Practical Machine Learning Course Project

Fan Wang

May 16, 2016

Synopsis

People use wearable devices to 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 predict the manner in which they did the exercise. I will create a report describing how to build the model, how to use cross validation, find the expected out of sample error is, and decide the prediction model to use. I will also use the prediction model to predict 20 different test cases.

Data Source

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. 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.

Load the packages

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.2.5
```

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

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

```
## Warning: package 'ggplot2' was built under R version 3.2.3
```

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

```
## Warning: package 'rpart.plot' was built under R version 3.2.5
```

```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.2.5
```

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

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.2.5
```

```
## 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
```

Load the data, and do some basic data analyses

```
set.seed(3388)
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", na.strings
testing <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", na.strings =
dim(training); dim(testing)
```

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

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

```
str(training) #Results hidden
str(testing)
```

Split data into two smaller data sets

```
inTrain <- createDataPartition(y=training$classe, p=0.7, list = FALSE)
train1 <- training[inTrain,]
train2 <- training[-inTrain,]
dim(train1); dim(train2)
```

```
## [1] 13737   160
```

```
## [1] 5885  160
```

Clean and transform the data

Remove variables with near zero variance

```
nzv1 <- nearZeroVar(train1, saveMetrics = TRUE)
nzv2 <- nearZeroVar(train2, saveMetrics = TRUE)
train1 <- train1[, nzv1$nzv == FALSE]
train2 <- train2[, nzv2$nzv == FALSE]
```

Remove the first seven variables, as they do not seem to contribute to the prediction.

```
train1 <- train1[, -(1:7)]
train2 <- train2[, -(1:7)]
```

Remove variables that have too many NA values (75%).

```
NA75pct <- sapply(train1, function(x) mean(is.na(x))) > 0.75
NA75pct2 <- sapply(train2, function(x) mean(is.na(x))) > 0.75
train1 <- train1[, NA75pct==FALSE]
train2 <- train2[, NA75pct2==FALSE]
```

Further reducing the number of features by removing variables that exist in train2 and testing data set, but not in the train1 data set. We will not include "classe" variables in testing data set.

```
train1_var <- colnames(train1)
train2 <- train2[, train1_var]
final_var <- colnames(train1[, -52])
testing <- testing[, final_var]
dim(train1); dim(train2); dim(testing)
```

```
## [1] 13737    52
```

```
## [1] 5885   52
```

```
## [1] 20 51
```

Build Model

We will begin prediction with decision tree

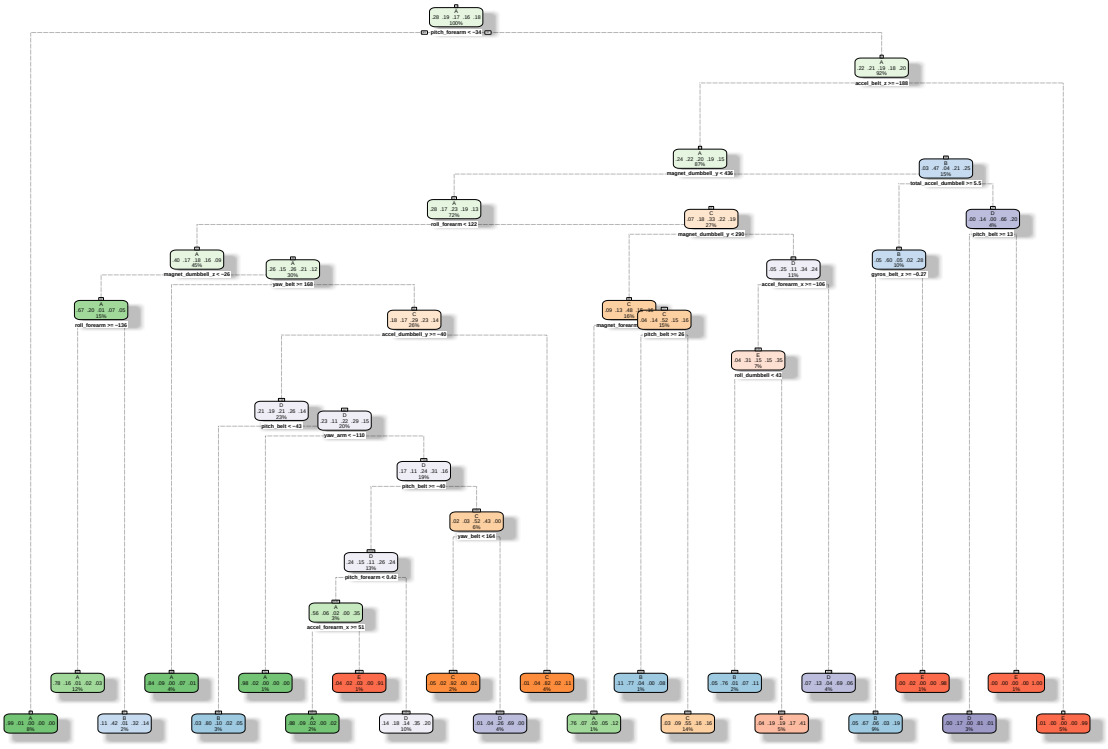
```
set.seed(138)
modfit_rpart <- rpart(classe ~ ., data = train1, method = "class")
pred_rpart <- predict(modfit_rpart, train2, type = "class")
confusionMatrix(pred_rpart, train2$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1469  159   17   29   33
##          B   60  636   58   75  156
##          C   27   89  739  148  164
##          D   87  194  139  644  122
##          E   31   61   73   68  607
```

```
##
## Overall Statistics
##
##                Accuracy : 0.6958
##                  95% CI : (0.6839, 0.7076)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6156
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8775   0.5584   0.7203   0.6680   0.5610
## Specificity            0.9435   0.9265   0.9119   0.8899   0.9515
## Pos Pred Value         0.8606   0.6457   0.6332   0.5430   0.7226
## Neg Pred Value         0.9509   0.8973   0.9392   0.9319   0.9058
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2496   0.1081   0.1256   0.1094   0.1031
## Detection Prevalence   0.2901   0.1674   0.1983   0.2015   0.1427
## Balanced Accuracy      0.9105   0.7424   0.8161   0.7790   0.7562
```

```
fancyRpartPlot(modfit_rpart)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2016−May−20 00:27:34 frankwang

We can see the prediction accuracy is about 70%. Next, we will try to predict using gbm method.

```
set.seed(353)
modfit_gbm <- train(classe ~ ., method = "gbm", data = train1, verbose = FALSE, trControl = trainControl
```

```
## Loading required package: gbm

## Warning: package 'gbm' was built under R version 3.2.5

## Loading required package: survival

##
## Attaching package: 'survival'

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

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.1

## Loading required package: plyr

## Warning: package 'plyr' was built under R version 3.2.3
```

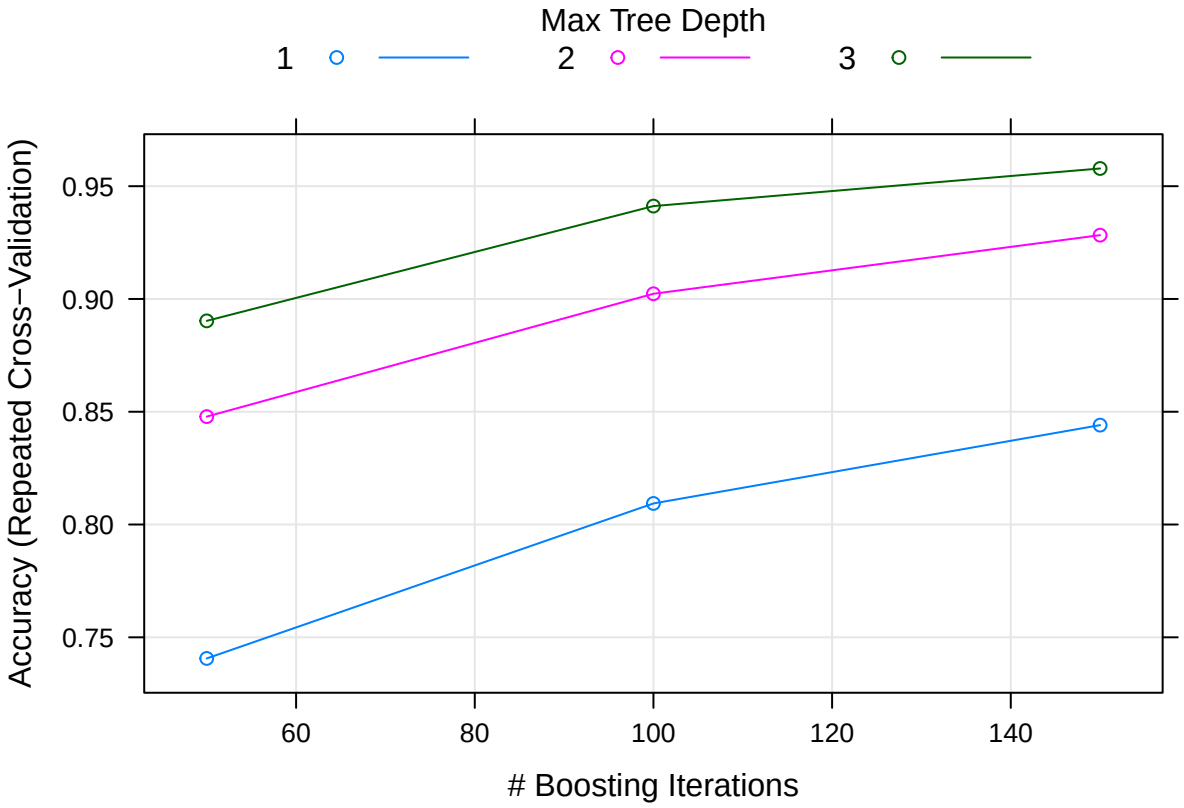
```
pred_gbm <- predict(modfit_gbm, train2)
confusionMatrix(pred_gbm, train2$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1644   43    0    1    1
##          B   22 1065   26    4    7
##          C    6   29  985   30   10
##          D    1    1   15  923   16
##          E    1    1    0    6 1048
##
## Overall Statistics
##
##                Accuracy : 0.9626
##                  95% CI : (0.9575, 0.9673)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9527
##  Mcnemar's Test P-Value : 2.759e-05
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9821   0.9350   0.9600   0.9575   0.9686
## Specificity            0.9893   0.9876   0.9846   0.9933   0.9983
## Pos Pred Value         0.9734   0.9475   0.9292   0.9655   0.9924
## Neg Pred Value         0.9929   0.9845   0.9915   0.9917   0.9930
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2794   0.1810   0.1674   0.1568   0.1781
## Detection Prevalence   0.2870   0.1910   0.1801   0.1624   0.1794
## Balanced Accuracy      0.9857   0.9613   0.9723   0.9754   0.9835
```

Accuracy has been significantly increased to 96%. Plot the results

```
plot(modfit_gbm)
```



The last prediction method we will use is random forest.

```
set.seed(158)
modfit_rf <- train(classe ~ ., method = "rf", data = train1, trControl =  trainControl(method = "cv", nu
pred_rf <- predict(modfit_rf, train2)
confusionMatrix(pred_rf, train2$classe)
```

```
## Confusion Matrix and Statistics
##
```

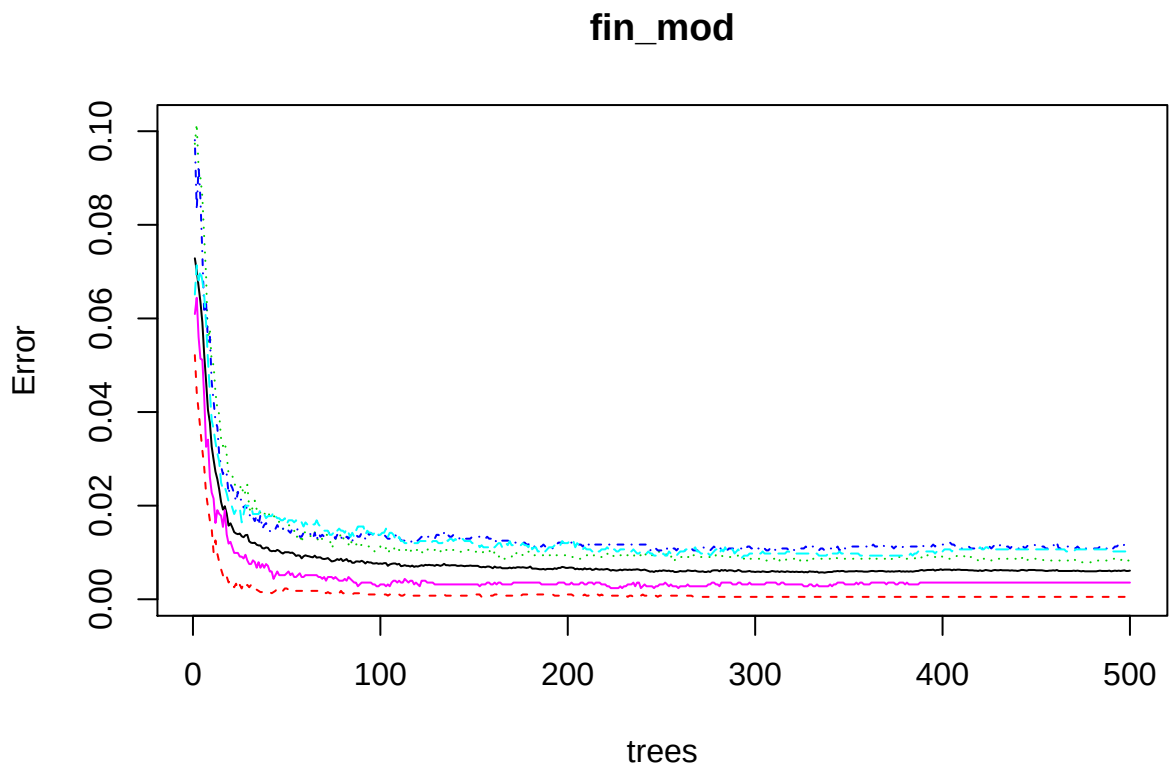
```
##           Reference
## Prediction    A    B    C    D    E
##          A 1670   15    0    0    0
##          B    2 1120   10    0    0
##          C    1    3 1013    5    3
##          D    0    1    3  956    1
##          E    1    0    0    3 1078
##
## Overall Statistics
##
##                Accuracy : 0.9918
##                  95% CI : (0.9892, 0.994)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9897
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9833   0.9873   0.9917   0.9963
## Specificity            0.9964   0.9975   0.9975   0.9990   0.9992
## Pos Pred Value         0.9911   0.9894   0.9883   0.9948   0.9963
## Neg Pred Value         0.9990   0.9960   0.9973   0.9984   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2838   0.1903   0.1721   0.1624   0.1832
## Detection Prevalence   0.2863   0.1924   0.1742   0.1633   0.1839
## Balanced Accuracy      0.9970   0.9904   0.9924   0.9953   0.9977
```

Accuracy is 99.2% and the expected out of sample error is about 0.8%, so we will use this model to predict on the testing data set. Plot the results. We can see that it uses 500 trees, and try 26 variables at each split. Error rate decreases as more trees are involved.

```
fin_mod <- modfit_rf$finalModel
fin_mod
```

```
##
## 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: 26
##
##         OOB estimate of  error rate: 0.61%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    2    0    0    0 0.0005120328
## B   15 2636    5    1    1 0.0082768999
## C    0   18 2368    9    1 0.0116861436
## D    0    0   22 2229    1 0.0102131439
## E    0    2    2    5 2516 0.0035643564
```

```
plot(fin_mod)
```



##Predict on testing data set Random forest gives best prediction accuracy, so we will use it to predict on testing data set.

```
pred_test <- predict(modfit_rf, testing)
pred_test
```

```
##  [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
```

Conclusion

After comparing three different types of prediction models, I decided to choose the one that uses random forest algorithm. It gives 99.8% accuracy. The expected out of sample error is very low (0.2%). Following are the "classe" results from predicting on testing set. [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