Coursera - Practical Machine Learning - Quiz4

Jean-Luc BELLIER

18 janvier 2017

Question 1

For this quiz we will be using several R packages. R package versions change over time, the right answers have been checked using the following versions of the packages.

AppliedPredictiveModeling: v1.1.6

caret: v6.0.47

ElemStatLearn: v2012.04-0

pgmm: v1.1

rpart: v4.1.8

gbm: v2.1

lubridate: v1.3.3

forecast: v5.6

e1071: v1.6.4

If you aren't using these versions of the packages, your answers may not exactly match the right answer, but hopefully should be close.

Load the vowel.train and vowel.test data sets:

```
library(ElemStatLearn)
```

```
## Warning: package 'ElemStatLearn' was built under R version 3.3.2
```

```
data(vowel.train)
data(vowel.test)
```

Set the variable y to be a factor variable in both the training and test set. Then set the seed to 33833. Fit (1) a random forest predictor relating the factor variable y to the remaining variables and (2) a boosted predictor using the "gbm" method. Fit these both with the train() command in the caret package.

What are the accuracies for the two approaches on the test data set? What is the accuracy among the test set samples where the two methods agree?

- RF Accuracy = 0.9881GBM Accuracy = 0.8371Agreement Accuracy = 0.9983
- RF Accuracy = 0.9987GBM Accuracy = 0.5152Agreement Accuracy = 0.9985
- RF Accuracy = 0.6082GBM Accuracy = 0.5152Agreement Accuracy = 0.6361
- RF Accuracy = 0.6082GBM Accuracy = 0.5152Agreement Accuracy = 0.5152

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.2

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 3.3.2

## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
set.seed(33833)
str(vowel.train)
```

```
## 'data.frame':    528 obs. of  11 variables:
##  $ y   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ x.1 : num  -3.64 -3.33 -2.12 -2.29 -2.6 ...
##  $ x.2 : num  0.418 0.496 0.894 1.809 1.938 ...
##  $ x.3 : num  -0.67 -0.694 -1.576 -1.498 -0.846 ...
##  $ x.4 : num  1.779 1.365 0.147 1.012 1.062 ...
##  $ x.5 : num  -0.168 -0.265 -0.707 -1.053 -1.633 ...
##  $ x.6 : num  1.627 1.933 1.559 1.06 0.764 ...
##  $ x.7 : num  -0.388 -0.363 -0.579 -0.567 0.394 0.217 0.322 -0.435 -0.512 -0.466 ...
##  $ x.8 : num  0.529 0.51 0.676 0.235 -0.15 -0.246 0.45 0.992 0.928 0.702 ...
##  $ x.9 : num  -0.874 -0.621 -0.809 -0.091 0.277 0.238 0.377 0.575 -0.167 0.06 ...
##  $ x.10: num  -0.814 -0.488 -0.049 -0.795 -0.396 -0.365 -0.366 -0.301 -0.434 -0.836 ...
```

```r
vowel.train$y <- as.factor(vowel.train$y)
vowel.test$y <- as.factor(vowel.test$y)

# Create Random Forest model and Gradient Boosting model on train set
modelRF <- train(y~.,data=vowel.train,method="rf")
```

```
## Loading required package: randomForest

## Warning: package 'randomForest' was built under R version 3.3.2

## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
modelGBM <- train(y~.,data=vowel.train, method="gbm",verbose=FALSE)
```

```
## Loading required package: gbm

## Warning: package 'gbm' was built under R version 3.3.2

## Loading required package: survival

## Warning: package 'survival' was built under R version 3.3.2

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.1

## Loading required package: plyr
```

```
##
## Attaching package: 'plyr'

## The following object is masked from 'package:ElemStatLearn':
##
##     ozone
```

```r
# Compute the prediction of both models on test set
test_RF <- predict(modelRF,newdata=vowel.test)
test_GBM <- predict(modelGBM,newdata=vowel.test)
confusionMatrix(vowel.test$y,test_RF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11
##         1  35  6  1  0  0  0  0  0  0  0  0
##         2   1 27 10  2  0  0  0  0  2  0  0
##         3   0  3 30  0  0  6  0  0  0  0  3
##         4   0  0  3 30  0  8  0  0  0  0  1
##         5   0  0  0  3 19 16  3  0  0  0  1
##         6   0  0  0  0  6 24  0  0  0  0 12
##         7   0  2  0  0  9  2 26  0  3  0  0
##         8   0  0  0  0  0  0  7 30  5  0  0
##         9   0  1  0  0  0  0  5  5 24  2  5
##         10  3 14  3  0  0  0  0  0  1 21  0
##         11  0  1  0  1  0  7  3  0 12  0 18
##
## Overall Statistics
##
##                Accuracy : 0.6147
##                  95% CI : (0.5686, 0.6593)
##     No Information Rate : 0.1364
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5762
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.89744  0.50000  0.63830  0.83333  0.55882  0.38095
## Specificity           0.98345  0.96324  0.97108  0.97183  0.94626  0.95489
## Pos Pred Value        0.83333  0.64286  0.71429  0.71429  0.45238  0.57143
## Neg Pred Value        0.99048  0.93571  0.95952  0.98571  0.96429  0.90714
## Prevalence            0.08442  0.11688  0.10173  0.07792  0.07359  0.13636
## Detection Rate        0.07576  0.05844  0.06494  0.06494  0.04113  0.05195
## Detection Prevalence  0.09091  0.09091  0.09091  0.09091  0.09091  0.09091
## Balanced Accuracy     0.94044  0.73162  0.80469  0.90258  0.75254  0.66792
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.59091  0.85714  0.51064   0.91304   0.45000
## Specificity           0.96172  0.97190  0.95663   0.95216   0.94313
## Pos Pred Value        0.61905  0.71429  0.57143   0.50000   0.42857
## Neg Pred Value        0.95714  0.98810  0.94524   0.99524   0.94762
## Prevalence            0.09524  0.07576  0.10173   0.04978   0.08658
```

```
## Detection Rate        0.05628  0.06494  0.05195   0.04545   0.03896
## Detection Prevalence  0.09091  0.09091  0.09091   0.09091   0.09091
## Balanced Accuracy     0.77632  0.91452  0.73363   0.93260   0.69656
```

```
confusionMatrix(vowel.test$y,test_GBM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11
##         1  30  9  1  0  0  0  0  0  0  2  0
##         2   0 21  8  1  0  1  0  0  5  0  6
##         3   0  1 11  9  0 14  0  0  0  0  7
##         4   0  0  3 21  3 14  0  0  0  0  1
##         5   0  0  0  3 19 11  7  0  0  0  2
##         6   0  0  0  0  4 30  2  0  0  0  6
##         7   0  1  0  1  1  1 36  2  0  0  0
##         8   0  0  0  0  0  0  6 29  7  0  0
##         9   0  0  0  0  0  1  3 10 28  0  0
##         10  2 12  0  0  0  0  1  2  6 19  0
##         11  0  0  0  0  0  6 14  0 18  0  4
##
## Overall Statistics
##
##                Accuracy : 0.5368
##                  95% CI : (0.4901, 0.583)
##     No Information Rate : 0.1688
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4905
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.93750  0.47727  0.47826  0.60000  0.70370  0.38462
## Specificity           0.97209  0.94976  0.92938  0.95082  0.94713  0.96875
## Pos Pred Value        0.71429  0.50000  0.26190  0.50000  0.45238  0.71429
## Neg Pred Value        0.99524  0.94524  0.97143  0.96667  0.98095  0.88571
## Prevalence            0.06926  0.09524  0.04978  0.07576  0.05844  0.16883
## Detection Rate        0.06494  0.04545  0.02381  0.04545  0.04113  0.06494
## Detection Prevalence  0.09091  0.09091  0.09091  0.09091  0.09091  0.09091
## Balanced Accuracy     0.95480  0.71352  0.70382  0.77541  0.82542  0.67668
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.52174  0.67442  0.43750   0.90476  0.153846
## Specificity           0.98473  0.96897  0.96482   0.94785  0.912844
## Pos Pred Value        0.85714  0.69048  0.66667   0.45238  0.095238
## Neg Pred Value        0.92143  0.96667  0.91429   0.99524  0.947619
## Prevalence            0.14935  0.09307  0.13853   0.04545  0.056277
## Detection Rate        0.07792  0.06277  0.06061   0.04113  0.008658
## Detection Prevalence  0.09091  0.09091  0.09091   0.09091  0.090909
## Balanced Accuracy     0.75324  0.82170  0.70116   0.92630  0.533345
```

```r
confusionMatrix(test_RF,test_GBM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5  6  7  8  9 10 11
##         1  31  3  1  0  0  0  0  0  0  4  0
##         2   0 38  3  1  0  1  2  0  5  0  4
##         3   1  1 19  6  0 10  0  0  0  2  8
##         4   0  0  0 24  1  9  0  0  2  0  0
##         5   0  0  0  0 17  3 11  2  0  0  1
##         6   0  0  0  3  8 45  7  0  0  0  0
##         7   0  1  0  0  1  0 39  1  2  0  0
##         8   0  0  0  0  0  0  0 33  2  0  0
##         9   0  0  0  1  0  0  2  4 40  0  0
##         10  0  1  0  0  0  0  1  3  3 15  0
##         11  0  0  0  0  0 10  7  0 10  0 13
##
## Overall Statistics
##
##                Accuracy : 0.6797
##                  95% CI : (0.635, 0.722)
##     No Information Rate : 0.1688
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6449
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.96875  0.86364  0.82609  0.68571  0.62963   0.5769
## Specificity           0.98140  0.96172  0.93622  0.97190  0.96092   0.9531
## Pos Pred Value        0.79487  0.70370  0.40426  0.66667  0.50000   0.7143
## Neg Pred Value        0.99764  0.98529  0.99036  0.97418  0.97664   0.9173
## Prevalence            0.06926  0.09524  0.04978  0.07576  0.05844   0.1688
## Detection Rate        0.06710  0.08225  0.04113  0.05195  0.03680   0.0974
## Detection Prevalence  0.08442  0.11688  0.10173  0.07792  0.07359   0.1364
## Balanced Accuracy     0.97507  0.91268  0.88115  0.82881  0.79527   0.7650
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.56522  0.76744  0.62500   0.71429   0.50000
## Specificity           0.98728  0.99523  0.98241   0.98186   0.93807
## Pos Pred Value        0.88636  0.94286  0.85106   0.65217   0.32500
## Neg Pred Value        0.92823  0.97658  0.94217   0.98633   0.96919
## Prevalence            0.14935  0.09307  0.13853   0.04545   0.05628
## Detection Rate        0.08442  0.07143  0.08658   0.03247   0.02814
## Detection Prevalence  0.09524  0.07576  0.10173   0.04978   0.08658
## Balanced Accuracy     0.77625  0.88133  0.80371   0.84807   0.71904
```

Question 2

Load the Alzheimer's data using the following commands:

```r
library(caret)
library(gbm)
set.seed(3433)
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version
## 3.3.2
```

```r
data(AlzheimerDisease)
adData = data.frame(diagnosis,predictors)
inTrain = createDataPartition(adData$diagnosis, p = 3/4)[[1]]
training = adData[ inTrain,]
testing = adData[-inTrain,]
```

Set the seed to 62433 and predict diagnosis with all the other variables using a random forest ("rf"), boosted trees ("gbm") and linear discriminant analysis ("lda") model. Stack the predictions together using random forests ("rf"). What is the resulting accuracy on the test set? Is it better or worse than each of the individual predictions?

- Stacked Accuracy: 0.80 is better than random forests and lda and the same as boosting.
- Stacked Accuracy: 0.69 is better than all three other methods
- Stacked Accuracy: 0.80 is better than all three other methods
- Stacked Accuracy: 0.76 is better than lda but not random forests or boosting.

```r
library(caret)
library(gbm)
set.seed(62433)
modelRF <- train(diagnosis~.,data=training, method="rf")
modelGBM <- train(diagnosis~.,data=training, method="gbm",verbose=FALSE)
modelLDA <- train(diagnosis~.,data=training, method="lda")
```

```
## Loading required package: MASS

## Warning: package 'MASS' was built under R version 3.3.2

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear
```

```r
# result on test set
predRF <- predict(modelRF,newdata=testing)
predGBM <- predict(modelGBM,newdata=testing)
predLDA <- predict(modelLDA,newdata=testing)

# Error on the models
Accuracy_RF <- confusionMatrix(testing$diagnosis,predRF)
Accuracy_RF$overall[1]
```

```
##  Accuracy
## 0.7682927
```

```r
Accuracy_GBM <- confusionMatrix(testing$diagnosis,predGBM)
Accuracy_GBM$overall[1]
```

```
##  Accuracy
## 0.7926829
```

```r
Accuracy_LDA <- confusionMatrix(testing$diagnosis,predLDA)
Accuracy_LDA$overall[1]
```

```
##  Accuracy
## 0.7682927
```

```
combDF <- data.frame(predRF, predGBM,predLDA,diagnosis=testing$diagnosis)
predCombModel <- train(diagnosis~., data=combDF, method="rf")
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
combPred <- predict(predCombModel,data=combDF)
Accuracy_Comb <- confusionMatrix(testing$diagnosis,combPred)
Accuracy_Comb$overall[1]
```

```
## Accuracy
## 0.804878
```

*** Answer : *** ***Stacked Accuracy: 0.80 is better than random forests and lda and the same as boosting***. We can notice that the accuracy for random forest and LDA are identical, and the lowest among all the models (0.769). GBM has an accuracy of 0.793, which is quite similar to the stacked accuracy.

Question 3

Load the concrete data with the commands:

```
set.seed(3523)
library(AppliedPredictiveModeling)
data(concrete)
inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]
training = concrete[ inTrain,]
testing = concrete[-inTrain,]
```

Set the seed to 233 and fit a lasso model to predict Compressive Strength. Which variable is the last coefficient to be set to zero as the penalty increases? (Hint: it may be useful to look up ?plot.enet).

- Cement
- BlastFurnaceSlag
- CoarseAggregate
- FineAggregate

```
set.seed(233)
library(elasticnet)
```

```
## Warning: package 'elasticnet' was built under R version 3.3.2
```

```
## Loading required package: lars
```

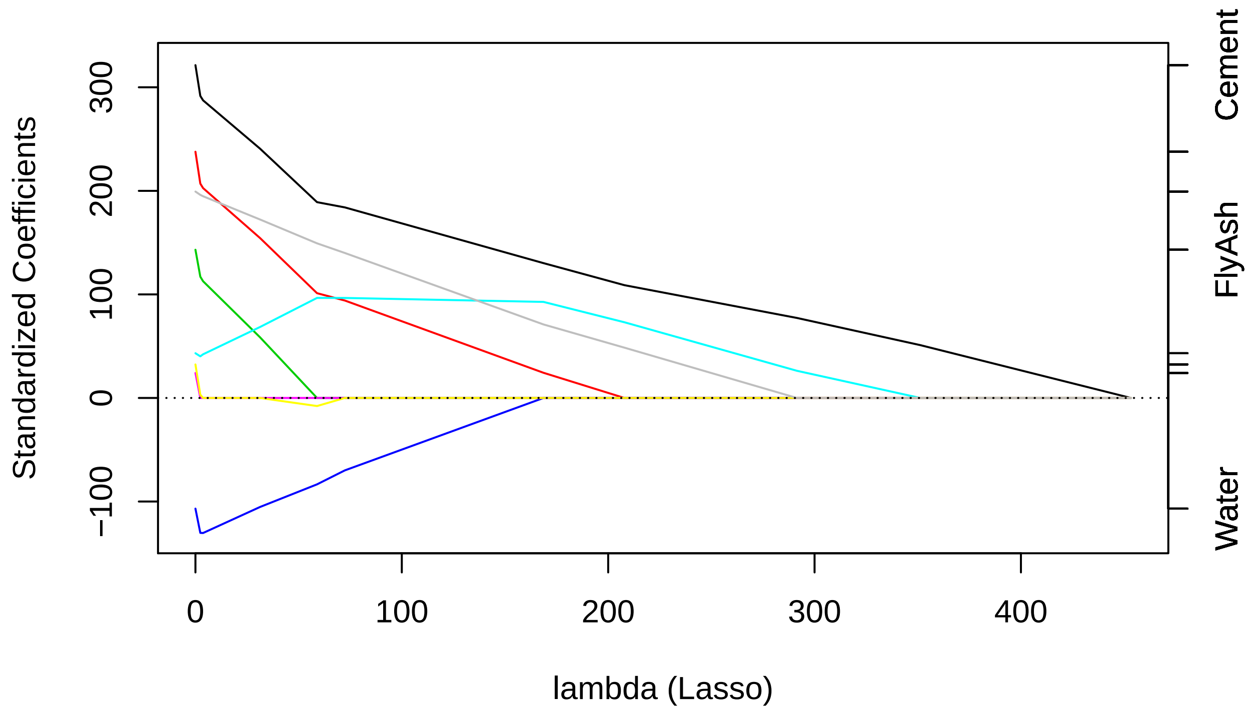
```
## Warning: package 'lars' was built under R version 3.3.2
```

```
## Loaded lars 1.2
```

```
Model_Lasso <- train(CompressiveStrength~.,data=training, method="lasso")
class(Model_Lasso)
```

```
## [1] "train"         "train.formula"
```

```
plot.enet(Model_Lasso$finalModel,xvar="penalty",use.color = TRUE)
```



Answer : The last coefficient to be set to zero as penalty increases is : Cement.

Question 4

Load the data on the number of visitors to the instructors blog from here:

https://d396qusza40orc.cloudfront.net/predmachlearn/gaData.csv

Using the commands:

```r
library(lubridate) # For year() function below
```

```
## Warning: package 'lubridate' was built under R version 3.3.2

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:plyr':
##
##     here

## The following object is masked from 'package:base':
##
##     date
```

```r
dat = read.csv("gaData.csv")
training = dat[year(dat$date) < 2012,]
testing = dat[(year(dat$date)) > 2011,]
```

```r
tstrain = ts(training$visitsTumblr)
tstrain
```

```
## Time Series:
## Start = 1
## End = 365
## Frequency = 1
##   [1]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [15]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [29]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [43]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [57]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [71]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [85]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
##  [99]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [113]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [127]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [141]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [155]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [169]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [183]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [197]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [211]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [225]    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## [239]    0    0    0    0    0    0    0    0    0    0    0    0   10   14
## [253]   10    8   95   44   15   15   34   26   17   63   35   45   50   82
## [267]   35   29   52   53  118   93   58   56   52   47  250  121   91   93
## [281]   41   52  580  230  119   99  109  252   97  176  196  145  128  142
## [295]   81   64  108   68   77  138   89   57   60   69  184  356  118  214
## [309]  145  114  191  131  157  632  758  191  140  305  313  378  306  251
## [323]  142  164  178  175  261  320  142  124  386 4997 1372  657  562  420
## [337]  229  156  255  262  253  304  202   90   76  232  212  306  196  319
## [351]  185  377  307  270  334  295  120   74   55   75   81  159  121   72
## [365]   59
```

Fit a model using the bats() function in the forecast package to the training time series. Then forecast this model for the remaining time points. For how many of the testing points is the true value within the 95% prediction interval bounds?

- 100%
- 94%
- 96%
- 92%

```r
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.3.2

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.3.2

##
## Attaching package: 'zoo'

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

```
## Loading required package: timeDate
## Warning: package 'timeDate' was built under R version 3.3.2
## This is forecast 7.3
```

```
model_bats <- bats(tstrain)
fcast <- forecast(model_bats,level=95,h = dim(testing)[1])
names(fcast)
```

```
## [1] "model"     "mean"      "level"     "x"         "upper"     "lower"
## [7] "fitted"    "method"    "residuals"
```

```
sum(testing$visitsTumblr >=fcast$lower & testing$visitsTumblr <=fcast$upper)/dim(testing)[1]
```

```
## [1] 0.9617021
```

Answer : The proportion of testing points having their true value within the 95% prediction interval bounds is nearly 96%.

Question 5

Load the concrete data with the commands:

```
set.seed(3523)
library(AppliedPredictiveModeling)
data(concrete)
inTrain = createDataPartition(concrete$CompressiveStrength, p = 3/4)[[1]]
training = concrete[ inTrain,]
testing = concrete[-inTrain,]
```

Set the seed to 325 and fit a support vector machine using the e1071 package to predict Compressive Strength using the default settings. Predict on the testing set. What is the RMSE?

- 6.72
- 6.93
- 107.44
- 11543.39

```
set.seed(325)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.3.2
##
## Attaching package: 'e1071'
## The following objects are masked from 'package:timeDate':
##
##     kurtosis, skewness
```

```
model_svm <- svm(CompressiveStrength~.,data=training)
pred_svm <- predict(model_svm,newdata=testing)
error <- pred_svm - testing$CompressiveStrength
sqrt(mean(error^2))
```

```
## [1] 6.715009
```

Answer : The RMSE is 6.72.