Practical Machine Learning Project

goose

March 17, 2020

Overview

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Course Instructions

The goal of the project is to predict the manner in which they did the exercise. This is the “classe” variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Submission Instructions

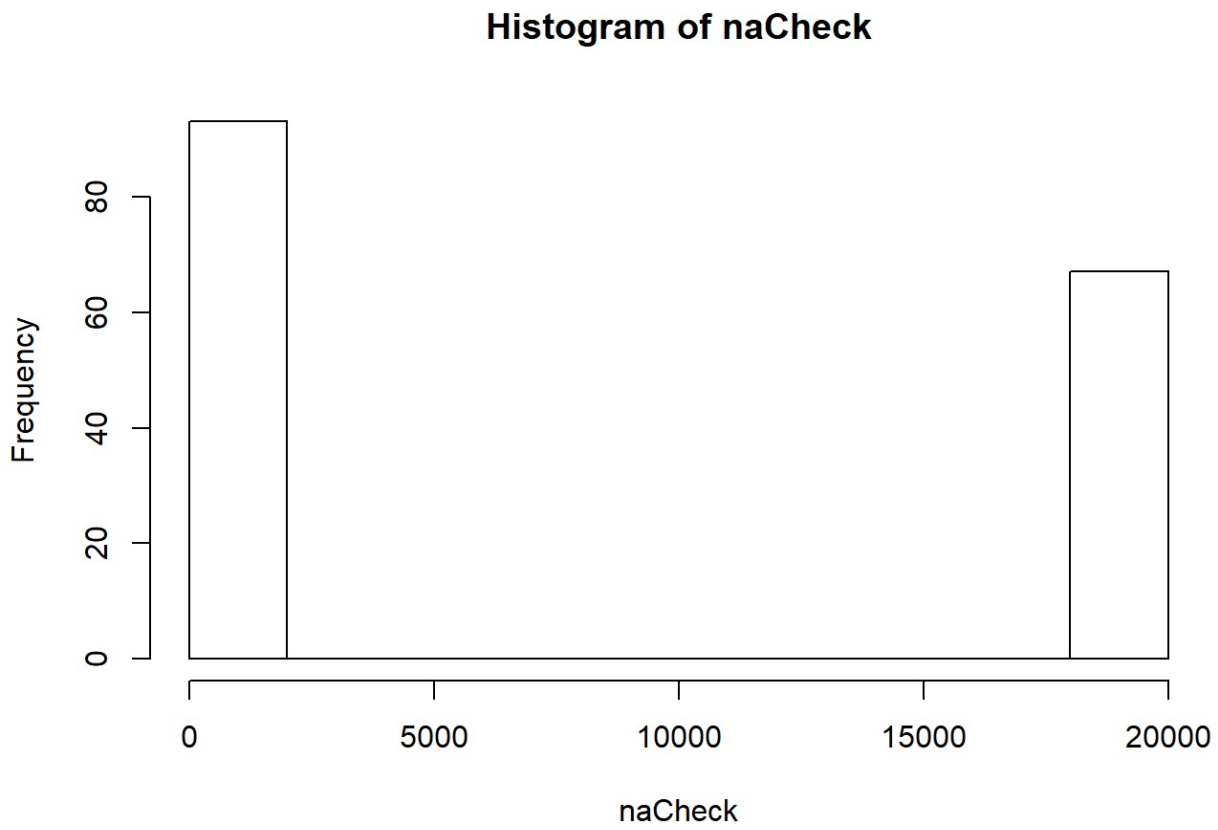
Your submission for the Peer Review portion should consist of a link to a Github repo with your R markdown and compiled HTML file describing your analysis. Please constrain the text of the writeup to < 2000 words and the number of figures to be less than 5. It will make it easier for the graders if you submit a repo with a gh-pages branch so the HTML page can be viewed online (and you always want to make it easy on graders :-).

Data Cleaning

Many of the data are missing from the data set, let’s take a quick look to see the structure of the missing data.

```
#count the missing data for each variable
naCheck <- apply(sapply(rawData,is.na),2,sum)

#Produce a histogram to visualize the missing data.
hist(naCheck)
```



Very clearly, the data for a variable is either completely missing, or completely present. This makes it easy: we will remove all variables with missing data. Further, since the test data is missing even more variables, we will remove all variables with missing data in the test set, as they will not be available for predictions in the use-case scenario.

```
#identify and remove variables in the test data set with missing variables as w
ell as the problem id variables.
cleanQuiz <- rawQuiz[,colSums(is.na(rawQuiz[,]))==0]
cleanQuiz <- cleanQuiz[,-c(1,60)]

#remove the same variables from the rest of the data as well as the problem id
variables
xData<- select(rawData,names(cleanQuiz[,-c(1,60)]))

#store the target variable
yData <- rawData$classe

#verify that the data are free of missing values
sum(is.na(cleanQuiz))
```

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

```
sum(is.na(yData))
```

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

```
sum(is.na(xData))
```

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

Next, we will remove the converted timestamp variable and knit together the two raw timestamp parts to make sure we get the entire value for the model.

```
#drop the factorized cvtd_timestamp data. the same data is captured by the raw_
timestamps and these are more precise
cleanQuiz <- select(cleanQuiz, -cvtd_timestamp)
xData <- select(xData, -cvtd_timestamp)

#create a feature to piece the two timestamps back together
xData$combinedTimestamp <- as.numeric(paste0(xData$raw_timestamp_part_1,xData$r
aw_timestamp_part_2))
cleanQuiz$combinedTimestamp <- as.numeric(paste0(cleanQuiz$raw_timestamp_part_
1,cleanQuiz$raw_timestamp_part_2))
```

Finally, we will create dummy variables for the variables with factor variables. (It turns out that `train` does this automatically for data expressed as a formula, but I did not know that at the time.)

```
#create dummy variables for factor variables
xObj <- dummyVars(~., xData)
xMatrix <- predict(xObj, xData)
xDummies <- as.data.frame(xMatrix)
quizMatrix <- predict(xObj, cleanQuiz)
quizDummies <- as.data.frame(quizMatrix)
```

Finally, we will partition the data and create one dataframe for modeling.

```
# set seed for reproducibility
set.seed(1234)

#Partition the data
trainIndex <- createDataPartition(yData, p = .8, list = F)
xTrain <- xDummies[trainIndex,]
yTrain <- yData[trainIndex]
xTest <- xDummies[-trainIndex,]
yTest <- yData[-trainIndex]

#Create Model Dataframe
modelDF <- cbind(xTrain,class = yTrain)
testDF <- cbind(xTest,class = yTest)
```

Classification Algorithms

Algorithms useful for classification include:

Logistic Regression Naive Bayes Classifier Nearest Neighbor Support Vector Machines Decision Trees Boosted Trees Random Forest Neural Networks Linear Discriminate Analysis Quadratic Discriminate Analysis

Of these, logistic regressions, SVMs, LDA's, and QDA's are best suited for binary classification models. For this reason, we will concern ourselves primarily with the remaining algorithms for this project. Specifically, we will conduct two nearest neighbor models, a simple neural network, and a random forest.

Model Training

Next we will train our models. The seed will be set prior to each model training for reproducibility of the individual models. Additionally, we will use the built in cross-validation function of the `train` function to conduct cross-validation for all models except knn2 which used a bootstrap validation for comparison.

```
# set seed for reproducibility
set.seed(1234)

#train the model
knnFit1 <- train(class ~., data = modelDF,
      method = "knn",
      preProc = c("center", "scale"),
      tuneLength = 1,
      trControl = trainControl(method = "cv")
      )
```

```
# set seed for reproducibility
set.seed(1234)

#train the model
knnFit2 <- train(class ~., data = modelDF,
      method = "knn",
      preProc = c("center", "scale"),
      tuneLength = 1,
      trControl = trainControl(method = "boot")
      )
```

```
# set seed for reproducibility
set.seed(1234)

#train the model
nnetFit <- train(class ~., data = modelDF,
      method = "nnet",
      preProcess = "range",
      tuneLength = ,
      trace = FALSE,
      maxit = 100,
      trControl = trainControl(method = "cv")
      )
```

```
# set seed for reproducibility
set.seed(1234)

#train the model
randomForestFit <- train(class ~., data = modelDF,
      method = "rf",
      tuneLength = 1,
      trControl = trainControl(method = "cv")
      )
```

Model Testing

Next we will test the models. Since this model will be used to answer the quiz, our primary concern is the percent of classifications which are correct. This will, therefore, be the criteria by which we evaluate our models. The code for the first model evaluation will be shown, the rest are included in the appendix for thoroughness.

```
knnFit1Pred <- predict(knnFit1, xTest)
(knn1Score <- sum((knnFit1Pred==yTest))/length(yTest))
```

```
## [1] 0.9536069
```

```
knn2Score
```

```
## [1] 0.9525873
```

```
nnetScore
```

```
## [1] 0.6803467
```

```
randomForestScore
```

```
## [1] 0.9992353
```

The best model had an accuracy of 99.9235279% and out of sample error of 0.0764721% which is more than enough to take our quiz and unlikely to be improved by combining models via a method such as stacking or ensembling. If we had been unsatisfied with the accuracy, we could have explored more models such as naive bayes, ADABoost, or a deep neural net instead of a simple one. Additionally we could have explored more parameters or created a combined model to improve our accuracy. Let's look a little closer at the randomForest performance:

```
confusionMatrix(yTest,randomForestFitPred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    0    0    0    0
##          B    1  758    0    0    0
##          C    0    2  682    0    0
##          D    0    0    0  643    0
##          E    0    0    0    0  721
##
## Overall Statistics
##
##                Accuracy : 0.9992
##                  95% CI : (0.9978, 0.9998)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.999
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9974   1.0000   1.0000   1.0000
## Specificity            1.0000   0.9997   0.9994   1.0000   1.0000
## Pos Pred Value         1.0000   0.9987   0.9971   1.0000   1.0000
## Neg Pred Value         0.9996   0.9994   1.0000   1.0000   1.0000
## Prevalence             0.2847   0.1937   0.1738   0.1639   0.1838
## Detection Rate         0.2845   0.1932   0.1738   0.1639   0.1838
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9996   0.9985   0.9997   1.0000   1.0000
```

Quiz Results:

We will now use the model to take the quiz:

```
predict(randomForestFit, quizDummies)
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```