QF301. Homework #4.

2021-11-08

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.

By filling out the following fields, you are signing this pledge. No assignment will get credit without being pledged.

Name: Siddharth Iyer

CWID: 10447455

Date: 11/8/2021

Instructions

In this assignment, you should use R markdown to answer the questions below. Simply type your R code into embedded chunks as shown above. When you have completed the assignment, knit the document into a PDF file, and upload both the .pdf and .Rmd files to Canvas.

```
CWID = 10447455 #Place here your Campus wide ID number, this will personalize
#your results, but still maintain the reproducible nature of using seeds.
#If you ever need to reset the seed in this assignment, use this as your seed
#Papers that use -1 as this CWID variable will earn 0's so make sure you change
#this value before you submit your work.
personal = CWID %% 10000
set.seed(personal)#You can reset the seed at any time in your code,
#but please always set it to this seed.
```

Question 1 (40pt)

Question 1.1

Use the quantmod package to obtain the daily adjusted close prices 2 different stocks. You should have at least two years of data for both assets. You should inspect the dates for your data to make sure you are including everything appropriately. Create a data frame of the daily log returns both both stocks along with the lagged returns (2 lags). Also include the direction (positive or negative) for both stocks in the current time point (not lagged). You may wish to remove the date from your data frame for later analysis. Print the first 6 lines of your data frame. (You may use the same two stocks as in Homework 2 or 3.)

Solution:

```
library(quantmod)
stocks = c("AAPL", "AMD")
getSymbols(stocks, from="2018-01-01", to="2021-10-14")
```

```
## [1] "AAPL" "AMD"
```

```
AAPL = AAPL$AAPL.Adjusted
AMD = AMD$AMD.Adjusted

aapl_log_rets = dailyReturn(AAPL, type="log")[-1]
amd_log_rets = dailyReturn(AMD, type="log")[-1]

aapl_l0 = aapl_log_rets[-2:-1]
aapl_l1 = as.numeric(stats::lag(aapl_log_rets,k=1))[-2:-1]
aapl_l2 = as.numeric(stats::lag(aapl_log_rets,k=2))[-2:-1]
aapl_dir = (aapl_l0>0)+0
amd_l0 = amd_log_rets[-2:-1]
amd_l1 = as.numeric(stats::lag(amd_log_rets,k=1))[-2:-1]
amd_l2 = as.numeric(stats::lag(amd_log_rets,k=2))[-2:-1]
amd_dir = (amd_l0>0)+0


lagged_df = data.frame(aapl_l0, aapl_l1, aapl_l2, aapl_dir, amd_l0, amd_l1, amd_l2, amd_dir)
colnames(lagged_df) <- c("aapl_l0", "aapl_l1", "aapl_l2", "aapl_dir", "amd_l0", "amd_l1", "amd_l2", "amd
head(lagged_df)
```

```
##                  aapl_l0       aapl_l1       aapl_l2 aapl_dir       amd_l0
## 2018-01-05  0.0113212226  0.0046341324 -0.0001744342        1 -0.020000667
## 2018-01-08 -0.0037212005  0.0113212226  0.0046341324        0  0.033115609
## 2018-01-09 -0.0001149321 -0.0037212005  0.0113212226        0 -0.038178911
## 2018-01-10 -0.0002292799 -0.0001149321 -0.0037212005        0  0.011774737
## 2018-01-11  0.0056640074 -0.0002292799 -0.0001149321        1  0.014938037
## 2018-01-12  0.0102734456  0.0056640074 -0.0002292799        1 -0.009933857
##                 amd_l1      amd_l2 amd_dir
## 2018-01-05  0.04817154  0.05061000       0
## 2018-01-08 -0.02000067  0.04817154       1
## 2018-01-09  0.03311561 -0.02000067       0
## 2018-01-10 -0.03817891  0.03311561       1
## 2018-01-11  0.01177474 -0.03817891       1
## 2018-01-12  0.01493804  0.01177474       0
```

Question 1.2

Split your data into training and testing sets (80% training and 20% test).

Run a logistic regression for the direction of one of your stock returns as a function of the lagged returns (2 lags) for both stocks. This should be of the form $\log(p_{1,t}/(1-p_{1,t})) = \beta_0 + \beta_{1,1}r_{1,t-1} + \beta_{1,2}r_{1,t-2} + \beta_{2,1}r_{2,t-1} + \beta_{2,2}r_{2,t-2}$. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
# split into train and test samples
N <- nrow(lagged_df)
train = sample(N, N*.8, replace = FALSE)

logistic.reg = glm(aapl_dir~aapl_l1+aapl_l2+amd_l1+amd_l2 , data=lagged_df , family=binomial, subset = 
summary(logistic.reg)
```

```
##
## Call:
## glm(formula = aapl_dir ~ aapl_l1 + aapl_l2 + amd_l1 + amd_l2,
##     family = binomial, data = lagged_df, subset = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6968  -1.2379   0.9449   1.1031   1.3867
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.18087    0.07391   2.447  0.01440 *
## aapl_l1     -13.45276    4.44405  -3.027  0.00247 **
## aapl_l2      -2.87728    4.06439  -0.708  0.47899
## amd_l1        3.00326    2.51772   1.193  0.23293
## amd_l2        4.91616    2.45204   2.005  0.04497 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1047.5  on 759  degrees of freedom
## Residual deviance: 1033.7  on 755  degrees of freedom
## AIC: 1043.7
##
## Number of Fisher Scoring iterations: 4
```

```
coef(logistic.reg)
```

```
## (Intercept)     aapl_l1     aapl_l2      amd_l1      amd_l2
##   0.1808731 -13.4527631  -2.8772806   3.0032597   4.9161643
```

```
summary(logistic.reg)$coef
```

```
##                Estimate Std. Error    z value   Pr(>|z|)
## (Intercept)   0.1808731 0.07391287  2.4471130 0.01440057
## aapl_l1     -13.4527631 4.44404586 -3.0271432 0.00246877
## aapl_l2      -2.8772806 4.06439329 -0.7079238 0.47899259
## amd_l1        3.0032597 2.51772498  1.1928466 0.23292946
## amd_l2        4.9161643 2.45204284  2.0049259 0.04497097
```

```
logistic.probs=predict(logistic.reg,type="response") # Predict the probability for direction on trainin

logistic.probs.test = predict(logistic.reg , newdata=lagged_df[-train, ], type="response") # Compute pr

# Use probabilities to make predictions
y.logistic.pred=rep(0,190) # Create repeated vector of "0"
y.logistic.pred[logistic.probs.test>.5] = 1 # Predict "1" if logistic regression gives greater probabil

# Evaluate the accuracy
#table(y.logistic.pred , lagged_df$aapl_l0[-train]) # Confusion matrix of results
```

```
logistic.acc = mean(y.logistic.pred==lagged_df$aapl_l0[-train]) # Directly compute the accuracy
logistic.acc
```

```
## [1] 0
```

Question 1.3

Consider the same classification problem as in Question 1.2 but with a Naive Bayes classifier. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
library("e1071")
nb = naiveBayes(aapl_dir~aapl_l1+aapl_l2+amd_l1+amd_l2, data=lagged_df, subset=train)

y.nb.prob = predict(nb , newdata=lagged_df[-train, ] , type="raw") # Compute probabilities
y.nb.pred = (y.nb.prob[,1] < y.nb.prob[,2])+0 # 1st column is "0", 2nd column is "1"

# Evaluate the accuracy
nb.acc = mean(y.nb.pred==lagged_df$aapl_dir[-train]) # Directly compute the accuracy
nb.acc
```

```
## [1] 0.5526316
```

```
table(y.nb.pred , lagged_df$aapl_dir[-train])
```

```
##
## y.nb.pred  0  1
##         0 21 14
##         1 71 84
```

Question 1.4

Consider the same classification problem as in Question 1.2 but with a Random Forest classifier with 300 trees and 2 predictors in each tree. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
library(randomForest)
rf = randomForest(aapl_dir~aapl_l1+aapl_l2+amd_l1+amd_l2, data=lagged_df, subset=train, ntree=300, mtry=
y.pred.rf = predict(rf, lagged_df[-train,])
rf.MSE = mean((y.pred.rf - lagged_df$aapl_l0[-train])^2)
rf.MSE
```

```
## [1] 0.2913605
```

Question 1.5

Consider the same classification problem as in Question 1.2 but with a neural network of your own design with at least 1 hidden layer and at least 3 hidden nodes. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
library(keras)

df1 = lagged_df[c("aapl_dir", "aapl_l0", "aapl_l1", "aapl_l2", "amd_l1", "amd_l2")]
rownames(df1) <- NULL

nn.reg = keras_model_sequential() %>% # Creating a model can sometimes take a bit of time
  layer_flatten(input_shape = c(4)) %>%
  layer_dense(units = 3, activation = "relu") %>%
  layer_dense(1 , activation = "linear")

nn.reg %>% # State the loss function and optimizer (adam is a good choice usually)
  compile(
    loss = "mean_squared_error",
    optimizer = "adam"
  )
```

Question 1.6

Consider the same classification problem as in Question 1.2 but with a support vector machine using a radial basis kernel. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
svm.model = svm(aapl_dir~aapl_l1+aapl_l2+amd_l1+amd_l2, data=lagged_df,kernal="tradial", subset = train)
summary(svm.model)
```

```
##
## Call:
## svm(formula = aapl_dir ~ aapl_l1 + aapl_l2 + amd_l1 + amd_l2, data = lagged_df,
##     kernal = "tradial", subset = train)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.25
##     epsilon:  0.1
##
##
## Number of Support Vectors:  707
```

```
names(svm.model)
```

```
##  [1] "call"            "type"            "kernel"          "cost"
##  [5] "degree"          "gamma"           "coef0"           "nu"
##  [9] "epsilon"         "sparse"          "scaled"          "x.scale"
## [13] "y.scale"         "nclasses"        "levels"          "tot.nSV"
## [17] "nSV"             "labels"          "SV"              "index"
## [21] "rho"             "compprob"        "probA"           "probB"
## [25] "sigma"           "coefs"           "na.action"       "fitted"
## [29] "decision.values" "residuals"       "terms"
```

```
svm.pred=predict(svm.model,newdata = lagged_df[-train, ])
table(predict=svm.pred,truth=lagged_df$aapl_dir[-train]) #Confusion matrix with manually given names
```

```
##                       truth
## predict                0 1
##   -0.100512896728862   0 1
##   -0.0587585617617872  1 0
##   -0.00424619263806325 1 0
##   0.0837042372621165   0 1
##   0.0998454959795024   1 0
##   0.152372144671843    1 0
##   0.223226923886572    1 0
##   0.227988366596204    0 1
##   0.233475251778643    1 0
##   0.243233687610055    0 1
##   0.255714720207109    0 1
##   0.270932187075734    1 0
##   0.277098303565053    1 0
##   0.289940797191496    0 1
##   0.328844581624857    1 0
##   0.357359969316911    1 0
##   0.369110316614454    1 0
##   0.37345117195233     1 0
##   0.403157506832517    0 1
##   0.415395540955277    1 0
##   0.430595300863478    0 1
##   0.44112723122816     0 1
##   0.4461140672672      1 0
##   0.449796442522514    0 1
##   0.464747590735302    0 1
##   0.490426684657866    0 1
##   0.497463508196686    1 0
##   0.500405426925971    1 0
##   0.501202033658076    0 1
##   0.502996428802343    1 0
##   0.508485633846852    0 1
##   0.556894599257252    0 1
##   0.57773011690262     1 0
##   0.588105058002938    0 1
##   0.590482911453332    1 0
##   0.590582508549911    0 1
##   0.59076935361302     1 0
```

```
##   0.600658530501668    1 0
##   0.604973875047902    0 1
##   0.610473074408375    1 0
##   0.612003117416674    0 1
##   0.651607627406077    1 0
##   0.65978516407293     1 0
##   0.662656250515724    1 0
##   0.670428185046889    1 0
##   0.674387966130632    0 1
##   0.699667075227574    0 1
##   0.706686606431703    1 0
##   0.720572598354056    1 0
##   0.72477833213866     1 0
##   0.726545729644776    1 0
##   0.730717978366653    1 0
##   0.731999281711657    1 0
##   0.733033897704184    1 0
##   0.737861959416117    0 1
##   0.748432910109843    1 0
##   0.750715687106972    0 1
##   0.754539845363683    0 1
##   0.760217529738532    0 1
##   0.760251897688436    0 1
##   0.763128950846316    0 1
##   0.772426545301527    0 1
##   0.776587604353258    0 1
##   0.777072492871203    0 1
##   0.784235580091774    1 0
##   0.788065515740465    1 0
##   0.794085207000022    1 0
##   0.798820944184902    1 0
##   0.803679943783166    1 0
##   0.804348510033517    0 1
##   0.811786186540702    0 1
##   0.813022325786593    0 1
##   0.815480823062047    0 1
##   0.817850979777111    1 0
##   0.8201403599787      1 0
##   0.824638461385592    0 1
##   0.825217393432674    1 0
##   0.826775102978764    0 1
##   0.828840599836311    0 1
##   0.831510609796689    0 1
##   0.834327649230839    0 1
##   0.834543981425207    1 0
##   0.83609422709308     0 1
##   0.837242694114593    0 1
##   0.839428194888685    1 0
##   0.840063272176373    0 1
##   0.841076604343582    0 1
##   0.841412545783233    1 0
##   0.841587468517449    0 1
##   0.846838926900527    1 0
##   0.849570228213512    0 1
```

```
##   0.849807857951307    0 1
##   0.85098311286076     1 0
##   0.85371141278552     0 1
##   0.855840561526915    1 0
##   0.861069505447748    1 0
##   0.861955697972947    1 0
##   0.862910511041721    0 1
##   0.864333710566603    0 1
##   0.864915588100626    1 0
##   0.865580715220418    0 1
##   0.873998691414818    0 1
##   0.874672162833147    0 1
##   0.874919600399113    0 1
##   0.875550550573524    0 1
##   0.876184273961201    0 1
##   0.877500146742482    0 1
##   0.880965837802134    1 0
##   0.884728135824221    1 0
##   0.88612300235248     1 0
##   0.887290430104038    1 0
##   0.890695607503028    0 1
##   0.891545002655573    0 1
##   0.893555416654263    0 1
##   0.893849565941636    0 1
##   0.900141042675674    0 1
##   0.901340936097625    0 1
##   0.904056882447247    1 0
##   0.904744888775162    1 0
##   0.906295564742891    0 1
##   0.906656112573003    0 1
##   0.90681481714355     0 1
##   0.907737259319226    0 1
##   0.909154800398487    0 1
##   0.91236479075177     1 0
##   0.91439638061388     1 0
##   0.91470574092271     0 1
##   0.914967559876448    1 0
##   0.915094374631043    1 0
##   0.916179993592528    0 1
##   0.919181215256285    0 1
##   0.922299057317857    0 1
##   0.923402675782049    0 1
##   0.92536741091659     1 0
##   0.926664644433614    0 1
##   0.928369095787941    1 0
##   0.928722207034968    1 0
##   0.929909830316259    1 0
##   0.9302924138614      1 0
##   0.93140043760521     1 0
##   0.931537470216518    1 0
##   0.9323098439547      1 0
##   0.933189432496173    1 0
##   0.93390805892556     1 0
##   0.934269334779434    1 0
```

```
##   0.934736955589048    1 0
##   0.93646820727392     0 1
##   0.93659336895032     0 1
##   0.936634255681845    0 1
##   0.936778314513592    0 1
##   0.937762199553929    0 1
##   0.939153169904377    0 1
##   0.940310596576753    1 0
##   0.941189315916011    1 0
##   0.941715570284815    1 0
##   0.942061577138168    0 1
##   0.942455100530629    0 1
##   0.943308250044473    1 0
##   0.944651215964573    0 1
##   0.94526511247302     1 0
##   0.945740484122683    0 1
##   0.946274908532078    0 1
##   0.948590267630599    0 1
##   0.948614421809473    0 1
##   0.948845991849587    0 1
##   0.949931442574476    0 1
##   0.950543495340776    0 1
##   0.950579642871484    0 1
##   0.950942113426602    0 1
##   0.953681711793542    1 0
##   0.956357127907391    1 0
##   0.956856580629658    0 1
##   0.957368322430214    1 0
##   0.959628878959868    0 1
##   0.962723594536246    0 1
##   0.964178050651552    1 0
##   0.964833430270909    1 0
##   0.968889943646479    1 0
##   0.971185322869104    0 1
##   0.972357134125579    0 1
##   0.978002114865359    1 0
##   0.97967247428717     1 0
##   0.980159888750638    1 0
##   0.987723237239237    1 0
##   0.99485884796418     1 0
##   1.00022676526213     1 0
##   1.01740227341017     1 0
##   1.02573097740648     1 0
##   1.02575188336979     0 1
##   1.06174228607123     0 1
```

```
mean((svm.pred==lagged_df$aapl_dir))
```

```
## [1] 0
```

Question 2 (10pt)

Question 2.1

Of the methods considered in Question 1, which would you recommend in practice? Explain briefly (1 paragraph) why you choose this fit.

Solution:

Naive Bayes is fast, has the highest accuracy, and is one of the leaset complex models we have implemented.

Question 3 (40pt)

Question 3.1

Using the same data as in Question 1.1, partition the predicted (current) stock returns into 5 possible "market directions". Add this data to your data frame and print the first 6 lines. Briefly (2-3 sentences) justify the choices of cutoff levels for the partitioning.

Solution:

```
direction = rep("Bear",length(lagged_df$aapl_dir))
quantile(lagged_df$aapl_l0,c(1/5,2/5,3/5,4/5))
```

```
##          20%          40%          60%          80%
## -0.011036549 -0.002147314  0.005481309  0.014547875
```

```
direction[lagged_df$aapl_l0 > quantile(lagged_df$aapl_l0,1/5)] = "Down"
direction[lagged_df$aapl_l0 > quantile(lagged_df$aapl_l0,2/5)] = "Flat"
direction[lagged_df$aapl_l0 > quantile(lagged_df$aapl_l0,3/5)] = "Up"
direction[lagged_df$aapl_l0 > quantile(lagged_df$aapl_l0,4/5)] = "Bull"

# Check how well this actually splits data:
sum(direction == "Bear")
```

```
## [1] 190
```

```
sum(direction == "Down")
```

```
## [1] 190
```

```
sum(direction == "Flat")
```

```
## [1] 190
```

```
sum(direction == "Up")
```

```
## [1] 190
```

```
sum(direction == "Bull")
```

```
## [1] 190
```

I partitioned them into bear, down, flat, up, bull markets... this is just easiest because normal distribution will take more time.

Question 3.2

Run a logistic regression for the generalized directions produced in Question 1.1 of one of your stock returns as a function of the lagged returns (2 lags) for both stocks.
Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
library(nnet)
df.multiclass = data.frame(direction = as.factor(direction), lagged_df$aapl_l1, lagged_df$aapl_l2, lagg

logistic.reg.new = multinom(direction ~ . , data=df.multiclass , subset = train)
```

```
## # weights:  30 (20 variable)
## initial  value 1223.172813
## iter  10 value 1203.763537
## iter  20 value 1202.600632
## iter  30 value 1202.402003
## iter  40 value 1202.401520
## final  value 1202.401447
## converged
```

```
# Let's classify all our test points
y.logistic.pred.new = predict(logistic.reg.new , newdata = df.multiclass[-train,])

# Consider the accuracy
logistic.acc.new = mean(y.logistic.pred.new==direction[-train]) # Directly compute the accuracy
logistic.acc.new
```

```
## [1] 0.1736842
```

```
table(y.logistic.pred.new , direction[-train]) # Confusion matrix of results
```

```
##
## y.logistic.pred.new Bear Bull Down Flat Up
##                Bear   10   12    8    4 12
##                Bull    8    8    5    2  5
##                Down    6    6    3    4  8
##                Flat   15   13   26   11 14
##                Up      2    2    3    2  1
```

I don't understand this graph at all.

Question 3.3

Consider the same classification problem as in Question 3.2 but with a Naive Bayes classifier. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
# Enter your R code here!
```

Question 3.4

Consider the same classification problem as in Question 3.2 but with a Random Forest classifier with 300 trees and 2 predictors in each tree. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
# Enter your R code here!
```

Question 3.5

Consider the same classification problem as in Question 3.2 but with a neural network of your own design with at least 1 hidden layer and at least 3 hidden nodes. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
# Enter your R code here!
```

Question 3.6

Consider the same classification problem as in Question 3.2 but with a support vector machine using a radial basis kernel. Use the same train/test split as in Question 1.2. Evaluate the performance of this model with the test accuracy and confusion matrix.

Solution:

```
# Enter your R code here!
```

Question 4 (10pt)

Question 4.1

Of the methods considered in Question 3, which would you recommend in practice? Explain briefly (1 paragraph) why you choose this fit.

Solution: