Practical Machine Learning

Toni Rosati

April 23, 2016

Rpubs

http://rpubs.com/Toni/174529 (http://rpubs.com/Toni/174529)

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).

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

Data

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). If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv) The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

```
#training dataset
whole_train<-read.csv("pml-training.csv", na.strings=c("NA","#DIV/0!",""))
#testing dataset
testdata<-read.csv("pml-testing.csv", na.strings=c("NA","#DIV/0!",""))
```

Cleaning the Data

It is unnecessary to have data that are precalculated, so I have removed the columns labeled “max_”, “min_”, “kurtosis”, “skewness”, “stddev”, “var_”, “avg_”, and “amplitude”. This is done for both the training set and the testing set.

```
#remove anything that says max_ min_ kurtosis skewness stddev var_ avg_ amplitude
train<-whole_train[,c(8:11,37:49,60:68,84:86,102,113:124,140,151:160)]
test<-testdata[,c(8:11,37:49,60:68,84:86,102,113:124,140,151:160)]
```

Creating Training Subsets

Since the training dataset is large enough, I've subset it into a training and a probe set. This will allow me to validate accuracy of the model before applying it to the test set.

```
inTrainPart<- createDataPartition(y=train$classe, p=.5, list=FALSE)
train1<-train[inTrainPart,]
train2<-train[-inTrainPart,]
#The two subsets have equivalent Classe
summary(train1$classe)
```

```
##    A    B    C    D    E
## 2790 1899 1711 1608 1804
```

```
summary(train2$classe)
```

```
##    A    B    C    D    E
## 2790 1898 1711 1608 1803
```

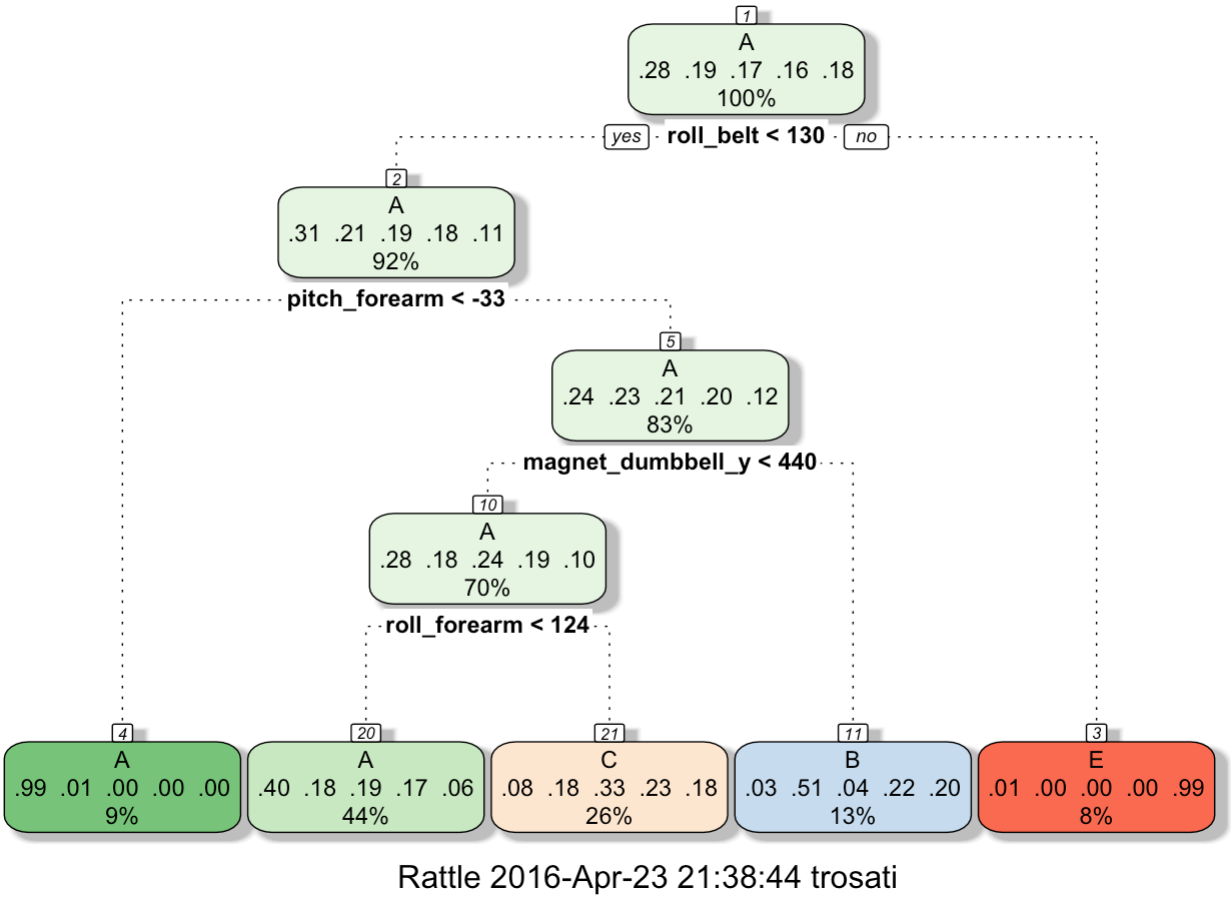
Fit a model using Recursive Partitioning and Regression Trees (r_part)

Based on the Coursera forums, I've opted to build models using rpart and random forest. I'm starting with rpart because the computational time is much shorter.

```
set.seed(459)
fit1_rpart<-train(classe~., data=train1, method="rpart")
fit1_rpart
```

```
## CART
##
## 9812 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 9812, 9812, 9812, 9812, 9812, 9812, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa       Accuracy SD  Kappa SD
##   0.03574480  0.5035986  0.35047809  0.03112679   0.04923510
##   0.05985949  0.4090051  0.19540278  0.05950623   0.10046277
##   0.11734549  0.3140128  0.04438142  0.03831717   0.06043108
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0357448.
```

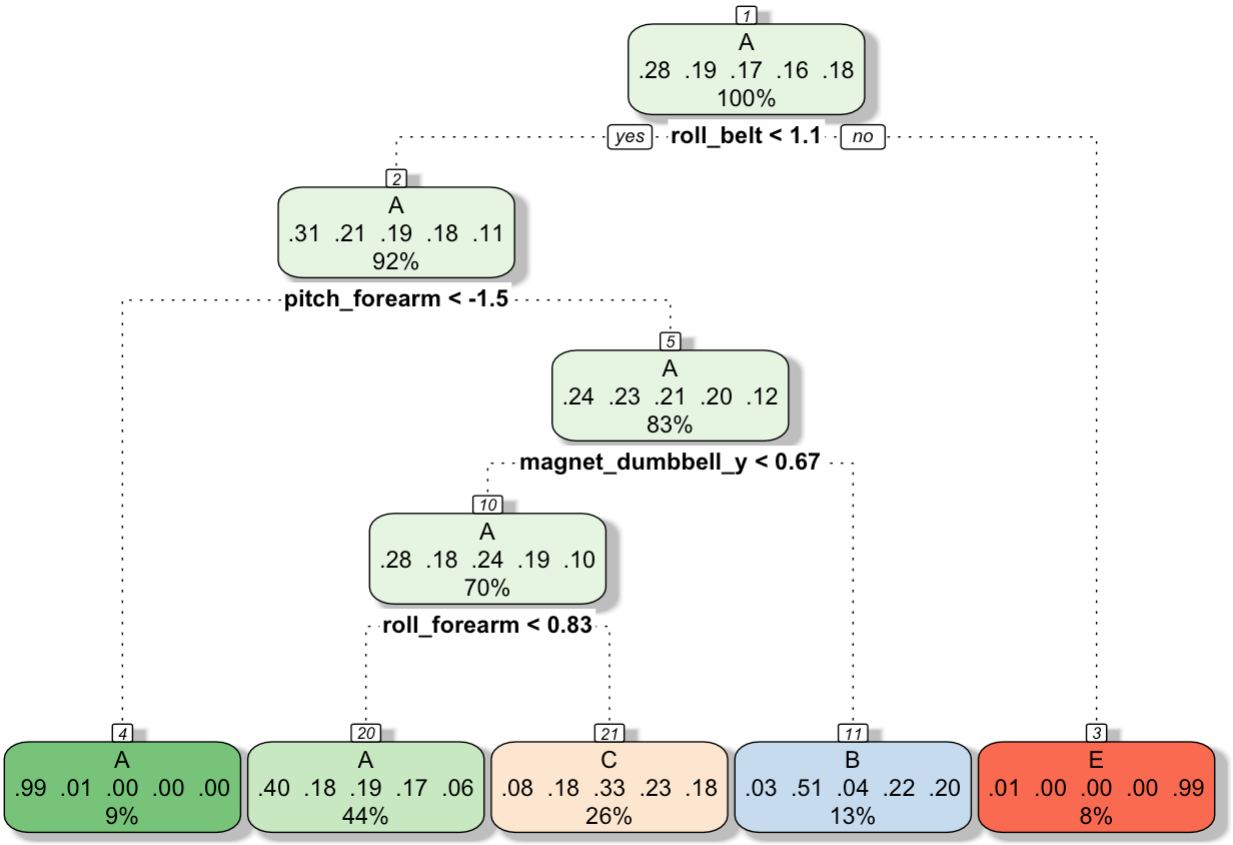
```
fancyRpartPlot(fit1_rpart$finalModel)
```



```
fit2_rpart <- train(train1$classe ~ .,  preProcess=c("center", "scale"), data = train1, method="rpa
rt")
print(fit2_rpart, digits=3)
```

```
## CART
##
## 9812 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 9812, 9812, 9812, 9812, 9812, 9812, ...
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.0357  0.510     0.3614  0.0373       0.0589
##   0.0599  0.407     0.1942  0.0605       0.1006
##   0.1173  0.330     0.0693  0.0405       0.0627
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0357.
```

```
fancyRpartPlot(fit2_rpart$finalModel)
```

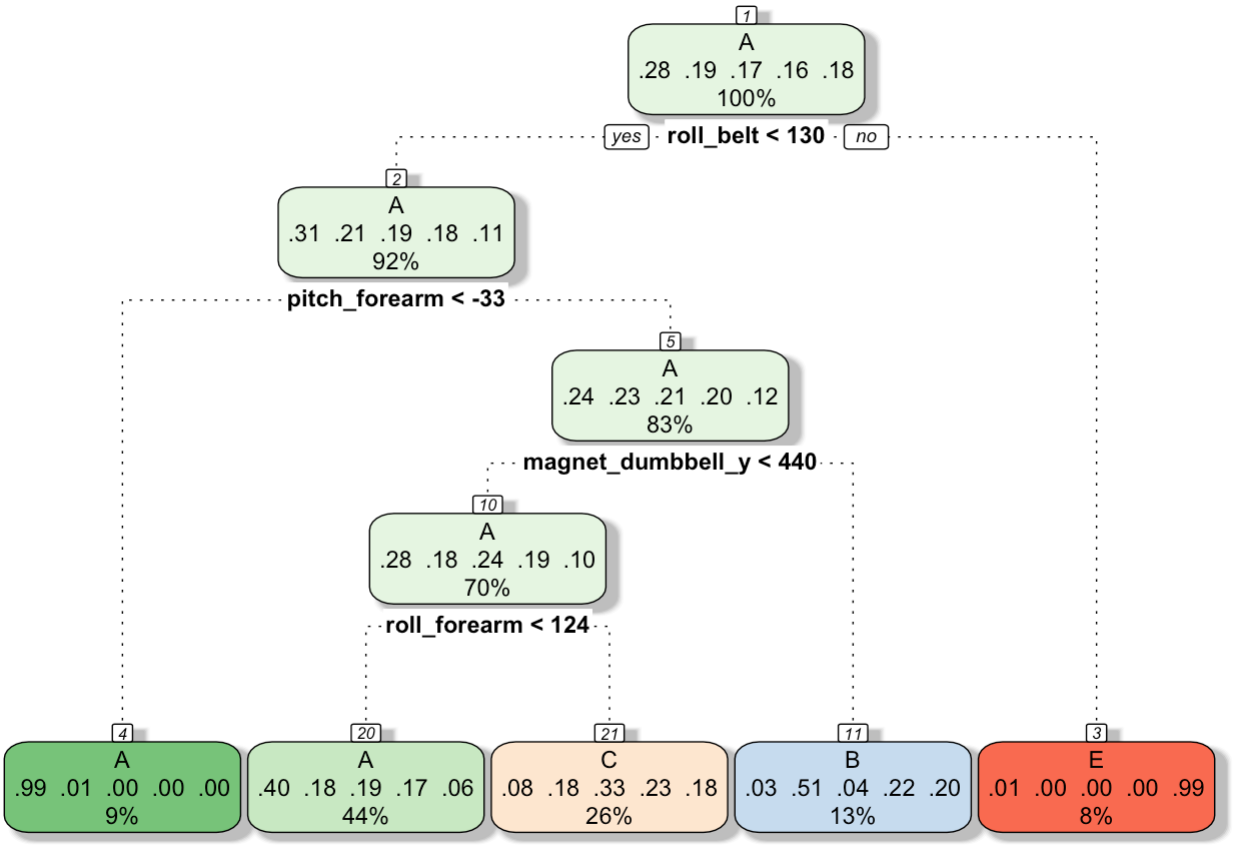


Rattle 2016-Apr-23 21:39:00 trosati

```
fit3_rpart <- train(classe ~ .,  trControl=trainControl(method = "cv", number = 4), data = train1,
method="rpart")
print(fit3_rpart, digits=3)
```

```
## CART
##
## 9812 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 7358, 7358, 7361, 7359
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.0357  0.506     0.3539  0.0105       0.0135
##   0.0599  0.430     0.2309  0.0724       0.1211
##   0.1173  0.323     0.0593  0.0449       0.0685
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0357.
```

```
fancyRpartPlot(fit3_rpart$finalModel)
```



Rattle 2016-Apr-23 21:39:05 trosati

Fit a Random Forest Model

Based on the Coursera forums, I've opted to build models using rpart and random forest.

```
set.seed(459)
fit1_rf<-train(classe~., data=train1, method="rf")
print(fit1_rf, digits=3)
```

```
## Random Forest
##
## 9812 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 9812, 9812, 9812, 9812, 9812, 9812, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.982     0.977  0.00332      0.00421
##   27    0.983     0.979  0.00293      0.00370
##   52    0.973     0.966  0.00491      0.00621
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

Model Selection (predict on train2) Cross Validation

```
predict_train1v2 <- predict(fit1_rpart, newdata=train2)
print(confusionMatrix(predict_train1v2, train2$classe), digits=2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2536  810  790  705  271
##          B   41  647   54  297  232
##          C  204  441  867  606  498
##          D    0    0    0    0    0
##          E    9    0    0    0  802
##
## Overall Statistics
##
##                Accuracy : 0.49
##                  95% CI : (0.48, 0.5)
##     No Information Rate : 0.28
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.34
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity              0.91    0.341    0.507     0.00    0.445
## Specificity              0.63    0.921    0.784     1.00    0.999
## Pos Pred Value           0.50    0.509    0.331      NaN    0.989
## Neg Pred Value           0.95    0.853    0.883     0.84    0.889
## Prevalence               0.28    0.193    0.174     0.16    0.184
## Detection Rate           0.26    0.066    0.088     0.00    0.082
## Detection Prevalence     0.52    0.130    0.267     0.00    0.083
## Balanced Accuracy        0.77    0.631    0.645     0.50    0.722
```

```
#sensitivity and specificity are terrible

predict_train1v2_2 <- predict(fit2_rpart, newdata=train2)
print(confusionMatrix(predict_train1v2_2, train2$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2536  810  790  705  271
##          B   41  647   54  297  232
##          C  204  441  867  606  498
##          D    0    0    0    0    0
##          E    9    0    0    0  802
##
## Overall Statistics
##
##                Accuracy : 0.4946
##                  95% CI : (0.4847, 0.5045)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3394
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9090  0.34089  0.50672   0.0000  0.44481
## Specificity            0.6330  0.92113  0.78405   1.0000  0.99888
## Pos Pred Value         0.4961  0.50905  0.33142      NaN  0.98890
## Neg Pred Value         0.9459  0.85350  0.88268   0.8361  0.88877
## Prevalence             0.2844  0.19348  0.17441   0.1639  0.18379
## Detection Rate         0.2585  0.06595  0.08838   0.0000  0.08175
## Detection Prevalence   0.5211  0.12956  0.26667   0.0000  0.08267
## Balanced Accuracy      0.7710  0.63101  0.64538   0.5000  0.72185
```

```
#still not great

predict_train1v2_3 <- predict(fit3_rpart, newdata=train2)
print(confusionMatrix(predict_train1v2_3, train2$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2536  810  790  705  271
##          B   41  647   54  297  232
##          C  204  441  867  606  498
##          D    0    0    0    0    0
##          E    9    0    0    0  802
##
## Overall Statistics
##
##                Accuracy : 0.4946
##                  95% CI : (0.4847, 0.5045)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3394
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9090  0.34089  0.50672   0.0000  0.44481
## Specificity            0.6330  0.92113  0.78405   1.0000  0.99888
## Pos Pred Value         0.4961  0.50905  0.33142      NaN  0.98890
## Neg Pred Value         0.9459  0.85350  0.88268   0.8361  0.88877
## Prevalence             0.2844  0.19348  0.17441   0.1639  0.18379
## Detection Rate         0.2585  0.06595  0.08838   0.0000  0.08175
## Detection Prevalence   0.5211  0.12956  0.26667   0.0000  0.08267
## Balanced Accuracy      0.7710  0.63101  0.64538   0.5000  0.72185
```

```
#even worse

predict_train1v2_rf <- predict(fit1_rf, newdata=train2)
print(confusionMatrix(predict_train1v2_rf, train2$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2784   20    0    0    1
##          B    4 1868    7    3    4
##          C    0    9 1693   23    7
##          D    0    1   11 1580    8
##          E    2    0    0    2 1783
##
## Overall Statistics
##
##                Accuracy : 0.9896
##                  95% CI : (0.9874, 0.9915)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9868
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9978   0.9842   0.9895   0.9826   0.9889
## Specificity            0.9970   0.9977   0.9952   0.9976   0.9995
## Pos Pred Value         0.9925   0.9905   0.9775   0.9875   0.9978
## Neg Pred Value         0.9991   0.9962   0.9978   0.9966   0.9975
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2838   0.1904   0.1726   0.1611   0.1818
## Detection Prevalence   0.2859   0.1923   0.1766   0.1631   0.1822
## Balanced Accuracy      0.9974   0.9910   0.9923   0.9901   0.9942
```

```
#results are fantastic!
```

Prediction on Testing Set

```
predictions <- predict(fit1_rf, test[,-53])
predictions
```

```
##  [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
```

Expected Out of Sample Error

The expected error is 1.03% for the Random Forest model.

```
print(fit1_rf$finalModel, digits=3)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 1.23%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 2782    5    3    0    0 0.002867384
## B   26 1862   11    0    0 0.019483939
## C    0   19 1683    9    0 0.016364699
## D    0    2   27 1578    1 0.018656716
## E    0    4    5    9 1786 0.009977827
```

Conclusion

While Recursive Partitioning and Regression Trees (r_part) allowed for the display of a Fancy Plot and processed much quicker, it did not provide nearly the level of accuracy that Random Forrest could.