Homework #4

Larry Lei Hua

Mar 17, 2019

In total there are 30 pts:

(1) (5 pts) Homework policy and presentation of the homework.
  - If the files are not complete or not zipped into one file (should be one zipped file with one pdf and one Rmd), deduct 2 pts
  - If the overall presentation is messy and not organized, deduct 3 pts
  - If the data is also submitted, deduct 1 pt

(2) Please check whether the following components are included:
  - (5 pts) data preparation
  - (4 pts) futures rollovers
  - (4 pts) randomforest models
  - (5 pts) grid search and parameter tuning
  - (3 pts) purged k-fold cross validation with embargo
  - (4 pts) summary of the performance and choices of candidate models
    - do not deduct pts if the test data was not used to evaluate the performance of the candidate model

---

(Please read the Homework Policy before you start)

Description: In this homework, you will gain firsthand experience of analyzing unstructured high frequency financial market data using (shallow) machine learning methods. Consider yourself as a quantitative analyst, and you are analyzing a futures dataset provided directly by a data vendor, which is AlgoSeek. Your goal is to provide models and features that are potentially useful for quantitative strategists / traders.

Dataset: Please download the tick data from Compass 2g, and you will be using E-mini SP500 data from the folders 201603 and 201604. Please note that the license of the data only allows you to do homework and projects for this course, and you should delete the data if you do not take the course.

Practices:

1. Preprocess the unstructured data so that you will have a labeled dataset of a sufficiently large sample size. You can label the price movement direction based on the volume weighted average prices (vwap) of each minute, and take advantage of the 10 levels limit order books to construct any potentially useful features. Some R functions from the fmlr package can be used, such as, fmlr::read_algoseek_futures_fullDepth(), fmlr::istar_CUSUM(), and fmlr::label_meta().
2. This is a relatively open assignment, and there are some necessary sub-tasks, including, but not limited to the following:
   - futures rollovers
   - randomforest models (sequential bootstrap is optional)
   - grid search and parameter tuning with purged k-fold cross validation with embargo

Your grade for this homework will be based on how much you follow the homework policy, the completeness of the practices, and whether relevant methods are appropriately applied.

```r
rm(list = setdiff(ls(), lsf.str()))
library(randomForestFML)
library(fmlr)
library(caret)
library(ROCR)

data_is_loaded <- TRUE
data_folder <- "~/datasets/algoseek"
data_folder_out <- file.path(data_folder, "out")
if(!file.exists(data_folder_out)) dir.create(data_folder_out)

data_range <- stringr::str_replace_all(
  as.character(as.Date(as.Date("2016-03-01"):as.Date("2016-03-31"),
                       origin = "1970-01-01")), "-", "")
pre_roll_date <- "20160310"
roll_date <- "20160311"

contracts <- rep("ES/ESH6", length(data_range))
```

```r
contracts[as.numeric(data_range) >= as.numeric(roll_date)] <- "ES/ESM6"

files <- paste0(data_range,".zip")

######################################
# prepare 1 minute data and features #
######################################

if(data_is_loaded == FALSE)
{
  sapply( (1:length(files)), function(i){

    tmp <- tryCatch(fmlr::read_algoseek_futures_fullDepth( file.path(data_folder, files[i]),
                                                            whichData = paste0(contracts[i], ".csv") ),
                    error=function(e) NA, warning=function(w) NA)

    if(!is.na(tmp)){
      tmp <- tmp[[1]]

      ########
      # SELL #
      ########
      data_sell <- subset(tmp, tmp$Side=="SELL")
      data_sell_smh <- aggregate(list(data_sell$p1*data_sell$v1,
                                      data_sell$p2*data_sell$v2,
                                      data_sell$p3*data_sell$v3,
                                      data_sell$p4*data_sell$v4,
                                      data_sell$p5*data_sell$v5,
                                      data_sell$p6*data_sell$v6,
                                      data_sell$p7*data_sell$v7,
                                      data_sell$p8*data_sell$v8,
                                      data_sell$p9*data_sell$v9,
                                      data_sell$p10*data_sell$v10,
                                      data_sell$v1, data_sell$v2, data_sell$v3, data_sell$v4,
                                      data_sell$v5, data_sell$v6, data_sell$v7, data_sell$v8,
                                      data_sell$v9, data_sell$v10),
                                 by=list(data_sell$m, data_sell$h), sum)
      names(data_sell_smh) <- c("m","h","p1","p2","p3","p4","p5","p6","p7","p8","p9","p10",
```

```
                          "v1","v2","v3","v4","v5","v6","v7","v8","v9","v10")
data_sell_smh$vwap1 <- data_sell_smh$p1 / data_sell_smh$v1
data_sell_smh$vwap2 <- data_sell_smh$p2 / data_sell_smh$v2
data_sell_smh$vwap3 <- data_sell_smh$p3 / data_sell_smh$v3
data_sell_smh$vwap4 <- data_sell_smh$p4 / data_sell_smh$v4
data_sell_smh$vwap5 <- data_sell_smh$p5 / data_sell_smh$v5
data_sell_smh$vwap6 <- data_sell_smh$p6 / data_sell_smh$v6
data_sell_smh$vwap7 <- data_sell_smh$p7 / data_sell_smh$v7
data_sell_smh$vwap8 <- data_sell_smh$p8 / data_sell_smh$v8
data_sell_smh$vwap9 <- data_sell_smh$p9 / data_sell_smh$v9
data_sell_smh$vwap10 <- data_sell_smh$p10 / data_sell_smh$v10

data_sell_smh <- data_sell_smh[, names(data_sell_smh)
                               %in% c("m","h", "vwap1","vwap2","vwap3","vwap4","vwap5",
                                      "vwap6","vwap7","vwap8","vwap9","vwap10")]


#######
# BUY #
#######
data_buy <- subset(tmp, Side=="BUY")
data_buy_smh <- aggregate(list(data_buy$p1*data_buy$v1,
                               data_buy$p2*data_buy$v2,
                               data_buy$p3*data_buy$v3,
                               data_buy$p4*data_buy$v4,
                               data_buy$p5*data_buy$v5,
                               data_buy$p6*data_buy$v6,
                               data_buy$p7*data_buy$v7,
                               data_buy$p8*data_buy$v8,
                               data_buy$p9*data_buy$v9,
                               data_buy$p10*data_buy$v10,
                               data_buy$v1, data_buy$v2, data_buy$v3, data_buy$v4,
                               data_buy$v5, data_buy$v6, data_buy$v7, data_buy$v8,
                               data_buy$v9, data_buy$v10),
                          by=list(data_buy$m, data_buy$h), sum)
names(data_buy_smh) <- c("m","h","p1","p2","p3","p4","p5","p6","p7","p8","p9","p10",
                         "v1","v2","v3","v4","v5","v6","v7","v8","v9","v10")
data_buy_smh$vwap1 <- data_buy_smh$p1 / data_buy_smh$v1
data_buy_smh$vwap2 <- data_buy_smh$p2 / data_buy_smh$v2
```

```r
data_buy_smh$vwap3 <- data_buy_smh$p3 / data_buy_smh$v3
data_buy_smh$vwap4 <- data_buy_smh$p4 / data_buy_smh$v4
data_buy_smh$vwap5 <- data_buy_smh$p5 / data_buy_smh$v5
data_buy_smh$vwap6 <- data_buy_smh$p6 / data_buy_smh$v6
data_buy_smh$vwap7 <- data_buy_smh$p7 / data_buy_smh$v7
data_buy_smh$vwap8 <- data_buy_smh$p8 / data_buy_smh$v8
data_buy_smh$vwap9 <- data_buy_smh$p9 / data_buy_smh$v9
data_buy_smh$vwap10 <- data_buy_smh$p10 / data_buy_smh$v10

data_buy_smh <- data_buy_smh[, names(data_buy_smh)
                             %in% c("m","h", "vwap1","vwap2","vwap3","vwap4","vwap5",
                                    "vwap6","vwap7","vwap8","vwap9","vwap10")]

####################
# combine BUY/SELL #
####################
data_buy_sell <- merge(data_buy_smh, data_sell_smh, by=c("h","m"), all=T,
                       suffixes = c("_buy","_sell"), sort=FALSE)

reorder <- c("h","m",
             "vwap10_buy","vwap9_buy","vwap8_buy","vwap7_buy","vwap6_buy",
             "vwap5_buy","vwap4_buy","vwap3_buy","vwap2_buy","vwap1_buy",
             "vwap1_sell","vwap2_sell","vwap3_sell","vwap4_sell","vwap5_sell",
             "vwap6_sell","vwap7_sell","vwap8_sell","vwap9_sell","vwap10_sell")

data_buy_sell <- data_buy_sell[, sapply(reorder, function(x){which(x==names(data_buy_sell))})]

# impute NA by the previous non-NA
data_buy_sell$vwap10_buy <- zoo::na.locf(data_buy_sell$vwap10_buy)
data_buy_sell$vwap9_buy <- zoo::na.locf(data_buy_sell$vwap9_buy)
data_buy_sell$vwap8_buy <- zoo::na.locf(data_buy_sell$vwap8_buy)
data_buy_sell$vwap7_buy <- zoo::na.locf(data_buy_sell$vwap7_buy)
data_buy_sell$vwap6_buy <- zoo::na.locf(data_buy_sell$vwap6_buy)
data_buy_sell$vwap5_buy <- zoo::na.locf(data_buy_sell$vwap5_buy)
data_buy_sell$vwap4_buy <- zoo::na.locf(data_buy_sell$vwap4_buy)
data_buy_sell$vwap3_buy <- zoo::na.locf(data_buy_sell$vwap3_buy)
data_buy_sell$vwap2_buy <- zoo::na.locf(data_buy_sell$vwap2_buy)
data_buy_sell$vwap1_buy <- zoo::na.locf(data_buy_sell$vwap1_buy)
```

```r
      data_buy_sell$vwap10_sell <- zoo::na.locf(data_buy_sell$vwap10_sell)
      data_buy_sell$vwap9_sell <- zoo::na.locf(data_buy_sell$vwap9_sell)
      data_buy_sell$vwap8_sell <- zoo::na.locf(data_buy_sell$vwap8_sell)
      data_buy_sell$vwap7_sell <- zoo::na.locf(data_buy_sell$vwap7_sell)
      data_buy_sell$vwap6_sell <- zoo::na.locf(data_buy_sell$vwap6_sell)
      data_buy_sell$vwap5_sell <- zoo::na.locf(data_buy_sell$vwap5_sell)
      data_buy_sell$vwap4_sell <- zoo::na.locf(data_buy_sell$vwap4_sell)
      data_buy_sell$vwap3_sell <- zoo::na.locf(data_buy_sell$vwap3_sell)
      data_buy_sell$vwap2_sell <- zoo::na.locf(data_buy_sell$vwap2_sell)
      data_buy_sell$vwap1_sell <- zoo::na.locf(data_buy_sell$vwap1_sell)

      rm(list=c("data_buy", "data_buy_smh", "data_sell", "data_sell_smh", "tmp")); gc()
      write.table(data_buy_sell, file = file.path(data_folder_out, paste0(data_range[i], "_M1.csv")), sep=",", row.names = F)
    }

  } )
}

####################
# Futures rollover #
####################

# load those data preprocessed from above
dat_roll <- read.csv( file.path( data_folder_out, paste0(roll_date, "_M1.csv")) )
dat_roll_pre <- read.csv( file.path( data_folder_out, paste0(pre_roll_date, "_M1.csv")))
nam <- names(dat_roll)

gap_roll <- (dat_roll$vwap1_buy[1] + dat_roll$vwap1_sell[1])/2 -
  (dat_roll_pre$vwap1_buy[nx <- nrow(dat_roll_pre)] + dat_roll_pre$vwap1_sell[nx])/2
files_out <- list.files(data_folder_out)
inx_add_gap <- as.numeric(stringr::str_sub(files_out, 1, 8)) < as.numeric(roll_date)

dat <- NULL
for(k in 1:length(files_out)){
  tmp <- read.csv( file.path(data_folder_out, files_out[k]) )
  if(inx_add_gap[k] == TRUE) tmp[,-c(1,2)] <- tmp[,-c(1,2)] + gap_roll
  dat <- rbind(dat, tmp)
}
```

```r
#############################################
# Label data using mid prices based on vwap #
#############################################

# use mid price to label the data
# you can also use some other reasonable prices
dat$mid_price <- (dat$vwap1_buy + dat$vwap1_sell)/2
ndat <- nrow(dat)

tt_split <- 2:1 # train/test split
idx_train <- (1:floor(tt_split[1]/sum(tt_split)*ndat))
idx_test <- setdiff( (1:ndat), idx_train)
train_data <- dat[idx_train,]
test_data <- dat[idx_test,]

###################################################
# vectors for two parameters to be tuned
hvec <- seq(0.2, 1, length=5)
trgtvec <- seq(0.001, 0.005, length=4)
k <- 5 # k-fold CV
gam <- 0.01 # embargo parameter
run <- FALSE # whether run the grid search?
###################################################
if(run==TRUE)
{
  rst <- NULL
  cat("ih", "itrgt", "iFold", "h", "trgt", "acc", "auc", "F1", "logloss",
      "train:-1", "train:1", "test:-1", "test:1", "\n")

  for(ih in 1:length(hvec))
  {
    for(jtrgt in 1:length(trgtvec))
    {
      ###################################################
      i_CUSUM <- fmlr::istar_CUSUM(train_data$mid_price, h=hvec[ih])  # <---------------- tuning parameter 1
      n_Event <- length(i_CUSUM)

      events <- data.frame(t0=i_CUSUM+1,
```

```r
                       t1 = i_CUSUM+300,  # <--- 300 can also be tuned
                       trgt = rep(trgtvec[jtrgt], n_Event), # <---------------- tuning parameter 2
                       side=rep(0,n_Event))
ptSl <- c(1,1)

out0 <- fmlr::label_meta(train_data$mid_price, events, ptSl, ex_vert = T)
tb <- table(out0$label)
if(length(tb) != 2) next

# feature matrix
fMat0 <- train_data[out0$t1Fea, !names(train_data)%in%c("m","h")]

# here first order difference is used, you can also consider using fracDiff for each
# feature and for each combination of the tuning parameters
fMat <- rbind( rep(NA, ncol(fMat0)),  apply(fMat0, 2,  function(x){diff(x)} ) )

# t1Fea and tLabel have to be included in order to use purged k-CV
allSet <- data.frame(Y=as.factor(out0$label), fMat, t1Fea=out0$t1Fea, tLabel=out0$tLabel)

# exclude NA at the beginning of the indicators
idx_NA <- apply(allSet,1,function(x){sum(is.na(x))>0 })
allSet <- subset(allSet, !idx_NA)
nx <- nrow(allSet)

#####################################
# prepare data for purged k-fold CV #
#####################################
CVobj <- fmlr::purged_k_CV(allSet, k=k, gam=gam)

##################
## randomforest ##
##################
# set.seed(1)
for(i in 1:k)
{
  trainSet <- CVobj[[i]]$trainSet
  trainSet <- trainSet[,!names(trainSet)%in%c("t1Fea", "tLabel")]
```

```
if(any(table(trainSet$Y)==0)) next

testSet <- CVobj[[i]]$testSet
testSet <- testSet[,!names(testSet)%in%c("t1Fea", "tLabel")]

# automatically choose mtry
# use sink to avoid displaying numbers on screen
sink("~/tmp"); mtry <- randomForestFML::tuneRF(trainSet[,-1], trainSet$Y, trace = FALSE, plot=FALSE); sink()
mtry <- mtry[which.min(mtry[,2]),1]

fit <- randomForestFML(Y ~ ., data = trainSet, mtry = mtry, importance = FALSE, ntrees = 500)

pre <- predict(fit, newdata = testSet)
acc <- mean(testSet$Y==pre)

# can also use R caret package to calculate F1 score
# predictions <- predict(fit, newdata=testSet)
precision <- posPredValue(pre, testSet$Y, positive="1")
recall <- sensitivity(pre, testSet$Y, positive="1")
F1 <- (2 * precision * recall) / (precision + recall)

roc_prob <- predict(fit, newdata=testSet, type="prob")
pred <- prediction(roc_prob[,2], testSet$Y)
# the 2nd column is where the label "1" is
# the default order of factors -1 and 1 is -1 < 1
# so "1" is treated as positive, and a ligher prob.
# means being closer to "1"

auc <- tryCatch(performance(pred, measure = "auc")@y.values[[1]],
                error=function(e) NA, warning=function(w) NA)

# logloss / cross entropy loss
logloss <- MLmetrics::LogLoss(roc_prob[,2], as.numeric(testSet$Y==1))

rst <- rbind(rst, c(ih, jtrgt, i, hvec[ih], trgtvec[jtrgt],
                    acc, auc, F1, logloss, table(trainSet$Y), table(testSet$Y)))
cat(ih, jtrgt, i, hvec[ih], trgtvec[jtrgt], acc, auc, F1, logloss,
    table(trainSet$Y), table(testSet$Y), "\n")
```

```r
      }
    } # end of jtrgt loop
  } # end of ih loop

  rst <- data.frame(rst)
  names(rst) <- c("ih", "jtrgt", "iCV", "hCUSUM", "trgt", "acc", "auc", "F1", "logloss",
                  "train0", "train1", "test0", "test1")
  write.csv(rst, "~/Dropbox/Teaching/STAT430/homework/hw04/rst.csv", row.names = F)

}

########################
# Organize the results #
########################

perfCV <- read.csv("~/Dropbox/Teaching/STAT430/homework/hw04/rst.csv", header = T)

# remove those records of which either acc, auc, or F1 are not available
perfCV <- subset(perfCV, (!is.na(acc))&(!is.na(auc))&(!is.na(F1))&(!is.na(logloss)))

cnt <- aggregate(perfCV$acc, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=length)
acc <- aggregate(perfCV$acc, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
auc <- aggregate(perfCV$auc, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
f1 <- aggregate(perfCV$F1, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
logloss <- aggregate(perfCV$logloss, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
train1 <- aggregate(perfCV$train1, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
train0 <- aggregate(perfCV$train0, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
test1 <- aggregate(perfCV$test1, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)
test0 <- aggregate(perfCV$test0, by=list(perfCV$hCUSUM, perfCV$trgt), FUN=mean)

# disable warning
options(warn=-1)

# combine results by merging multiple data.frame together
mer <- Reduce(function(...) merge(..., by=c("Group.1","Group.2")),
              list(cnt, acc, auc, f1,logloss, train1,train0,test1,test0))
names(mer) <- c("hCUSUM", "trgt", "kCV", "acc", "auc", "f1", "logloss",
                "train1","train0","test1", "test0")
```

```r
# rule out those the k-fold CV hasn't been successfully conducted
mer <- subset(mer, kCV==5)

####################
# Rank the results #
####################

options(digits = 3)

# rank by logloss
rstlogloss <- mer[order(mer$logloss, decreasing=F),]
rstlogloss
```

```
##    hCUSUM    trgt kCV   acc   auc    f1 logloss train1 train0 test1  test0
## 2     0.2 0.00233   5 0.572 0.513 0.706   0.698   4007   2562  1023  672.8
## 10    0.6 0.00233   5 0.578 0.523 0.706   0.703   1783   1113   454  288.4
## 14    0.8 0.00233   5 0.574 0.513 0.703   0.704   1299    813   334  211.2
## 1     0.2 0.00100   5 0.511 0.515 0.556   0.707   4381   4061  1114 1030.0
## 18    1.0 0.00233   5 0.589 0.535 0.706   0.708   1002    643   258  167.2
## 6     0.4 0.00233   5 0.554 0.492 0.687   0.715   2522   1630   641  423.2
## 3     0.2 0.00367   5 0.624 0.517 0.757   0.716   2729   1402   708  368.0
## 9     0.6 0.00100   5 0.514 0.514 0.573   0.717   1863   1671   476  424.4
## 7     0.4 0.00367   5 0.616 0.487 0.751   0.717   1753    909   454  237.0
## 11    0.6 0.00367   5 0.631 0.494 0.761   0.718   1269    626   329  163.4
## 5     0.4 0.00100   5 0.487 0.479 0.546   0.719   2700   2447   687  620.2
## 15    0.8 0.00367   5 0.596 0.485 0.733   0.723    920    474   239  123.8
## 17    1.0 0.00100   5 0.539 0.535 0.581   0.726   1052    947   268  240.6
## 13    0.8 0.00100   5 0.501 0.488 0.558   0.731   1366   1215   348  308.2
## 19    1.0 0.00367   5 0.583 0.449 0.721   0.770    720    368   186   96.8
## 4     0.2 0.00500   5 0.666 0.519 0.789   0.779   1770    752   463  204.0
## 8     0.4 0.00500   5 0.642 0.514 0.770   0.844   1145    503   298  136.6
## 12    0.6 0.00500   5 0.645 0.556 0.774   0.944    829    345   216   94.2
## 16    0.8 0.00500   5 0.620 0.505 0.754   1.039    600    259   156   71.8
## 20    1.0 0.00500   5 0.606 0.535 0.740   1.334    473    208   123   57.2
```

```r
# rank by f1 scores
rstF1 <- mer[order(mer$f1, decreasing=T),]
rstF1
```

```
##    hCUSUM    trgt kCV   acc   auc    f1 logloss train1 train0 test1  test0
## 4     0.2 0.00500   5 0.666 0.519 0.789   0.779   1770    752   463  204.0
## 12    0.6 0.00500   5 0.645 0.556 0.774   0.944    829    345   216   94.2
## 8     0.4 0.00500   5 0.642 0.514 0.770   0.844   1145    503   298  136.6
## 11    0.6 0.00367   5 0.631 0.494 0.761   0.718   1269    626   329  163.4
## 3     0.2 0.00367   5 0.624 0.517 0.757   0.716   2729   1402   708  368.0
## 16    0.8 0.00500   5 0.620 0.505 0.754   1.039    600    259   156   71.8
## 7     0.4 0.00367   5 0.616 0.487 0.751   0.717   1753    909   454  237.0
## 20    1.0 0.00500   5 0.606 0.535 0.740   1.334    473    208   123   57.2
## 15    0.8 0.00367   5 0.596 0.485 0.733   0.723    920    474   239  123.8
## 19    1.0 0.00367   5 0.583 0.449 0.721   0.770    720    368   186   96.8
## 18    1.0 0.00233   5 0.589 0.535 0.706   0.708   1002    643   258  167.2
## 10    0.6 0.00233   5 0.578 0.523 0.706   0.703   1783   1113   454  288.4
## 2     0.2 0.00233   5 0.572 0.513 0.706   0.698   4007   2562  1023  672.8
## 14    0.8 0.00233   5 0.574 0.513 0.703   0.704   1299    813   334  211.2
## 6     0.4 0.00233   5 0.554 0.492 0.687   0.715   2522   1630   641  423.2
## 17    1.0 0.00100   5 0.539 0.535 0.581   0.726   1052    947   268  240.6
## 9     0.6 0.00100   5 0.514 0.514 0.573   0.717   1863   1671   476  424.4
## 13    0.8 0.00100   5 0.501 0.488 0.558   0.731   1366   1215   348  308.2
## 1     0.2 0.00100   5 0.511 0.515 0.556   0.707   4381   4061  1114 1030.0
## 5     0.4 0.00100   5 0.487 0.479 0.546   0.719   2700   2447   687  620.2
```

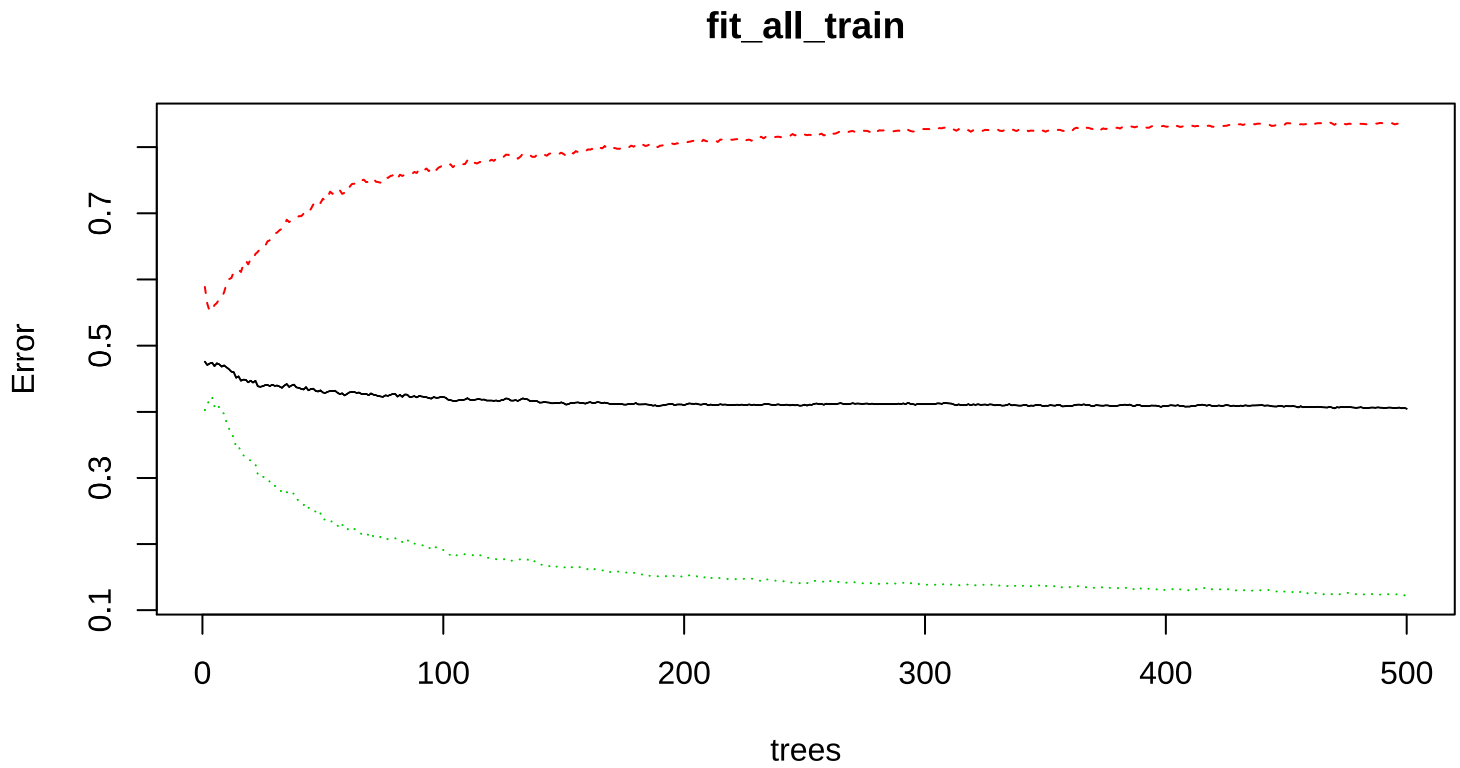
```
#######################################################################
# select candidate model based on logloss and evaluate with test_data #
#######################################################################
h_selected <- 0.2
trgt_selected <- 0.00233 # based on logloss
# trgt_selected <- 0.005 # based on f1

# re-run the model with all train_data
i_CUSUM <- fmlr::istar_CUSUM(train_data$mid_price, h=h_selected)
n_Event <- length(i_CUSUM)
events <- data.frame(t0 = i_CUSUM+1,
                     t1 = i_CUSUM+300,
                     trgt = rep(trgt_selected, n_Event),
                     side = rep(0,n_Event))
ptSl <- c(1,1)
out0 <- fmlr::label_meta(train_data$mid_price, events, ptSl, ex_vert = T)
table(out0$label)
```

```
##
##   -1    1
## 3365 5120
```

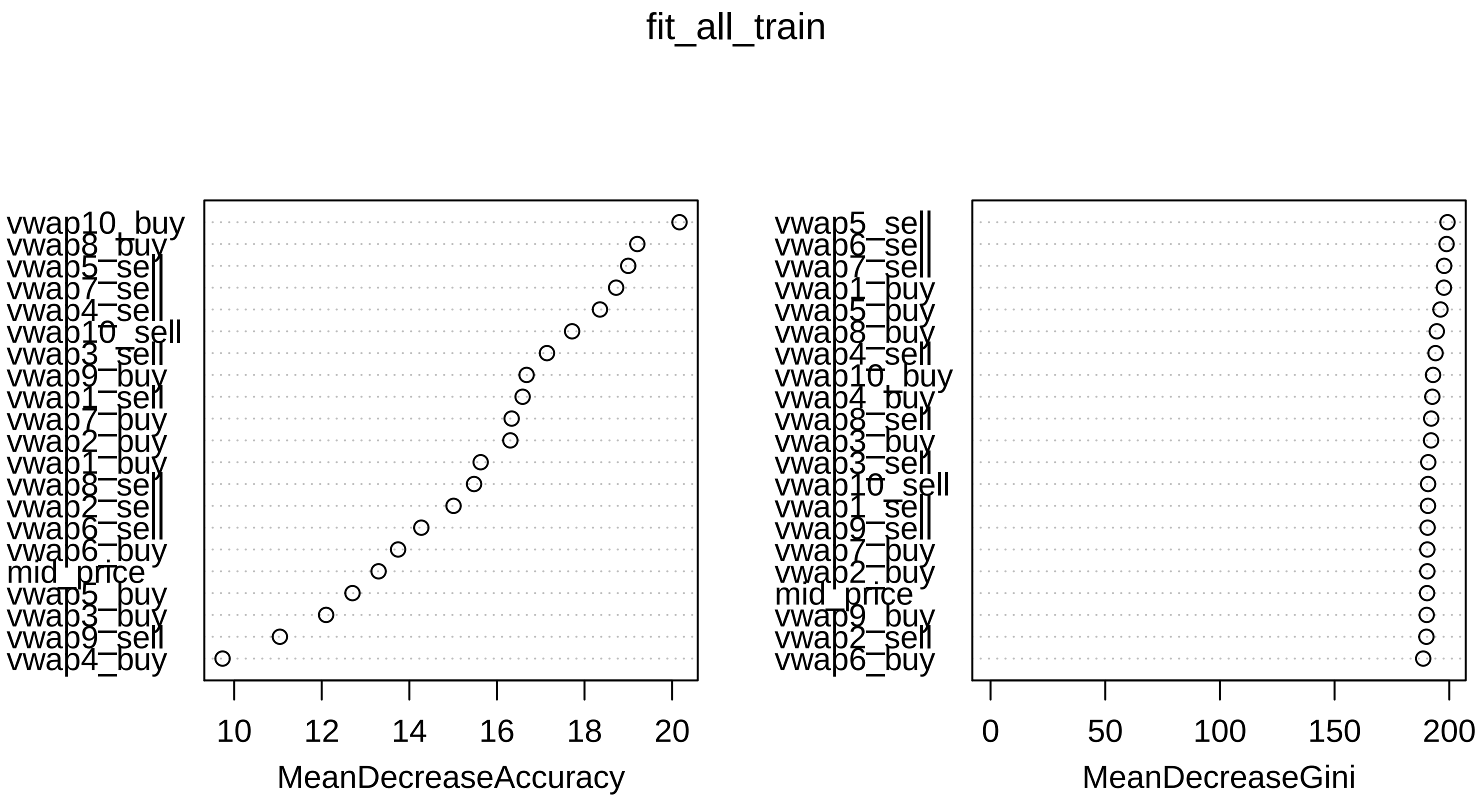
```r
fMat0 <- train_data[out0$t1Fea, !names(train_data)%in%c("m","h")]
fMat <- rbind( rep(NA, ncol(fMat0)),  apply(fMat0, 2,  function(x){diff(x)} ) )
allSet_train <- data.frame(Y=as.factor(out0$label), fMat)
idx_NA <- apply(allSet_train,1,function(x){sum(is.na(x))>0 })
allSet_train <- subset(allSet_train, !idx_NA)

mtry <- randomForestFML::tuneRF(allSet_train[,-1], allSet_train$Y, trace = FALSE, plot=FALSE)
```

```
## 0.00245 0.05
## -0.0185 0.05
```

```r
mtry <- mtry[which.min(mtry[,2]),1]
fit_all_train <- randomForestFML(Y ~ ., data = allSet_train, mtry = mtry, importance = TRUE, ntrees = 800)
plot(fit_all_train)
```




```r
varImpPlot(fit_all_train)
```




```r
# run the above fitted model for all test_data
i_CUSUM <- fmlr::istar_CUSUM(test_data$mid_price, h=h_selected)
n_Event <- length(i_CUSUM)
events <- data.frame(t0 = i_CUSUM+1,
                     t1 = i_CUSUM+300,
                     trgt = rep(trgt_selected, n_Event),
                     side = rep(0,n_Event))
ptSl <- c(1,1)
out0 <- fmlr::label_meta(test_data$mid_price, events, ptSl, ex_vert = T)
table(out0$label)
```

```
##
##   -1    1
## 1423 1535
```

```r
fMat0 <- test_data[out0$t1Fea, !names(test_data)%in%c("m","h")]
fMat <- rbind( rep(NA, ncol(fMat0)),  apply(fMat0, 2,  function(x){diff(x)} ) )
allSet_test <- data.frame(Y=as.factor(out0$label), fMat)
idx_NA <- apply(allSet_test,1,function(x){sum(is.na(x))>0 })
allSet_test <- subset(allSet_test, !idx_NA)

pre <- predict(fit_all_train, newdata = allSet_test)
cat("Confusion Matrix", "\n")
```

```
## Confusion Matrix
```

```r
table(allSet_test$Y, pre == 1) # associate TRUE with "1"
```

```
##
##      FALSE TRUE
##   -1   177 1246
##   1    185 1349
```

```r
acc <- mean(allSet_test$Y==pre)
precision <- posPredValue(pre, allSet_test$Y, positive="1")
recall <- sensitivity(pre, allSet_test$Y, positive="1")
F1 <- (2 * precision * recall) / (precision + recall)
cat("acc, precision, recall, F1", "\n")
```

```
## acc, precision, recall, F1
```

```r
cat(c(acc, precision, recall, F1))
```

```
## 0.516 0.52 0.879 0.653
```

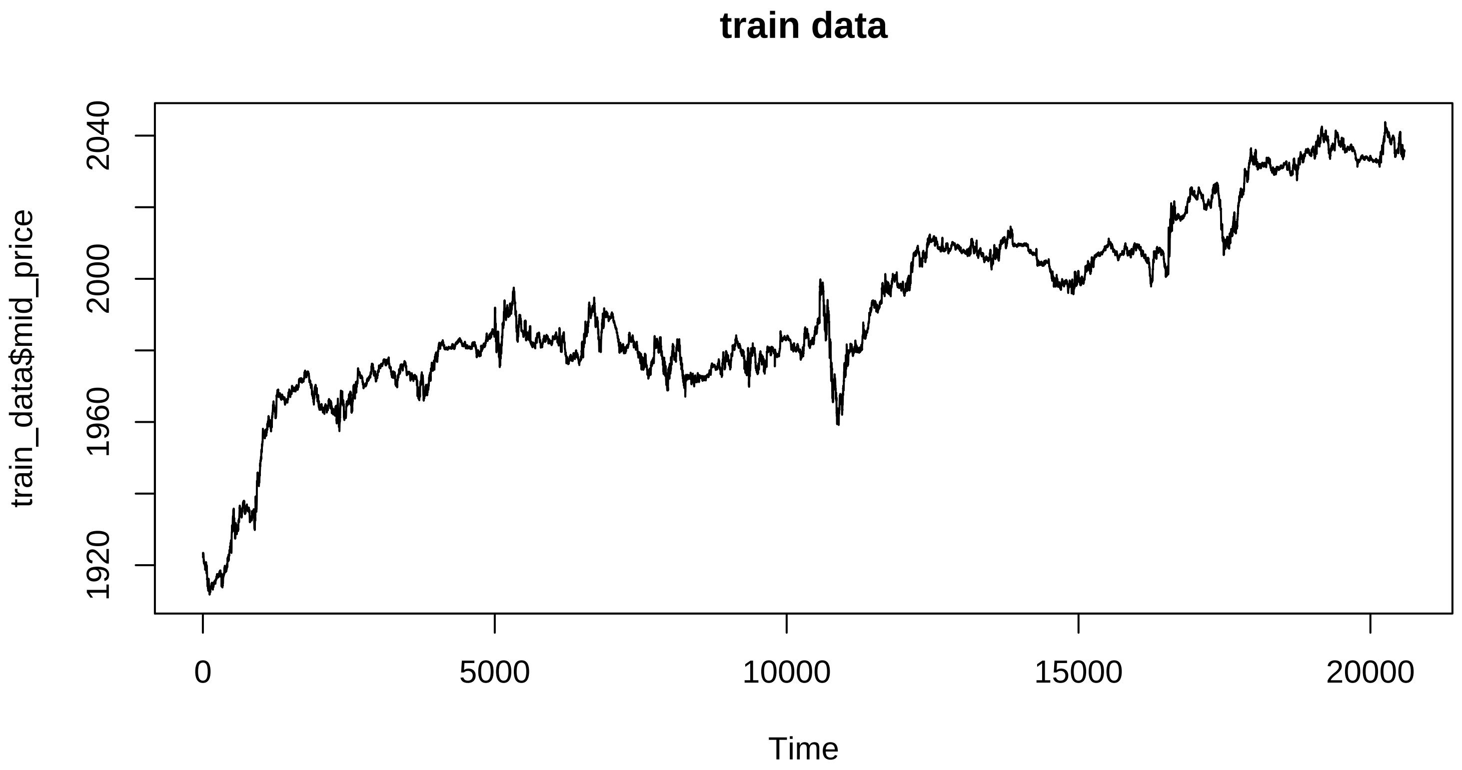
```r
acc_lucky(table(allSet_train$Y), table(allSet_test$Y), acc)
```

```
## $my_accuracy
## [1] 0.516
##
## $p_random_guess
## [1] 0.039
##
## $p_educated_guess
## [1] 0.089
##
## $mean_random_guess
```

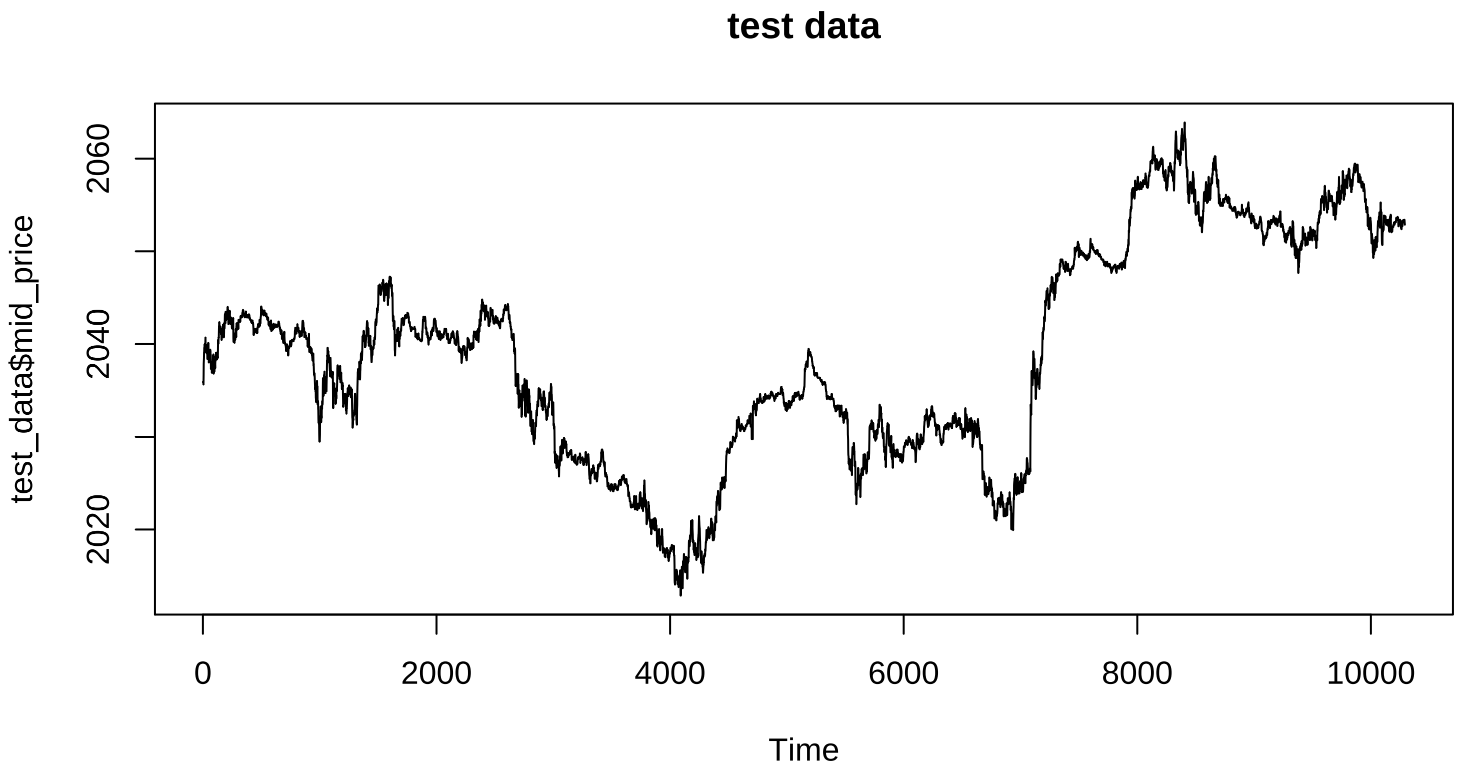
```
## [1] 0.5
##
## $mean_educated_guess
## [1] 0.504
##
## $acc_majority_guess
## [1] 0.519
```

From the confusion matrix, we can notice that the model did a good job for predicting the upward price movements in the unseen test set. However, the performance for predicting the downward movements in the test set was much worse. This is quite reasonable by looking at the following basic patterns of the training set and the test set: the training set has a lot more upward movements, while the test set does not have.

```
plot.ts(train_data$mid_price, main="train data")
```




```
plot.ts(test_data$mid_price, main="test data")
```


test data
test_data$mid_price
2020
2040
2060
0
2000
4000
6000
8000
10000
Time