QF301. Homework #4.

2022-11-08

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: Arjun Koshal

CWID: 10459064

Date: 11/05/2022

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 = 10459064 #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)

getSymbols(c("AMZN","TSLA"),from="2018-01-01",to="2021-12-31")
```

```
## [1] "AMZN" "TSLA"
```

```
rAMZN = dailyReturn(AMZN$AMZN.Adjusted,type="log")
rTSLA = dailyReturn(TSLA$TSLA.Adjusted,type="log")

rAMZN1 = as.numeric(lag(rAMZN,k=1))[-(1:2)]
rAMZN2 = as.numeric(lag(rAMZN,k=2))[-(1:2)]
rAMZN = as.numeric(rAMZN)[-(1:2)]
rAMZN.dir = (rAMZN > 0) + 0
rTSLA1 = as.numeric(lag(rTSLA,k=1))[-(1:2)]
rTSLA2 = as.numeric(lag(rTSLA,k=2))[-(1:2)]
rTSLA = as.numeric(rTSLA)[-(1:2)]
rTSLA.dir = (rTSLA > 0) + 0

df = data.frame(rAMZN,rAMZN.dir,rTSLA,rTSLA.dir,rAMZN1,rTSLA1,rAMZN2,rTSLA2)
head(df)
```

```
##         rAMZN rAMZN.dir        rTSLA rTSLA.dir      rAMZN1       rTSLA1
## 1 0.004466030         1 -0.008324529         0 0.012694401 -0.010285832
## 2 0.016033295         1  0.006210381         1 0.004466030 -0.008324529
## 3 0.014321625         1  0.060754725         1 0.016033295  0.006210381
## 4 0.004664811         1 -0.008118266         0 0.014321625  0.060754725
## 5 0.001300360         1  0.003320920         1 0.004664811 -0.008118266
## 6 0.017661409         1  0.009364661         1 0.001300360  0.003320920
##        rAMZN2       rTSLA2
## 1 0.000000000  0.000000000
## 2 0.012694401 -0.010285832
## 3 0.004466030 -0.008324529
## 4 0.016033295  0.006210381
## 5 0.014321625  0.060754725
## 6 0.004664811 -0.008118266
```

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:

```
train = sample(length(rAMZN),0.8*length(rAMZN),replace=FALSE)

logistic.reg = glm(rAMZN.dir ~ rAMZN1 + rAMZN2 + rTSLA1 + rTSLA2 , data=df , subset=train , family=binor
summary(logistic.reg)
```

```
##
## Call:
## glm(formula = rAMZN.dir ~ rAMZN1 + rAMZN2 + rTSLA1 + rTSLA2,
##     family = binomial, data = df, subset = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4770  -1.2197   0.9927   1.1263   1.4184
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.12883    0.07135   1.806   0.0710 .
## rAMZN1      -2.00711    3.95915  -0.507   0.6122
## rAMZN2      -1.53355    3.93218  -0.390   0.6965
## rTSLA1      -3.35167    1.87451  -1.788   0.0738 .
## rTSLA2       2.20770    1.92141   1.149   0.2506
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1111.7  on 803  degrees of freedom
## Residual deviance: 1105.2  on 799  degrees of freedom
## AIC: 1115.2
##
## Number of Fisher Scoring iterations: 4
```

```
logistic.prob=predict(logistic.reg,df[-train,])

logistic.pred=rep(0,length(logistic.prob))
logistic.pred[logistic.prob>.5] = 1

mean(logistic.pred==df$rAMZN.dir[-train])
```

```
## [1] 0.3930348
```

```
table(predict=logistic.pred , truth=df$rAMZN.dir[-train])
```

```
##        truth
## predict   0   1
##       0  77 121
##       1   1   2
```

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")
df.x = data.frame(rAMZN1,rTSLA1,rAMZN2,rTSLA2)

nb = naiveBayes(df.x[train,] , df$rAMZN.dir[train])
```

```r
nb.prob = predict(nb , newdata=df.x[-train,] , type="raw")
head(nb.prob)
```

```
##              0         1
## [1,] 0.4850410 0.5149590
## [2,] 0.4721470 0.5278530
## [3,] 0.4529759 0.5470241
## [4,] 0.5132381 0.4867619
## [5,] 0.5087249 0.4912751
## [6,] 0.3775865 0.6224135
```

```r
nb.pred = (nb.prob[,1] < nb.prob[,2])+0

mean(nb.pred==df$rAMZN.dir[-train])
```

```
## [1] 0.5572139
```

```r
table(predict=nb.pred , truth=df$rAMZN.dir[-train])
```

```
##        truth
## predict  0  1
##       0 20 31
##       1 58 92
```

```r
nb.prob.train = predict(nb , newdata=df.x[train,] , type="raw")
nb.pred.train = (nb.prob.train[,1] < nb.prob.train[,2]) + 0
mean(nb.pred.train==df$rAMZN.dir[train])
```

```
## [1] 0.5261194
```

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:

```r
library(randomForest)

rAMZN.factor = as.factor(rAMZN.dir)
df.tree = data.frame(rAMZN.dir=rAMZN.factor , rAMZN1,rTSLA1,rAMZN2,rTSLA2)

rf.class = randomForest(rAMZN.dir ~ .,data=df.tree,subset=train,mtry=2,ntree=300,importance=TRUE)
rf.class
```

```
##
## Call:
##  randomForest(formula = rAMZN.dir ~ ., data = df.tree, mtry = 2,      ntree = 300, importance = TRUE
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 53.73%
## Confusion matrix:
##     0   1 class.error
## 0 138 240   0.6349206
## 1 192 234   0.4507042
```

```
rf.pred = predict(rf.class,newdata=df.tree[-train,])

mean(rf.pred==df.tree$rAMZN.dir[-train])
```

```
## [1] 0.5422886
```

```
table(predict=rf.pred,truth=df.tree$rAMZN.dir[-train])
```

```
##        truth
## predict  0  1
##       0 33 47
##       1 45 76
```

```
rf.pred.train = predict(rf.class,newdata=df.tree[train,])
mean(rf.pred.train==df$rAMZN.dir[train])
```

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

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:

```
See python code
```

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:

```
rAMZN.factor = as.factor(rAMZN.dir)
df.svm = data.frame(rAMZN.dir=rAMZN.factor , rAMZN1,rTSLA1,rAMZN2,rTSLA2)

tune.out = tune(svm,rAMZN.dir ~ .,data=df.svm[train,],kernel="radial",ranges=list(cost=c(0.001,.01,.1,1
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost gamma
##  0.001   0.5
##
## - best performance: 0.4702623
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-03   0.5 0.4702623 0.02415648
## 2  1e-02   0.5 0.4702623 0.02415648
## 3  1e-01   0.5 0.4702623 0.02415648
## 4  1e+00   0.5 0.5025000 0.05775885
## 5  5e+00   0.5 0.5011883 0.05632808
## 6  1e+01   0.5 0.5161265 0.04777448
## 7  1e+02   0.5 0.5211265 0.05684626
## 8  1e-03   1.0 0.4702623 0.02415648
## 9  1e-02   1.0 0.4702623 0.02415648
## 10 1e-01   1.0 0.4702623 0.02415648
## 11 1e+00   1.0 0.5148765 0.03626001
## 12 5e+00   1.0 0.5137500 0.06297765
## 13 1e+01   1.0 0.5113426 0.06108912
## 14 1e+02   1.0 0.4990586 0.07530384
## 15 1e-03   2.0 0.4702623 0.02415648
## 16 1e-02   2.0 0.4702623 0.02415648
## 17 1e-01   2.0 0.4702623 0.02415648
## 18 1e+00   2.0 0.5073920 0.04992964
## 19 5e+00   2.0 0.4964198 0.07575265
## 20 1e+01   2.0 0.4716049 0.08667379
## 21 1e+02   2.0 0.4889969 0.06696559
## 22 1e-03   3.0 0.4702623 0.02415648
## 23 1e-02   3.0 0.4702623 0.02415648
## 24 1e-01   3.0 0.4702623 0.02415648
## 25 1e+00   3.0 0.5261574 0.04763754
## 26 5e+00   3.0 0.4976543 0.06032215
## 27 1e+01   3.0 0.4926852 0.05527403
## 28 1e+02   3.0 0.4926235 0.07656664
## 29 1e-03   4.0 0.4702623 0.02415648
## 30 1e-02   4.0 0.4702623 0.02415648
## 31 1e-01   4.0 0.4702623 0.02415648
## 32 1e+00   4.0 0.5162809 0.04384251
## 33 5e+00   4.0 0.5075617 0.05016005
## 34 1e+01   4.0 0.5038889 0.06129917
```

```
## 35 1e+02   4.0 0.4888426 0.06253778
```

```
best.svm.class = tune.out$best.model
best.svm.class$cost
```

```
## [1] 0.001
```

```
best.svm.class$gamma
```

```
## [1] 0.5
```

```
summary(best.svm.class)
```

```
##
## Call:
## best.tune(method = svm, train.x = rAMZN.dir ~ ., data = df.svm[train,
##     ], ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5,
##     1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.001
##
## Number of Support Vectors:  763
##
##  ( 378 385 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
best.svm.pred=predict(best.svm.class,newdata = df.svm[-train,])

mean((best.svm.pred==df.svm$rAMZN.dir[-train]))
```

```
## [1] 0.6119403
```

```
table(predict=best.svm.pred,truth=df.svm$rAMZN.dir[-train])
```

```
##        truth
## predict   0   1
##       0   0   0
##       1  78 123
```

```r
best.svm.pred.train = predict(best.svm.class,newdata = df.svm[train,])
mean(best.svm.pred.train==df$rAMZN.dir[train])
```

```
## [1] 0.5298507
```

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:

Though Random forest appears to have performed the best for me in the test data, it has a training accuracy of 100%. This gives strong indications of overfitting and could result in poor performance on other (new) data. Instead, I would recommend the neural network even though SVM has slightly higher test accuracy. This is because the neural network provides more balanced predictions.

Random forest performed the best for me in the test data; however, it has a training accuracy of 100%. This probably means that the data was overfitting and would most likely imply poor performance on other data

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:

```r
quantile(rAMZN,c(0.2,0.4,0.6,0.8))
```

```
##          20%          40%          60%          80%
## -0.011432761 -0.001858162  0.005126694  0.013490184
```

```r
rAMZN.dir5 = rep("DDown",length(rAMZN.dir))
rAMZN.dir5[rAMZN > -0.01] = "Down"
rAMZN.dir5[rAMZN > -0.0025] = "Flat"
rAMZN.dir5[rAMZN > 0.0025] = "Up"
rAMZN.dir5[rAMZN > 0.01] = "UUp"

sum(rAMZN.dir5 == "DDown")
```

```
## [1] 222
```

```
sum(rAMZN.dir5 == "Down")
```

```
## [1] 165
```

```
sum(rAMZN.dir5 == "Flat")
```

```
## [1] 144
```

```
sum(rAMZN.dir5 == "Up")
```

```
## [1] 199
```

```
sum(rAMZN.dir5 == "UUp")
```

```
## [1] 275
```

```
df = data.frame(rAMZN,rAMZN.dir,rAMZN.dir5,rTSLA,rTSLA.dir,rAMZN1,rTSLA1,rAMZN2,rTSLA2)
head(df)
```

```
##         rAMZN rAMZN.dir rAMZN.dir5        rTSLA rTSLA.dir      rAMZN1
## 1 0.004466030         1         Up -0.008324529         0 0.012694401
## 2 0.016033295         1        UUp  0.006210381         1 0.004466030
## 3 0.014321625         1        UUp  0.060754725         1 0.016033295
## 4 0.004664811         1         Up -0.008118266         0 0.014321625
## 5 0.001300360         1       Flat  0.003320920         1 0.004664811
## 6 0.017661409         1        UUp  0.009364661         1 0.001300360
##          rTSLA1      rAMZN2       rTSLA2
## 1 -0.010285832 0.000000000  0.000000000
## 2 -0.008324529 0.012694401 -0.010285832
## 3  0.006210381 0.004466030 -0.008324529
## 4  0.060754725 0.016033295  0.006210381
## 5 -0.008118266 0.014321625  0.060754725
## 6  0.003320920 0.004664811 -0.008118266
```

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)

logistic.reg = multinom(rAMZN.dir5 ~ rAMZN1 + rAMZN2 + rTSLA1 + rTSLA2 , data=df , subset = train)
```

```
## # weights:  30 (20 variable)
## initial  value 1293.988082
## iter  10 value 1261.231766
## iter  20 value 1258.261915
## iter  30 value 1258.250688
## final  value 1258.249923
## converged
```

```
summary(logistic.reg)
```

```
## Call:
## multinom(formula = rAMZN.dir5 ~ rAMZN1 + rAMZN2 + rTSLA1 + rTSLA2,
##     data = df, subset = train)
##
## Coefficients:
##      (Intercept)    rAMZN1    rAMZN2      rTSLA1   rTSLA2
## Down  -0.3112787 -7.862107 -3.508649 -0.07679674 6.380442
## Flat  -0.5094570 13.287297  6.909072 -2.84335786 3.960596
## Up    -0.1910080 -2.961411  3.187844 -4.41931938 3.818332
## UUp    0.2145021 -4.463688 -5.040209 -3.87628542 5.196774
##
## Std. Errors:
##      (Intercept)   rAMZN1   rAMZN2   rTSLA1   rTSLA2
## Down   0.1161869 6.479734 6.330763 2.995812 3.079238
## Flat   0.1247934 6.689720 6.724008 3.107279 3.218670
## Up     0.1119557 6.277908 6.197927 2.905824 3.013450
## UUp    0.1012300 5.694587 5.579218 2.651664 2.720293
##
## Residual Deviance: 2516.5
## AIC: 2556.5
```

```
logistic.pred = predict(logistic.reg , newdata = df[-train,])

mean(logistic.pred==df$rAMZN.dir5[-train]) # Test accuracy
```

```
## [1] 0.2288557
```

```
table(predict=logistic.pred , truth=df$rAMZN.dir5[-train]) # Confusion matrix of results
```

```
##        truth
## predict DDown Down Flat Up UUp
##   DDown     7    6    4  9  13
##   Down      0    1    0  0   0
##   Flat      2    0    0  1   2
##   Up        1    0    0  2   1
##   UUp      29   24   25 38  36
```

```
logistic.pred.train = predict(logistic.reg , newdata = df[train,])
mean(logistic.pred.train==df$rAMZN.dir5[train])
```

```
## [1] 0.2773632
```

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:

```
nb = naiveBayes(df.x[train,] , df$rAMZN.dir5[train])

nb.prob = predict(nb , newdata=df.x[-train,] , type="raw")
head(nb.prob)
```

```
##          DDown       Down        Flat         Up       UUp
## [1,] 0.1177756 0.23132331 0.229665813 0.28107840 0.1401569
## [2,] 0.1152037 0.23946225 0.200107133 0.30368052 0.1415464
## [3,] 0.1057241 0.28532086 0.160683634 0.29892600 0.1493454
## [4,] 0.1526273 0.17775038 0.309719993 0.19426867 0.1656336
## [5,] 0.1387090 0.22856018 0.233356451 0.25406625 0.1453081
## [6,] 0.4144476 0.01659864 0.002051151 0.01012925 0.5567734
```

```
max.prob = pmax(nb.prob[,1],nb.prob[,2],nb.prob[,3],nb.prob[,4],nb.prob[,5])

nb.pred = rep("DDown",length(max.prob))
nb.pred[nb.prob[,1] == max.prob] = "DDown"
nb.pred[nb.prob[,2] == max.prob] = "Down"
nb.pred[nb.prob[,3] == max.prob] = "Flat"
nb.pred[nb.prob[,4] == max.prob] = "Up"
nb.pred[nb.prob[,5] == max.prob] = "UUp"

mean(nb.pred==df$rAMZN.dir5[-train]) #Test accuracy
```

```
## [1] 0.2338308
```

```
table(pred=nb.pred , true=df$rAMZN.dir5[-train]) #Test confusion matrix
```

```
##        true
## pred    DDown Down Flat Up UUp
##   DDown     4    2    1  5   5
##   Down      4    1    4  3   2
##   Flat      4    5    2  1  10
##   Up       13   14   15 27  22
##   UUp      14    9    7 14  13
```

```
nb.prob.train = predict(nb , newdata=df.x[train,] , type="raw")
max.prob.train = pmax(nb.prob.train[,1],nb.prob.train[,2],nb.prob.train[,3],nb.prob.train[,4],nb.prob.t
nb.pred.train = rep("DDown",length(max.prob.train))

nb.pred.train[nb.prob.train[,1] == max.prob.train] = "DDown"
```

```
nb.pred.train[nb.prob.train[,2] == max.prob.train] = "Down"
nb.pred.train[nb.prob.train[,3] == max.prob.train] = "Flat"
nb.pred.train[nb.prob.train[,4] == max.prob.train] = "Up"
nb.pred.train[nb.prob.train[,5] == max.prob.train] = "UUp"
mean(nb.pred.train==df$rAMZN.dir5[train])
```

```
## [1] 0.2649254
```

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:

```
rAMZN.factor = as.factor(rAMZN.dir5)
df.tree = data.frame(rAMZN.dir5=rAMZN.factor , rAMZN1,rTSLA1,rAMZN2,rTSLA2)

rf.class = randomForest(rAMZN.dir5 ~ .,data=df.tree,subset=train,mtry=2,ntree=300,importance=TRUE)
rf.class
```

```
##
## Call:
##  randomForest(formula = rAMZN.dir5 ~ ., data = df.tree, mtry = 2,      ntree = 300, importance = TRUI
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 76.74%
## Confusion matrix:
##       DDown Down Flat Up UUp class.error
## DDown    54   18   17 26  68   0.7049180
## Down     25   22    8 29  50   0.8358209
## Flat     22   13   14 28  38   0.8782609
## Up       35   22   17 19  56   0.8724832
## UUp      61   25   25 34  78   0.6502242
```

```
rf.pred = predict(rf.class,newdata=df.tree[-train,])

mean(rf.pred==df.tree$rAMZN.dir5[-train])
```

```
## [1] 0.2686567
```

```
table(predict=rf.pred,truth=df.tree$rAMZN.dir5[-train])
```

```
##        truth
## predict DDown Down Flat Up UUp
```

```
##   DDown    13   11    7 13   6
##   Down      2    1    3 11   8
##   Flat      3    2    3  4   5
##   Up        6    5    5  9   5
##   UUp      15   12   11 13  28
```

```
rf.pred.train = predict(rf.class,newdata=df.tree[train,])
mean(rf.pred.train==df$rAMZN.dir5[train])
```

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

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:

```
See python code
```

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:

```
rAMZN.factor = as.factor(rAMZN.dir5)
df.svm = data.frame(rAMZN.dir5=rAMZN.factor , rAMZN1,rTSLA1,rAMZN2,rTSLA2)

tune.out = tune(svm,rAMZN.dir5 ~ ., data=df.svm[train,],kernel="radial",ranges=list(cost=c(0.001,.01,.1
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost gamma
##  0.001   0.5
##
## - best performance: 0.722608
##
## - Detailed performance results:
##     cost gamma     error dispersion
```

```
## 1  1e-03   0.5 0.7226080 0.03353627
## 2  1e-02   0.5 0.7226080 0.03353627
## 3  1e-01   0.5 0.7226080 0.03353627
## 4  1e+00   0.5 0.7561728 0.03426026
## 5  5e+00   0.5 0.7910340 0.05092449
## 6  1e+01   0.5 0.7922531 0.04663705
## 7  1e+02   0.5 0.8059259 0.03311776
## 8  1e-03   1.0 0.7226080 0.03353627
## 9  1e-02   1.0 0.7226080 0.03353627
## 10 1e-01   1.0 0.7226080 0.03353627
## 11 1e+00   1.0 0.7910340 0.03477826
## 12 5e+00   1.0 0.7910340 0.04159761
## 13 1e+01   1.0 0.7947222 0.03850403
## 14 1e+02   1.0 0.7959877 0.03731279
## 15 1e-03   2.0 0.7226080 0.03353627
## 16 1e-02   2.0 0.7226080 0.03353627
## 17 1e-01   2.0 0.7226080 0.03353627
## 18 1e+00   2.0 0.7811111 0.03548557
## 19 5e+00   2.0 0.7784877 0.03537900
## 20 1e+01   2.0 0.7810494 0.02094459
## 21 1e+02   2.0 0.7734722 0.05085940
## 22 1e-03   3.0 0.7226080 0.03353627
## 23 1e-02   3.0 0.7226080 0.03353627
## 24 1e-01   3.0 0.7226080 0.03353627
## 25 1e+00   3.0 0.7799074 0.03666464
## 26 5e+00   3.0 0.7710802 0.04407294
## 27 1e+01   3.0 0.7611111 0.03595852
## 28 1e+02   3.0 0.7723457 0.04564035
## 29 1e-03   4.0 0.7226080 0.03353627
## 30 1e-02   4.0 0.7226080 0.03353627
## 31 1e-01   4.0 0.7226080 0.03353627
## 32 1e+00   4.0 0.7699383 0.04246447
## 33 5e+00   4.0 0.7648920 0.04087990
## 34 1e+01   4.0 0.7586574 0.04933454
## 35 1e+02   4.0 0.7723457 0.05239580
```

```
best.svm.class = tune.out$best.model
best.svm.class$cost
```

```
## [1] 0.001
```

```
best.svm.class$gamma
```

```
## [1] 0.5
```

```
summary(best.svm.class)
```

```
##
## Call:
## best.tune(method = svm, train.x = rAMZN.dir5 ~ ., data = df.svm[train,
##     ], ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5,
##     1, 2, 3, 4)), kernel = "radial")
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.001
##
## Number of Support Vectors:  800
##
##  ( 183 134 219 149 115 )
##
##
## Number of Classes:  5
##
## Levels:
##  DDown Down Flat Up UUp
```

```
best.svm.pred=predict(best.svm.class,newdata = df.svm[-train,])

mean((best.svm.pred==df.svm$rAMZN.dir5[-train]))
```

```
## [1] 0.2587065
```

```
table(predict=best.svm.pred,truth=df.svm$rAMZN.dir5[-train])
```

```
##        truth
## predict DDown Down Flat Up UUp
##   DDown     0    0    0  0   0
##   Down      0    0    0  0   0
##   Flat      0    0    0  0   0
##   Up        0    0    0  0   0
##   UUp      39   31   29 50  52
```

```
best.svm.pred.train = predict(best.svm.class,newdata = df.svm[train,])
mean(best.svm.pred.train==df$rAMZN.dir5[train])
```

```
## [1] 0.2773632
```

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:

Neural network appears to be the best since the accuracy is the best.