Practical Machine Learning Course Project - Quantified Self Movement Data Analysis

English Garden

June 19, 2015

Introduction

Here we provide you an introduction of this assignment, quoted from the Practical Machine Learning course site.

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 is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise. More information about the data is available from this website. See the section on the Weight Lifting Exercise Dataset.

Get and Clean the Data

```r
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
```

Download the Data

```r
trainUrl <-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
trainFile <- "./training.csv"
testFile  <- "./testing.csv"

if (!file.exists(trainFile)) {
  download.file(trainUrl, destfile=trainFile, method="curl")
}
if (!file.exists(testFile)) {
  download.file(testUrl, destfile=testFile, method="curl")
}
```

Read the Data

After downloading the data from the source website, we read the two csv files into two data frames.

```
trainingRaw <- read.csv("./training.csv")
testingRaw <- read.csv("./testing.csv")
dim(trainingRaw)
```

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

```
dim(testingRaw)
```

```
## [1]  20 160
```

The training data set contains 19622 observations and 160 variables. The testing data set contains 20 observations and 160 variables. The "classe" variable in the training set is the outcome to predict.

Clean the data

First, we preprocess the training data set by removing the near zero variance predictors.

```
dataNZV <- nearZeroVar(trainingRaw, saveMetrics=TRUE)
nzvNames <- row.names(dataNZV[dataNZV$nzv!=FALSE,])
indexNZV<-names(trainingRaw) %in% nzvNames
trainingSet2 <- trainingRaw[!indexNZV]
```

Next, we select and remove the columns that have more than 60% NA entries, followed by removing the intermediate data.

```
dim(trainingSet2)
```

```
## [1] 19622   100
```

```
x<-NULL
for(i in 1:length(trainingSet2)) { #for every column in the training dataset
      if( sum( is.na( trainingSet2[, i] ) ) /nrow(trainingSet2) >= .6 )
            x<-c(x,i)
}
length(x)
```

```
## [1] 41
```

```
trainingSet3 <- trainingSet2[-x]
dim(trainingSet3)
```

```
## [1] 19622    59
```

```
trainingSet <- trainingSet3
rm(trainingSet3, trainingSet2)
```

Next, we remove the three columns that do not contribute much to the accelerometer measurements and perform the same cleaning operations on the testing data.

```
classe <- trainingSet$classe
trainRemove <- grepl("^X|timestamp|window", names(trainingSet))
trainingSet <- trainingSet[, !trainRemove]
trainCleaned <- trainingSet[, sapply(trainingSet, is.numeric)]
trainCleaned$classe <- classe
testRemove <- grepl("^X|timestamp|window", names(testingRaw))
testingRaw <- testingRaw[, !testRemove]
testCleaned <- testingRaw[, sapply(testingRaw, is.numeric)]
```

Now, we completed cleaning both the training and testing data sets. The cleaned training data set contains 19622 observations and 53 variables. The cleaned testing data set contains 20 observations and 53 variables. The “classe” variable is still in the cleaned training set; the last column of the testing data is `problem_id`.

Slice the data

Next, we split the cleaned training set into a training data set (70%) and a validation data set (30%). We reserve the validation data set for conducting cross validation in future steps.

```
set.seed(123456) # For reproducibile purpose
inTrain <- createDataPartition(y=trainCleaned$classe, p=0.7, list=FALSE)
trainingSet <- trainCleaned[inTrain, ]
testingSet <- trainCleaned[-inTrain, ]
dim(trainingSet)
```

```
## [1] 13737    53
```

```
dim(testingSet)
```

```
## [1] 5885   53
```

Data Modeling

We first build a **decision tree** for activity recognition, validate this model with the validation dataset, compute the confusion matrix as follows.

```
library(rpart)
fitDT <- rpart(classe ~ ., data=trainingSet, method="class")
predictionDT <- predict(fitDT, testingSet, type = "class")
confusionMatrix(predictionDT, testingSet$classe)
```

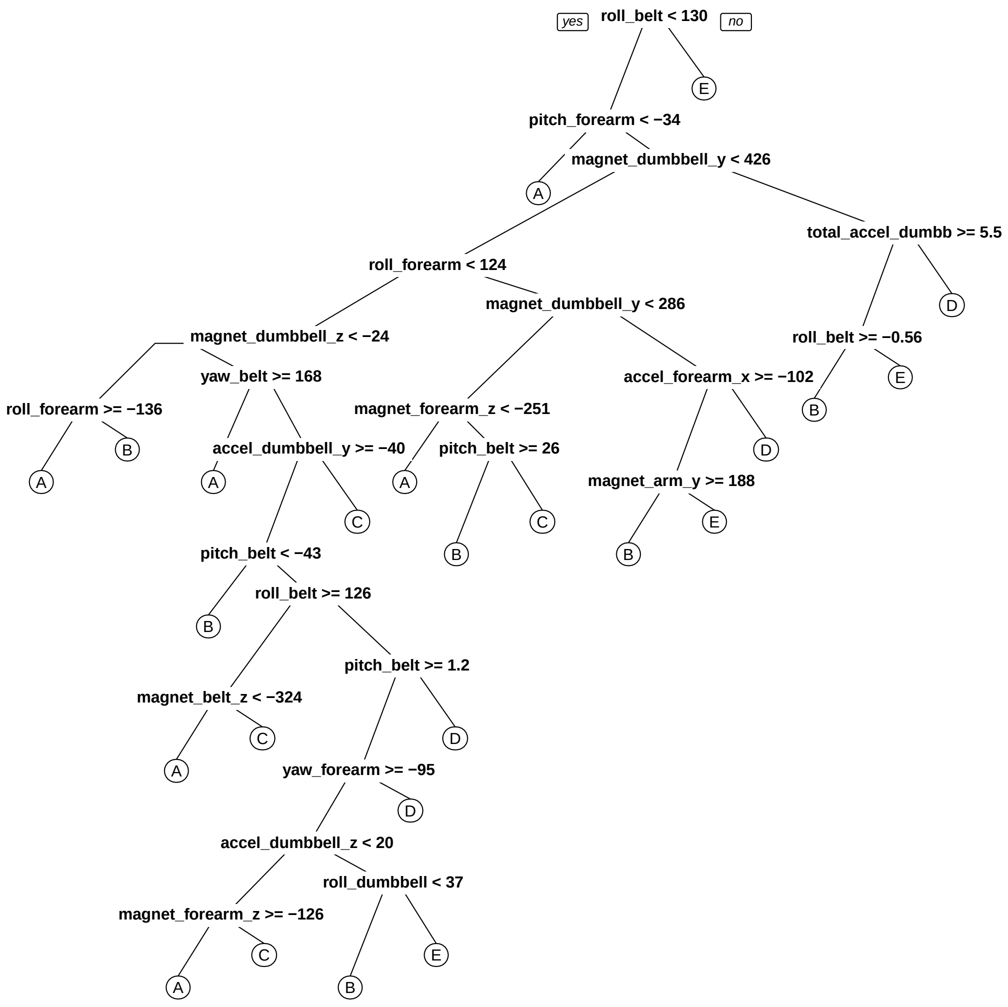
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1476  168   24   44   14
##          B   65  720  105   86  111
##          C   50  108  813  148  123
##          D   54   89   51  623   59
##          E   29   54   33   63  775
##
## Overall Statistics
##
##                Accuracy : 0.7489
##                  95% CI : (0.7376, 0.7599)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.682
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8817   0.6321   0.7924   0.6463   0.7163
## Specificity            0.9406   0.9227   0.9117   0.9486   0.9627
## Pos Pred Value         0.8552   0.6624   0.6546   0.7112   0.8124
## Neg Pred Value         0.9524   0.9127   0.9541   0.9319   0.9377
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2508   0.1223   0.1381   0.1059   0.1317
## Detection Prevalence   0.2933   0.1847   0.2110   0.1489   0.1621
## Balanced Accuracy      0.9112   0.7774   0.8521   0.7974   0.8395
```

Now to visualise the decision tree, we plot it here:

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

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

```
prp(fitDT)
```



We then fit a predictive model for activity recognition using **Random Forest** algorithm because it automatically selects important variables and is robust to correlated covariates and the outliers in general. We use **5-fold cross validation** when applying the algorithm. Please note this process can take a couple minutes to complete and be cautious of doubling the folds to 10 on a low-end computer hardware.

```
controlRf <- trainControl(method="cv", 5)
fitRF <- train(classe ~ ., data=trainingSet, method="rf", trControl=controlRf, ntree=250)
fitRF
```

```
## Random Forest
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
##
## Summary of sample sizes: 9421, 9420, 9421, 9422, 9420
##
## Resampling results across tuning parameters:
##
```

```
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9882811  0.9851744  0.004252211  0.005380315
##   27    0.9887908  0.9858194  0.003085007  0.003902986
##   52    0.9807233  0.9756145  0.002594969  0.003286314
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

We then estimate the performance of the model on the validation data set. As expected, Random Forests yields better results than the decision tree approach.

```
predictionRF <- predict(fitRF, testingSet)
confusionMatrix(predictionRF, testingSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231   15    0    0    0
##          B    1 1495    7    1    1
##          C    0    8 1355   23    3
##          D    0    0    6 1262    3
##          E    0    0    0    0 1435
##
## Overall Statistics
##
##                Accuracy : 0.9913
##                  95% CI : (0.989, 0.9933)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.989
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9848   0.9905   0.9813   0.9951
## Specificity            0.9973   0.9984   0.9948   0.9986   1.0000
## Pos Pred Value         0.9933   0.9934   0.9755   0.9929   1.0000
## Neg Pred Value         0.9998   0.9964   0.9980   0.9963   0.9989
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1905   0.1727   0.1608   0.1829
## Detection Prevalence   0.2863   0.1918   0.1770   0.1620   0.1829
## Balanced Accuracy      0.9984   0.9916   0.9926   0.9900   0.9976
```

```
accuracy <- postResample(predictionRF, testingSet$classe)
accuracy
```

```
##  Accuracy     Kappa
## 0.9913332 0.9890352
```

```
estimate_error <- 1 - as.numeric(confusionMatrix(testingSet$classe, predictionRF)$overall[1])
estimate_error
```

```
## [1] 0.008666837
```

So, the estimated accuracy of the model is 0.9913332 and the estimated out-of-sample error is 0.0086668.

Predicting for Test Data Set

Finally, we apply the random forests model to the original testing data set downloaded from the data source, after moving the `problem_id` column first.

```
result <- predict(fitRF, testCleaned[, -length(names(testCleaned))])
result
```

```
##  [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
```

Generating files for submission

We use the following function to generate files with predictions to submit for assignment.

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