Practical Machine Language Project: Write Up

dahra

Thursday, August 14, 2015

Background

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 a study, six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience. We made sure that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).

(Ref. [1])

Objectives

This report shows predictions on the manner in which 6 people performed barbell lifts correctly and incorrectly in 5 different ways. The data was collected using accelerometers on the belt, forearm, arm, and dumbell.

The 5 different ways the exercises were performed are identified in the training data set as the “classe” variable. The classes are described as follows: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

This report describes - how the model was built, - how cross validation was used, - the expected out of sample error, - and the reasons choices were made.

Lastly, the prediction model was used to predict 20 different test cases.

Method / Expectation

Method of building model/algorithm

Test models with (1) classification/ decision tree algorithm and (2) random forest algorithm. Choose the model with the highest accuracy as the final model.

Expectation

Expect a better performance using Random Forest.

Single decision trees often have high variance or high bias. Random Forests attempts to mitigate the problems of high variance and high bias by averaging to find a natural balance between the two extremes. (REF[5])

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. (REF[4])

vs.

Random forests which operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random forests correct for decision trees' habit of overfitting to their training set. (REF[4])

Data

Outcome variable : classe Number predictor variables: 52

Load & Read Data

```
####Go to working directory
#setwd("PracMachLearning/")

####Download data

dest_train = "train.csv"
dest_test = "test.csv"

if (!file.exists(dest_train)) {
  url_train <- ("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
  download.file(url_train,destfile="train.csv")
} else {
  print("test.csv exists")
}

if (!file.exists(dest_test)) {
  url_test <- ("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
  download.file(url_test,destfile="test.csv")
} else {
  print("test.csv exists")
}

###Load data into R using read.csv

traindata <- read.csv("train.csv", na.strings=c("", "NA"))
testdata <- read.csv("test.csv", na.strings=c("", "NA"))
```

Packages & Libraries

```
if (!require("caret")) {
  install.packages("caret", repos="http://cran.rstudio.com/")
  library("caret")
}
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.2.1
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.2.2
```

```
if (!require("e1071")) {
  install.packages("e1071", repos="http://cran.rstudio.com/")
  library("e1071")
}
```

```
## Loading required package: e1071
```

```
## Warning: package 'e1071' was built under R version 3.2.2
```

```
if (!require("rpart")) {
  install.packages("rpart", repos="http://cran.rstudio.com/")
  library("rpart")
}
```

```
## Loading required package: rpart
```

```
if (!require("randomForest")) {
  install.packages("randomForest", repos="http://cran.rstudio.com/")
  library("randomForest")

}
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.2.2
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

Clean Data

Datasets come sometimes with predictors that take an unique value across samples. Such uninformative predictor is more common than you might think. This kind of predictor is not only non-informative, it can break some models you may want to fit to your data (see example below). Even more common is the presence of predictors that are almost constant across samples. One quick and dirty solution is to remove all predictors that satisfy some threshold criterion related to their variance. (Ref. [2])

Drop Zero Variance Predictor Columns

Used function “nearZeroVar” from the caret package to (1.) remove predictors that have one unique value across samples (zero variance predictors), and (2.) remove predictors that have both (2a) few unique values relative to the number of samples and (2b) large ratio of the frequency of the most common value to the frequency of the second most common value (near-zero variance predictors).

```
#Original number of columns in training and testing datasets
ncol(traindata)
```

```
## [1] 160
```

```
ncol(testdata)
```

```
## [1] 160
```

```
#Zero Variance Predictors - nearZeroVar function

near0 <- nearZeroVar(traindata)
traindata <- traindata[-near0]
testdata <- testdata[-near0]

#New number of columns in training and testing datasets
ncol(traindata)
```

```
## [1] 117
```

```
ncol(testdata)
```

```
## [1] 117
```

Remove columns that have mainly missing values

```
removemissingtrain <- apply(traindata,2,function(x) {sum(is.na(x))})
traindata2 <- traindata[,which(removemissingtrain == 0)]

removemissingtest <- apply(testdata,2,function(x) {sum(is.na(x))})
testdata2 <- testdata[,which(removemissingtest == 0)]

#new col count for training
ncol(traindata2)
```

```
## [1] 59
```

```
#old col count for training
ncol(traindata)
```

```
## [1] 117
```

```
#new col count for testing
ncol(testdata2)
```

```
## [1] 59
```

```
#old col count for testing
ncol(testdata)
```

```
## [1] 117
```

Remove columns that are not relevant exercise data (drop id, username, windows and time columns)

Col 1 is id,

Col 2 is user_name , and

Col 3-5 are the timestamps.

Col 6 num_windows

```
finaltraindata <- traindata2[, -c(1,2,3,4,5,6)]
#New number of columns in training data
ncol(finaltraindata)
```

```
## [1] 53
```

```
finaltestdata <- testdata2[, -c(1,2,3,4,5,6)]
#New number of columns in training data
ncol(finaltestdata)
```

```
## [1] 53
```

List of Predictors

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"     "classe"
```

Plot: frequency of classe in training data

```
plot(finaltraindata$classe , ylim=c(0,6000), col="light blue",ylab="Frequency", xlab="Cla
sse Variable", main="Clean Training Data\nFreq of Classes")
```

Cross validation

Sample training data set randomly without replacement Training data: 70% of orig Training data Testing data: 30% of orig Training data The final test will be against the original testing data set using the best fit model.

create training and test sets

```
#70% training ; 30% testing
inTrain <- createDataPartition(y=finaltraindata$classe, p=0.7, list=FALSE)
trainset <- finaltraindata[inTrain,]
testset  <- finaltraindata[-inTrain,]
```

Dimension of Original and Training Dataset

```
rbind("original dataset" = dim(traindata),"training set" = dim(trainset))
```

```
##                   [,1] [,2]
## original dataset 19622  117
## training set     13737   53
```

Predictions

1. Classificaton Tree Prediction

Recursive partitioning helps us explore the stucture of a set of data, while developing easy to visualize decision rules for predicting a categorical (classification tree) or continuous (regression tree) outcome.

Grow the tree

```
#Classification Tree
classtree <- rpart(classe ~ ., data=trainset, method="class")
```

Determining the Smallest Cross-validated Error

The following code fragment automatically selects the complexity parameter associated with the smallest cross-validated error.

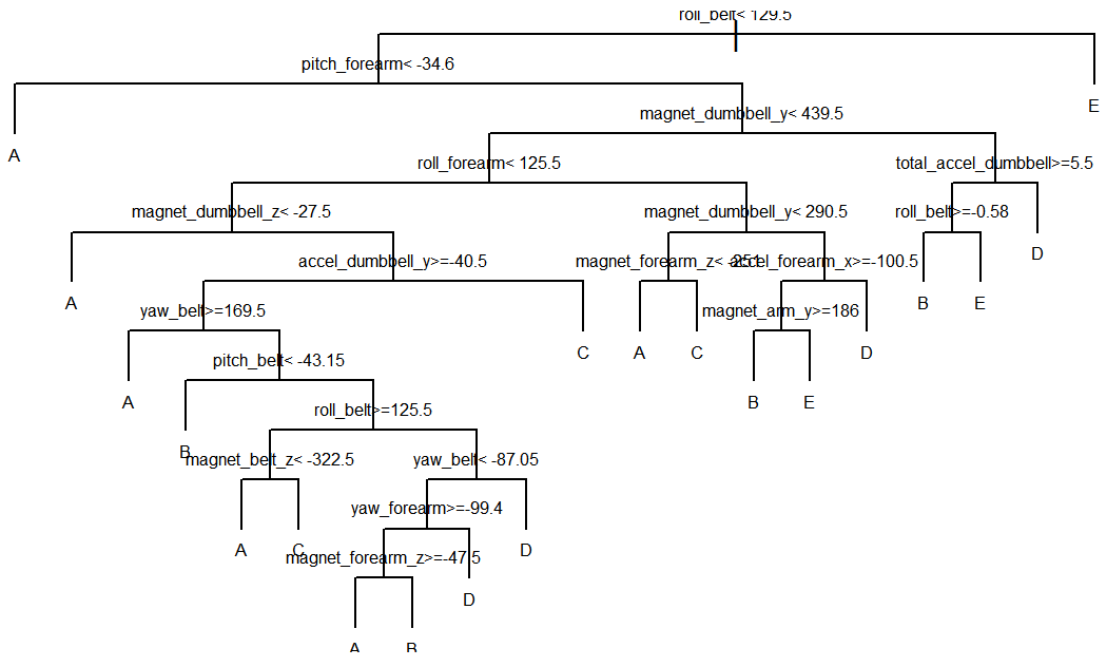
```
classtree$cptable[which.min(classtree$cptable[,"xerror"]),"CP"]
```

```
## [1] 0.01
```

Plot of Classification Tree

The basic idea of a classification tree is to first start with all variables in one group; imagine all the points in the above scatter plot. Then find some characteristic that best separates the groups. Then continue this process until the partitions have sufficiently homogeneous or are too small. (Ref[3])

```
plot(classtree,
        uniform=TRUE,
     main="Classification Tree")
text(classtree, cex=.5)
```



Prediction01

```
predict01 <- predict(classtree, testset, type = "class")
```

confusionMatrix

Cross-tabulation of observed and predicted classes with associated statistics.

```
confusionMatrix(predict01, testset$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1473  237   28  107   42
##          B   37  648  125   50  124
##          C   41  134  776  133  117
##          D   76   85   72  636   94
##          E   47   35   25   38  705
##
## Overall Statistics
##
##                Accuracy : 0.7201
##                  95% CI : (0.7085, 0.7316)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6446
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8799   0.5689   0.7563   0.6598   0.6516
## Specificity            0.9017   0.9292   0.9125   0.9336   0.9698
## Pos Pred Value         0.7806   0.6585   0.6461   0.6604   0.8294
## Neg Pred Value         0.9497   0.8998   0.9466   0.9334   0.9251
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2503   0.1101   0.1319   0.1081   0.1198
## Detection Prevalence   0.3206   0.1672   0.2041   0.1636   0.1444
## Balanced Accuracy      0.8908   0.7491   0.8344   0.7967   0.8107
```

2. Random Forest Prediction

Random forests improve predictive accuracy by generating a large number of bootstrapped trees (based on random samples of variables), classifying a case using each tree in this new “forest”, and deciding a final predicted outcome by combining the results across all of the trees (an average in regression, a majority vote in classification). Breiman and Cutler’s random forest approach is implimented via the randomForest package.

```
randforest <- randomForest(classe ~. , data=trainset, method="class")
```

Prediction02

```
predict02 <- predict(randforest, testset, type = "class")
```

confusionMatrix

Cross-tabulation of observed and predicted classes with associated statistics.

```
confusionMatrix(predict02, testset$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    7    0    0    0
##          B    0 1126    6    0    0
##          C    0    6 1020    7    1
##          D    0    0    0  956    4
##          E    0    0    0    1 1077
##
## Overall Statistics
##
##                Accuracy : 0.9946
##                  95% CI : (0.9923, 0.9963)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9931
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9886   0.9942   0.9917   0.9954
## Specificity            0.9983   0.9987   0.9971   0.9992   0.9998
## Pos Pred Value         0.9958   0.9947   0.9865   0.9958   0.9991
## Neg Pred Value         1.0000   0.9973   0.9988   0.9984   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1913   0.1733   0.1624   0.1830
## Detection Prevalence   0.2856   0.1924   0.1757   0.1631   0.1832
## Balanced Accuracy      0.9992   0.9937   0.9956   0.9954   0.9976
```

Results

As expected, Classification Tree did not improve the performance, so the Random Forest model was used.

Classification Tree Accuracy: 0.7314

Random Forest Accuracy: 0.9939

Submit Random Forest algorithm

Predicts the outcome levels based on the untouched Testing Dataset using Random Forest prediction algorithm

```
#Prediction for Submission
predictsubmit <- predict(randforest, finaltestdata)

predictsubmit
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

```
# Function to Create submission files
pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    files = paste0("problem_id_",i,".txt")
    write.table(x[i],file=files,col.names=FALSE,row.names=FALSE,quote=FALSE)
  }
}

pml_write_files(predictsubmit)
```

References

[1] The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har). [2] http://www.r-bloggers.com/near-zero-variance-predictors-should-we-remove-them/ (http://www.r-bloggers.com/near-zero-variance-predictors-should-we-remove-them/) [3]http://davetang.org/muse/2013/03/12/building-a-classification-tree-in-r/ (http://davetang.org/muse/2013/03/12/building-a-classification-tree-in-r/) [4] Wikipedia [5]http://www.datasciencecentral.com/profiles/blogs/random-forests-algorithm (http://www.datasciencecentral.com/profiles/blogs/random-forests-algorithm)