Machine Learning Project

Pedro Márquez

Monday, July 20, 2015

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 (see the section on the Weight Lifting Exercise Dataset).

Data

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.

What you should submit

The goal of your project is to predict the manner in which they did the exercise. This is the "**classe**" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

1. Your submission should consist of a link to a Github repo with your R markdown and compiled HTML file describing your analysis. Please constrain the text of the writeup to < 2000 words and the number of figures to be less than 5. It will make it easier for the graders if you submit a repo with a gh-pages branch so the HTML page can be viewed online (and you always want to make it easy on graders :-).
2. You should also apply your machine learning algorithm to the 20 test cases available in the test data above. Please submit your predictions in appropriate format to the programming assignment for automated grading. See the programming assignment for additional details.

Reproducibility

Due to security concerns with the exchange of R code, your code will not be run during the evaluation by your classmates. Please be sure that if they download the repo, they will be able to view the compiled HTML version of your analysis.

Load all required libraries:

```
library(caret)
```

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

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

```
## Loading required package: RGtk2
## Rattle: A free graphical interface for data mining with R.
## Version 3.5.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

In order to get the results here reported, set seed as follows:

```
set.seed(7531)
```

Load the training and test data sets, taking into consideration that data has different strings we interprete as **NA**:

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

testData <- read.csv("pml-testing.csv",
                     na.strings=c("NA","#DIV/0!",""))

dim(trainData)
```

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

```
dim(testData)
```

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

Cleaning the data

Before doing any prediction, we have to clean the data.

NearZeroVariance Variables: Datasets come sometimes with predictors that take an unique value across samples. This kind of predictor is not only non-informative, it can break some models you may want to fit to your data. Even more common is the presence of predictors that are almost constant across samples. One quick and dirty solution is to remove all predictors that satisfy some threshold criterion related to their variance.

```
TrainDataNZV <- nearZeroVar(trainData, saveMetrics=TRUE)
head(TrainDataNZV,10)
```

```
##                        freqRatio percentUnique zeroVar   nzv
## X                       1.000000  100.00000000   FALSE FALSE
## user_name               1.100679    0.03057792   FALSE FALSE
## raw_timestamp_part_1    1.000000    4.26562022   FALSE FALSE
## raw_timestamp_part_2    1.000000   85.53154622   FALSE FALSE
## cvtd_timestamp          1.000668    0.10192641   FALSE FALSE
## new_window             47.330049    0.01019264   FALSE  TRUE
## num_window              1.000000    4.37264295   FALSE FALSE
## roll_belt               1.101904    6.77810621   FALSE FALSE
## pitch_belt              1.036082    9.37722964   FALSE FALSE
## yaw_belt                1.058480    9.97349913   FALSE FALSE
```

By default, a predictor is classified as near-zero variance if the percentage of unique values in the samples is less than {10%} and when the frequency ratio mentioned above is greater than 19 (95/5). These default values can be changed by setting the arguments **uniqueCut** and **freqCut**.

We can explore which ones are the zero variance predictors, and which ones are the near-zero variance predictors:

```
TrainDataNZV[TrainDataNZV[,"zeroVar"] + TrainDataNZV[,"nzv"] > 0,]
```

```
##                          freqRatio percentUnique zeroVar  nzv
## new_window                47.33005    0.01019264   FALSE TRUE
## kurtosis_yaw_belt          0.00000    0.00000000    TRUE TRUE
## skewness_yaw_belt          0.00000    0.00000000    TRUE TRUE
## amplitude_yaw_belt         0.00000    0.00509632    TRUE TRUE
## avg_roll_arm              77.00000    1.68178575   FALSE TRUE
## stddev_roll_arm           77.00000    1.68178575   FALSE TRUE
## var_roll_arm              77.00000    1.68178575   FALSE TRUE
## avg_pitch_arm             77.00000    1.68178575   FALSE TRUE
## stddev_pitch_arm          77.00000    1.68178575   FALSE TRUE
## var_pitch_arm             77.00000    1.68178575   FALSE TRUE
## avg_yaw_arm               77.00000    1.68178575   FALSE TRUE
## stddev_yaw_arm            80.00000    1.66649679   FALSE TRUE
## var_yaw_arm               80.00000    1.66649679   FALSE TRUE
## max_roll_arm              25.66667    1.47793293   FALSE TRUE
## min_roll_arm              19.25000    1.41677709   FALSE TRUE
## min_pitch_arm             19.25000    1.47793293   FALSE TRUE
## amplitude_roll_arm        25.66667    1.55947406   FALSE TRUE
## amplitude_pitch_arm       20.00000    1.49831821   FALSE TRUE
## kurtosis_yaw_dumbbell      0.00000    0.00000000    TRUE TRUE
## skewness_yaw_dumbbell      0.00000    0.00000000    TRUE TRUE
## amplitude_yaw_dumbbell     0.00000    0.00509632    TRUE TRUE
## kurtosis_yaw_forearm       0.00000    0.00000000    TRUE TRUE
## skewness_yaw_forearm       0.00000    0.00000000    TRUE TRUE
## max_roll_forearm          27.66667    1.38110284   FALSE TRUE
## min_roll_forearm          27.66667    1.37091020   FALSE TRUE
## amplitude_roll_forearm    20.75000    1.49322189   FALSE TRUE
## amplitude_yaw_forearm      0.00000    0.00509632    TRUE TRUE
## avg_roll_forearm          27.66667    1.64101519   FALSE TRUE
```

```
## stddev_roll_forearm     87.00000    1.63082255   FALSE TRUE
## var_roll_forearm        87.00000    1.63082255   FALSE TRUE
## avg_pitch_forearm       83.00000    1.65120783   FALSE TRUE
## stddev_pitch_forearm    41.50000    1.64611151   FALSE TRUE
## var_pitch_forearm       83.00000    1.65120783   FALSE TRUE
## avg_yaw_forearm         83.00000    1.65120783   FALSE TRUE
## stddev_yaw_forearm      85.00000    1.64101519   FALSE TRUE
## var_yaw_forearm         85.00000    1.64101519   FALSE TRUE
```

There are 9 **zeroVar** predictors, and 36 **nzv** predictors.

Now, we delete near-zero variance variables, columns not useful, and all columns with too many "NA"s (more than 70%):

```
trainData <- trainData[-c(1:7)]
TrainDataNZV <- nearZeroVar(trainData)
newTrainData <- trainData[,-TrainDataNZV]

NAs <- sapply(colnames(newTrainData),
              function(x)
                  if(sum(is.na(newTrainData[, x])) >
                     0.7 * nrow(newTrainData))
                     return(TRUE)
                  else
                     return(FALSE)
              )

cat("We found",sum(NAs),"variables with too many NAs")
```

```
## We found 65 variables with too many NAs
```

```
newTrainData <- newTrainData[, !NAs]
cat("Final training data dimensions:",dim(newTrainData))
```

```
## Final training data dimensions: 19622 53
```

We partition the training data to get the test data from it:

```
part <- createDataPartition(y=newTrainData$classe, p=0.6, list=FALSE)
newTestData <- newTrainData[-part, ]
newTrainData <- newTrainData[part, ]
dim(newTrainData)
```

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

```
dim(newTrainData)
```

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

Identifying Correlated Predictors

While there are some models that thrive on correlated predictors (such as pls), other models may benefit from reducing the level of correlation between the predictors.

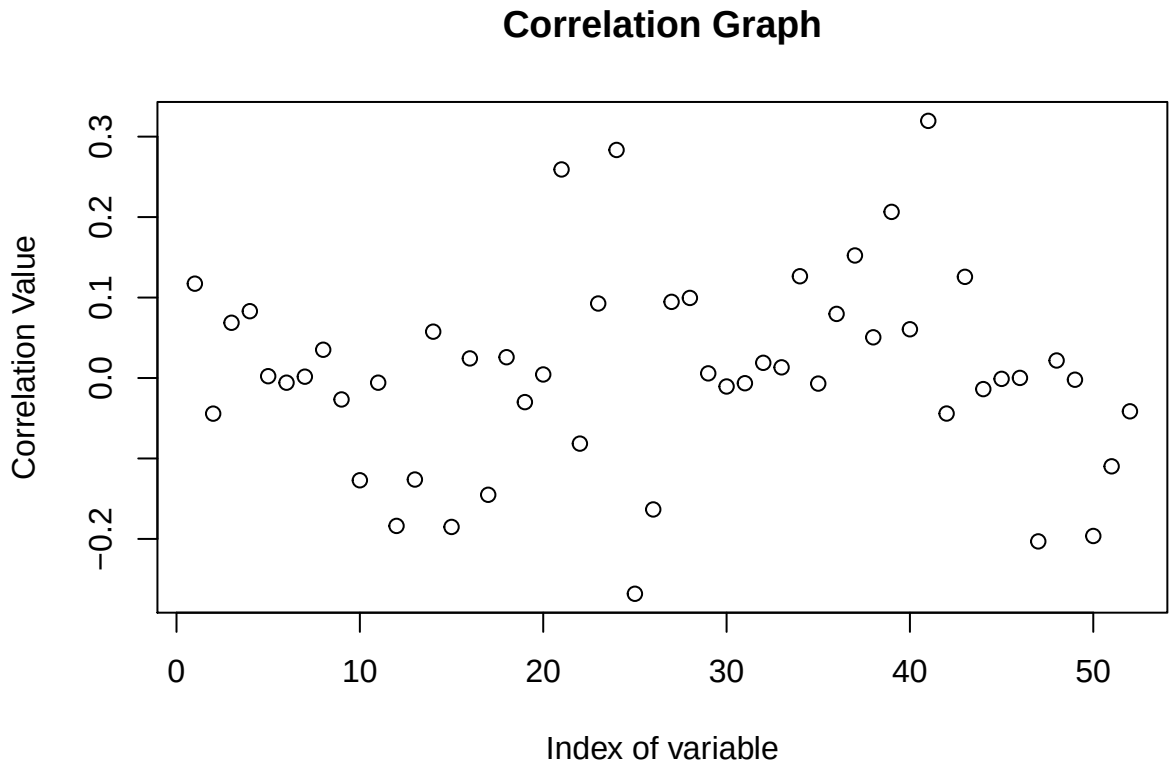
Spearman's coefficient, like any correlation calculation, is appropriate for both continuous and discrete variables, including ordinal variables. The Spearman correlation increases in magnitude as $\mathbf{X}$ and $\mathbf{Y}$ become closer to being perfect monotone functions of each other.

```
nums <- sapply(newTrainData, is.numeric)
M <- cor(newTrainData[,nums],
       as.numeric(newTrainData$classe),method="spearman")
print(M)
```

```
##                             [,1]
## roll_belt             0.1172708284
## pitch_belt           -0.0442978085
## yaw_belt              0.0686588065
## total_accel_belt      0.0830503756
## gyros_belt_x          0.0022519462
## gyros_belt_y         -0.0058123439
## gyros_belt_z          0.0016404097
## accel_belt_x          0.0351132943
## accel_belt_y         -0.0266563530
## accel_belt_z         -0.1271128457
## magnet_belt_x        -0.0058100762
## magnet_belt_y        -0.1838174231
## magnet_belt_z        -0.1261775559
## roll_arm              0.0576134656
## pitch_arm            -0.1850905866
## yaw_arm               0.0243542482
## total_accel_arm      -0.1451471351
## gyros_arm_x           0.0258309803
## gyros_arm_y          -0.0299579314
## gyros_arm_z           0.0043933123
## accel_arm_x           0.2592063954
## accel_arm_y          -0.0815286304
## accel_arm_z           0.0926034874
## magnet_arm_x          0.2834484469
## magnet_arm_y         -0.2681507481
## magnet_arm_z         -0.1633631215
## roll_dumbbell         0.0946468185
## pitch_dumbbell        0.0995128590
## yaw_dumbbell          0.0057624220
## total_accel_dumbbell -0.0105414759
## gyros_dumbbell_x     -0.0064668216
## gyros_dumbbell_y      0.0188746679
## gyros_dumbbell_z      0.0132502203
## accel_dumbbell_x      0.1264629000
## accel_dumbbell_y     -0.0069393541
## accel_dumbbell_z      0.0796831579
## magnet_dumbbell_x     0.1523002426
## magnet_dumbbell_y     0.0505820359
## magnet_dumbbell_z     0.2064838593
```

```
## roll_forearm          0.0605684637
## pitch_forearm         0.3195803609
## yaw_forearm          -0.0441522818
## total_accel_forearm   0.1257353853
## gyros_forearm_x      -0.0137942052
## gyros_forearm_y      -0.0010525939
## gyros_forearm_z       0.0001226287
## accel_forearm_x      -0.2030848272
## accel_forearm_y       0.0217576878
## accel_forearm_z      -0.0021384253
## magnet_forearm_x     -0.1963403271
## magnet_forearm_y     -0.1097703058
## magnet_forearm_z     -0.0414124578
```

```
plot(M, xlab="Index of variable",
     ylab="Correlation Value",
     main="Correlation Graph")
```



It becomes apparent from the graph and values that there are not predictors that strongly correlate with `classe`, so we will explore other methods.

Decision Trees

Decision trees helps us explore the stucture of data, while developing easy to visualize decision rules for predicting a categorical classification tree outcome. We look for the tree with the highest cross-validated error less than the minimum cross-validated error plus the standard deviation of the error at that tree.

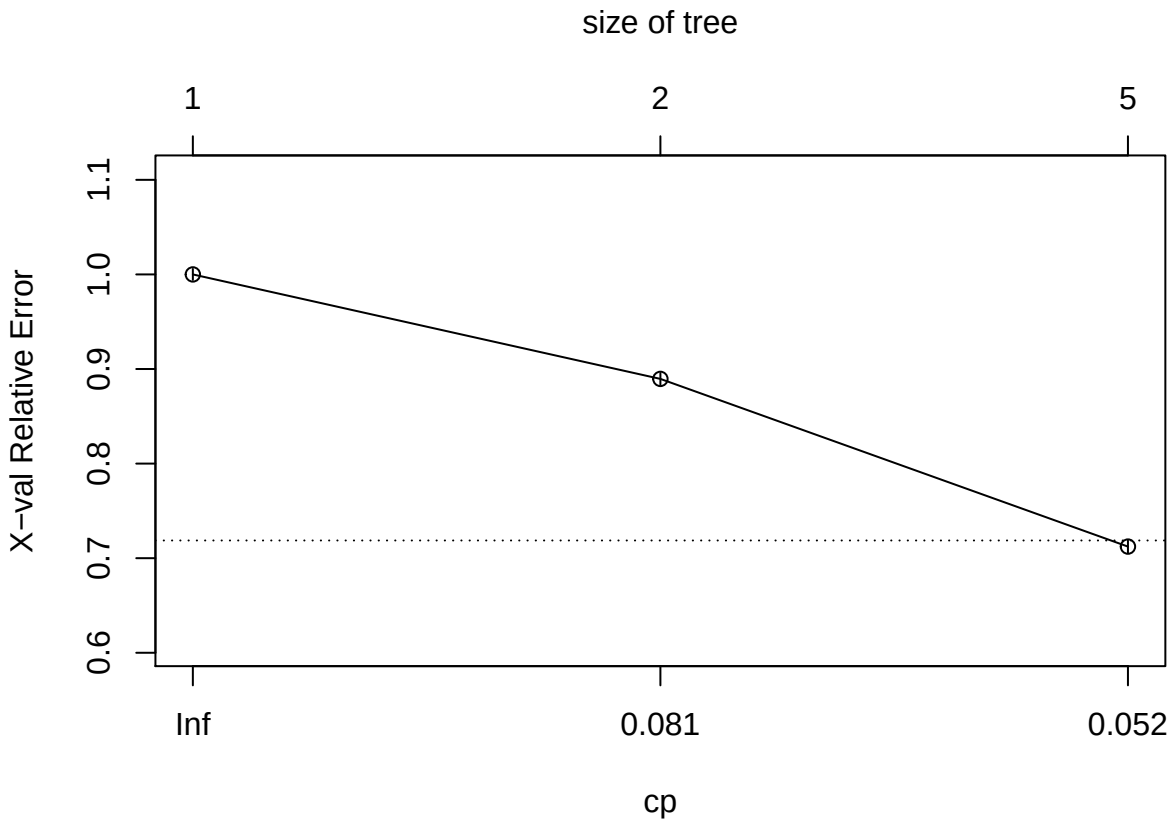
```
mycontrol = rpart.control(cp = 0, xval = 10)
fittree = rpart(classe ~., method = "class",
     data = newTrainData, control = mycontrol)
minXerror <- min(fittree$cptable[,"xerror"])
row <- fittree$cptable[which(fittree$cptable[,"xerror"]==minXerror),]
newcpt <- row[1] + minXerror
cptrows <- fittree$cptable[which(fittree$cptable[,"CP"] <= newcpt),]
maxmincpt <- max(cptrows[,"CP"])
nrow <- which(fittree$cptable[,"CP"] == maxmincpt)
cptarg <- sqrt(fittree$cptable[nrow,1]*fittree$cptable[nrow+1,1])
prunedtree = prune(fittree,cp=cptarg)
```

We now prune the decision tree at the best `complexity parameter`:

```
DT.fit <- rpart(classe ~ ., data=newTrainData, method="class", cp=cptarg)
printcp(DT.fit)
```

```
##
## Classification tree:
## rpart(formula = classe ~ ., data = newTrainData, method = "class",
##     cp = cptarg)
##
## Variables actually used in tree construction:
## [1] magnet_dumbbell_y pitch_forearm     roll_belt         roll_forearm
##
## Root node error: 8428/11776 = 0.71569
##
## n= 11776
##
##         CP nsplit rel error  xerror      xstd
## 1 0.111533      0   1.00000 1.00000 0.0058081
## 2 0.059366      1   0.88847 0.88953 0.0061929
## 3 0.045739      4   0.71037 0.71227 0.0064367
```

```
plotcp(DT.fit)
```



Now let's do the predictions:

```
pred.DT <- predict(DT.fit, newTestData, type = "class")
confusionMatrix(pred.DT, newTestData$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2031  638  611  596  214
##          B   34  508   44  221  183
##          C  161  372  713  469  362
##          D    0    0    0    0    0
##          E    6    0    0    0  683
##
## Overall Statistics
##
##                Accuracy : 0.5015
##                  95% CI : (0.4904, 0.5127)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3484
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9099  0.33465  0.52120   0.0000  0.47365
## Specificity            0.6332  0.92383  0.78944   1.0000  0.99906
## Pos Pred Value         0.4966  0.51313  0.34328      NaN  0.99129
## Neg Pred Value         0.9465  0.85268  0.88646   0.8361  0.89395
## Prevalence             0.2845  0.19347  0.17436   0.1639  0.18379
## Detection Rate         0.2589  0.06475  0.09087   0.0000  0.08705
## Detection Prevalence   0.5213  0.12618  0.26472   0.0000  0.08782
## Balanced Accuracy      0.7716  0.62924  0.65532   0.5000  0.73636
```

Boosting

Boosting is a machine learning ensemble meta-algorithm for reducing bias primarily and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones. A weak learner is defined to be a classifier which is only slightly correlated with the true classification. In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.

```
set.seed(7531)
fitControl <- trainControl(method = "cv",
                           number = 10,
                           repeats = 10)
gbmFit <- train(classe ~ ., data = newTrainData,
                method = "gbm",
                trControl = fitControl,
                verbose = FALSE)
```

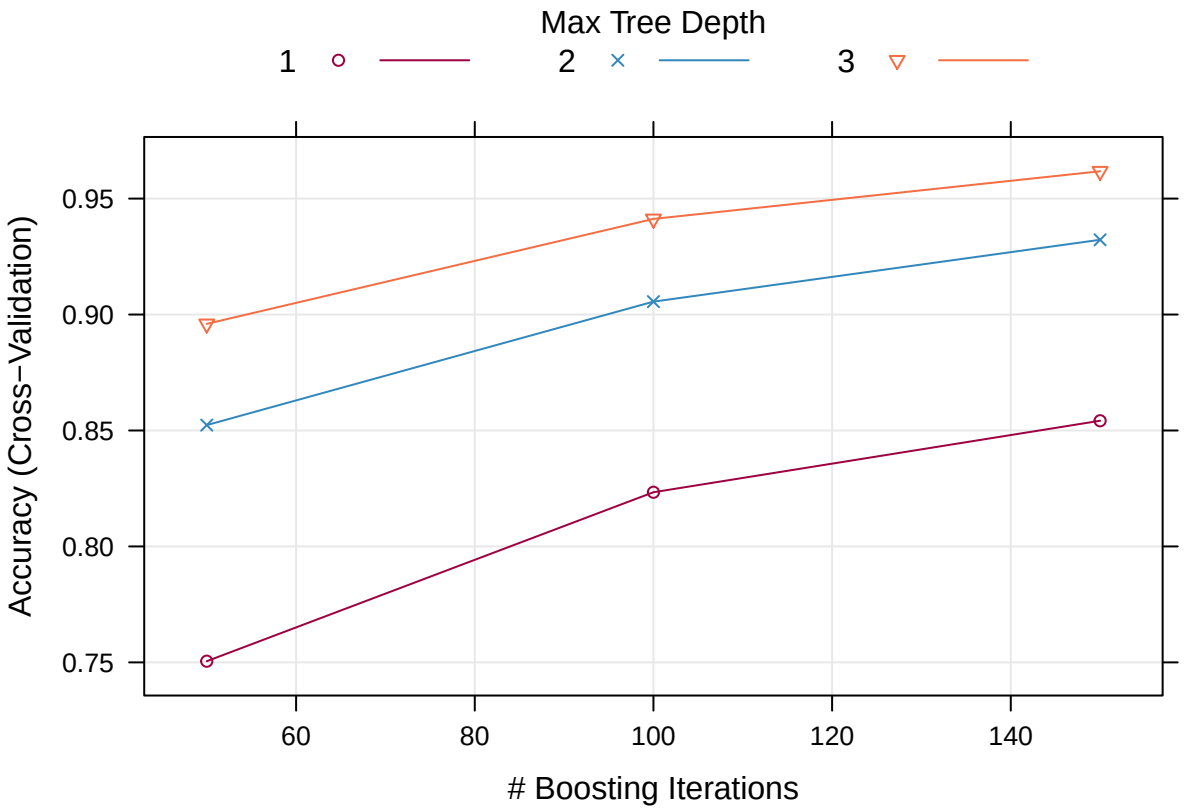
```
## Loading required package: gbm
## Loading required package: survival
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.1
## Loading required package: plyr
```

```
gbmFit
```

```
## Stochastic Gradient Boosting
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 10601, 10598, 10598, 10598, 10598, 10597, ...
```

```
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.7505050  0.6834559  0.010317940
##   1                  100      0.8233648  0.7763930  0.008186901
##   1                  150      0.8541930  0.8154830  0.006848621
##   2                   50      0.8523256  0.8128685  0.007117277
##   2                  100      0.9055687  0.8804763  0.010807156
##   2                  150      0.9322337  0.9142502  0.006100124
##   3                   50      0.8959729  0.8683288  0.007395508
##   3                  100      0.9412345  0.9256430  0.005641616
##   3                  150      0.9617849  0.9516540  0.005266735
##   Kappa SD
##   0.013086607
##   0.010365572
##   0.008681012
##   0.009033814
##   0.013675301
##   0.007718580
##   0.009348084
##   0.007157029
##   0.006656385
##
## 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.
```

```
trellis.par.set(caretTheme())
plot(gbmFit)
```

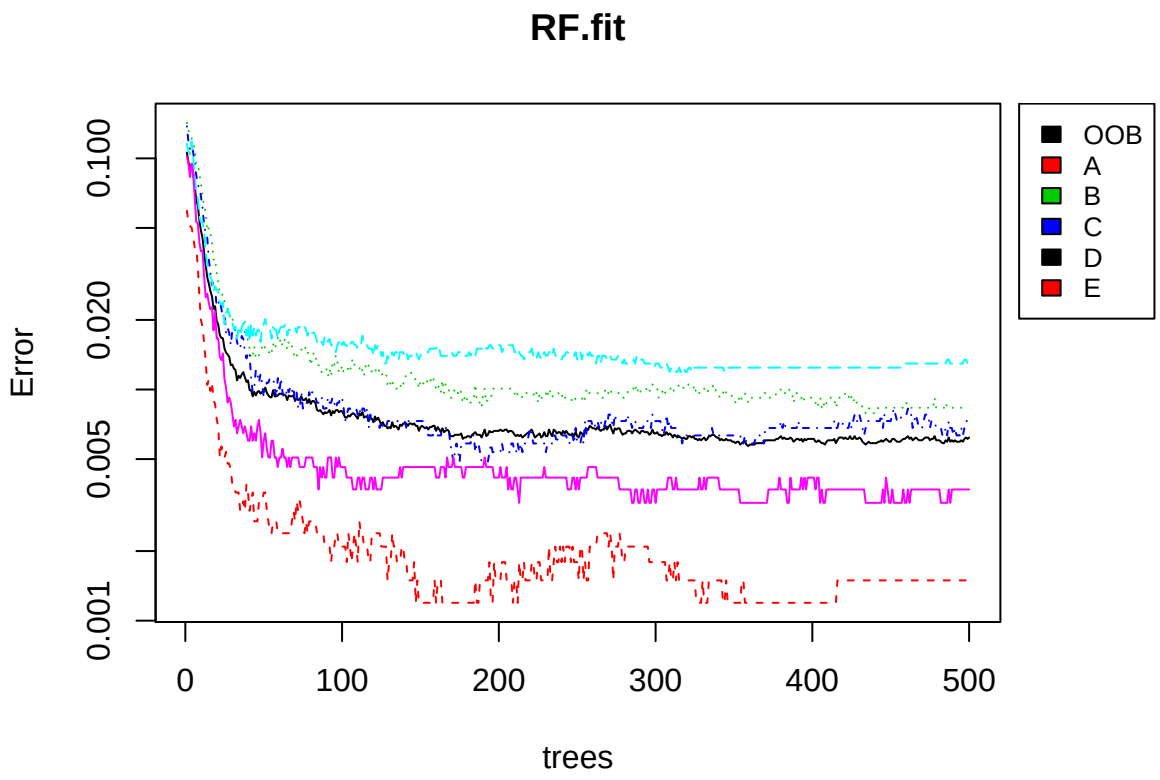


Random Forest

Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance of the final model.

```
set.seed(7531)
RF.fit <- randomForest(classe ~. , data=newTrainData, method="class")

layout(matrix(c(1,2),nrow=1), width=c(4,1))
par(mar=c(5,4,4,0)) #No margin on the right side
plot(RF.fit, log="y")
par(mar=c(5,0,4,2)) #No margin on the left side
plot(c(0,1),type="n", axes=F, xlab="", ylab="")
legend("top", colnames(RF.fit$err.rate),col=1:4,cex=0.8,fill=1:4)
```



Let's do the prediction with random forest:

```
pred.RF <- predict(RF.fit, newTestData, type="class")
confusionMatrix(pred.RF, newTestData$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2228   11    0    0    0
##          B    2 1507    7    0    0
##          C    2    0 1355   13    1
##          D    0    0    6 1271    6
##          E    0    0    0    2 1435
##
## Overall Statistics
##
##                Accuracy : 0.9936
##                  95% CI : (0.9916, 0.9953)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9919
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9928   0.9905   0.9883   0.9951
## Specificity            0.9980   0.9986   0.9975   0.9982   0.9997
## Pos Pred Value         0.9951   0.9941   0.9883   0.9906   0.9986
## Neg Pred Value         0.9993   0.9983   0.9980   0.9977   0.9989
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2840   0.1921   0.1727   0.1620   0.1829
## Detection Prevalence   0.2854   0.1932   0.1747   0.1635   0.1832
## Balanced Accuracy      0.9981   0.9957   0.9940   0.9933   0.9974
```

Files to submit

It seems that Random Forest performed better, as expected, so we use it to submit the files:

```
vars <- names(newTrainData[,-52])
finalTest <- testData[vars]
pred.RF <- predict(RF.fit, finalTest, type="class")
```

```
pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,
                quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}

pml_write_files(pred.RF)
```