Practical Machine Learning: Assignment Write Up

Thisomimie

January 29, 2016

Executive Summary

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.

Data Description

Training data can be downloaded from Training Data

Testing data can be downloaded from Testing Data

The data for this project comes from this original source: http://groupware.les.inf.puc-rio.br/har.

Exploratory Data Analysis

```
## download the training data, put it the working directory and assign it to a dataframe called "train"
train <- read.csv("data/pml-training.csv", na.strings=c("","NA"))

## download the testing data, put it the working directory and assign it to a dataframe called "test"
test <- read.csv("data/pml-testing.csv", na.strings=c("","NA"))

## load relevant packages into the R environment, installing them if necessary
pckgInstall <- function(x) {
    for (i in x) {
        ## "require" returns TRUE invisibly if it was able to load package
        if (!require(i, character.only = TRUE)) {
            ## if package was not able to be loaded, install it
            install.packages(i, dependencies = TRUE, repos="http://cran.r-project.org/")
            ## load package after installing
            require (i, character.only = TRUE)
        }
    }
}

## assign names of required packages to "pkgs"
pckgs <- c("R2HTML", "caret", "doParallel", "randomForest")

## load/install packages
pckgInstall(pckgs)
```

```
## Loading required package: R2HTML
## Loading required package: caret

## Warning: package 'caret' was built under R version 3.2.3

## Loading required package: lattice
## Loading required package: ggplot2
## Loading required package: doParallel
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: parallel
## Loading required package: randomForest
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
dim(train)
```

```
## [1] 19622   160
```

Based on the lecture notes, below code is used to create training and testing dataset. 60% data is considered for the training set and remaining 40% is considered for testing the model.

Data Cleaning

Before we can move forward with data analysis, we split the training set into two for cross validation purposes. We randomly subsample 60% of the set for training purposes (actual model building), while the 40% remainder will be used only for testing, evaluation and accuracy measurement.

```
set.seed(12345)
library(caret)
inTrain <- createDataPartition(y=train$classe, p=0.6, list=FALSE)
myTraining <- train[inTrain, ]
myTesting <- train[-inTrain, ]
dim(myTraining)
```

```
## [1] 11776   160
```

```
dim(myTesting)
```

```
## [1] 7846  160
```

Both the training data and test data supplied at the course website contain 160 columns of information.

Now we are going to remove the NearZeroVariance variables using nearZeroVar() function from the 'caret' package to remove features that have a near zero variance as, again, these are unlikely to be predictive.

```
nzv <- nearZeroVar(myTraining, saveMetrics=TRUE)
myTraining <- myTraining[,nzv$nzv==FALSE]
```

The first column of the myTraining data set have to be remove to ensure prediction algorithm can run smoothly

```
myTraining <- myTraining[c(-1)]
```

Clean variables with more than 60% NA

```
trainingV3 <- myTraining
for(i in 1:length(myTraining)) {
    if( sum( is.na( myTraining[, i] ) ) /nrow(myTraining) >= .7) {
        for(j in 1:length(trainingV3)) {
            if( length( grep(names(myTraining[i]), names(trainingV3)[j]) ) == 1)  {
                trainingV3 <- trainingV3[ , -j]
            }
        }
    }
}

# Set back to the original variable name
myTraining <- trainingV3
rm(trainingV3)
```

Transform the myTesting and testing data sets

```
clean1 <- colnames(myTraining)
clean2 <- colnames(myTraining[, -58])  # remove the classe column
myTesting <- myTesting[clean1]         # allow only variables in myTesting that are also in myTraining
testing <- test[clean2]             # allow only variables in testing that are also in myTraining

dim(myTesting)
```

```
## [1] 7846   58
```

```
dim(testing)
```

```
## [1] 20 57
```

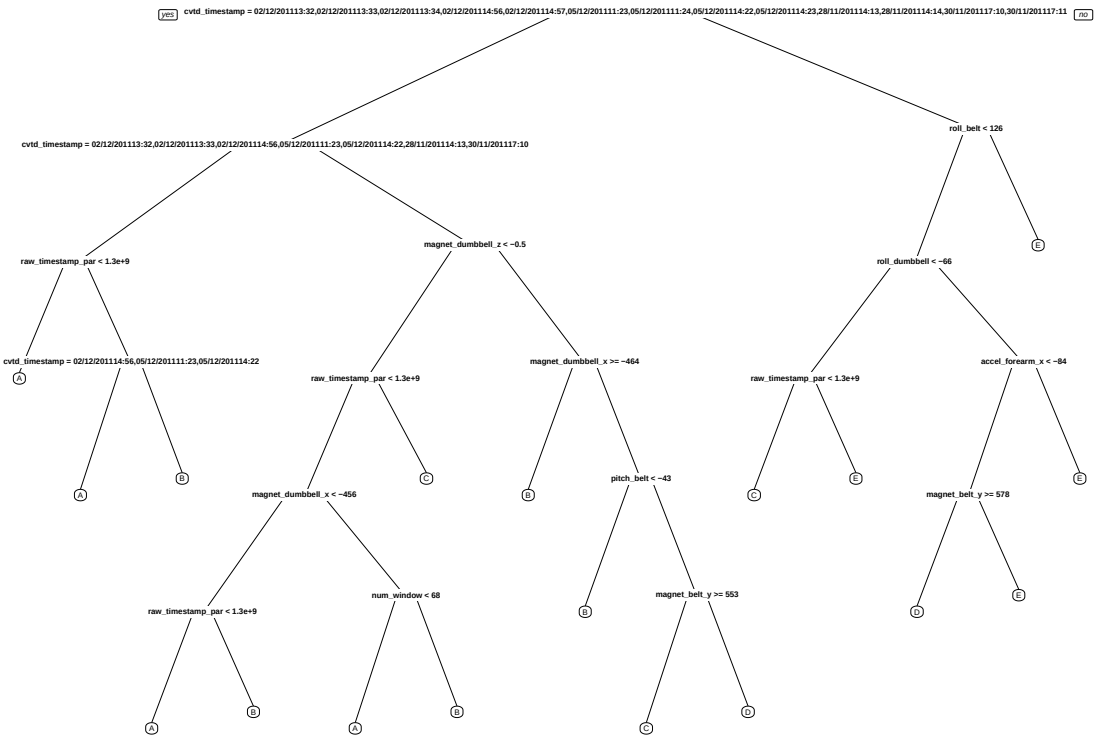
Coerce the data into the same type

```
for (i in 1:length(testing) ) {
    for(j in 1:length(myTraining)) {
        if( length( grep(names(myTraining[i]), names(testing)[j]) ) == 1)  {
            class(testing[j]) <- class(myTraining[i])
        }
    }
}

# To get the same class between testing and myTraining
testing <- rbind(myTraining[2, -58] , testing)
testing <- testing[-1,]
```

Prediction Algorithm

We will use the randomForest function (in the randomForest package) to fit the predictor to the training set. In the computer used for this analysis the default number of trees (500) gives a reasonable tradeoff between training time and accuracy. In more powerful machines that number can be increased for (slightly) better predictions.

```
library(rpart)
library(rpart.plot)

set.seed(12345)
modFitA1 <- rpart(classe ~ ., data=myTraining, method="class")
prp(modFitA1)
```



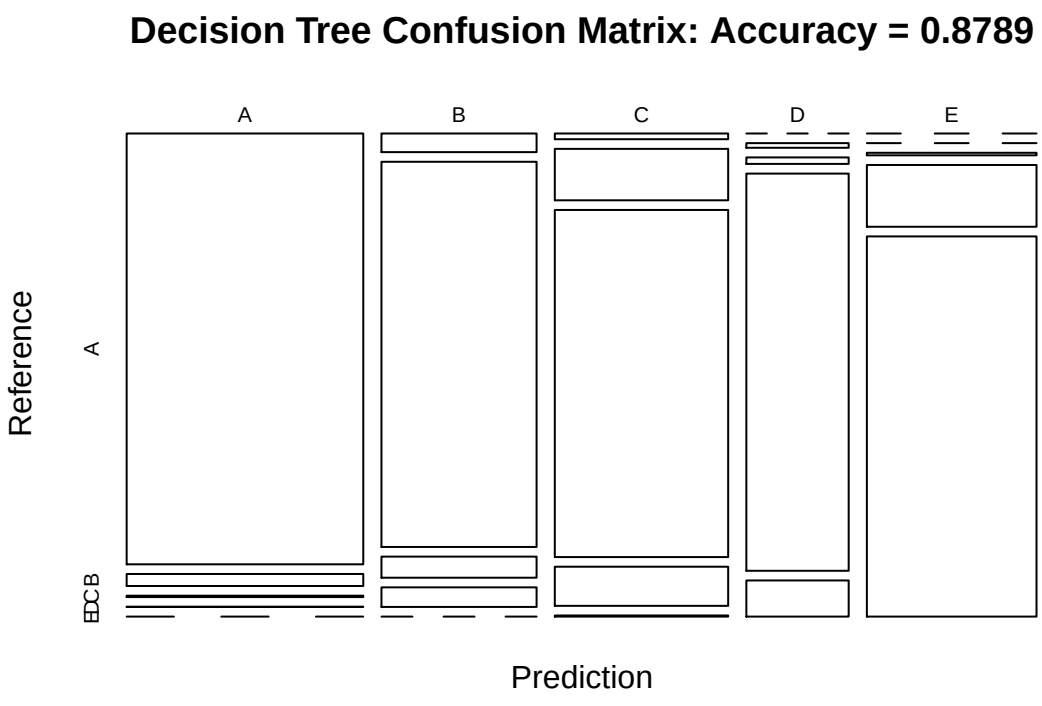
i. Prediction with Decision Trees

```
predictionsA1 <- predict(modFitA1, myTesting, type = "class")
cmtree <- confusionMatrix(predictionsA1, myTesting$classe)
cmtree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2150   60    7    1    0
##          B   61 1260   69   64    0
##          C   21  188 1269  143    4
```

```
##          D    0   10   14  857   78
##          E    0    0    9  221 1360
##
## Overall Statistics
##
##                Accuracy : 0.8789
##                  95% CI : (0.8715, 0.8861)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8468
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9633   0.8300   0.9276   0.6664   0.9431
## Specificity            0.9879   0.9693   0.9450   0.9845   0.9641
## Pos Pred Value         0.9693   0.8666   0.7809   0.8936   0.8553
## Neg Pred Value         0.9854   0.9596   0.9841   0.9377   0.9869
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2740   0.1606   0.1617   0.1092   0.1733
## Detection Prevalence   0.2827   0.1853   0.2071   0.1222   0.2027
## Balanced Accuracy      0.9756   0.8997   0.9363   0.8254   0.9536
```

```
plot(cmtree$table, col = cmtree$byClass, main = paste("Decision Tree Confusion Matrix: Accuracy =", rou
```

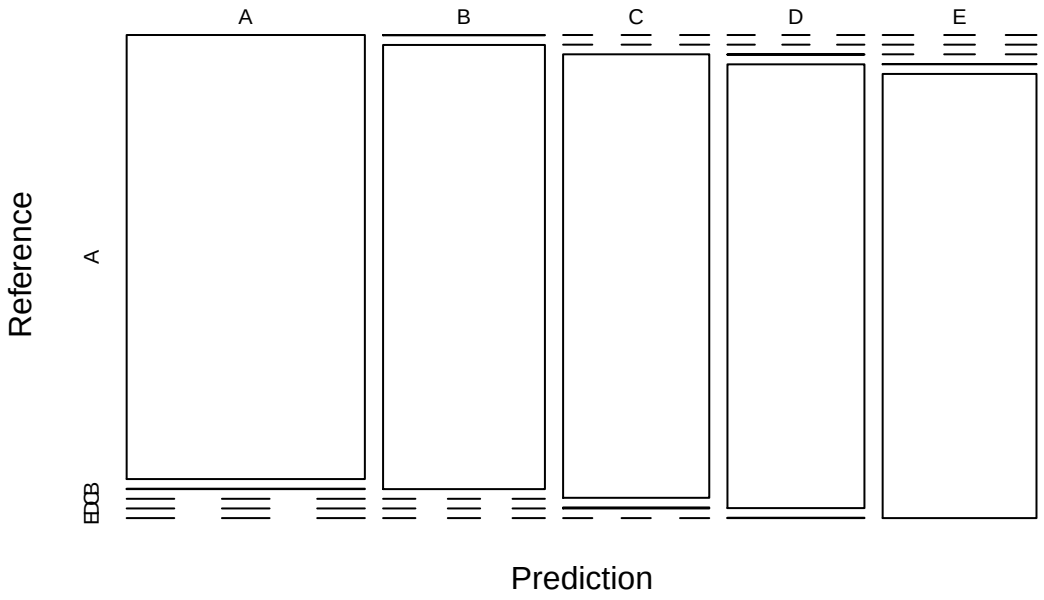


ii. Prediction with Random Forest

```r
set.seed(12345)
modFitB1 <- randomForest(classe ~ ., data=myTraining, ntree=500)
predictionB1 <- predict(modFitB1, myTesting, type = "class")
ranfor_conmat <- confusionMatrix(predictionB1, myTesting$classe)
ranfor_conmat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    2    0    0    0
##          B    1 1516    0    0    0
##          C    0    0 1367    3    0
##          D    0    0    1 1282    1
##          E    0    0    0    1 1441
##
## Overall Statistics
##
##                Accuracy : 0.9989
##                  95% CI : (0.9978, 0.9995)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9985
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9987   0.9993   0.9969   0.9993
## Specificity            0.9996   0.9998   0.9995   0.9997   0.9998
## Pos Pred Value         0.9991   0.9993   0.9978   0.9984   0.9993
## Neg Pred Value         0.9998   0.9997   0.9998   0.9994   0.9998
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1932   0.1742   0.1634   0.1837
## Detection Prevalence   0.2846   0.1933   0.1746   0.1637   0.1838
## Balanced Accuracy      0.9996   0.9993   0.9994   0.9983   0.9996
```

```r
plot(ranfor_conmat$table, col = cmtree$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =
```



Applying the Model to the Testing Subsample.

Random Forests gave an Accuracy in the myTesting dataset of 99.89%, which was more accurate than the Decision Trees. The expected out-of-sample error is 100-99.89 = 0.11%.

```
## predict the testing set and store the results in a character vector
predictionB2 <- predict(modFitB1, testing, type = "class")

## display the predictions
predictionB2
```

```
##  2 31  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
##  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
```

```
## create a directory called "test_output," suppressing warnings if the
## directory already exists
dir.create(file.path("test_output"), showWarnings = FALSE)

## create a function to write the files to be submitted
writefiles = function(x){
    n = length(x)
    for(i in 1:n){
        filename = paste0("test_output/problem_id_", i, ".txt")
        write.table(x[i], file = filename, quote = FALSE, row.names = FALSE,
                    col.names=FALSE)
    }
}
```

```
## then create the files, one for each prediction
writefiles(predictionB2)
```

Conclusion

The random forest learner is well suited to generating accurate predictions for this specific dataset. While a single stratification of the feature set (decision tree) wasn???t able to accurately predict the response, a bootstrapped ensemble of decision trees were.

The accuracy obtained (accuracy = 99.89%, and out-of-sample error = 0.11%) is obviously highly suspicious as it is never the case that machine learning algorithms are that accurate, and a mere 85% if often a good accuracy result.

Applying the random forest model to predict the 20 test cases from the Submission portion of the Course Project resulted in a 100% classification rate.

Reference

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz3ybOQ0F2X