Recognizing Human Activity in Weight Lifting Exercise Data

Project Write-up for Practical Machine Learning

by L. Jackson on 31 January 2016

The purpose of this project is to use data from accelerometers on the belt, forearm, arm, and dumbell of six participants to predict how they performed a Dumbbell Biceps Curl. Particpants were asked to perform barbell lifts correctly and incorrectly in different ways.

Specifically: “Participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).”

Source: Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human ’13) . Stuttgart, Germany: ACM SIGCHI, 2013.

This report describes how a classification model was built, how cross validation was used, what the expected out of sample error is, and why the choices made in developing the model were made. Finally, the model is used to classify 20 different test cases.

Results can be reproduced using the R code in this document, randomForest package version 4.6-12 and data from these links:

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

```
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Data Development Process

```
directory <- "C:/Users/lajackso/Documents/GitHub/practicalmachinelearning/"
fileData <- paste0(directory,"pml-training.csv")
fileTest <- paste0(directory,"pml-testing.csv")
data <- read.csv(fileData, na.strings = c("", "NA","#DIV/0!"))
finalTest <- read.csv(fileTest, na.strings = c("", "NA","#DIV/0!"))
# colnames(data)
setdiff(colnames(data),colnames(finalTest))
```

```
## [1] "classe"
```

```
setdiff(colnames(finalTest),colnames(data))
```

```
## [1] "problem_id"
```

```
features <- intersect(colnames(data),colnames(finalTest))
drop <- colnames(finalTest)[apply(is.na(finalTest), 2, any)]
drop <- c(drop,"X","user_name","raw_timestamp_part_1","raw_timestamp_part_2","cvtd_timest
amp","new_window","num_window")
keep <- setdiff(features,drop)
# Verify cases are complete.
sum(complete.cases(finalTest[,keep]))
```

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

```
sum(complete.cases(data[,keep]))
```

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

```
data <- data[,c("classe",keep)]
# set.seed(1959); data <- data[sample(nrow(data)),]
finalTest <- finalTest[,c("problem_id",keep)]
```

First load the two sets of data. One data set is 19,622 observations on 160 variables and the other is 20 observations on the same 160 variables less the ‘classe’ variable (which is replaced with ‘problem_id’). Some feature data values are ‘NA’.

Remove features from the data sets that are not from accelerometers. Also remove features with an ‘NA’ value in any of the 20 observations of the second data set. This produces a data set with 19,622 observations on 52 feature variables with no missing data.

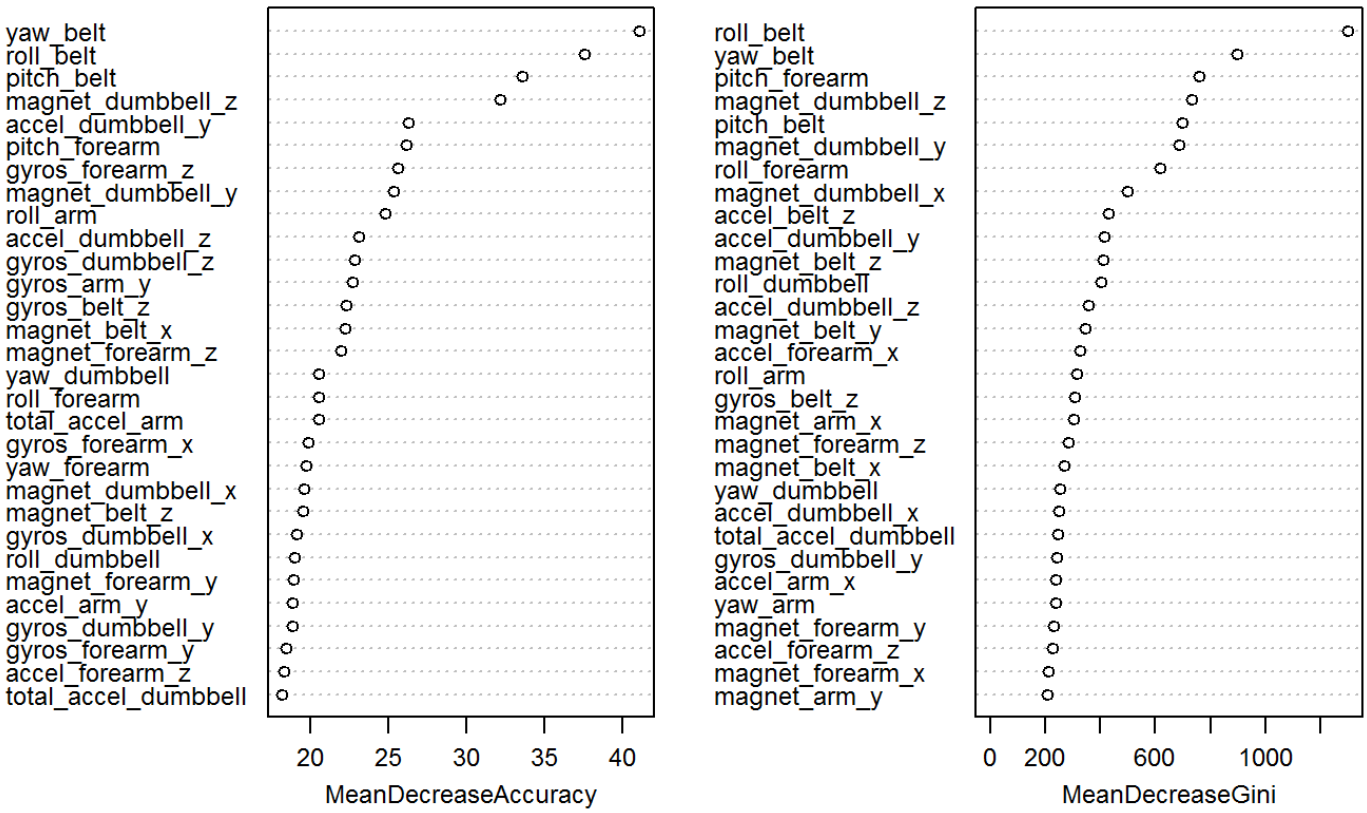
Random Forest Classification Model

Fit a Random Forest of 200 classification trees.

```
# Random Forest
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:dplyr':
##
##     combine
```

```
set.seed(1959)
model.rf <- randomForest(classe~.,data,ntree=200,importance=TRUE)
varImpPlot(model.rf,cex=0.75,main="Variable Importance in the Model")
```



Above are two plots of variable importance in the model. One plot presents the mean decrease in accuracy (MDA) of the forest when the values of a variable are randomly permuted in the out-of-bag samples. The other plot presents the mean decrease in node impurity (MDI) from splitting on the variable, averaged over all trees, as measured by the Gini index. Both of these variable importance measures have demonstrated practical utility.

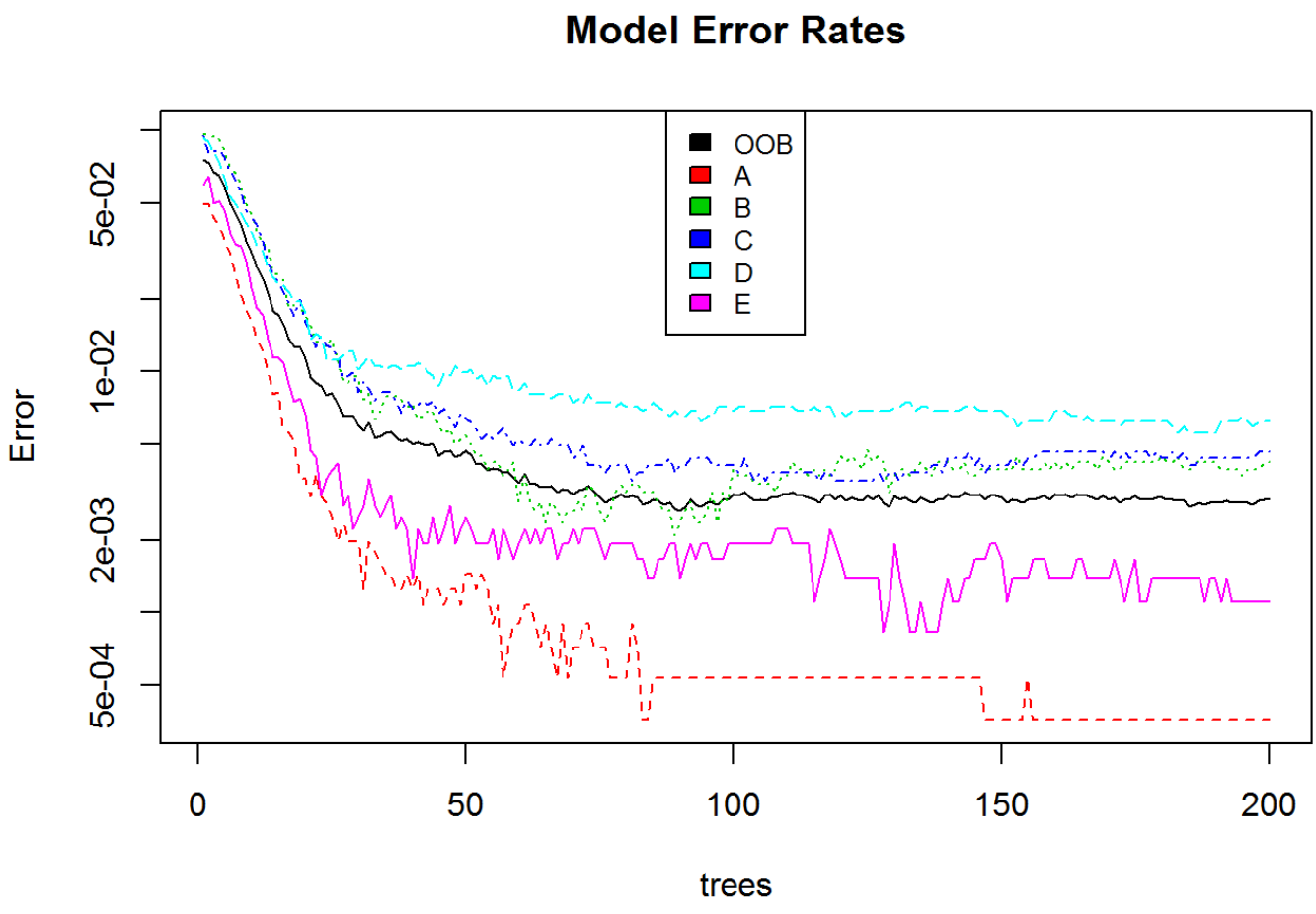
Expected Out of Sample Error

```
print(model.rf)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = data, ntree = 200,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.3%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5578    1    0    0    1 0.0003584229
## B   11 3781    5    0    0 0.0042138530
## C    0   14 3406    2    0 0.0046756283
## D    0    0   19 3196    1 0.0062189055
## E    0    0    0    4 3603 0.0011089548
```

The confusion matrix above cross tabulates reference class (row) versus out-of-bag (OOB) predicted class (column) and estimates the error rate for each class. Class error rates range from 0.0004 to 0.0062. The OOB estimated error rate is 0.003 (0.3%).

```
plot(model.rf,col=1:6,log="y", main="Model Error Rates")
legend("top", colnames(model.rf$err.rate),col=1:6,cex=0.8,fill=1:6)
```

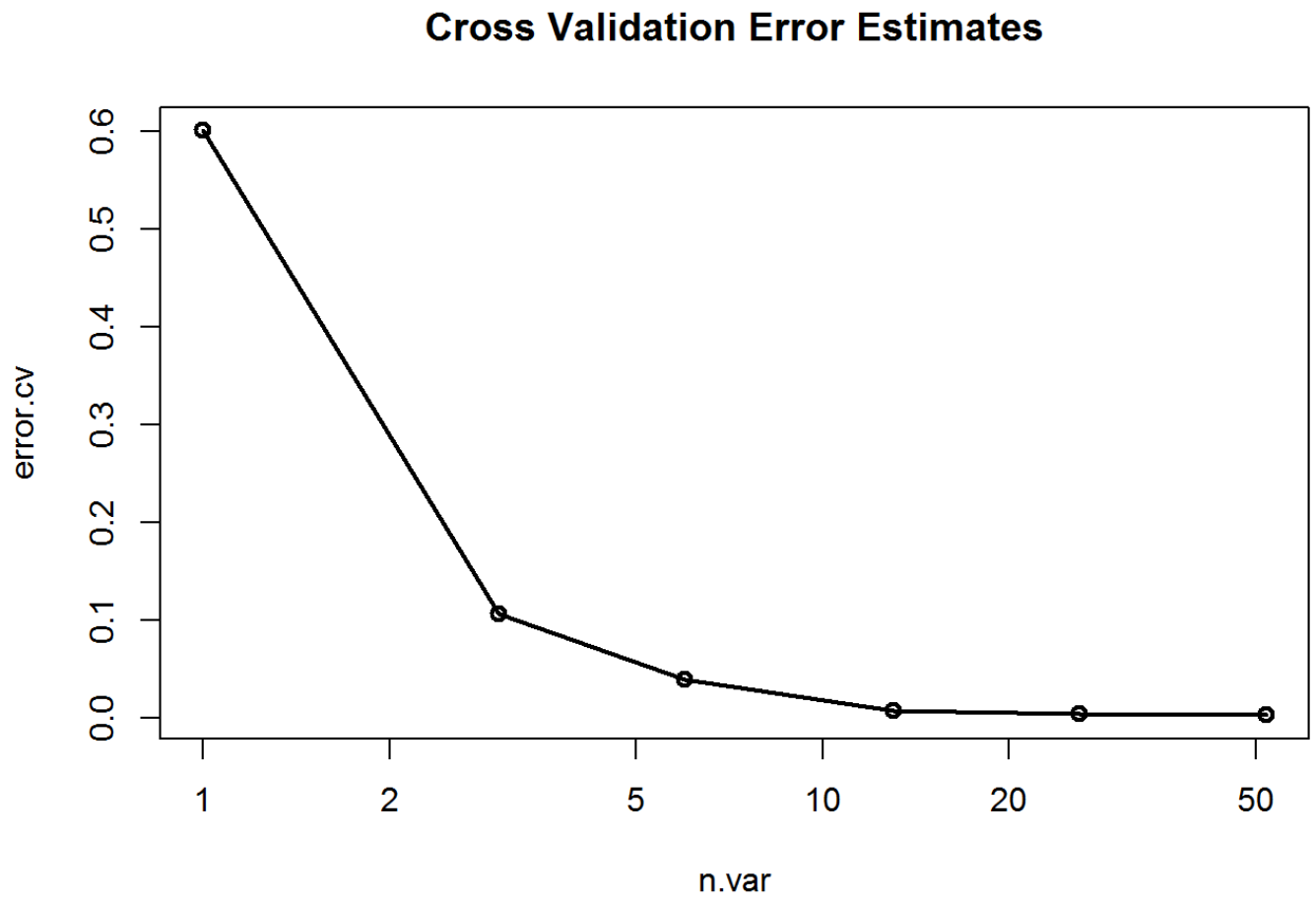


Plotted above are error rates for each class and OOB as a function of the number of trees in the model. OOB error rate stabilizes after as few as 90 trees. Class B exhibits the highest error rate, but rates are reasonably balanced among the classes. Balancing class error rate would increase the overall OOB error rate and is not necessary. (Note that the y-axis scale is logarithmic.)

Cross Validation Out of Sample Error Estimate

The rfcv function in the randomForest package shows the cross-validated prediction performance of models with sequentially reduced number of predictors (ranked by variable importance) via a nested cross-validation procedure. There is no need to reduce the number of predictors for this model, but the error estimate for all 52 predictors is relevant. Below rfcv is applied with 5-fold cross validation.

```
set.seed(1959)
result <- rfcv(data[,2:53], data$classe, cv.fold=5, step=0.5, trees=200)
with(result, plot(n.var, error.cv, log="x", type="o", lwd=2, main="Cross Validation Error
Estimates"))
```




```
result$error.cv
```

```
##          52          26          13           6           3           1
## 0.003465498 0.005096320 0.007797370 0.039955152 0.106665987 0.600856182
```

The 5-fold cross-validation estimate of the error rate for all 52 predictors is 0.003 (0.3%).

Notes on Bagged Classifiers and Error Estimates.

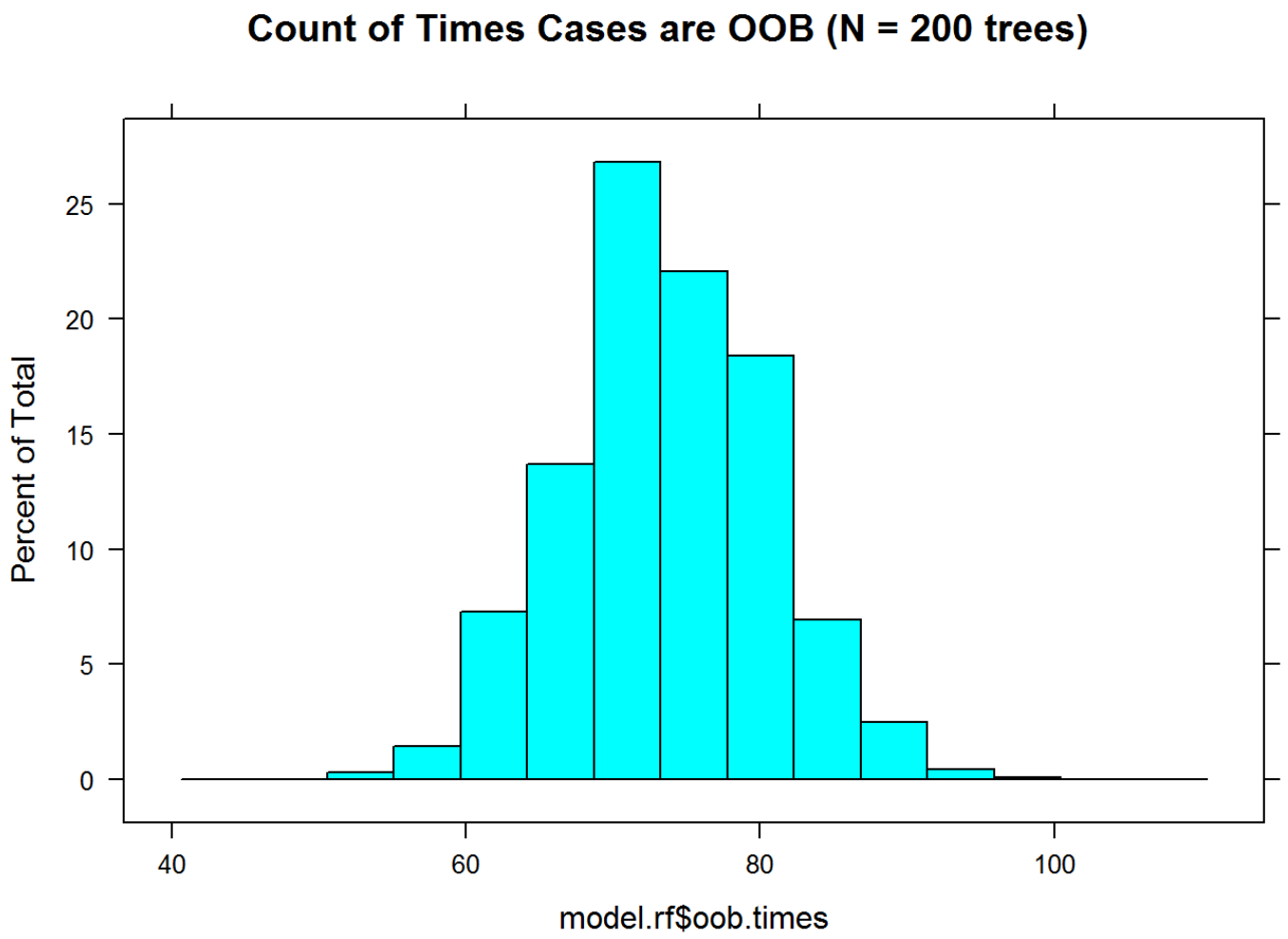
The study of error estimates for bagged classifiers by Leo Breiman (“Bagging Predictors” in Machine Learning, 24, pp. 123-140 (1996)), gives empirical evidence that the out-of-bag estimate is as accurate as using a test set of the same size as the training set. This obviates the need for a hold-out test set.

Breiman claims there is no need for additional cross-validation or a separate test set to get an unbiased estimate of the test set error in a random forest, because this error is estimated during model fitting. Each tree is constructed using a different bootstrap sample from the data. For each tree, an out-of-bag classification is obtained for the left out cases. All trees not built with a case vote on the class of that case. Comparing the winners to the actual classes in the training data produces an out-of-bag error rate for each class and an overall out-of-bag error rate.

```
summary(model.rf$oob.times)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   44.00   69.00   74.00   73.64   78.00  107.00
```

```
library(lattice)
histogram(model.rf$oob.times,main="Count of Times Cases are OOB (N = 200 trees)")
```



For this model the number of times a case was out-of-bag ranged from 44 to 107 with a median of 74. The histogram above shows the distribution.

Final Word on Out of Sample Error

The expected out of sample error is 0.003 (0.3%). Cross validation validates the OOB estimate. This rate is in line with the 99.4% accuracy rate from the 1/3 hold-out model below.

In the 5-fold cross validation, the error estimate was the average misclassification rate of five 200-tree models each built in turn using 80% of the data and tested on the remaining 20% of the data. Thus all cases were in the test set exactly once. None of those five models is identical to the ultimate model. The OOB estimate uses trees in the ultimate model, but estimates the error rate using only 22% to 54% of the trees for a given case (44 to 107 as noted above).

Model Development Choices

I consulted several sources on model selection considering factors such as purpose (multi-class classification); training method (batch versus incremental); if observed, partially observed or unobserved (outcome data is present for all cases); amount of data (~20k cases); number of features (~52 is modest and not large compared to the amount of data); and desired accuracy (best score is the goal). I also considered the number of parameters to tune in the model (fewer preferred) and training time (not a significant factor since I will train the predictor only once).

I decided to explore three ensemble methods appropriate to black-box classification with many features and a moderate number of cases. Those methods are support vector classification (SVC), adaptive boosting, and random forest classification. I partitioned the data based on `classe` with 1/3 in a hold-out test set.

Below are notes on the three methods. Detailed results are omitted.

```
library(caret)
```

```
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
##
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library(rpart)
library(mlbench)
library(adabag)
library(e1071)
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
##
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
# partition data for training and test
set.seed(1959)
inTrain = createDataPartition(data$classe, p = 2/3, list = FALSE)
training = data[inTrain,]
testing = data[-inTrain,]
```

Support Vector Classification using Support Vector Machines

SVC with svm (using `e1071` package) basically finds optimal separating hyperplanes between classes. With k=5 classes, it uses the `one-against-one'-approach, in which k(k-1)/2 = 10 binary classifiers are trained and the appropriate class is found by a voting scheme.

```
set.seed(1959)
model.svc <- svm(classe~.,data=training)
pred.svc <- predict(model.svc,testing)
confusionMatrix(pred.svc,testing$classe)$overall[1]
```

```
##  Accuracy
## 0.9368405
```

Adaptive Boosting with Classification Trees

Adaptive Boosting (using `adabag` package) fits classification trees as single classifiers using the AdaBoost.M1 algorithm. By default, a bootstrap sample of the training set is drawn using the weights for each observation on that iteration and alpha=1/2ln((1-err)/err) is the weight updating coefficient. In adaptive boosting, 'weak learners' are combined into a weighted sum to produce a boosted classifier. The algorithm is adaptive, because subsequent weak learners are tweaked in favor of those instances misclassified previously.

```
set.seed(1959)
model.ada <- boosting(classe~., data=training)
pred.ada <- predict(model.ada,testing)
confusionMatrix(pred.ada$class,testing$classe)$overall[1]
```

```
##  Accuracy
## 0.9206301
```

Random Forest with Classification Trees

```
set.seed(1959)
model.rfi <- randomForest(classe ~ ., training, importance=FALSE)
pred.rfi <- predict(model.rfi,testing)
confusionMatrix(pred.rfi,testing$classe)$overall[1]
```

```
##  Accuracy
## 0.9946475
```

Selecting and Tuning the Ultimate Model

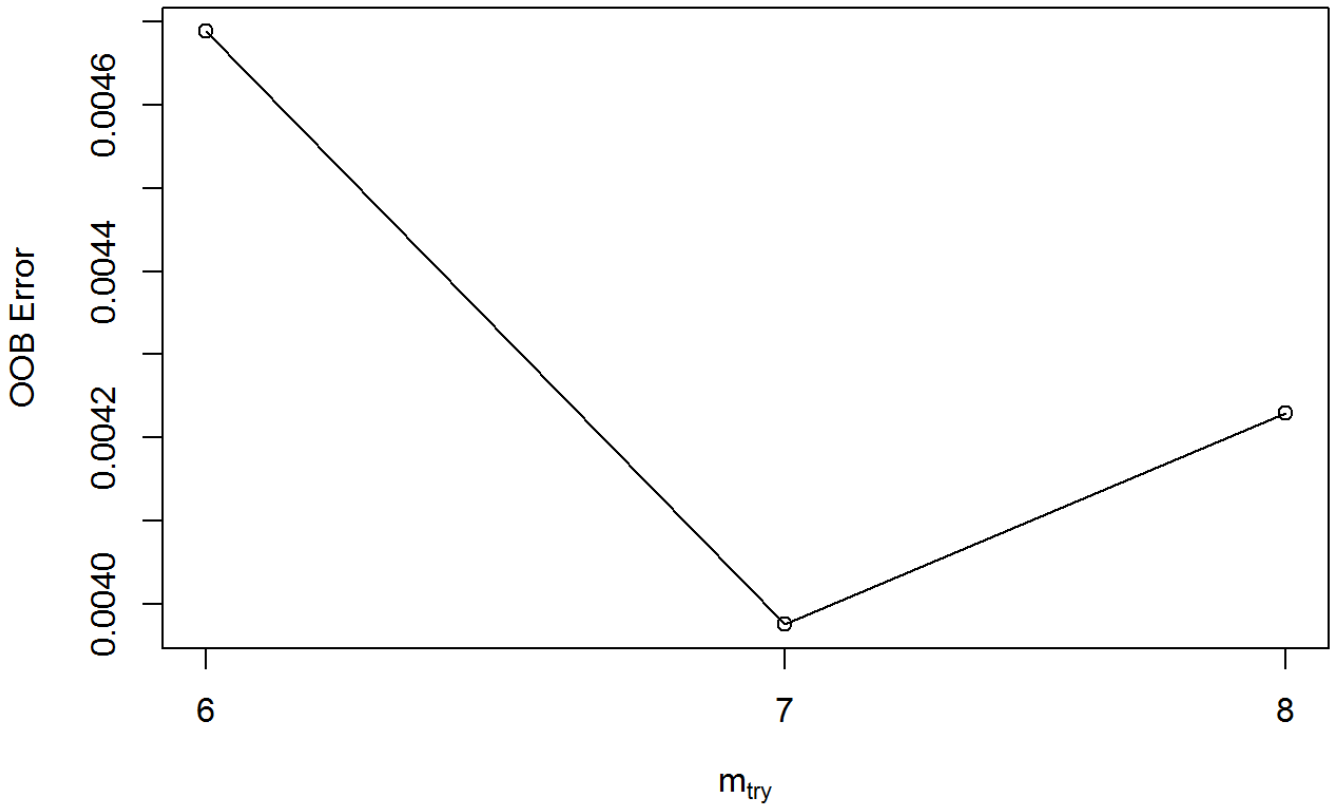
The random forest produced the best accuracy result. The random forest result were very good so I did minimal tuning. I reduced the number of trees from the 500 to 200 based on a plot of error rates versus tree size (example: Model Error Rates, above). Random forests do not overfit, so that was not a consideration; however, a smaller forest computes faster and I judged that 200 trees provided sufficient OOB instances for each case to estimate the error rate.

The random forest approach did not require additional effort for cross validation or the use of a hold-out test or validation data set per Breiman (above); therefore, I used the entire data set for the ultimate model.

I found that the randomForest package performed slightly better than party or caret with method="rf". Using `tuneRF` from randomForest (below), I determined that I did not need to consider changing the number of variables randomly selected as candidates at each split from the default 7 (~ square root of 52 features). According to Breiman, this is the only adjustable parameter to which random forests is somewhat sensitive.

```
set.seed(1959)
tuneRF(data[,2:53], data$classe, mtryStart=7, stepFactor=.9)
```

```
## mtry = 7  OOB error = 0.4%
## Searching left ...
## mtry = 8     OOB error = 0.42%
## -0.06410256 0.05
## Searching right ...
## mtry = 6     OOB error = 0.47%
## -0.1794872 0.05
```




```
##       mtry    OOBError
## 6.OOB    6 0.004688615
## 7.OOB    7 0.003975130
## 8.OOB    8 0.004229946
```

Ultimate Model Prediction

```
predict(model.rf,finalTest)
```

```
##  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
```

The model predicted these classes for the final test data scoring 20 out of 20. With an error rate of 0.003 we would expect to get this score 94% of the time.