Machine Learning Project

Ronald Thisted

August 23, 2014

Contents

Executive Summary

We construct a machine-learning algorithm to predict activity quality, represented by five classes of activity "A" through "E", based on activity monitor data. To build this algorithm, we first do some data cleaning, remove unusable variables, and remove variables that are not suitable for prediction (such as session labels or timestamps) that are not related to activity. Next, we look at three kinds of machine-learning algorithms, and we select random forests as being highly accurate as well as relatively inexpensive computationally. We then evaluate likely out-of-sample prediction errors based on this model, taking advantage of both the internal cross validation that is built into the random forest algorithm, and also assessing the effect of number of predictors on prediction accuracy using cross validation.

Download and preprocess data

The R code for these steps is included in the Rmd file, but is omitted from the display file. The steps we took are outlined briefly below.

First, we downloaded the training and test data sets into two data frames, `trainSet` and `testSet`. Based on a preliminary look at the training data set in Excel and R, some of the variables have lots of NAs, and some have entries for only a subset of rows corresponding to `new_window=="yes"`. For these variables, the automatic reading of the .csv file puts the numeric values in quotes, so that they needed to be converted back to numerics. After further analysis, it appeared that rows with `new_window=="yes"` actually represented summary statistics based on a collection of other entries in the data set. For this reason, we omitted all of the rows with `new_window=="yes"`, and we omitted those variables that only had values in those rows.

Because the original Excel data set mixed two kinds of rows—raw data rows and summary rows—some entries for some variables contained character data in the summary rows. As a result, those variables appeared to be factor variables when imported into R. We identified those variables and, after deleting the summary rows, converted those variables back to their numeric values. This temporary reduced training data set is called `dssub`.

Omitting non-predictors

Since the timestamps uniquely identify occasions in the training set (but are independent of anything that will appear in the test set), they can be used to perfectly predict (in-sample) class, but are useless for out-of-sample prediction. Similary user name is not helpful for out-of-sample prediction. Using the command `table(dssub$num_window,dssub$user_name)` (output omitted), it is also clear that each window number is

used only for a single user. Thus, these variables should be omitted from any prediction algorithm. These are variables 1:5 and 7 in the data set.

We also restrict the prediction model to be based on the cases for which `new_window=="no"`, which is variable number 6 in the data set. Thus, columns 8:60 are used for model building. The reduced data set that omits the specified rows and columns is called `tidyTrain`.

Building alternative machine-learning models

We started with a simple recursive partitioning prediction model (CART), followed by k-nearest neighbors, random forests, and then bagged trees. At each stage, we compared the performance of the current algorithm to its best competitor thus far, provisionally keeping the model that best trades off accuracy and computational efficiency. We then focused most of our attention on the best performing model to examine issues of out-of-sample accuracy, variable importance, and cross-validated error.

At each stage, we present only the results that are most useful in assessing performance and deciding between algorithms. The R code that does the calculation and produces this report is contained in the *knitr* document `MLProject.Rmd`. Fuller output is contained in the html file `MLProject-working.html`.

CART model (recursive partitioning)

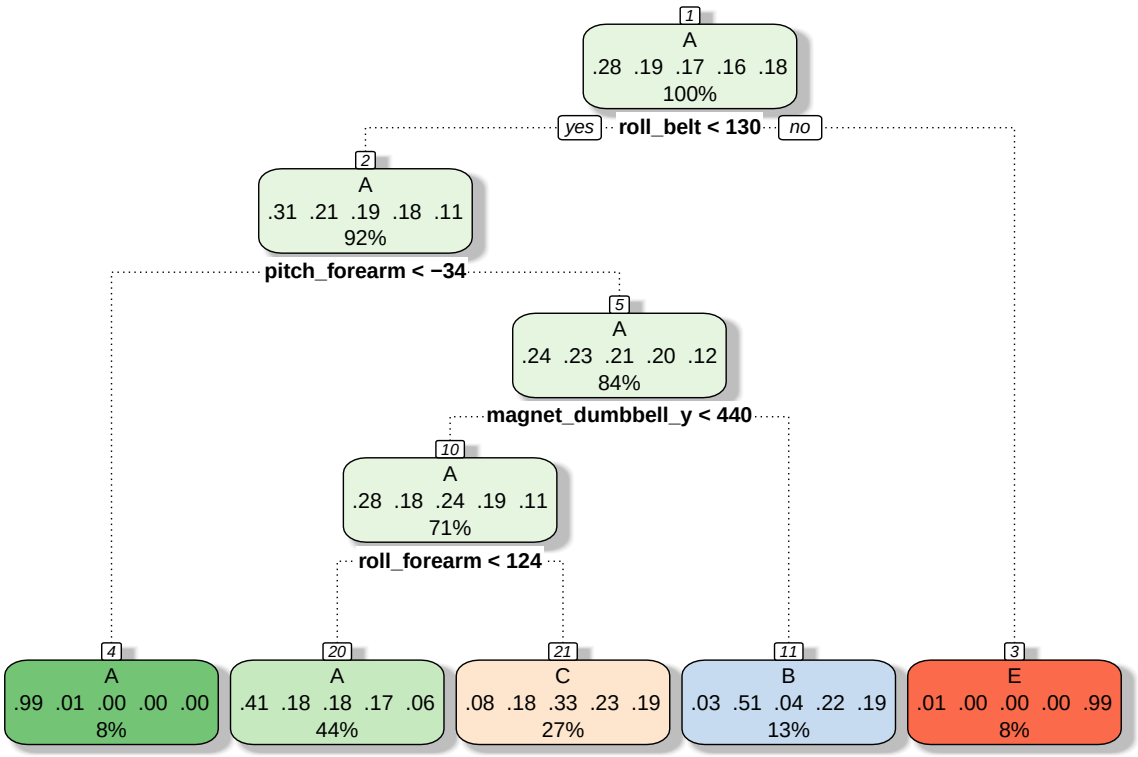
Our first machine learning algorithm is the simple CART model of recursive partioning.

```
set.seed(9943413)
ptm <- proc.time()
model1 <- train(classe ~ .,
                data=tidyTrain, method="rpart")
confuse1 <- confusionMatrix(predict(model1),tidyTrain$classe)
```

The confusion matrix (table of predicted vs actual classes) is

```
##           Reference
## Prediction    A    B    C    D    E
##          A 4981 1551 1550 1419  516
##          B   80 1259  108  556  476
##          C  397  908 1694 1172  947
##          D    0    0    0    0    0
##          E   13    0    0    0 1589
```

The figure below shows the CART algorithm based on the training data set.



The class-specific accuracies ("Positive predictive values") are given in the table below. Note that, since no cases were predicted to be in Class D, the accuracy is undefined (that is, 0 correct out of 0 predictions).

```
## Class: A Class: B Class: C Class: D Class: E
##   0.4973   0.5079   0.3310      NaN   0.9919
```

The overall accuracy is 0.4956, so the error rate is 0.5044.

The main advantage of the CART model is that it is easy to understand and display, as the Figure above indicates. However, the accuracy of the CART model is poor (less than 50%), and none of the cases are correctly classified into class D. The time needed to fit this model was -1 minute, 54 seconds.

k-nearest neighbor model

As a second potential machine-learning algorithm, we considered a k-nearest neighbor classifier. Before we carry out this calculation, we standardize the predictors so that they are not on different scales (thereby giving each predictor an equal weight in the distance calculations when calculating "nearness").

```
set.seed(9341355)
ptm2 <- proc.time()
model2 <- train(classe ~ ., data=tidyTrain,
                preProcess=c("center", "scale"), method="knn")
confuse2 <- confusionMatrix(predict(model2),tidyTrain$classe)
```

The confusion matrix (table of predicted vs actual classes) is

```
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 5461   26    1    2    1
##          B    8 3661    8    1    6
##          C    0   26 3328   43    5
##          D    1    4   14 3097    3
##          E    1    1    1    4 3513
```

The class-specific accuracies ("Positive predictive values") are given in the table below. .

```
## Class: A Class: B Class: C Class: D Class: E
##   0.9945   0.9938   0.9782   0.9929   0.9980
```

The overall accuracy is 0.9919, so the error rate is 0.0081.

Note the generally good predictive performance (over 99%). However, the computation time needed to fit the model (-2 minutes, 8 seconds) was substantial.

Because this model is a substantially better classifier than the CART model, we did not consider the CART model further.

Random forests

Our third candidate machine learning algorithm was the random forest algorithm of Leo Breiman. This algorithm creates a large collection ("ensemble") of trees which are combined as the end result of the algorithm. Our algorithm uses 400 trees. In the case of classification problems such as this one, each of the trees gets one "vote" as to the correct classification for a case, and the majority vote becomes the prediction. An important feature of random forests is that multiple trees are calculated using bootstrap selection of cases (and bootstrap selection of variables as each node of the tree is grown). As a result, for each tree, multiple cases are left out of the computation. These are called "out of bag" cases, and the predictions for the out-of-bag cases is used to calculate a projected error rate for new (out of sample) cases. Because they were not part of the original training process for the tree, they provide an unbiased estimate of error rates.

```
set.seed(9434193)
ptm <- proc.time()
model3 <- randomForest(classe ~ ., data=tidyTrain, ntree=400)
confuse3 <- confusionMatrix(predict(model3),tidyTrain$classe)
```

The confusion matrix (table of predicted vs actual classes) is

```
##           Reference
## Prediction    A    B    C    D    E
##          A 5469   10    0    0    0
##          B    1 3705   10    0    0
##          C    0    3 3338   23    0
##          D    0    0    4 3122    3
##          E    1    0    0    2 3525
```

The class-specific accuracies ("Positive predictive values") are given in the table below. .

```
## Class: A Class: B Class: C Class: D Class: E
##   0.9982   0.9970   0.9923   0.9978   0.9991
```

The overall ("OOB") accuracy is 0.997, so the error rate is 0.003.

In-sample prediction accuracy is very good—over 99%. The estimated out-of-sample prediction error rate ("OOB estimate of error rate") is a remarkable 0.3%. [We examine how much to believe this below.]

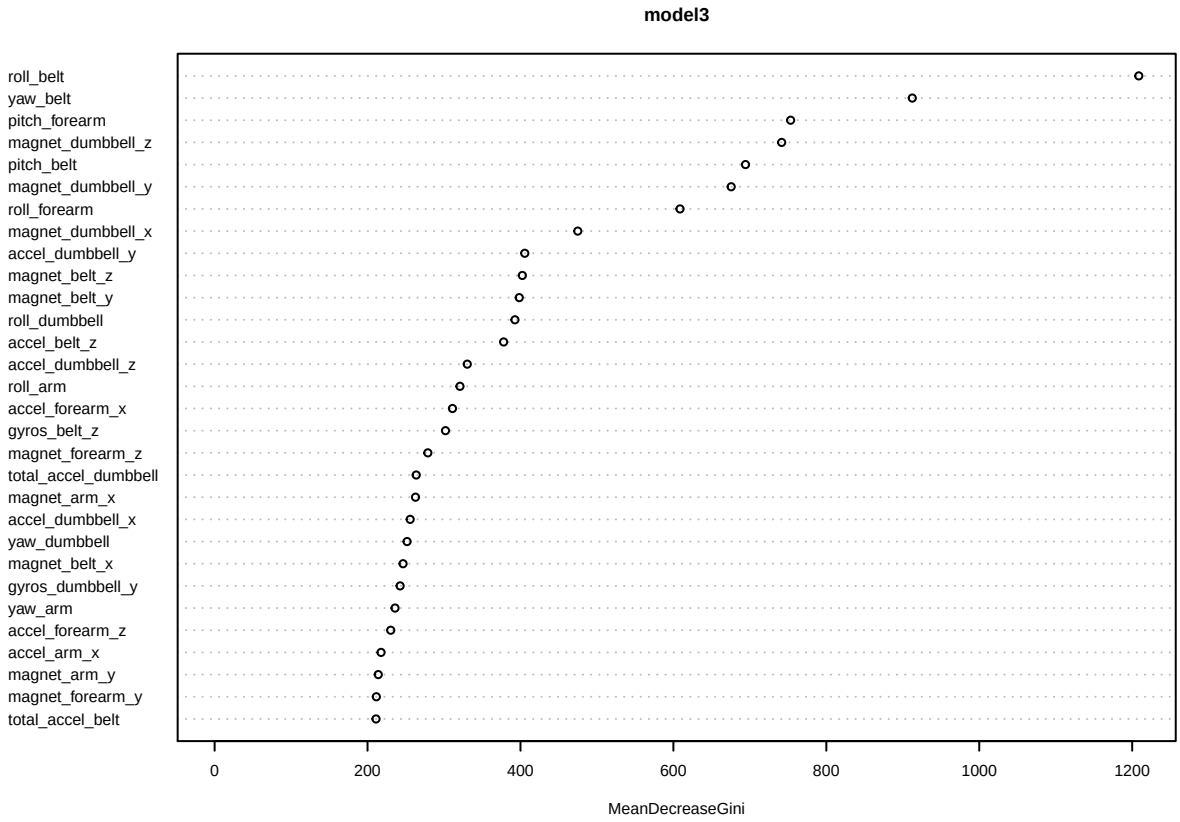
The elapsed computation time of -36 minutes and
55 seconds is much more reasonable than for the k-nn algorithm, and the accuracy of the random forest is slightly better. In addition, the predicted classes for the test data set were identical for k-nn and random forests. For these reasons, we selected the random forest algorithm as our final model. We examine some of its properties below.

Relative importance of variables Breiman suggested an approach for determining the relative importance of individual predictors in the final random forest ensemble. The calculations below list the top 15 (of 52) predictors and the display gives a graphical depiction of the variables' relative importance for prediction.

```
idx <- order(-varImp(model3))
foo<-varImp(model3)[idx,]
bar <- data.frame(var=attributes(varImp(model3))$row.names[idx], foo)
head(bar, 10)
```

```
##                  var    foo
## 1          roll_belt 1208.6
## 2            yaw_belt  912.4
## 3      pitch_forearm  753.3
## 4  magnet_dumbbell_z  741.6
## 5         pitch_belt  694.3
## 6  magnet_dumbbell_y  675.6
## 7       roll_forearm  608.7
## 8  magnet_dumbbell_x  474.9
## 9   accel_dumbbell_y  405.6
## 10     magnet_belt_z  402.5
```

```
varImpPlot(model3, cex=0.5)
```

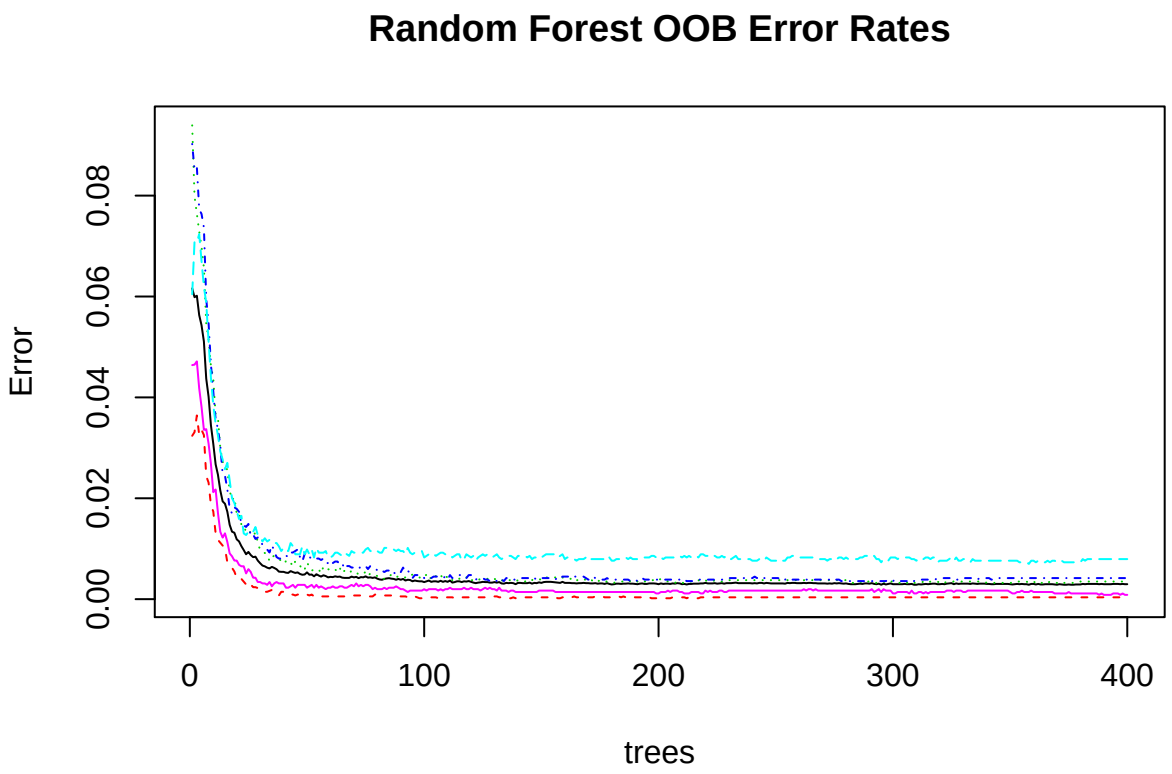


Out of sample error for random forest model

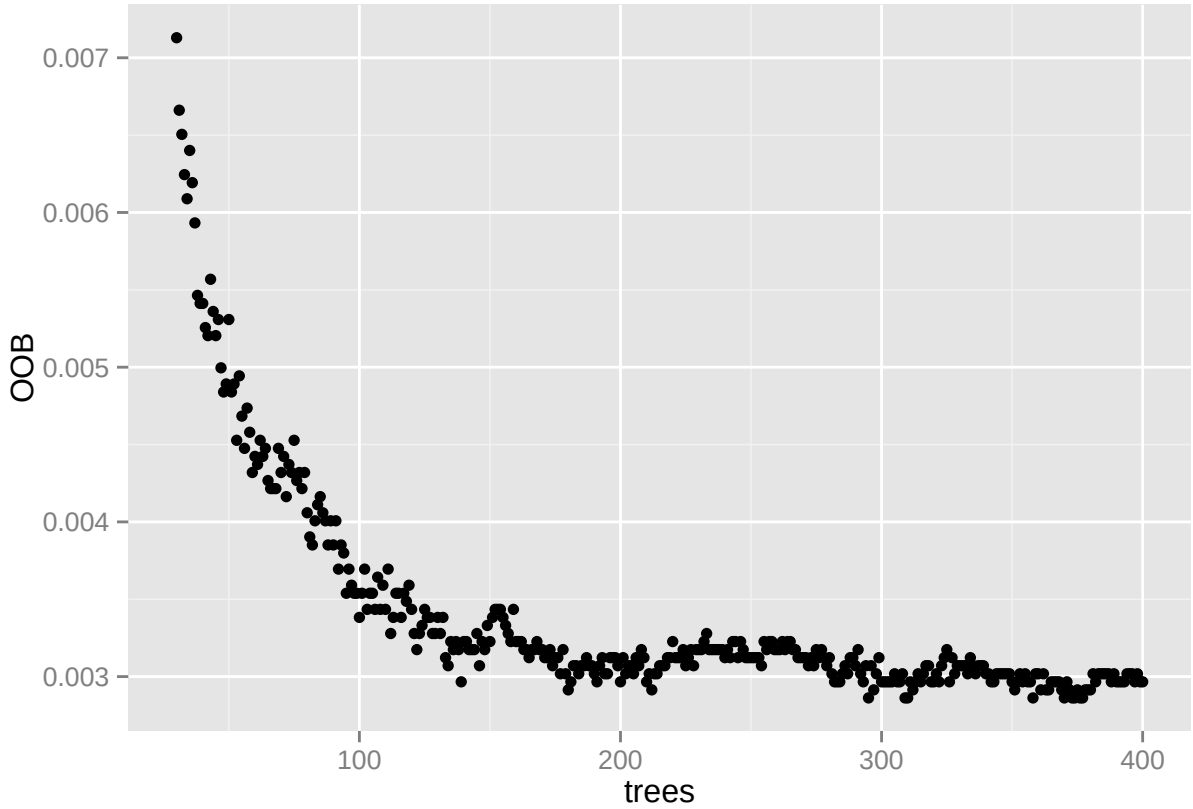
The calculations below show out-of-bag (OOB) error rates for random forests built on varying number of trees. The error rates are shown for each class, as well as an overall (averaged) out-of-bag error rate. The final model above was calculated using 400 trees, by which time the error rate has stabilized; indeed, 200 trees probably suffices.

```
##       ntree      OOB        A        B        C        D        E
##  [1,]     1 0.061606 0.032387 0.093931 0.090238 0.060449 0.046404
##  [2,]     2 0.059854 0.033119 0.079839 0.085117 0.070866 0.046490
##  [3,]     5 0.054289 0.033562 0.069851 0.076356 0.067021 0.038098
##  [4,]    10 0.030813 0.017391 0.043774 0.041166 0.038486 0.021210
##  [5,]    30 0.007129 0.001828 0.009683 0.011933 0.012393 0.003401
##  [6,]   100 0.003383 0.000183 0.004572 0.004177 0.008262 0.001984
##  [7,]   200 0.002966 0.000183 0.003765 0.003878 0.007944 0.001134
##  [8,]   300 0.002966 0.000366 0.003497 0.003580 0.008262 0.001134
##  [9,]   400 0.002966 0.000366 0.003497 0.004177 0.007944 0.000850
```

The first plot shows the error rates for each class of activity. Note that the plot has five curves, one for each of the five activity classes. Class D is hardest to predict with error rate about 0.8%. This is still remarkably good. Class E is easiest to predict, with error rates of about 0.1%. The other classes have error rates of about 0.3%.



The plot below shows, for random forest models with 30 or greater trees, the overall out-of-bag error estimates. Note that by 200 trees the prediction errors have stabilized at about 0.3%. Our model uses 400 trees.



Because these are unbiased estimates based on out-of-bag samples, that is, observations in the training set that were held out at each stage of the random forest creation, they give a good indication of the error rates

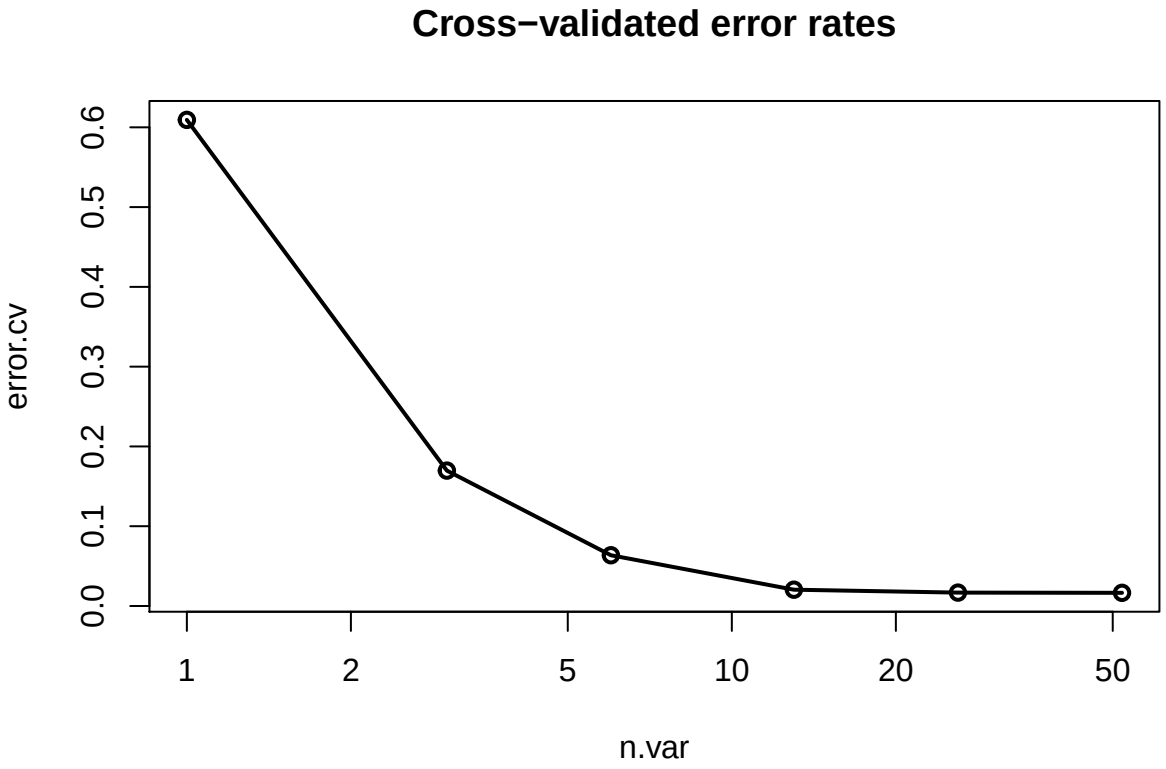
that could be expected in out-of-sample classification. I would expect classification error rates in test samples generated in the same manner as the training samples to be about 0.3–0.5%.

Cross validation for random forest model

The random forest algorithm uses cross validation as an intrinsic component of its tree-building, so in some sense, cross validation has already been done.

To get some idea of the effect of number of variables used to construct the random forest, we looked at a 40% sample of the training set and did cross validation. Note that the error rates in this plot are estimated based on only 40% of the data, and are therefore higher than the error rates obtained by training on the full data set as we do for our final model.

```
##      52      26      13       6       3       1
## 0.01652 0.01678 0.02042 0.06360 0.16972 0.60931
```



Accuracy of OOB error-rate predictions

To show that the OOB error rates are good forecasts of (new) out-of-sample error rates, I again used the split training data set that I divided into two pieces: a 40% sample on which I re-ran the random forest procedure, and the remaining 60% of the original training set to use as a "test set" with known true classes that I will use to validate the error-rate estimates.

The table below compares the OOB estimated error rates for each class for a new sample, based on the 40% of the original training set, with the actual error rates when the random forest is applied to the 60% of the data that was held out.

```
##           Overall Class: A Class: B Class: C Class: D Class: E
## actual    0.01206 0.007599  0.01923  0.02704 0.004852 0.003314
## predicted 0.01092 0.004111  0.01546  0.01268 0.023828 0.003541
```

Although for some classes the actual error rates are as much as twice the predicted rates, the overall accuracy is quite good, and corresponds closely to the actual error rates observed in the held-out sample.