Practical Machine Learning - Course Project

Alex Ho

December 18, 2016

Introduction

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.

The goal of this project was to take data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants (more information is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)), and based on this historical data, explore the efficacy of various models to predict what “classe” or type of exercise was being done.

After a prediction model was selected, this model was used to predict the “classe” for 20 different test cases.

Loading the Data

This chunk of data loads the necessary libraries and and data sets.

```
library(caret)
```

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

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

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

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

```
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

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

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

```
library(rpart)

# training data - https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
mypath <- c("C:\\courseproj_ml")
setwd(mypath)

mydest_train <- c("pml-training.csv")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile=m
ydest_train)
training <- read.csv(file=mydest_train, header=TRUE)

# test data - https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv
mydest_test <- c("pml-testing.csv")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile=my
dest_test)
testing <- read.csv(file=mydest_test, header=TRUE)
```

Preprocessing

Below are the steps for preprocessing, which consisted of determining which initial covariates (column names) had either no or a near-zero variance. Covariates with little variability contribute minimally to the model and including too many covariates may result in overfitting. Thus, the code chunks below remove all covariates that were TRUE for either “zerovar” (no variation) or “nzv” (near zero variation) from training and testing data.

```
# removing near zero covariates
# zerovar TRUE means that there is only one distinct value in the predictor
# nzv TRUE means that the predictor is a near zero variance predictor
# consider removing all zerovar = TRUE or nzv = TRUE ; the identifiers
# and "classe" are all still intact

nsv <- nearZeroVar(training, saveMetrics=TRUE)
nsvind <- mutate(nsv, include=!(nsv$zeroVar|nsv$nzv))
trainingnew <- training[,nsvind$include]

mynas <- matrix(0, nrow=1, ncol = dim(trainingnew)[2])
for (i in 1:dim(trainingnew)[2]){
  mynas[,i] <- sum(is.na(trainingnew[,i]))
}

#looks like columns 1:7, 11:26, 50:53, 57:62, 64:73, 86:88, 90 have 19216 NA values (out of 1962
2 obs)
removecols <- c(1:7, 11:26, 40, 50:53, 57:62, 64:73, 86:88, 90)
trainingnew2 <- trainingnew[,-removecols]

#remove the same columns from testing
testingnew <- testing[,nsvind$include]
testingnew2 <- testingnew[,-removecols]
```

Partitioning for Validation

In order to explore the efficacy of various models without using the data set aside ultimately for “testing”, it was necessary to create a “validation” set from the “training” data. The script for splitting the allocated “training” data into 60% training and 40% validation data is shown below.

```
#partition training set into training/test sets for cross validation (to decide
# what type of model to use, based on accuracy)
set.seed(234)
inTrain <- createDataPartition(trainingnew2$classe, p=0.6)[[1]]
validation <- trainingnew2[-inTrain,]
trainingnew3 <- trainingnew2[inTrain,]
trainingnew2 <- trainingnew3

dim(validation)
```

```
## [1] 7846   52
```

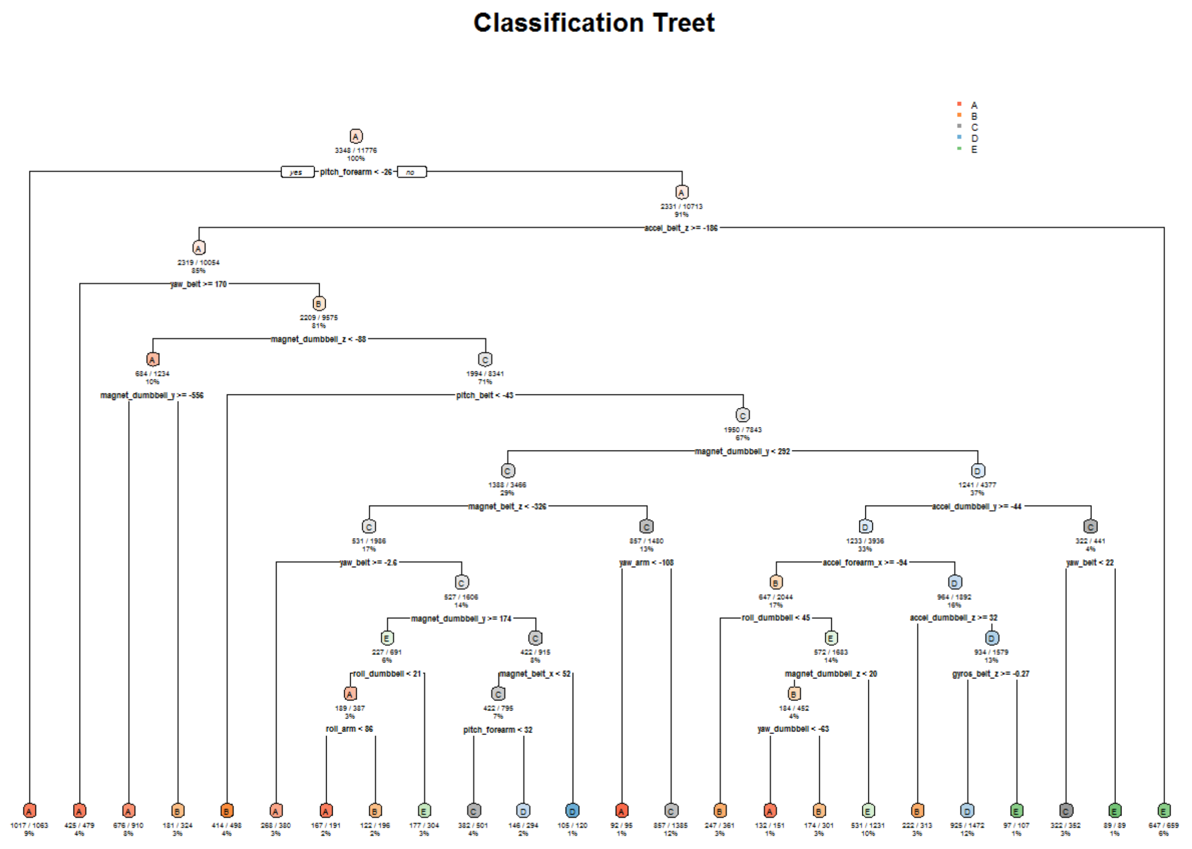
```
dim(trainingnew2)
```

```
## [1] 11776    52
```

Using Classification Trees

The chunk of code below generates the prediction model using classification trees. After the model is generated from the “training” data, the model was used to predict the exercise class (“classe”) on the “validation” data. There is also code to generate a Confusion Matrix to summarize various accuracy parameters of the prediction model, but the results are displayed in a subsequent section of this report.

```
# trees - accuracy 71%
mytree <- rpart(classe ~. ,method="class", data=trainingnew2)
predtree <- predict(mytree, validation, type="class")
conftree <- confusionMatrix(predtree, validation$classe)
# predict on testing data
predtreetesting <- predict(mytree, testingnew2, type="class")
rpart.plot(mytree,main="Classification Treet", extra=102, under=TRUE, faclen=0)
```



Using Random Forest

The chunk of code below generates the prediction model using random forests. After the model is generated from the “training” data, the model was used to predict the exercise class (“classe”) on the “validation” data. There is also code to generate a Confusion Matrix to summarize various accuracy parameters of the prediction model, but the results are displayed in a subsequent section of this report.

```
# random forest - accuracy 99%
set.seed(234)
library(randomForest)
myrf <- randomForest(classe ~., data=trainingnew2)
predrf <- predict(myrf, validation)
confrf <- confusionMatrix(predrf, validation$classe)
# predict on testing data
predrftesting <- predict(myrf, testingnew2)
```

Using Boosting

The chunk of code below generates the prediction model using boosting. After the model is generated from the “training” data, the model was used to predict the exercise class (“classe”) on the “validation” data. There is also code to generate a Confusion Matrix to summarize various accuracy parameters of the prediction model, but the results are displayed in a subsequent section of this report.

```
# boosting - accuracy 96%
set.seed(234)
mygbm <- train(classe ~., method="gbm", data=trainingnew2, verbose=FALSE,
trControl=trainControl(method="cv", number=3))
predgbm <- predict(mygbm, validation)
confgbm <- confusionMatrix(predgbm, validation$classe)
# predict on testing data
predgbmtesting <- predict(mygbm, testingnew2)
```

Comparing Confusion Matrices

This section displays the Confusion Matrices for Classification Trees, Random Forest, and Boosting, respectively. It can be seen that the Random Forest model has the highest accuracy (99%).

```
# confusion matrix - trees
conftree
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1835  198   14   28   95
##          B  115  906   75  155  115
##          C   25  122 1015  180   82
##          D  156  139   60  781   92
##          E  101  153  204  142 1058
## 
## Overall Statistics
## 
##                Accuracy : 0.7131
##                  95% CI : (0.703, 0.7231)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.6375
##  Mcnemar's Test P-Value : < 2.2e-16
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8221   0.5968   0.7420  0.60731   0.7337
## Specificity            0.9403   0.9273   0.9369  0.93186   0.9063
## Pos Pred Value         0.8456   0.6633   0.7128  0.63599   0.6381
## Neg Pred Value         0.9301   0.9056   0.9450  0.92369   0.9379
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2339   0.1155   0.1294  0.09954   0.1348
## Detection Prevalence   0.2766   0.1741   0.1815  0.15651   0.2113
## Balanced Accuracy      0.8812   0.7621   0.8394  0.76958   0.8200
```

```
# confusion matrix - random forest
confrf
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    6    0    0    0
##          B    0 1510    7    0    0
##          C    0    2 1361   23    0
##          D    0    0    0 1263    4
##          E    0    0    0    0 1438
## 
## Overall Statistics
## 
##                Accuracy : 0.9946
##                  95% CI : (0.9928, 0.9961)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9932
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9947   0.9949   0.9821   0.9972
## Specificity            0.9989   0.9989   0.9961   0.9994   1.0000
## Pos Pred Value         0.9973   0.9954   0.9820   0.9968   1.0000
## Neg Pred Value         1.0000   0.9987   0.9989   0.9965   0.9994
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1925   0.1735   0.1610   0.1833
## Detection Prevalence   0.2852   0.1933   0.1767   0.1615   0.1833
## Balanced Accuracy      0.9995   0.9968   0.9955   0.9908   0.9986
```

```
# confusion matrix - boosting
confgbm
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2207   77    0    1    4
##          B   16 1403   45    4   15
##          C    6   32 1293   49   14
##          D    2    0   23 1218   16
##          E    1    6    7   14 1393
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9577          
##                  95% CI : (0.953, 0.962)  
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9464          
##  Mcnemar's Test P-Value : 4.333e-11       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9888   0.9242   0.9452   0.9471   0.9660
## Specificity            0.9854   0.9874   0.9844   0.9938   0.9956
## Pos Pred Value         0.9642   0.9461   0.9275   0.9674   0.9803
## Neg Pred Value         0.9955   0.9819   0.9884   0.9897   0.9924
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2813   0.1788   0.1648   0.1552   0.1775
## Detection Prevalence   0.2917   0.1890   0.1777   0.1605   0.1811
## Balanced Accuracy      0.9871   0.9558   0.9648   0.9704   0.9808
```

Predicting on the Testing Data Using Random Forest

Below is the code chunk to display the “classe” results predicted on the 20 testing cases, based on the Random Forest model.

```
#choose random forest, and predict
predrftesting
```

```
##  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
```