Prediction

Parthasarathy Krishnamurthy

About the Project

The main aim of this project is to quantify how well people perform certain exercise activities. We use machine learning to predict the manner in which they did exercise. 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. We use data from accelorometer on the belt, forearm, arm and dumbbell of 6 participants.

Required R libraries

```
library(caret)
```

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

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

```
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

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

```
library(knitr)
library(doMC)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

Getting and loading data

> The data for this project was provided by http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har).

```
trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

training <- read.csv(url(trainUrl), na.strings=c("NA","#DIV/0!",""))
testing <- read.csv(url(testUrl), na.strings=c("NA","#DIV/0!",""))
```

Cleaning data

In order for the machine learning algorithms to work, we have to remove features which have null values. For this lets find null values percent in each column and we will subset the data.

```
naVal <- round(colMeans(is.na(training)), 2)
index <- which(naVal==0)[-1]
training <- training[, index]
testing <- testing[, index]
```

Also the 1st 6 columns are not very useful for our analysis. Also making all columns as numeric.

```
training <- training[, -(1:6)]
testing <- testing[, -(1:6)]
for(i in 1:(length(training)-1)){
    training[,i] <- as.numeric(training[,i])
    testing[,i] <- as.numeric(testing[,i])
}
```

Cross validation (Partitioning data)

The training data is to be partioned into 2 parts: data to train a model and data to test the model. I prefer partitioning at 60% for training and remaining for testing.

```
inTrain <- createDataPartition(training$classe, p=0.6, list=FALSE)
myTraining <- training[inTrain, ]
myTesting <- training[-inTrain, ]
dim(myTraining); dim(myTesting)
```

```
## [1] 11776    53
```

```
## [1] 7846   53
```

Machine Learning Models

Lets now build the machine learning models using two widely used algorithms.

Random Forest

```
registerDoMC(cores = 8)
rfFit <- randomForest(classe~., data = myTraining , method ="rf", prox = TRUE)
rfFit
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = myTraining, method = "rf",      prox =
TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.66%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3346    2    0    0    0 0.0005973716
## B   16 2256    7    0    0 0.0100921457
## C    0   11 2040    3    0 0.0068159688
## D    0    1   30 1898    1 0.0165803109
## E    0    1    1    5 2158 0.0032332564
```

```
rfPred <- predict(rfFit, myTesting)
confusionMatrix(rfPred, myTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    2    0    0    0
##          B    1 1510    8    0    0
##          C    0    6 1360   16    0
##          D    0    0    0 1269    8
##          E    0    0    0    1 1434
##
## 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            0.9996   0.9947   0.9942   0.9868   0.9945
## Specificity            0.9996   0.9986   0.9966   0.9988   0.9998
## Pos Pred Value         0.9991   0.9941   0.9841   0.9937   0.9993
## Neg Pred Value         0.9998   0.9987   0.9988   0.9974   0.9988
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1925   0.1733   0.1617   0.1828
## Detection Prevalence   0.2846   0.1936   0.1761   0.1628   0.1829
## Balanced Accuracy      0.9996   0.9967   0.9954   0.9928   0.9971
```

As seen from the summary above the accuracy is 99%.

Lets now use another algorithm and verify its accuracy before deciding the algorithm to use on test data.

Generalized Boosted Regression Models

```
gbmFit <- train(classe~., data = myTraining, method ="gbm", verbose = FALSE)
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loaded gbm 2.1.3
```

```
## Loading required package: plyr
```

```
gbmFit
```

```
## Stochastic Gradient Boosting
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 11776, 11776, 11776, 11776, 11776, 11776, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7497457  0.6828031
##   1                  100      0.8147067  0.7654477
##   1                  150      0.8474586  0.8069078
##   2                   50      0.8487017  0.8082179
##   2                  100      0.8999980  0.8733961
##   2                  150      0.9239709  0.9037589
##   3                   50      0.8893499  0.8598529
##   3                  100      0.9341772  0.9166755
##   3                  150      0.9534538  0.9410827
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

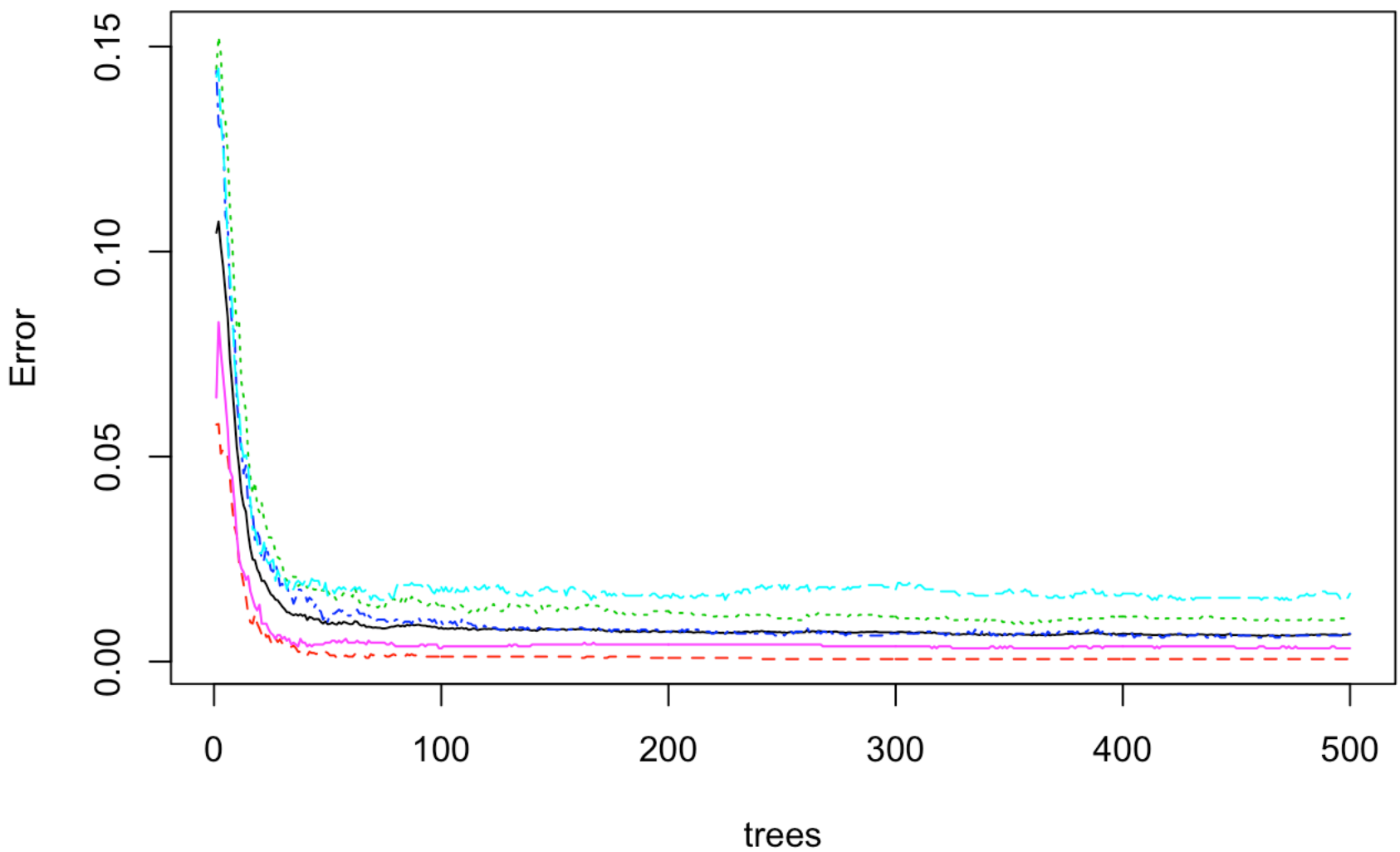
```
gbmPred <- predict(gbmFit, myTesting)
confusionMatrix(gbmPred, myTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2202   47    0    2    4
##          B   23 1428   36    4   20
##          C    5   38 1310   35    9
##          D    2    5   16 1240   22
##          E    0    0    6    5 1387
##
## Overall Statistics
##
##                Accuracy : 0.9644
##                  95% CI : (0.9601, 0.9684)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.955
##  Mcnemar's Test P-Value : 2.261e-08
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9866   0.9407   0.9576   0.9642   0.9619
## Specificity            0.9906   0.9869   0.9866   0.9931   0.9983
## Pos Pred Value         0.9765   0.9451   0.9377   0.9650   0.9921
## Neg Pred Value         0.9946   0.9858   0.9910   0.9930   0.9915
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2807   0.1820   0.1670   0.1580   0.1768
## Detection Prevalence   0.2874   0.1926   0.1781   0.1638   0.1782
## Balanced Accuracy      0.9886   0.9638   0.9721   0.9787   0.9801
```

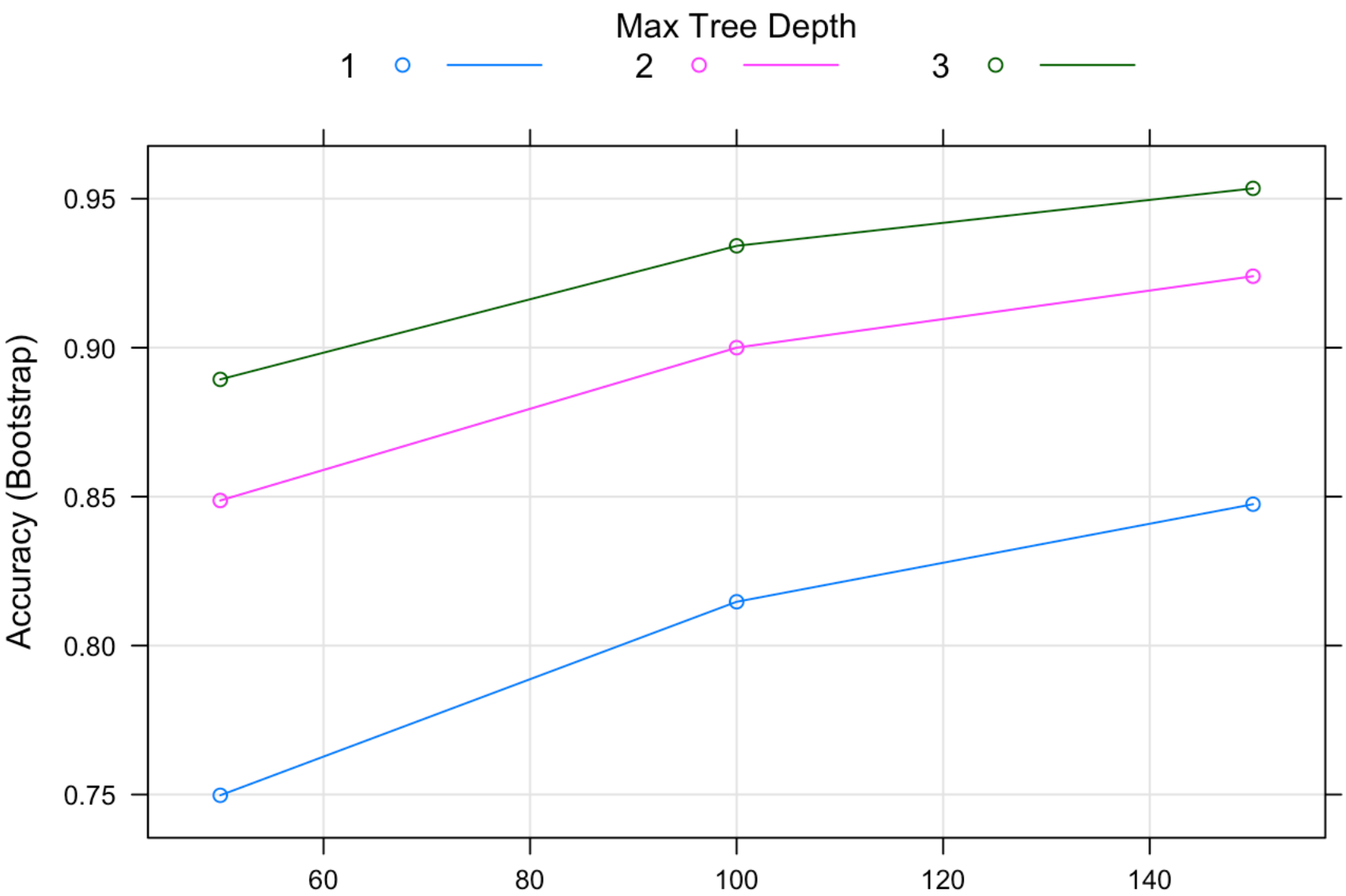
The accuracy of above algorithm is at 96%.

Let’s plot the 2 models.

```
plot(rfFit)
```




```
plot(gbmFit)
```



We can apply the Random Forest model for the testing set.

```
predict(rfFit, testing)
```

```
##  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
```