Qualitative activity recognition

FDJ

February 26, 2016

Executive Summary

The "Weight Lifting Exercises Dataset" from http://groupware.les.inf.puc-rio.br/har was used to train a "RandomForest" machine learning algorithm to predict the quality of performance of a specific dummbell exercise in 5 categories (A = exactly according to the specification, B-E various types of mistakes in performance of the exercise).

The predictors were a set of 52(=4*13) "Inertial Measurement Unit" sensor data obtained from 4 sensors attached to the participant (arm, forearm, dumbbell and (waist)belt), each sensor contributing 13 variables.

The model's accuracy on the trial data was 99.546 % (all correct classifications divided by total amount of cases = 19622). Interestingly, the error rate seems to vary inversely with the amount of cases present in the training data (very small error rate for category A but about 40 x times higher for category D, 0.03584229 % vs 1.46144279 %, see below confusion matrix and Fig. 1)

Not all sensors seem equally important: Fig. 2 shows that in the first 15 most important variables, the arm sensor only appears one time (last line), suggesting that possibly a model could be built not using this sensor at all.

The machine learning algorithm was applied to the 20 test cases available in the test data. All 20 test cases were predicted correctly with the model (data not shown, as they are usd to fill-in the assignment quiz)

Future directions: build the model to use only three of the sensors?

Data set

Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions (5 classes A-E):

- A = exactly according to the specification
- B = throwing the elbows to the front
- C = lifting the dumbbell only halfway
- D = lowering the dumbbell only halfway
- E = throwing the hips to the front

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

Structure of the data set

6 participants, data from 4 sensors (accelerometers on the belt, forearm, arm, and dumbell) per participant, 10 repetitions

Each sensor has a total of 38 items measured:

variables in dataset

- 1 = X = index?
- 2 = username
- 3 to 5 = timestamp data
- 6 to 7 = window related
- 4 blocks of 38 vars per sensor = 152 vars
- classe = classification of the exercise aa A to E

Each block of 38 vars consists of (13) sensor measurements, namely:

- roll, pitch, yaw, total_accel
- gyros_x, gyros_y, gyros_z
- accel_x, accel_y, accel_z
- magnet_x, magnet_y, magnet_z
- and of (25) derived measurements: (24) for roll, pitch, yaw (consisting of: kurtosis, skewness, max, min, amplitude, avg, stddev, var) and (1) for total_accel

The dataset has many rows where the derived measurements are not present, preventing the use of these measurements in model building Columns 1 to 7 equally do not seem relevant to prediction and where left out of the final training set.

Citation / Origin of data

The "Weight Lifting Exercises Dataset" was generously supplied by "Human Activity Recognition" academic group at http://groupware.les.inf.puc-rio.br/har. Citation: 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.

For the Coursera project data sets were downloaded as follows:

- training set: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
- test data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Data loading and predictor selection in R

```
#get the data, set NA correctly
fname <- "pml-training.csv"
ds <- read.csv(fname, na.strings = c("NA", "", "#DIV/0!"))
```

```
#reset some colums from "logical" to "numeric"
loginames<-c("kurtosis_yaw_belt", "skewness_yaw_belt", "kurtosis_yaw_dumbbell", "skewness_yaw_dumbbell"
lapply( c(1:6), function(x) {ds[loginames[x]]<<-as.numeric(unlist(ds[loginames[x]])); return(1)} )

#col numbers for cols with derived results
cols_derived_to_delete<-c(12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3

other_cols_to_delete<-c(1:7)
cols_to_delete<-c(other_cols_to_delete, cols_derived_to_delete)
cols_to_keep<-c(1:160)
```

```
tlogic<-cols_to_keep %in% cols_to_delete
ds_raw<-ds[, cols_to_keep[! tlogic] ]
dsd<-ds_raw

#remove cols with NA
dsd<-dsd[colSums(is.na(dsd)) == 0]
```

%==========

How we built the models

Model RF-OOB - Random forest with bagging (Out-Of-Bag OOB error rate)

We choose to implement a random forest model using the 13 variables per sensor as indicated above; thus we have 4*13=52 predictor variable for the dependent variable "classe", the exercise quality category. We used the train function from the caret package with the method "parRF", a parallell implementation of random forest to speed up execution. There is only one model tuning factor, "mtry", the number of variables randomly sampled as candidates at each (decision tree) split. The parameter "mtry" was left at its default value = (sqrt(p) ~ 7 where p is number of variables (52).

Random forest method combines Breiman's "bagging" idea and the random selection of features (the mtry parameter). The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging. Bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples. This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias.

As explained at https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#ooberr and on wikipedia https://en.wikipedia.org/wiki/Out-of-bag_error, in random forests, there is no need for cross-validation to get an unbiased estimate of the test set error. It is estimated internally, during the run.

The out-of-bag OOB-error rate is the mean prediction error on each training sample for row x, using only the trees that did not have row x in their bootstrap sample.The OOB error tends to level off after a number of trees have been fit (see Fig. 3).

How we used cross validation

As the OOB-error rate estimate is obtained during the model fitting procedure, there is no need to do separate cross validation (see above).

What we think the expected out of sample error is?

The OOB-error rate was 0.45% (see below) and is an estimate for the out of sample error rate. The OOB rate quickly decreases with increasing number of trees tested (see Fig. 3).

Justification of choices we made

see above

Model RF-CV - Random forest with cross-validation

To see the difference, a second random forest model was made, using cross-validation, using the same 13 variables per sensor as indicated above; thus we have 4*13=52 predictor variable for the dependent variable "classe", the exercise quality category. We used the train function from the caret package with the method "parRF", a parallell implementation of random forest to speed up execution.

How we used cross validation

The original data set is split up in a 60 % training data set and 40 % test data set (as per general guidelines for cross-validation); the RF-CV model is then trained on the training set.

What we think the expected out of sample error is?

The overall out-of-sample error (false) classification rate = 0.94 % (see below) when we apply the RF-CV model to the test data.

Compare RF-OOB with RF-CV

RF-OOB has OOB error rate estimate = 0.4535 % which compares favorably with the error rate (0.94 %) found with cross-validation for the RF-CV model. The better performance of the RF-OOB is probably due to the fact that it is trained on the whole data set.

Code: Results RF-OOB model (OOB error rate)

```
#load libs
library(caret)
```

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

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

```
library(randomForest)
```

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

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

```
#first model
set.seed(1956)
training<-dsd

#machine learning method
#parallell random forest
mlm="parRF"

#data
pdata=training
```

```r
#run separately in a terminal (takes > 30 mins)
#and save the model to disk (with var name modrfq)
mod_prf<-train(classe ~ ., method=mlm, data=pdata, importance = TRUE, do.trace = 50)
```

```r
#now just load from disk
load("mod_parRF_var_modrfq_001.Rdata")
mod_prf<-modrfq
print(mod_prf)
```

```
## Parallel Random Forest
##
## 19622 samples
##    52 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 19622, 19622, 19622, 19622, 19622, 19622, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9925990  0.9906372  0.001296168  0.001631792
##   27    0.9924195  0.9904093  0.001392158  0.001758891
##   52    0.9850777  0.9811204  0.003884601  0.004909248
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```r
#get the confusion matrix
fm<-mod_prf["finalModel"]
cfm<-fm[[1]]["confusion"]
```

```
## The confusion matrix shows correct / incorrect classifications
##
## $confusion
##      A    B    C    D    E  class.error
## A 5578    2    0    0    0 0.0003584229
## B   11 3781    5    0    0 0.0042138530
## C    0   17 3403    2    0 0.0055523086
## D    0    0   45 3169    2 0.0146144279
## E    0    0    0    5 3602 0.0013861935
```

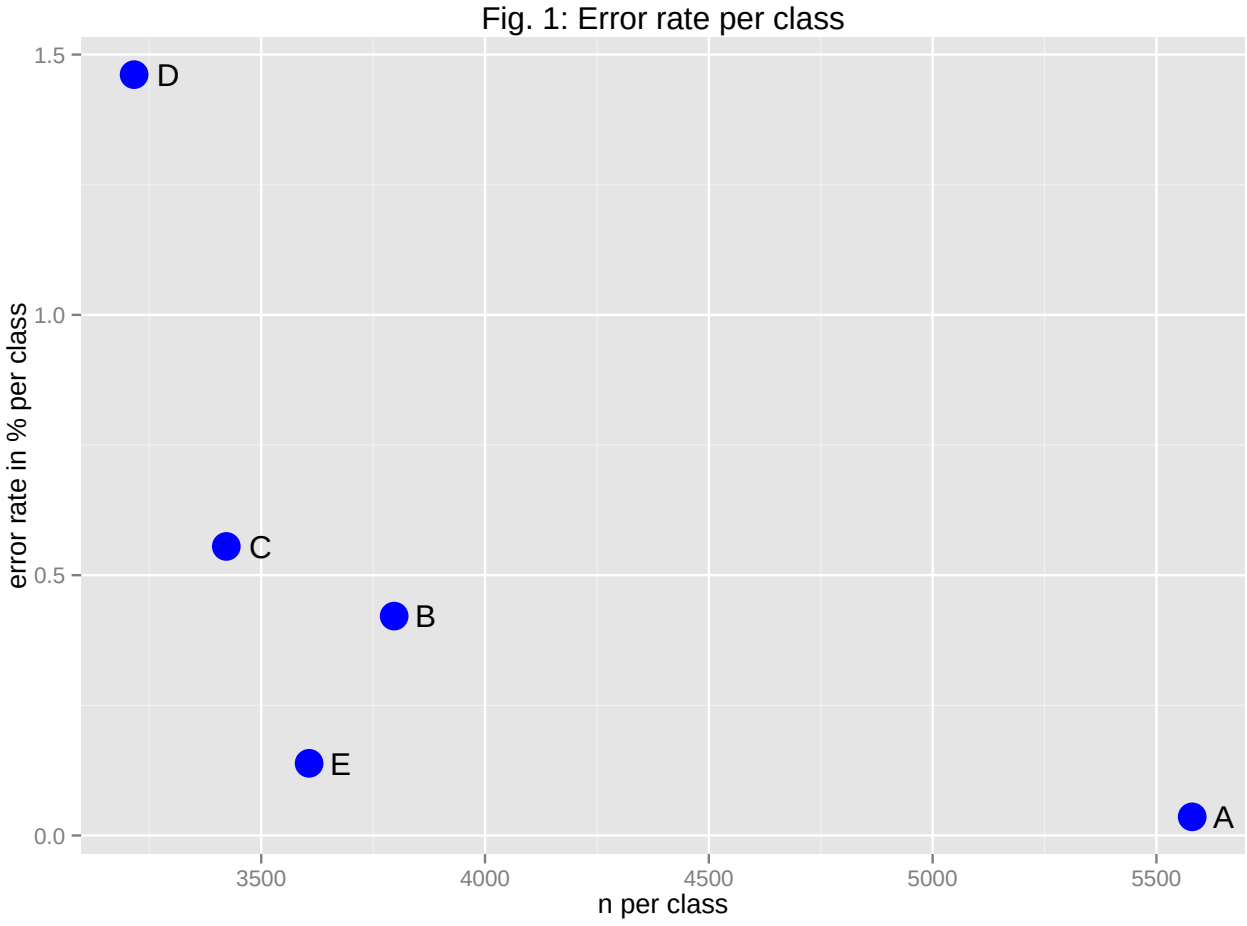
```r
ntotal<-sum(colSums(cfm$confusion[,1:5]))
#calculate all correct classifications, numbers on diagonal in cfm
sum<-0
lapply(c(1:5), function(x){sum<<-sum+cfm$confusion[x,x]})
correct_classifications<-sum
overall_correct_rate<-sum/ntotal
overall_wrong_classifications_rate <- (ntotal - correct_classifications)/ntotal
```

```
cat(sprintf("Accuracy rate is = %.6s %%\n",100*overall_correct_rate))
cat(sprintf("OOB error rate is = %.6s %%\n",100*overall_wrong_classifications_rate))
```

```
## Accuracy rate is = 99.546 %
## OOB error rate is = 0.4535 %
```

```
#plot error rate per class
n_per_class <- summary(dsd$classe)
err_per_class<-cfm$confusion[,6]

g <- ggplot(data.frame(npc=n_per_class, epc=100*(err_per_class)), aes(x = npc, y = epc ))
g <- g + geom_point( colour = "blue", size=6)
g <- g + xlab("n per class") + ylab("error rate in % per class") + ggtitle("Fig. 1: Error rate per class
g <- g + geom_text(label=c("A","B","C","D","E"), hjust=-1 )
print(g)
```

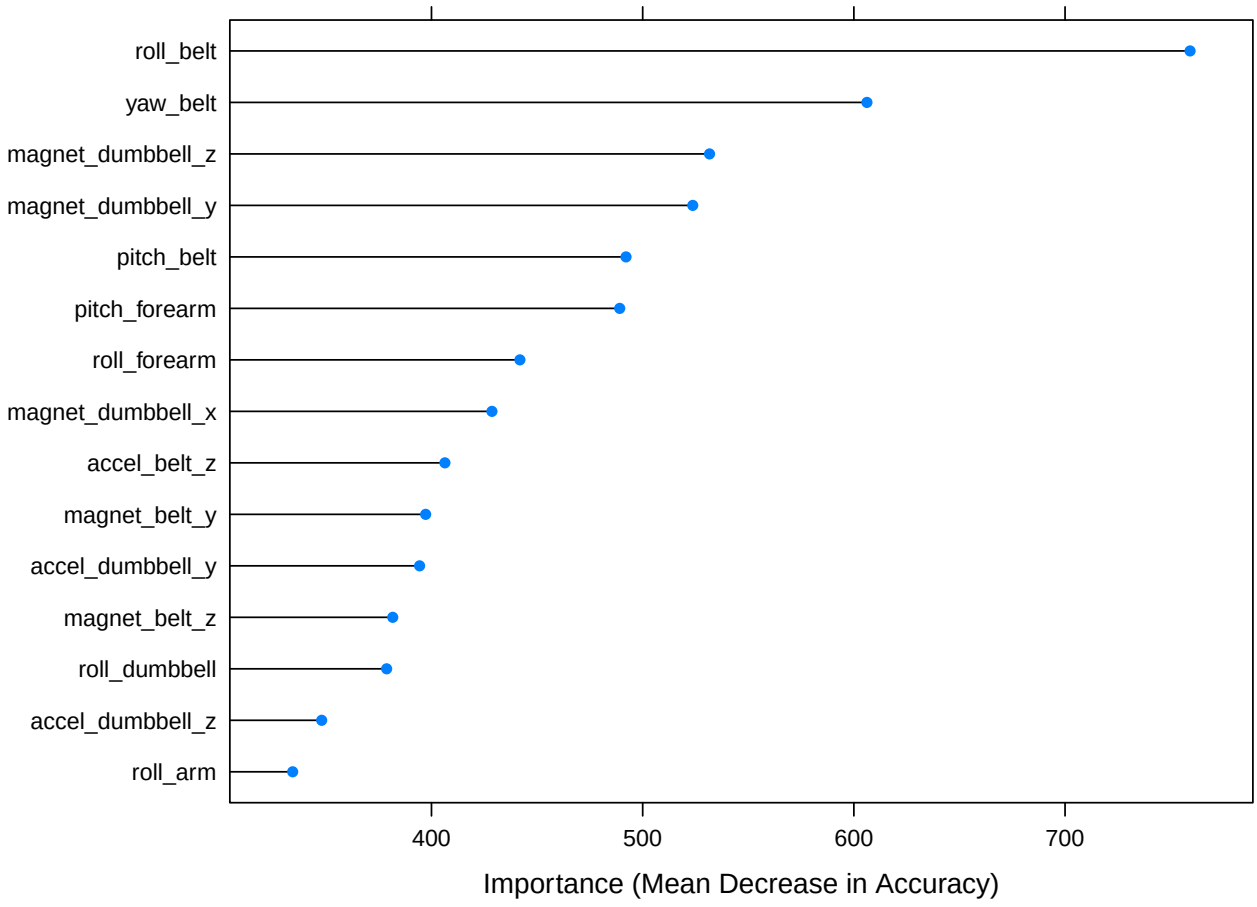



```
#plot variable importance
var_imp <- varImp(mod_prf, scale = FALSE)
```

```
## Loading required package: e1071
```

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

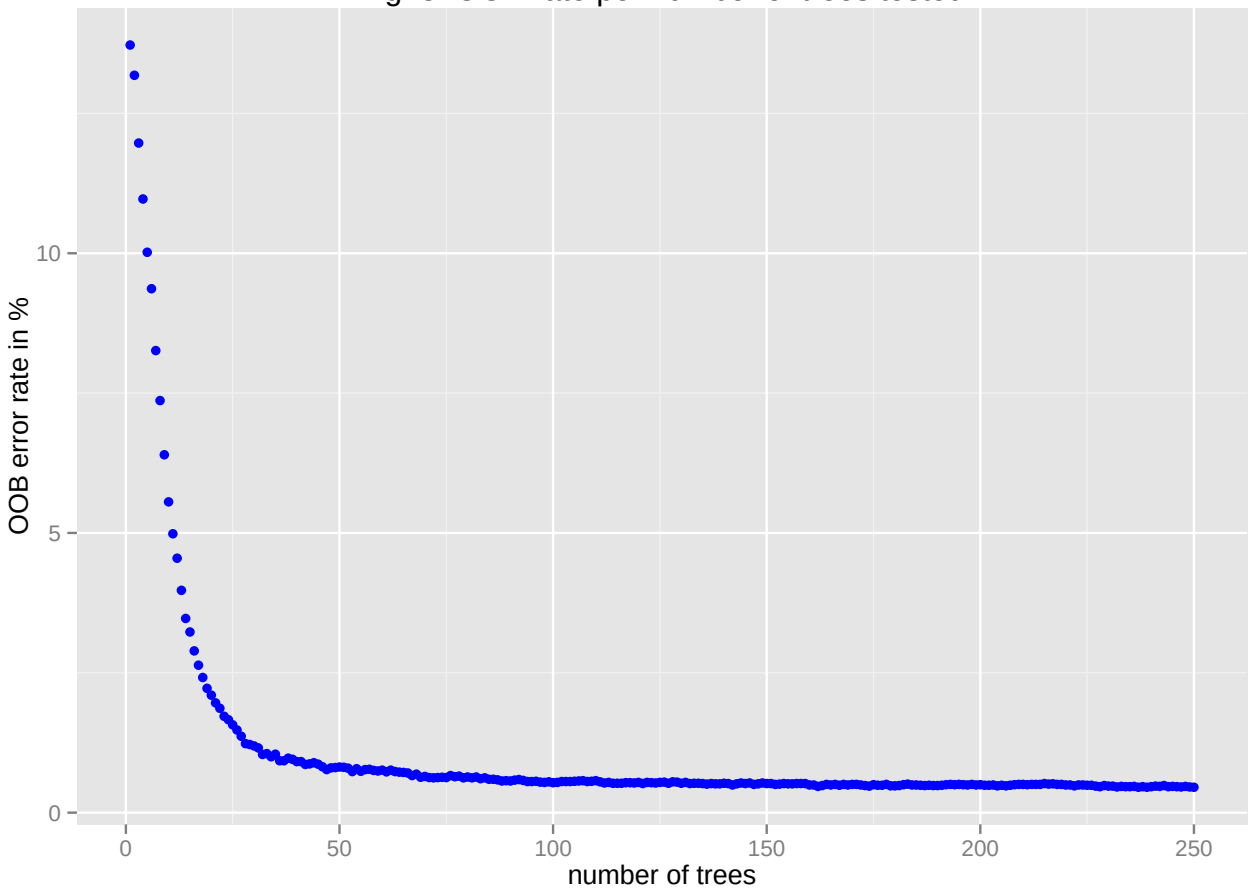
```
plot(var_imp, top = 15, main = "Fig. 2: Variable-Importance Plot", xlab = "Importance (Mean Decrease in
```




```
#plot oob-err as function of number of trees
ntree<-unlist(fm[[1]]["ntree"])
trees_idx<-c(1:ntree)
err_vec<-fm[[1]]["err.rate"]
oob_errs<-err_vec[[1]][,1]

g <- ggplot(data.frame(tidx=trees_idx, oer=100*oob_errs), aes(x = tidx, y = oer ))
g <- g + geom_point( colour = "blue", size=2)
g <- g + xlab("number of trees") + ylab("OOB error rate in %") + ggtitle("Fig. 3: OOB rate per number o
print(g)
```

Fig. 3: OOB rate per number of trees tested



Code: Results RF-CV model (using cross-validation)

```
#uses  2 cores
library(doParallel)
```

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

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

```
options(cores=2)
registerDoParallel()
set.seed(1956)

#split dsd into train = 60% test = 40% set
trainIndex <- createDataPartition(dsd$classe, p = .6, list = FALSE, times = 1)
dsTrain <- dsd[ trainIndex,]
dsTest  <- dsd[-trainIndex,]

#in script run in terminal
#save(dsTest, file="var_dsTest.Rdata")
```

```
#set train control
# 10-fold CV, repeated ten times
ptunelength=10
trc <- trainControl(method = "repeatedcv", number = 10, repeats = 10)

#metric
pmetric="Accuracy"

#machine learning method
mlm="parRF"

#mk train test data
#training<-dsd
pdata=dsTrain
```

```
#run in terminal and save
modrfcv<-train(classe ~ ., method=mlm, data=pdata, trControl=trc, tunelenght=ptunelength, ntree=1000,
```

```
#now just load the model from disk as var=modrfcv
load("mod_parRF_var_modrfcv_003.Rdata")

mod<-modrfcv

#make predictions for dsTest data
pred_mod<-predict(mod, newdata=dsTest)

cfm<-confusionMatrix(pred_mod, dsTest$classe)

#confusion mtx predicts
cat("Model RF-CV: Test Set Predicted data Confusion matrix\n")
print(cfm)
cat("\n\n")

#function to calculate overall true / false classifications
otfclass<-function(cfm) {

  nr<-nrow(cfm$table)
  nc<-ncol(cfm$table)

  #true classifications is along diagnal

  vecT<-vector()
  for ( idx in c(1:nr) ) {
    vecT<-c(vecT, cfm$table[idx,idx])
  }
  #the false classifications are along column as col = REF
  allTF<-colSums(cfm$table)
  vecF<-allTF-vecT

  rateT<-sum(vecT)/sum(allTF)
  rateF<-sum(vecF)/sum(allTF)

  rateT<-sum(vecT)/sum(allTF)
```

```
  rateF<-sum(vecF)/sum(allTF)

  cat(sprintf("Overall true classification rate = %4.2f %%\n", rateT*100))
  cat(sprintf("Overall false classification rate = %4.2f %%\n\n", rateF*100))
}

cat("Model RF-CV classification rates\n")
otfclass(cfm)
```

```
## Model RF-CV: Test Set Predicted data Confusion matrix
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2226   12    0    0    0
##          B    6 1495    9    0    1
##          C    0   11 1351   21    2
##          D    0    0    8 1264    3
##          E    0    0    0    1 1436
##
## Overall Statistics
##
##                Accuracy : 0.9906
##                  95% CI : (0.9882, 0.9926)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9881
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9973   0.9848   0.9876   0.9829   0.9958
## Specificity            0.9979   0.9975   0.9948   0.9983   0.9998
## Pos Pred Value         0.9946   0.9894   0.9755   0.9914   0.9993
## Neg Pred Value         0.9989   0.9964   0.9974   0.9967   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2837   0.1905   0.1722   0.1611   0.1830
## Detection Prevalence   0.2852   0.1926   0.1765   0.1625   0.1832
## Balanced Accuracy      0.9976   0.9912   0.9912   0.9906   0.9978
##
##
## Model RF-CV classification rates
## Overall true classification rate = 99.06 %
## Overall false classification rate = 0.94 %
```