Practical Machine Learning

Prediction Assignment

Gregory Roberts

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

The objective, in this project, is to use the data to predict the manner in which the participants accomplished the exercises.

Data

Download and load the data.

```
setwd("C:\\Coursera\\PracMachLearn")

if(!file.exists("pml-training.csv")){
    download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", des
tfile = "./pml-training.csv")
}
if(!file.exists("pml-testing.csv")){
    download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destf
ile = "./pml-testing.csv")
}

trainingOriginal = read.csv("pml-training.csv", na.strings=c("", "NA", "NULL"))
testingOriginal = read.csv("pml-testing.csv", na.strings=c("", "NA", "NULL"))
#dim(trainingOriginal)
```

Load libraries

```
suppressWarnings(library(caret))
```

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

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

```
suppressWarnings(library(tree))
suppressWarnings(library(caret))
suppressWarnings(library(rattle))
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
suppressWarnings(suppressMessages(library(randomForest)))
```

Removing NA and extraneous data

Remove variables that have an excess number of NA values.

```
training_na <- trainingOriginal[ , colSums(is.na(trainingOriginal)) == 0]
dim(training_na)
```

```
## [1] 19622    60
```

Remove extraneous variables that are unrelated to dependent variable.

```
extraneous = c('X', 'user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2', 'cvtd_timest
amp', 'new_window', 'num_window')
training_ext <- training_na[, -which(names(training_na) %in% extraneous)]
dim(training_ext)
```

```
## [1] 19622    53
```

Check variables that have low variance.

```
nZV= nearZeroVar(training_ext[sapply(training_ext, is.numeric)], saveMetrics = TRUE)
training_nzv = training_ext[,nZV[, 'nzv']==0]
dim(training_nzv)
```

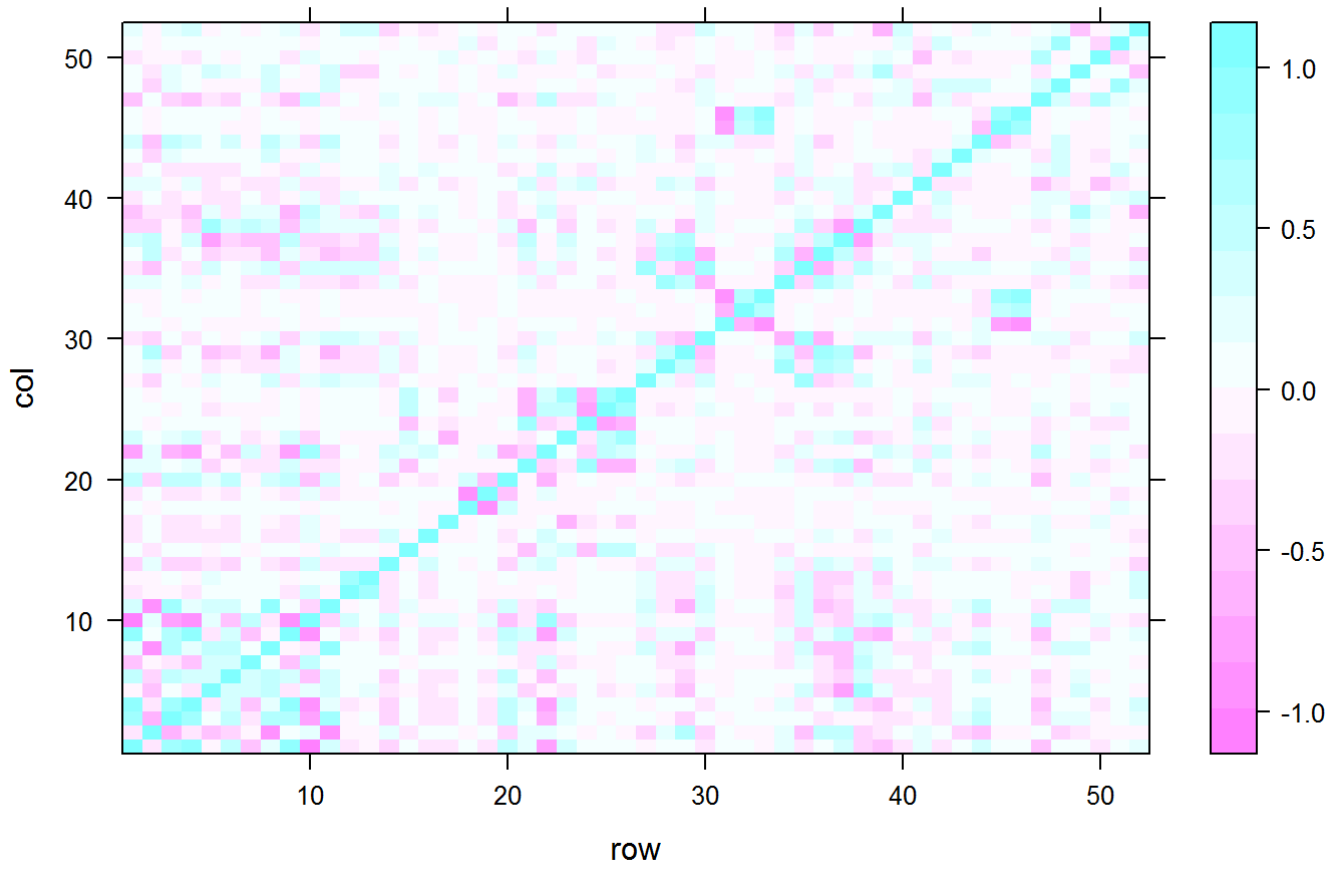
```
## [1] 19622    53
```

Removing correlated variables.

```
cor_matrix <- cor(na.omit(training_nzv[sapply(training_nzv, is.numeric)]))
dim(cor_matrix)
```

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

```
corr_dataframe <- expand.grid(row = 1:52, col = 1:52)
corr_dataframe$corr <- as.vector(cor_matrix)
levelplot(corr ~ row+ col, corr_dataframe)
```



Remove variables with high correlation.

```
remove_corr = findCorrelation(cor_matrix, cutoff = .90, verbose = TRUE)
```

```
## Compare row 10  and column  1 with corr  0.992
##   Means:  0.27 vs 0.168 so flagging column 10
## Compare row 1  and column  9 with corr  0.925
##   Means:  0.25 vs 0.164 so flagging column 1
## Compare row 9  and column  4 with corr  0.928
##   Means:  0.233 vs 0.161 so flagging column 9
## Compare row 8  and column  2 with corr  0.966
##   Means:  0.245 vs 0.157 so flagging column 8
## Compare row 19  and column  18 with corr  0.918
##   Means:  0.091 vs 0.158 so flagging column 18
## Compare row 46  and column  31 with corr  0.914
##   Means:  0.101 vs 0.161 so flagging column 31
## Compare row 46  and column  33 with corr  0.933
##   Means:  0.083 vs 0.164 so flagging column 33
## All correlations <= 0.9
```

```
training_corr = training_nzv[,-remove_corr]
dim(training_corr)
```

```
## [1] 19622    46
```

Returns 19622 samples and 46 variable.

Cross validate training and testing data

```
training_dataPart <- createDataPartition(y=training_corr$classe, p=0.7, list=FALSE)
training <- training_corr[training_dataPart,]; testing <- training_corr[-training_dataPart,]
dim(training)
```

```
## [1] 13737    46
```

```
dim(testing)
```

```
## [1] 5885   46
```

Returns 13737 samples and 46 variables for training, and 5885 samples and 46 variables for testing.

Analysis

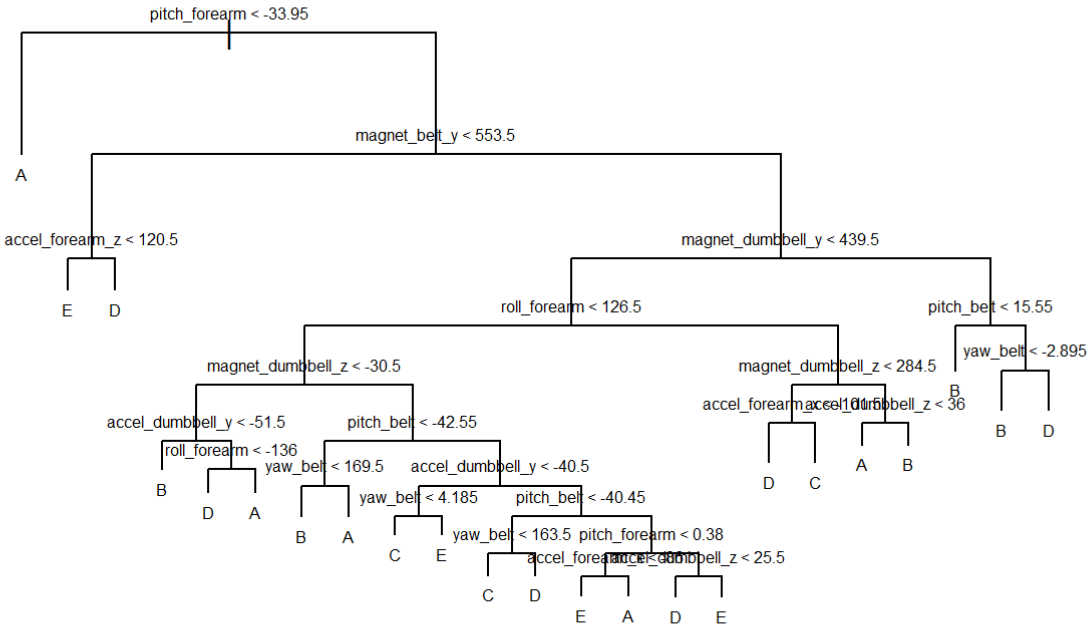
Regression Tree

Fit a tree to this data, summarize and plot it.

```
set.seed(12345)
tree_training = tree(classe~.,data=training)
summary(tree_training)
```

```
##
## Classification tree:
## tree(formula = classe ~ ., data = training)
## Variables actually used in tree construction:
##  [1] "pitch_forearm"     "magnet_belt_y"     "accel_forearm_z"
##  [4] "magnet_dumbbell_y" "roll_forearm"      "magnet_dumbbell_z"
##  [7] "accel_dumbbell_y"  "pitch_belt"        "yaw_belt"
## [10] "accel_forearm_x"   "accel_dumbbell_z"
## Number of terminal nodes:  23
## Residual mean deviance:  1.536 = 21060 / 13710
## Misclassification error rate: 0.3026 = 4157 / 13737
```

```
plot(tree_training)
text(tree_training,pretty=0, cex =.5)
```



The tree has an excess amount of branches that need to be pruned using “rpart” method.

Rpart method

```
tree_prune <- train(classe ~ .,method="rpart",data=training)
```

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

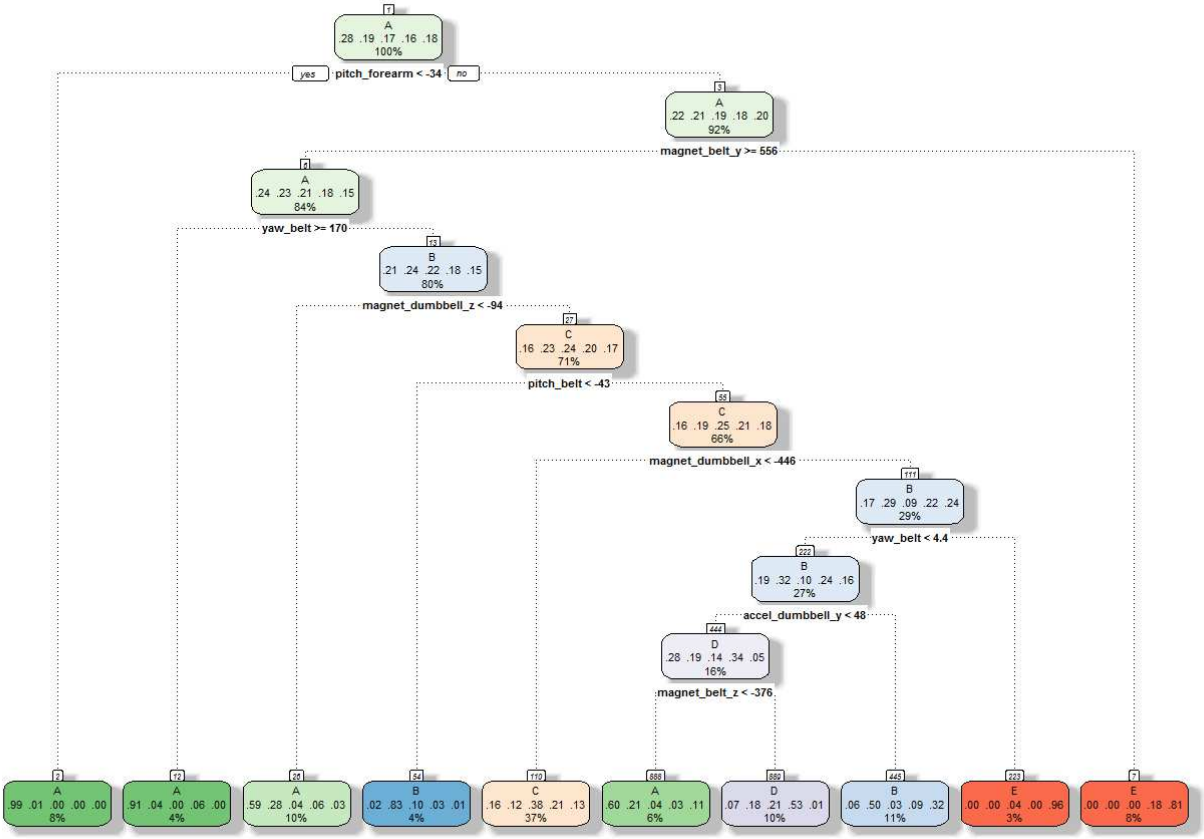
```
## Warning: package 'rpart' was built under R version 3.3.2
```

```
print(tree_prune$finalModel)
```

```
## n= 13737
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 13737 9831 A (0.28 0.19 0.17 0.16 0.18)
##     2) pitch_forearm< -33.95 1115    7 A (0.99 0.0063 0 0 0) *
##     3) pitch_forearm>=-33.95 12622 9824 A (0.22 0.21 0.19 0.18 0.2)
##       6) magnet_belt_y>=555.5 11589 8794 A (0.24 0.23 0.21 0.18 0.15)
##        12) yaw_belt>=169.5 561   51 A (0.91 0.036 0 0.055 0) *
##        13) yaw_belt< 169.5 11028 8400 B (0.21 0.24 0.22 0.18 0.15)
##          26) magnet_dumbbell_z< -94.5 1306  537 A (0.59 0.28 0.044 0.056 0.034) *
##          27) magnet_dumbbell_z>=-94.5 9722 7384 C (0.16 0.23 0.24 0.2 0.17)
##            54) pitch_belt< -42.95 593   98 B (0.02 0.83 0.1 0.03 0.01) *
##            55) pitch_belt>=-42.95 9129 6853 C (0.16 0.19 0.25 0.21 0.18)
##             110) magnet_dumbbell_x< -446.5 5095 3183 C (0.16 0.12 0.38 0.21 0.13) *
##             111) magnet_dumbbell_x>=-446.5 4034 2874 B (0.17 0.29 0.09 0.22 0.24)
##               222) yaw_belt< 4.42 3646 2486 B (0.19 0.32 0.095 0.24 0.16)
##                 444) accel_dumbbell_y< 48.5 2178 1440 D (0.28 0.19 0.14 0.34 0.048)
##                   888) magnet_belt_z< -376.5 843  333 A (0.6 0.21 0.036 0.034 0.11) *
##                   889) magnet_belt_z>=-376.5 1335  626 D (0.072 0.18 0.21 0.53 0.0075) *
##                 445) accel_dumbbell_y>=48.5 1468  732 B (0.057 0.5 0.029 0.091 0.32) *
##               223) yaw_belt>=4.42 388   16 E (0 0 0.041 0 0.96) *
##       7) magnet_belt_y< 555.5 1033  193 E (0.0029 0.0029 0.00097 0.18 0.81) *
```

Prettier plots

```
fancyRpartPlot(tree_prune$finalModel)
```



The results are close to the “tree” package.

Cross Validation

Use cross validation to check the performance of the tree.

```
tree_predict = predict(tree_training,testing,type="class")
pred_matrix = with(testing,table(tree_predict,classe))
sum(diag(pred_matrix))/sum(as.vector(pred_matrix)) # error rate
```

```
## [1] 0.6849618
```

This is not very accurate

```
tree_predict = predict(tree_prune,testing)
pred_matrix = with(testing,table(tree_predict,classe))
sum(diag(pred_matrix))/sum(as.vector(pred_matrix)) # error rate
```

```
## [1] 0.575531
```

The result from the “caret” package is lower.

Pruning tree

The tree is to large and requires cross validation to prune it.

```
cv.training = cv.tree(tree_training,FUN=prune.misclass)
cv.training
```

```
## $size
##  [1] 23 22 21 20 18 17 16 15 12 10  7  5  1
##
## $dev
##  [1] 4499 4572 4578 4637 5130 5372 5442 5462 6027 6027 6715 7282 9831
##
## $k
##  [1]     -Inf  72.0000  74.0000  95.0000 133.0000 147.0000 149.0000
##  [8] 159.0000 182.3333 189.0000 229.6667 272.5000 638.2500
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Evaluate pruned tree on test data.

```
prune_training = prune.misclass(tree_training,best=18)
tree_predict = predict(prune_training,testing,type="class")
pre_matrix = with(testing,table(tree_predict,classe))
sum(diag(pre_matrix))/sum(as.vector(pre_matrix)) # error rate
```

```
## [1] 0.6513169
```

Pruning produces a lower result and gives us a simpler tree.

Try Random Forest to improve accuracy.

Random Forests

Random Forests

Use of random forests tends to build a lot of bushy trees. Then they are averaged to reduce the variance.

```
require(randomForest)
set.seed(12345)
```

Trying a random forest, to see how well it performs.

```
training_ranForest = randomForest(classe~.,data=training,ntree=100, importance=TRUE)
training_ranForest
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training, ntree = 100,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0.7%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3898    5    2    0    1 0.002048131
## B   18 2630   10    0    0 0.010534236
## C    0   16 2376    4    0 0.008347245
## D    0    0   28 2222    2 0.013321492
## E    0    0    1    9 2515 0.003960396
```

Out-of Sample Accuracy

The model shows OOB estimate of .73% error rate.

```
tree_predict = predict(training_ranForest,testing,type="class")
pred_matrix = with(testing,table(tree_predict,classe))
sum(diag(pred_matrix))/sum(as.vector(pred_matrix)) # error rate
```

```
## [1] 0.9928632
```

0.99 means we got a very accurate estimate.

Conclusion

Now we can predict the testing data from the website.

```
answers <- predict(training_ranForest, testingOriginal)
answers
```

```
##  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
```

Those answers are going to submit to website for grading. It shows that this random forest model did a good job.