Practical Machine Learning Course Project

Abel Mendez S.

6/18/2016

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 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://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

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.

Project Goal

The goal of this project is to predict the manner in which the subjects did the exercise. This is the "classe" variable in the training set. I will use some of the other variables to predict the outcome of this particular variable.

In this report, I describe how I built the prediction model, the use of cross validation, the expected sample error, and explain the choices that I made for the final fitted model. Lastly, I will use the built model to predict 20 different test cases, and report the results.

Loading and Cleaning the Data

Getting the Data

I am loading the data directly from the URLs noted above. The identifying information (`user_name`, `raw_timestamp_part_1`, `raw_timestamp_part_2`, `cvtd_timestamp`, `new_window`, and `num_window`) is ommitted from the data, as these are not features which I'll use for the prediction models.

The values that contain the strings `NA`,`""`, and `#DIV/0!` are set to NA (missing).

```r
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

download.file( trainURL, "pml-training.csv" )
download.file( testURL, "pml-testing.csv" )

train.data <- read.csv( "pml-training.csv", na.strings = c( "", "NA", "#DIV/0!" ) )[,-c(1:7)]
test.data  <- read.csv( "pml-testing.csv", na.strings = c( "", "NA", "#DIV/0!" ) )[,-c(1:7)]
```

Cleaning the data

Some of the variables in the dataset may have little variance (for instance, they may have just a single value, or have more than one value, but one of these is contained in the vast majority of the cases, say 95% of them). Likewise, variables that have a large amount of missing values (say, morenthan 90%) will provide small value for prediction. I have taken these variables out of the training data, as they are most likely poor predictors.

```r
# Removing near zero variance columns
removeCols <- nearZeroVar( train.data )

trd <- train.data[ , -c( removeCols )]
tsd <- test.data[, -c( removeCols )]

# removing columns with high rate of NAs
N <- nrow(trd)
threshold_NA_rate <- 0.9

l <- apply(is.na( trd ),2, function(x) { sum(x) / N } )

removeCols2 <- which( as.numeric( l > threshold_NA_rate ) * seq( 1, length(colnames(trd))) > 0 )

trd <- trd[, -c(removeCols2)]
tsd <- tsd[, -c(removeCols2)]
```

Creating Data Partitions

We'll use a fixed seed for the data partitions. We'll use 70% of the train data for training of the models, and the remaining 30% for validation.

```r
set.seed(12345)

training <- createDataPartition( trd$classe, p=0.7, list=FALSE)
this.train <- trd[ training, ]
this.test  <- trd[ -training,]
```

Fitting Prediction Models

There are many machine learning algorithms. Some perform better than others, depending on the data. For this project, I am choosing to create three predictive algorithms: Decision Trees, LogitBoost, and Random

Forest. I use the in-sample training data for the model training, and the in-samle test data for validation. For each of the models, I am showing the confusion matrix and related statistics.

Decision Trees

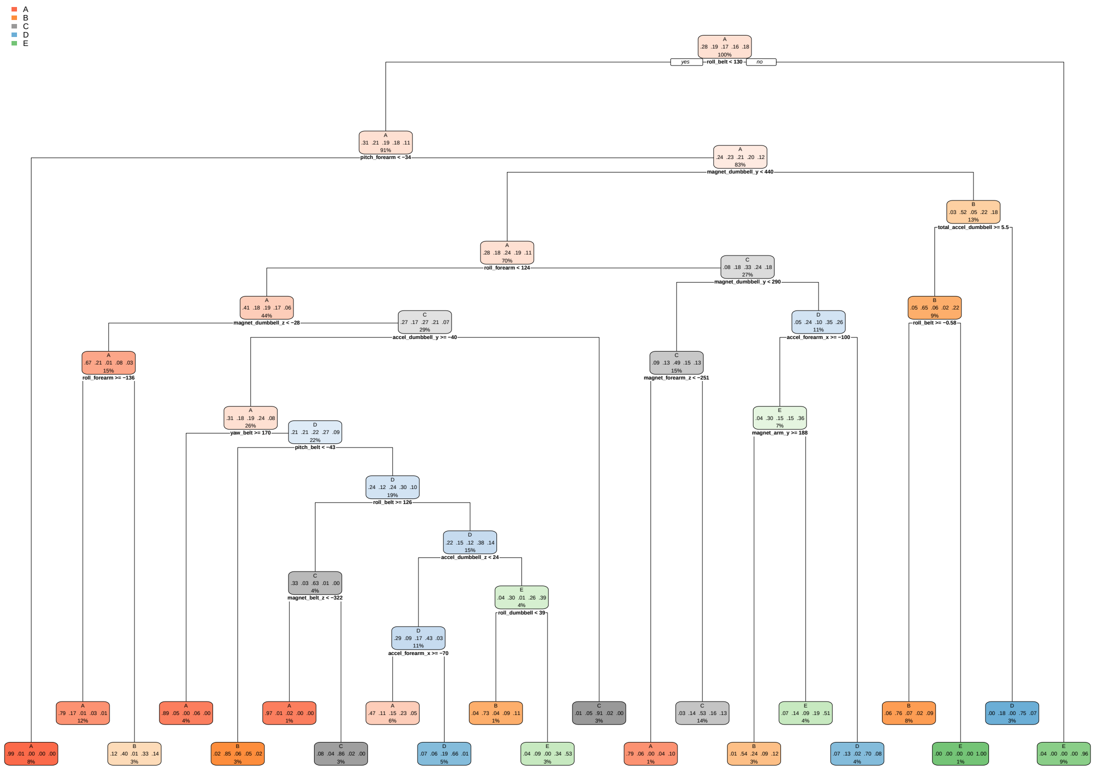
This type of model creates a decision tree which most matches the training data. The decision tree consists of *interior nodes* and *leaf nodes*. The *interior nodes* correspond to an input variable from the training data. Each *interior node* has one *leaf node* which represents the value of the predicted value given the values of the predictor variables in its path.

We use the `rpart` library when creating this model

```
dt_model <- rpart(classe ~ ., data=this.train, method="class")

# Predicting
dt_predict <- predict(dt_model, this.test, type = "class")

# Plotting
rpart.plot(dt_model)
```



```
# Testing results on myTesting data set
dt_confusion <- confusionMatrix(dt_predict, this.test$classe)
print(dt_confusion$table)
```

```
##            Reference
## Prediction    A    B    C    D    E
##          A 1498  196   69  106   25
##          B   42  669   85   86   92
##          C   43  136  739  129  131
##          D   33   85   98  553   44
##          E   58   53   35   90  790
```

```
cat("Accuracy:",dt_confusion$overall[[1]],"\n")
```

```
## Accuracy: 0.7220051
```

LogitBoost

Boosting is a way of combining a number of "weaker" prediction models into a more robust model, by using some sort of weighting method, such that the combined (linear additive) model yields better results than any of the component models. LogitBoost uses an iterative weighting process such that the number of weights (and thus input variables) is trimmed, without much loss of classification power, with an associated increase in computation speed.

```
tc <- trainControl( verboseIter=FALSE , preProcOptions="pca", allowParallel=TRUE)
loggrid <- expand.grid(nIter = 40)

lg_model <- train(classe ~ ., data = this.train, method = "LogitBoost", trControl = tc, tuneGrid =loggri
```

```
## Loading required package: caTools
```

```
lg_predict <- predict( lg_model, this.test )

# Testing results on myTesting data set
lg_confusion <- confusionMatrix( lg_predict, this.test$classe)
print(lg_confusion$table)
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1536  104   13   19    4
##          B   21  760   54    1   33
##          C    9   58  692   31   19
##          D    7   33   71  784   47
##          E    8    8    4    5  860
```

```
cat("Accuracy:",lg_confusion$overall[[1]],"\n")
```

```
## Accuracy: 0.8940359
```

Random Forest

```
rf_model <- randomForest(classe ~. , data=this.train, importance=TRUE)
rf_predict <- predict( rf_model, this.test, type = "class")
rf_confusion <- confusionMatrix( rf_predict, this.test$classe)
print(rf_confusion$table)
```

```
##            Reference
## Prediction    A    B    C    D    E
##          A 1673   11    0    0    0
##          B    1 1123    9    0    0
##          C    0    5 1016   14    0
##          D    0    0    1  950    5
##          E    0    0    0    0 1077
```

```
cat("Accuracy:",rf_confusion$overall[[1]],"\n")
```

```
## Accuracy: 0.9921835
```

Conclusion

Comparing the accuracy of the three models, it is clear that the Random Forest prediction model is the most accurate. In light of this, I will use this model against the out of sample test data. I will use the results of this prediction, as the output

```
final_predict <- predict( rf_model, tsd)

for(i in 1: length( final_predict) ){

    filename = paste("problem_id_",i,".txt", sep="")

    write.table( final_predict[i],
                 file=filename,
                 quote=FALSE,
                 row.names=FALSE,
                 col.names=FALSE)
    }
```