProf
##    28056.51       -0.02        5.18       -4.88       -0.57        5.26
##     MaxLoss
##       -5.00
```

```
plot(t1,market,theme='white',name='SP500')
```


```
## Rentability =  -1.036805 %
```

```
t2 <- trading.simulator(market,sig,'policy.2',list(exp.prof=0.05,bet=0.3))
summary(t2)
```

```
##
## == Summary of a Trading Simulation with  500  days ==
##
## Trading policy function :  policy.2
## Policy function parameters:
##   exp.prof  =  0.05
```

```
##   bet  =  0.3
##
## Transaction costs :  5
## Initial Equity    :  1e+06
## Final Equity      :  921129.9   Return :  -7.89 %
## Number of trading positions:  17
##
## Use function "tradingEvaluation()" for further stats on this simulation.
```

```
tradingEvaluation(t2)
```

```
##     NTrades       NProf    PercProf          PL         Ret   RetOverBH
##       17.00        6.00       35.29   -78870.08       -7.89      -14.74
##       MaxDD SharpeRatio     AvgProf     AvgLoss       AvgPL     MaxProf
##   121924.15       -0.04        5.12       -4.80       -1.30        5.26
##     MaxLoss
##       -5.00
```

```
#repeating the experiment with a different training and testing period
start <- 2000
len.tr <- 1000
len.ts <- 500
tr <- start:(start+len.tr-1)
ts <- (start+len.tr):(start+len.tr+len.ts-1)
s <- svm(Tform,Tdata.train[tr,],cost=10,gamma=0.01)
p <- predict(s,Tdata.train[ts,])
sig <- trading.signals(p,0.1,-0.1)
t2 <- trading.simulator(market,sig,'policy.2',list(exp.prof=0.05,bet=0.3))
summary(t2)
```

```
##
## == Summary of a Trading Simulation with  500  days ==
##
## Trading policy function :  policy.2
## Policy function parameters:
##   exp.prof  =  0.05
##   bet  =  0.3
##
## Transaction costs :  5
## Initial Equity    :  1e+06
## Final Equity      :  102011.3   Return :  -89.8 %
## Number of trading positions:  238
##
## Use function "tradingEvaluation()" for further stats on this simulation.
```

```
tradingEvaluation(t2)
```

```
##     NTrades       NProf    PercProf          PL         Ret   RetOverBH
##      238.00       70.00       29.41  -897988.74      -89.80      -96.65
##       MaxDD SharpeRatio     AvgProf     AvgLoss       AvgPL     MaxProf
##   905528.90       -0.08        5.26       -4.51       -1.64        5.26
```

```
##     MaxLoss
##       -5.90
```

Model Evaluation and Selection

Monte Carlo Estimates

```
MC.svmR <- function(form,train,test,b.t=0.1,s.t=-0.1,...) {
  require(e1071)
  t <- svm(form,train,...)
  p <- predict(t,test)
  trading.signals(p,b.t,s.t)
}
MC.svmC <- function(form,train,test,b.t=0.1,s.t=-0.1,...) {
  require(e1071)
  tgtName <- all.vars(form)[1]
  train[,tgtName] <- trading.signals(train[,tgtName],b.t,s.t)
  t <- svm(form,train,...)
  p <- predict(t,test)
  factor(p,levels=c('s','h','b'))
}
MC.nnetR <- function(form,train,test,b.t=0.1,s.t=-0.1,...) {
  require(nnet)
  t <- nnet(form,train,...)
  p <- predict(t,test)
  trading.signals(p,b.t,s.t)
}
MC.nnetC <- function(form,train,test,b.t=0.1,s.t=-0.1,...) {
  require(nnet)
  tgtName <- all.vars(form)[1]
  train[,tgtName] <- trading.signals(train[,tgtName],b.t,s.t)
  t <- nnet(form,train,...)
  p <- predict(t,test,type='class')
  factor(p,levels=c('s','h','b'))
}
MC.earth <- function(form,train,test,b.t=0.1,s.t=-0.1,...) {
  require(earth)
  t <- earth(form,train,...)
  p <- predict(t,test)
  trading.signals(p,b.t,s.t)
}
single <- function(form,train,test,learner,policy.func,...) {
  p <- do.call(paste('MC',learner,sep='.'),list(form,train,test,...))
  eval.stats(form,train,test,p,policy.func=policy.func)
}
slide <- function(form,train,test,learner,relearn.step,policy.func,...) {
  real.learner <- learner(paste('MC',learner,sep='.'),pars=list(...))
  p <- slidingWindowTest(real.learner,form,train,test,relearn.step)
  p <- factor(p,levels=1:3,labels=c('s','h','b'))
```

```
  eval.stats(form,train,test,p,policy.func=policy.func)
}
grow <- function(form,train,test,learner,relearn.step,policy.func,...) {
  real.learner <- learner(paste('MC',learner,sep='.'),pars=list(...))
  p <- growingWindowTest(real.learner,form,train,test,relearn.step)
  p <- factor(p,levels=1:3,labels=c('s','h','b'))
  eval.stats(form,train,test,p,policy.func=policy.func)
}

#The function eval.stats() uses two other functions to collect the precision
and recall of the signals, and several economic evaluation metrics
eval.stats <- function(form,train,test,preds,b.t=0.1,s.t=-0.1,...) {
  # Signals evaluation
  tgtName <- all.vars(form)[1]
  test[,tgtName] <- trading.signals(test[,tgtName],b.t,s.t)
  #Function sigs.PR() receives as arguments the predicted and true signals,
and calculates precision and recall for the sell, buy, and sell+buy signals
  st <- sigs.PR(preds,test[,tgtName])
  dim(st) <- NULL
  names(st) <- paste(rep(c('prec','rec'),each=3),
                     c('s','b','sb'),sep='.')

  # Trading evaluation
  #tradingEvaluation() obtains the economic metrics of a given trading record
  date <- rownames(test)[1]
  market <- GSPC[paste(date,"/",sep='')][1:length(preds),]
  trade.res <- trading.simulator(market,preds,...)

  c(st,tradingEvaluation(trade.res))
}

#The functions single(), slide(), and grow() are called from the Monte Carlo
routines with the proper parameters filled in so that we obtain the models we
want to compare
pol1 <- function(signals,market,op,money)
policy.1(signals,market,op,money,
           bet=0.2,exp.prof=0.025,max.loss=0.05,hold.time=10)

pol2 <- function(signals,market,op,money)
  policy.1(signals,market,op,money,
           bet=0.2,exp.prof=0.05,max.loss=0.05,hold.time=20)

pol3 <- function(signals,market,op,money)
  policy.2(signals,market,op,money,
           bet=0.5,exp.prof=0.05,max.loss=0.05)

# The list of learners we will use
TODO <- c('svmR','svmC','earth','nnetR','nnetC')

# The data sets used in the comparison
```

```
DSs <- list(dataset(Tform,Tdata.train,'SP500'))

# Monte Carlo (MC) settings used
MCsetts <- mcSettings(20,     # 20 repetitions of the MC exps
                      2540,   # ~ 10 years for training
                      1270,   # ~ 5 years for testing
                      1234)   # random number generator seed

# Variants to try for all learners
VARS <- list()
VARS$svmR   <- list(cost=c(10,150),gamma=c(0.01,0.001),
                    policy.func=c('pol1','pol2','pol3'))
VARS$svmC   <- list(cost=c(10,150),gamma=c(0.01,0.001),
                    policy.func=c('pol1','pol2','pol3'))
VARS$earth <- list(nk=c(10,17),degree=c(1,2),thresh=c(0.01,0.001),
                   policy.func=c('pol1','pol2','pol3'))
VARS$nnetR  <- list(linout=T,maxit=750,size=c(5,10),
                    decay=c(0.001,0.01),
                    policy.func=c('pol1','pol2','pol3'))
VARS$nnetC  <- list(maxit=750,size=c(5,10),decay=c(0.001,0.01),
                    policy.func=c('pol1','pol2','pol3'))

# main loop [takes ]
```

Result Analysis

```
#the results of all variants involving the five learning systems are the
below data files
load('svmR.Rdata')
load('svmC.Rdata')
load('earth.Rdata')
load('nnetR.Rdata')
load('nnetC.Rdata')

tgtStats <- c('prec.sb','Ret','PercProf',
              'MaxDD','SharpeRatio')
allSysRes <- join(subset(svmR,stats=tgtStats),
                  subset(svmC,stats=tgtStats),
                  subset(nnetR,stats=tgtStats),
                  subset(nnetC,stats=tgtStats),
                  subset(earth,stats=tgtStats),
                  by = 'variants')
rankSystems(allSysRes,5,maxs=c(T,T,T,F,T))

## $SP500
## $SP500$prec.sb
##           system score
## 1  slide.svmC.v5     1
## 2  slide.svmC.v6     1
## 3 slide.svmC.v13     1
```

```
## 4 slide.svmC.v14     1
## 5 slide.svmC.v21     1
##
## $SP500$Ret
##             system   score
## 1 single.nnetR.v12 97.4240
## 2  single.svmR.v11  3.4960
## 3  slide.nnetR.v15  2.6230
## 4  single.svmC.v12  0.7875
## 5   single.svmR.v8  0.6115
##
## $SP500$PercProf
##          system   score
## 1 grow.nnetR.v5 60.4160
## 2 grow.nnetR.v6 60.3640
## 3 slide.svmR.v3 60.3615
## 4  grow.svmR.v3 59.8710
## 5 grow.nnetC.v1 59.8615
##
## $SP500$MaxDD
##           system    score
## 1  slide.svmC.v5 197.3945
## 2  slide.svmC.v6 197.3945
## 3   grow.svmC.v5 197.3945
## 4   grow.svmC.v6 197.3945
## 5 slide.svmC.v13 399.2800
##
## $SP500$SharpeRatio
##           system score
## 1  slide.svmC.v5  0.02
## 2  slide.svmC.v6  0.02
## 3 slide.svmC.v13  0.02
## 4 slide.svmC.v14  0.02
## 5 slide.svmC.v21  0.02

#The function summary() can be applied to our loaded compExp objects
summary(subset(svmC,stats=c('Ret','RetOverBH','PercProf','NTrades'),
            vars=c('slide.svmC.v5','slide.svmC.v6')))

##
## == Summary of a  Monte Carlo  Experiment ==
##
##  20  repetitions Monte Carlo Simulation using:
##   seed =  1234
##   train size =  2540  cases
##   test size =  1270  cases
##
## * Data sets ::  SP500
## * Learners  ::  slide.svmC.v5, slide.svmC.v6
##
```

```
## * Summary of Experiment Results:
##
##
## -> Datataset:  SP500
##
##  *Learner: slide.svmC.v5
##                Ret  RetOverBH  PercProf   NTrades
## avg     0.0250000  -77.10350   5.00000 0.0500000
## std     0.1118034   33.12111  22.36068 0.2236068
## min     0.0000000 -128.01000   0.00000 0.0000000
## max     0.5000000  -33.77000 100.00000 1.0000000
## invalid 0.0000000    0.00000   0.00000 0.0000000
##
##  *Learner: slide.svmC.v6
##                Ret  RetOverBH  PercProf   NTrades
## avg     0.0250000  -77.10350   5.00000 0.0500000
## std     0.1118034   33.12111  22.36068 0.2236068
## min     0.0000000 -128.01000   0.00000 0.0000000
## max     0.5000000  -33.77000 100.00000 1.0000000
## invalid 0.0000000    0.00000   0.00000 0.0000000
```

```
fullResults <- join(svmR,svmC,earth,nnetC,nnetR,by='variants')
nt <- statScores(fullResults,'NTrades')[[1]]
rt <- statScores(fullResults,'Ret')[[1]]
pp <- statScores(fullResults,'PercProf')[[1]]
s1 <- names(nt)[which(nt > 20)]
s2 <- names(rt)[which(rt > 0.5)]
s3 <- names(pp)[which(pp > 40)]
namesBest <- intersect(intersect(s1,s2),s3)
summary(subset(fullResults,
               stats=tgtStats,
               vars=namesBest))
```

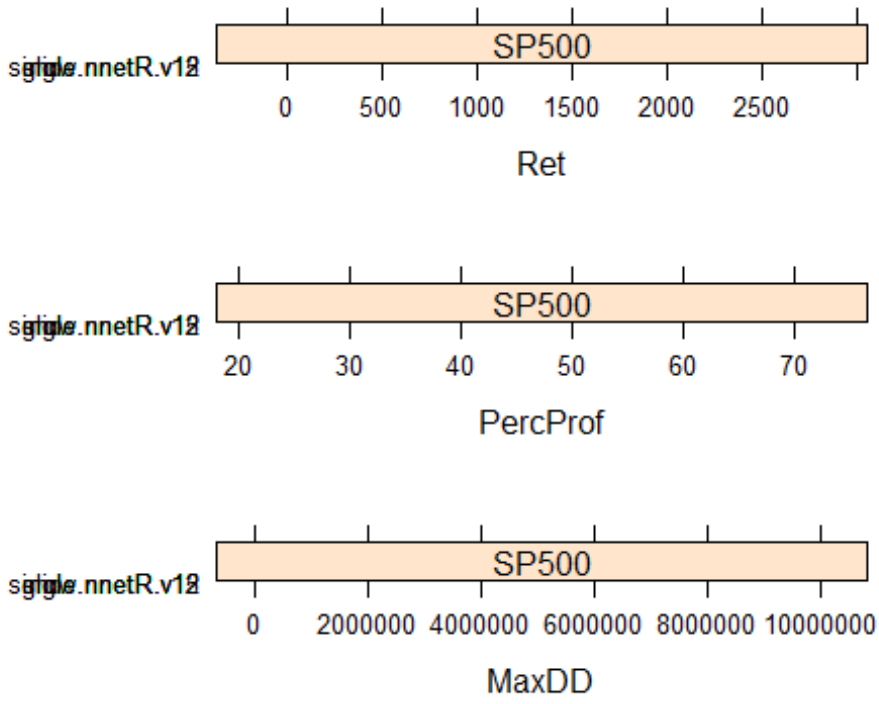
```
##
## == Summary of a  Monte Carlo  Experiment ==
##
##  20  repetitions Monte Carlo Simulation using:
##   seed =  1234
##   train size =  2540  cases
##   test size =  1270  cases
##
## * Data sets ::  SP500
## * Learners  ::  single.nnetR.v12, slide.nnetR.v15, grow.nnetR.v12
##
## * Summary of Experiment Results:
##
##
## -> Datataset:  SP500
##
##  *Learner: single.nnetR.v12
```

```
##            prec.sb       Ret PercProf      MaxDD SharpeRatio
## avg     0.12893147   97.4240  45.8860  1595761.4 -0.01300000
## std     0.06766129  650.8639  14.0488  2205913.7  0.03798892
## min     0.02580645 -160.4200  21.5000   257067.4 -0.08000000
## max     0.28695652 2849.8500  73.0800 10142084.7  0.04000000
## invalid 0.00000000    0.0000   0.0000        0.0  0.00000000
##
##  *Learner: slide.nnetR.v15
##            prec.sb      Ret  PercProf    MaxDD SharpeRatio
## avg     0.14028491  2.62300 54.360500 46786.28  0.01500000
## std     0.05111339  4.93178  8.339434 23526.07  0.03052178
## min     0.03030303 -7.03000 38.890000 18453.94 -0.04000000
## max     0.22047244  9.85000 68.970000 99458.44  0.05000000
## invalid 0.00000000  0.00000  0.000000     0.00  0.00000000
##
##  *Learner: grow.nnetR.v12
##            prec.sb        Ret PercProf     MaxDD SharpeRatio
## avg     0.18774920   0.544500 52.66200  41998.26  0.00600000
## std     0.07964205   4.334151 11.60824  28252.05  0.03408967
## min     0.04411765 -10.760000 22.22000  18144.11 -0.09000000
## max     0.33076923   5.330000 72.73000 121886.17  0.05000000
## invalid 0.00000000   0.000000  0.00000      0.00  0.00000000
```

```
#The following code carries out a statistical significance analysis of the
results using the function compAnalysis()
compAnalysis(subset(fullResults,
                    stats=tgtStats,
                    vars=namesBest))
```

```
## Warning in wilcox.test.default(comp@foldResults[, s, m, d],
## comp@foldResults[, : cannot compute exact p-value with ties

## Warning in wilcox.test.default(comp@foldResults[, s, m, d],
## comp@foldResults[, : cannot compute exact p-value with zeroes

## Warning in wilcox.test.default(comp@foldResults[, s, m, d],
## comp@foldResults[, : cannot compute exact p-value with ties

## Warning in wilcox.test.default(comp@foldResults[, s, m, d],
## comp@foldResults[, : cannot compute exact p-value with zeroes

##
## == Statistical Significance Analysis of Comparison Results ==
##
## Baseline Learner::    single.nnetR.v12  (Learn.1)
##
## ** Evaluation Metric::    prec.sb
##
## - Dataset: SP500
##        Learn.1    Learn.2 sig.2    Learn.3 sig.3
## AVG 0.12893147 0.14028491       0.18774920    +
```

```
## STD 0.06766129 0.05111339       0.07964205
##
## ** Evaluation Metric::    Ret
##
## - Dataset: SP500
##      Learn.1 Learn.2 sig.2  Learn.3 sig.3
## AVG  97.4240 2.62300    -  0.544500    -
## STD 650.8639 4.93178       4.334151
##
## ** Evaluation Metric::    PercProf
##
## - Dataset: SP500
##     Learn.1   Learn.2 sig.2  Learn.3 sig.3
## AVG 45.8860 54.360500    +  52.66200
## STD 14.0488  8.339434       11.60824
##
## ** Evaluation Metric::    MaxDD
##
## - Dataset: SP500
##     Learn.1  Learn.2 sig.2  Learn.3 sig.3
## AVG 1595761 46786.28    -- 41998.26    --
## STD 2205914 23526.07       28252.05
##
## ** Evaluation Metric::    SharpeRatio
##
## - Dataset: SP500
##         Learn.1    Learn.2 sig.2    Learn.3 sig.3
## AVG -0.01300000 0.01500000    +  0.00600000
## STD  0.03798892 0.03052178       0.03408967
##
## Legends:
## Learners -> Learn.1 = single.nnetR.v12 ; Learn.2 = slide.nnetR.v15 ;
Learn.3 = grow.nnetR.v12 ;
## Signif. Codes -> 0 '++' or '--' 0.001 '+' or '-' 0.05 ' ' 1

#We may have a better idea of the distribution of the scores on some of these
statistics across all 20 repetitions by plotting the compExp object
plot(subset(fullResults,
            stats=c('Ret','PercProf','MaxDD'),
            vars=namesBest))
```

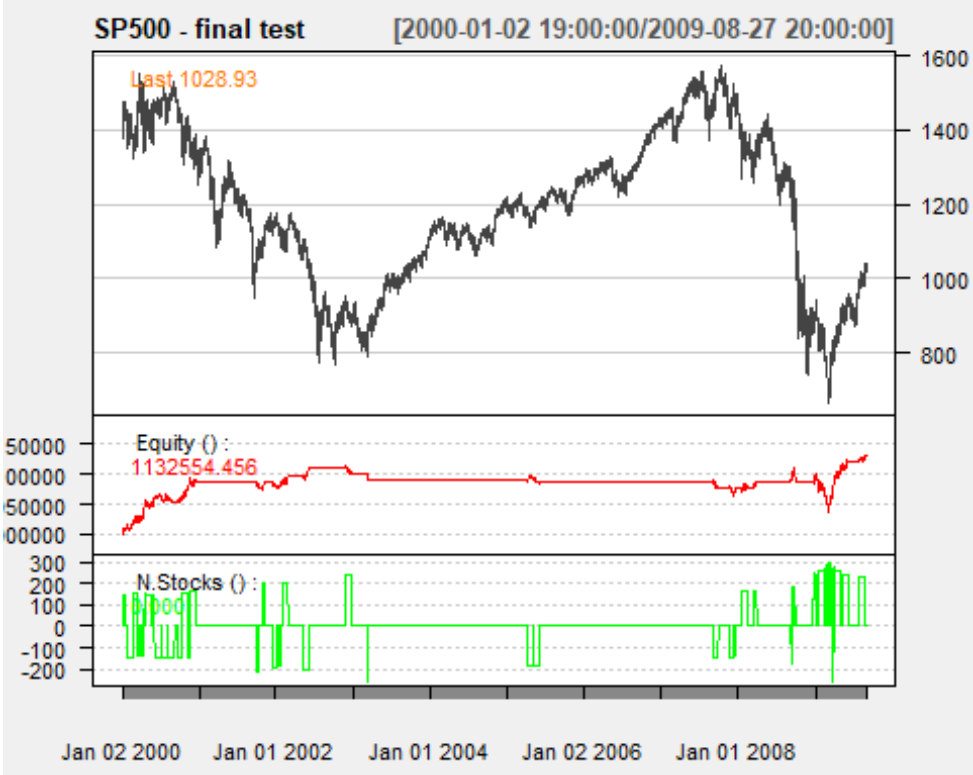



```
#we can check the configuration of this particular trading system using the
function getVariant()
getVariant('single.nnetR.v12',nnetR)

##
## Learner::  "single"
##
## Parameter values
##   learner  =  "nnetR"
##   linout  =  TRUE
##   trace  =  FALSE
##   maxit  =  750
##   size  =  10
##   decay  =  0.01
##   policy.func  =  "pol3"
```

The Trading System

```
#The following code obtains the evaluation statistics of these systems on the
9-year test period
data <- tail(Tdata.train,2540)
results <- list()
#for(name in namesBest) {
  #sys <- getVariant(name,fullResults)
  #results[[name]] <- runLearner(sys,Tform,data,Tdata.eval)
#}
```

```r
#results <- t(as.data.frame(results))

load('results.Rdata')
#We inspect the values of some of the main statistics
results[,c('Ret','RetOverBH','MaxDD','SharpeRatio','NTrades','PercProf')]

##                      Ret RetOverBH      MaxDD SharpeRatio NTrades PercProf
## single.nnetR.v12 -91.13    -61.26 1256121.55       -0.03     759    44.66
## slide.nnetR.v15   -6.16     23.71  107188.96       -0.01     132    48.48
## grow.nnetR.v12     1.47     31.34   84881.25        0.00      89    53.93

#The best model has the following characteristics
getVariant('grow.nnetR.v12',fullResults)

##
## Learner::  "grow"
##
## Parameter values
##   learner  =  "nnetR"
##   relearn.step  =  120
##   linout  =  TRUE
##   trace  =  FALSE
##   maxit  =  750
##   size  =  10
##   decay  =  0.001
##   policy.func  =  "pol2"

model <-
learner('MC.nnetR',list(maxit=750,linout=T,trace=F,size=10,decay=0.001))
preds <- growingWindowTest(model,Tform,data,Tdata.eval,relearn.step=120)

## ********************

signals <- factor(preds,levels=1:3,labels=c('s','h','b'))
date <- rownames(Tdata.eval)[1]
market <- GSPC[paste(date,"/",sep='')][1:length(signals),]
trade.res <- trading.simulator(market,signals,policy.func='pol2')

#plots the trading record of the system
plot(trade.res,market,theme='white',name='SP500 - final test')
```




```
## Rentability =  13.25545 %
```

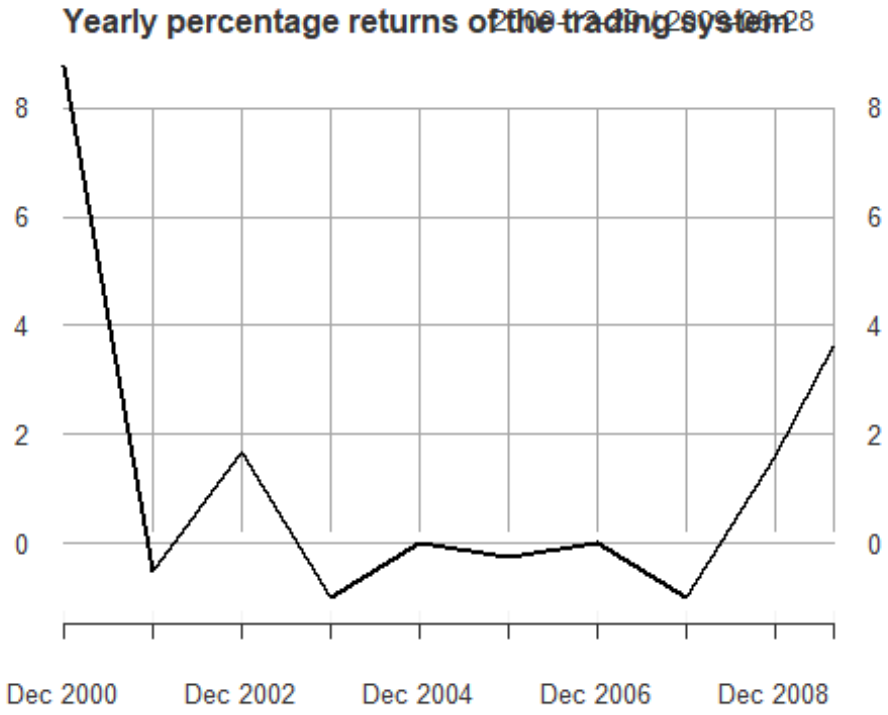
Performance Estimation

```
#Package PerformanceAnalytics provides an overwhelming set of tools for
analyzing the performance of any trading system
#install.packages("PerformanceAnalytics")
library(PerformanceAnalytics)

##
## Attaching package: 'PerformanceAnalytics'

## The following objects are masked from 'package:e1071':
##
##     kurtosis, skewness

## The following object is masked from 'package:graphics':
##
##     legend

rets <- Return.calculate(trade.res@trading$Equity)


chart.CumReturns(rets,main='Cumulative returns of the
strategy',ylab='returns')
```


```
yearlyReturn(as.xts(trade.res@trading$Equity))
```

```
##            yearly.returns
## 2000-12-29    0.087742766
## 2001-12-31   -0.005228831
## 2002-12-31    0.016706228
## 2003-12-31   -0.009993090
## 2004-12-31    0.000000000
## 2005-12-30   -0.002510516
## 2006-12-29    0.000000000
## 2007-12-31   -0.010017424
## 2008-12-31    0.016020630
## 2009-08-28    0.036424300
```

```
plot(100*yearlyReturn(as.xts(trade.res@trading$Equity)),
     main='Yearly percentage returns of the trading system')
abline(h=0,lty=2)
```




```
#table.CalendarReturns(rets)
#table.DownsideRisk(rets)

R <- as.xts(rets)
colnames(R) <- 'Equity'
table.DownsideRisk(R)

##                                 Equity
## Semi Deviation                  0.0010
## Gain Deviation                  0.0021
## Loss Deviation                  0.0020
## Downside Deviation (MAR=210%)   0.0084
## Downside Deviation (Rf=0%)      0.0010
## Downside Deviation (0%)         0.0010
## Maximum Drawdown                0.0702
## Historical VaR (95%)           -0.0019
## Historical ES (95%)            -0.0037
## Modified VaR (95%)             -0.0017
## Modified ES (95%)              -0.0017

table.CalendarReturns(R)

##       Jan  Feb  Mar  Apr  May  Jun  Jul Aug  Sep  Oct  Nov Dec Equity
## 2000 -0.2  0.2 -0.1  0.2  0.4 -0.2 -0.1   0  0.3 -0.1  0.0 0.0    0.3
## 2001  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0  0.0  0.5  0.0 0.2    0.7
## 2002  0.0  0.5  0.0  0.0  0.0  0.0  0.0   0  0.0  0.0 -0.1 0.0    0.4
## 2003  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0  0.0  0.0  0.0 0.0     NA
```

```
## 2004  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0  0.0  0.0  0.0 0.0     NA
## 2005  0.0  0.0  0.0 -0.2 -0.2  0.0  0.0   0  0.0  0.0  0.0 0.0   -0.4
## 2006  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0  0.0  0.0  0.0 0.0     NA
## 2007  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0 -0.3  0.0 -0.2 0.0   -0.4
## 2008  0.3  0.0  0.6  0.0  0.0  0.0  0.0   0  0.0  0.0  0.0 0.3    1.2
## 2009 -0.5 -0.5  0.3  0.1  0.5  0.0  0.0   0   NA   NA   NA  NA    0.0
```