Qualitative Activity Recognition/Prediction Using Machine Learning Models

parafac

Wednesday, March 10, 2015

Executive Summary

In this study, our goal is to use data from accelerometers to classify and predict the quality of activities performed. we build four basic machine learning models including recursive partitioning decision tree model (rpart), random forest model (rf), gradient boosted model (gbm), and Linear Discriminant Analysis model (lda). The random forest model and the gradient boosting model give the best accuracy on the validation data set, they also give the same predictions when applied to the testing data set. Future work may include cross validation and pre-processing with principal component analysis (PCA).

Introduction

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. And human activity recognition research has traditionally focused on discriminating between different activities, i.e. to predict "which" activity was performed at a specific point in time, but they rarely quantify how well they do it [2].

In this study, our goal is to use data from accelerometers to classify and predict the quality of activities performed. We perform exploratory data analysis, experiment with several machine learning algorithms, and derive conclusion from our analysis.

Data Source and Description

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) [1] [2].

The data were collected from accelerometers of 6 young male health participants, aged between 20-28 years, with little weight lifting experience. They were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in 5 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). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. [2]

The training data for this project are available here:

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

The test data are available here:

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

The R code for downloading the data

Software and Required Library

We use R version 3.1.1 and RStudio 0.98 on a 64-bit Windows 7 computer. The R packages that is required for the study include

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

```
## Warning: package 'rpart.plot' was built under R version 3.1.2
```

```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.1.2
```

Data Cleaning and Exploratory Data Analysis

The most difficult part of this study is data cleaning, that is, choosing the right set of variables for performing machine learning algorithms and predictions. The training data and the testing data are in nice csv format. They can be read in with the read.csv() function with little effort.

```
trainingData <- read.csv("./pml-training.csv", header=TRUE, sep=",", na.string=c("", " ", "NA", "#DI
V/0!"))
testingData  <- read.csv("./pml-testing.csv", header=TRUE, sep=",", na.string=c("", " ", "NA", "#DI
V/0!"))
```

There are 160 fields(columns) and 19622 records in the training data set. Not every variable is useful in predicting the quality of the activities, besides there are many missing values in the data. The process outlined here is the result of many trials. The may not be the best choice of variables, better alternative is possible if more time or better guideline is available. Pre-processing techniques such as the principal component analysis (PCA) may be useful, too.

First, we use the nearZeroVar() function to diagonse predictors that may have one or very few unique values (near zero variance).

```
nsv <- nearZeroVar(trainingData, saveMetric=TRUE)
sum(nsv$nzv)
```

```
## [1] 36
```

We manually inspect the output from nearZeroVar(), there are 36 near zero variance columns. We remove those predictors and a few other related predictors. The same process is also applied to the testing data set.

```
trainingSet <- trainingData[,!nsv$nzv]
testingSet  <- testingData[,!nsv$nzv]

jcols <- grep("amplitude_|avg_|kurtosi_|ls_|max_|min_|skewness_|stddev_|var_", names(trainingSet))

trainingSet <- trainingSet[,-jcols]
testingSet  <- testingSet[,-jcols]

dim(trainingSet)
```

```
## [1] 19622    68
```

```
dim(testingSet)
```

```
## [1] 20 68
```

There are 68 predictors remain after this process.

Since there are many missing values in the data, we decide to keep only those columns that have less than 20% of missing values, that is, columns with 80% qualified entries. We also remove the first column because it only represents the row index which is not useful in prediction. The same process is also applied to the testing data set. Furtherrmore, the classe variable (outcome variable) in the training set is converted into factor variable.

We make sure the training data set and the testing data set have the same varialbes except the last column which is the quality of activities we are going to predict.

```
colIdx <- which(colSums(is.na(trainingSet))/nrow(trainingSet) <= 0.20)
trainingSet <- trainingSet[, colIdx]
testingSet  <- testingSet[, colIdx]

trainingSet <- trainingSet[,-1]
testingSet  <- testingSet[,-1]

trainingSet$classe <- as.factor(trainingSet$classe)

dim(trainingSet)
```

```
## [1] 19622    58
```

```
dim(testingSet)
```

```
## [1] 20 58
```

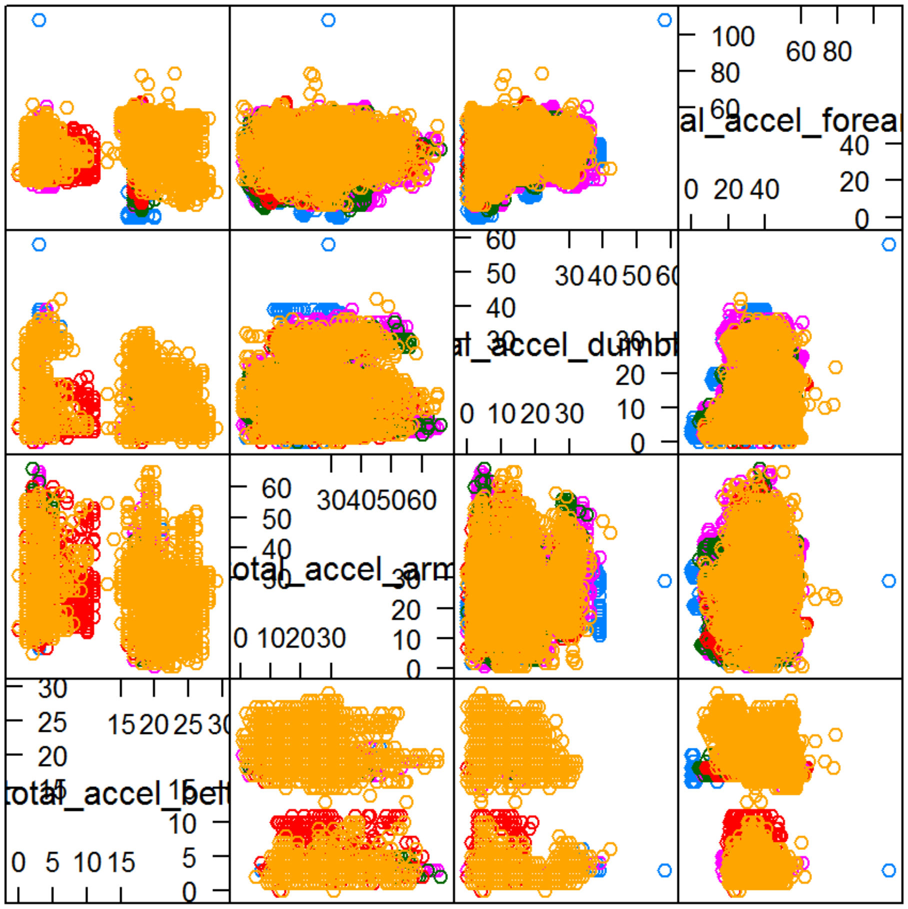
```
sum(names(trainingSet) != names(testingSet))
```

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

At the end of data cleaning and selection, we keep 58 columns (57 predictors and one outcome variable).

We plot the pairwise relationships between predictors “total_accel_belt”, “total_accel_arm”, “total_accel_dumbbell”, and “total_accel_forearm” using featurePlot() function.

```
featurePlot(x=trainingSet[,c("total_accel_belt", "total_accel_arm", "total_accel_dumbbell","total_acc
el_forearm")],
            y=trainingSet$classe, plot="pairs")
```



Scatter Plot Matrix

It’s hard to make conclusion from these plots.

Data Preparation and Splitting

We split the training data set into two parts: 60% for building the models (input into machine learning algorithms),
and 40% for testing the accuracy of the prediction models.

```
set.seed(3245)
inTrain <- createDataPartition(trainingSet$classe, p = 0.6)[[1]]
myTraining <- trainingSet[inTrain,]
myTesting  <- trainingSet[-inTrain,]
```

Machine Learning Models

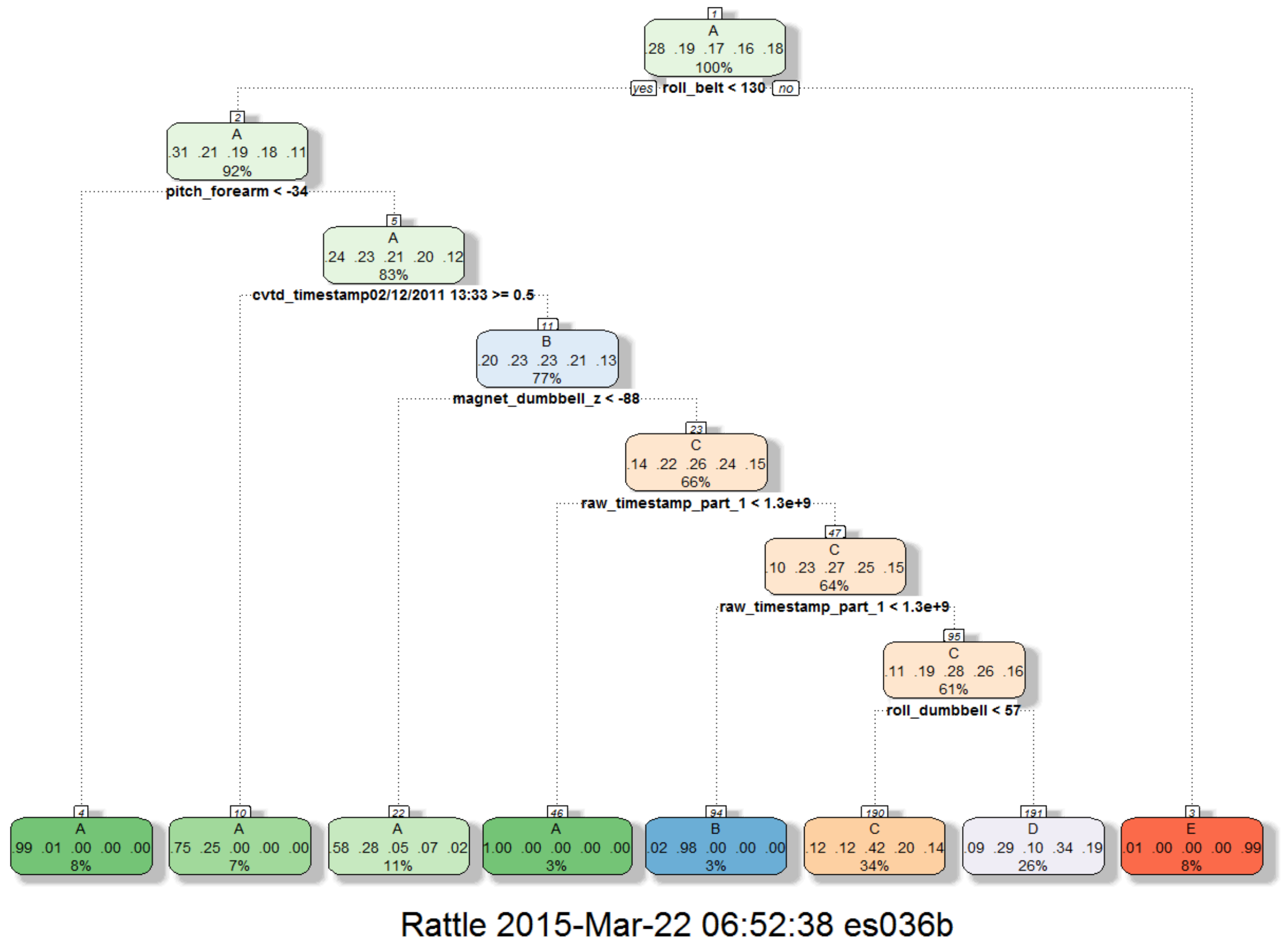
Recursive Partitioning Model (rpart)

We first the decision tree model using recursive partitioning algorithm. The decision tree model is usually easy to interprete the results.

```
modelFit_rpart <- train(classe ~ ., data=myTraining, method="rpart")
pred_rpart <- predict(modelFit_rpart, newdata=myTesting)
accuracy_rpart <- sum(myTesting$classe == pred_rpart)/length(myTesting$classe)
print(accuracy_rpart)
```

```
## [1] 0.5573541
```

```
fancyRpartPlot(modelFit_rpart$finalModel)
```



Unfortunately, we only get 56% of accuracy when applied the model to the validation set. The poor result may be related to the predictor selection.

Random Forest Model (rf)

Then we build a random forest model and test the prediction accuracy. It takes a very long time to build the model on a laptop with Core i5 CPU and 8GB of memory.

```
modelFit_rf <- train(classe ~ ., data=myTraining, method="rf")
```

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

```
pred_rf <- predict(modelFit_rf, newdata=myTesting)
accuracy_rf <- sum(myTesting$classe == pred_rf)/length(myTesting$classe)
print(accuracy_rf)
```

```
## [1] 0.9992353
```

Gradient Boosted Model (gbm)

Then we build a gradient boosted model. It also takes a long time to train this model.

```
modelFit_gbm <- train(classe ~ ., data=myTraining, method="gbm", verbose=FALSE)
```

```
## Loading required package: gbm
```

```
## Warning: package 'gbm' was built under R version 3.1.2
```

```
## Loading required package: survival
## Loading required package: splines
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: parallel
## Loaded gbm 2.1
## Loading required package: plyr
```

```
pred_gbm <- predict(modelFit_gbm, newdata=myTesting)
accuracy_gbm <- sum(myTesting$classe == pred_gbm)/length(myTesting$classe)
print(accuracy_gbm)
```

```
## [1] 0.9966862
```

Linear Discriminant Analysis (lda)

Finally we build the linear discriminant analysis model (lda).

```
modelFit_lda <- train(classe ~ ., data=myTraining, method="lda")
```

```
## Loading required package: MASS
```

```
## Warning: package 'MASS' was built under R version 3.1.3
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
pred_lda <- predict(modelFit_lda, newdata=myTesting)
accuracy_lda <- sum(myTesting$classe == pred_lda)/length(myTesting$classe)
print(accuracy_lda)
```

```
## [1] 0.8553403
```

Prediction Results

After we train the models, we apply these models to the testing data.

```
prediction_rpart <- predict(modelFit_rpart, newdata=testingSet)
print(prediction_rpart)
```

```
##  [1] C C C A A D C D A A D C B A C D C D C B
## Levels: A B C D E
```

```
prediction_rf <- predict(modelFit_rf, newdata=testingSet)
print(prediction_rf)
```

```
##  [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
```

```
prediction_gbm <- predict(modelFit_gbm, newdata=testingSet)
print(prediction_gbm)
```

```
##  [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
```

```
prediction_lda <- predict(modelFit_lda, newdata=testingSet)
print(prediction_lda)
```

```
##  [1] B B B A A E D C A A B C B A E E A B B B
## Levels: A B C D E
```

From the accuracy results in the previous section, the rf model and the gbm model have the highest precision when applied to the validation set (myTesting). In fact, both models give the same predictions to the testing data. We write the results from gbm model to external files.

```
write_prediction = 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)
    }
}

write_prediction(prediction_gbm)
```

Conclusion

In this study, we used data from accelerometers to classify and predict the quality of activities performed. Four machine learning models were perfomed on the training data, they are recursive partitioning decision tree model (rpart), random forest model (rf), gradient boosted model (gbm), and Linear Discriminant Analysis model (lda). We found the gbm model gave the best accuracy on the validation data set, we applied it to the testing data set.

Only the basic models were tested. Future work should include cross validation techniques such as k-fold cross validation and leave one out cross validation, better pre-processing techniques such as the principal component analysis (PCA), and other machine learning algorithms such as support vector machine (SVM).

References

[1] 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.

[2] http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

Appendix 1. Confusion Matrix Output

```
confusionMatrix(myTesting$classe, pred_rpart)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1670    3  371  181    7
##          B  373  247  351  547    0
##          C   36    2 1127  203    0
##          D   61    0  547  678    0
##          E   19    0  362  410  651
## 
## Overall Statistics
## 
##                Accuracy : 0.5574          
##                  95% CI : (0.5463, 0.5684)
##     No Information Rate : 0.3515          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.4443          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7735  0.98016   0.4086  0.33581  0.98936
## Specificity            0.9012  0.83263   0.9526  0.89566  0.88996
## Pos Pred Value         0.7482  0.16271   0.8238  0.52722  0.45146
## Neg Pred Value         0.9129  0.99921   0.7482  0.79558  0.99891
## Prevalence             0.2752  0.03212   0.3515  0.25733  0.08386
## Detection Rate         0.2128  0.03148   0.1436  0.08641  0.08297
## Detection Prevalence   0.2845  0.19347   0.1744  0.16391  0.18379
## Balanced Accuracy      0.8373  0.90639   0.6806  0.61573  0.93966
```

```
confusionMatrix(myTesting$classe, pred_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    1 1517    0    0    0
##          C    0    2 1364    2    0
##          D    0    0    0 1286    0
##          E    0    0    0    1 1441
##
## Overall Statistics
##
##                Accuracy : 0.9992
##                  95% CI : (0.9983, 0.9997)
##     No Information Rate : 0.2846
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.999
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9987   1.0000   0.9977   1.0000
## Specificity            1.0000   0.9998   0.9994   1.0000   0.9998
## Pos Pred Value         1.0000   0.9993   0.9971   1.0000   0.9993
## Neg Pred Value         0.9998   0.9997   1.0000   0.9995   1.0000
## Prevalence             0.2846   0.1936   0.1738   0.1643   0.1837
## Detection Rate         0.2845   0.1933   0.1738   0.1639   0.1837
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9998   0.9993   0.9997   0.9988   0.9999
```

```
confusionMatrix(myTesting$classe, pred_gbm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    1 1512    3    2    0
##          C    0    2 1352   14    0
##          D    0    0    0 1286    0
##          E    0    0    0    4 1438
##
## Overall Statistics
##
##                Accuracy : 0.9967
##                  95% CI : (0.9951, 0.9978)
##     No Information Rate : 0.2846
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9958
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9987   0.9978   0.9847   1.0000
## Specificity            1.0000   0.9991   0.9975   1.0000   0.9994
## Pos Pred Value         1.0000   0.9960   0.9883   1.0000   0.9972
## Neg Pred Value         0.9998   0.9997   0.9995   0.9970   1.0000
## Prevalence             0.2846   0.1930   0.1727   0.1665   0.1833
## Detection Rate         0.2845   0.1927   0.1723   0.1639   0.1833
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9998   0.9989   0.9977   0.9923   0.9997
```

```
confusionMatrix(myTesting$classe, pred_lda)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2025  187   20    0    0
##          B  184 1138  188    8    0
##          C    4  147 1169   46    2
##          D    0    4  139 1073   70
##          E    0    0    4  132 1306
## 
## Overall Statistics
##                                           
##                Accuracy : 0.8553          
##                  95% CI : (0.8474, 0.8631)
##     No Information Rate : 0.2821          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.8171          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9150   0.7710   0.7691   0.8523   0.9478
## Specificity            0.9633   0.9403   0.9685   0.9677   0.9790
## Pos Pred Value         0.9073   0.7497   0.8545   0.8344   0.9057
## Neg Pred Value         0.9665   0.9466   0.9458   0.9716   0.9888
## Prevalence             0.2821   0.1881   0.1937   0.1605   0.1756
## Detection Rate         0.2581   0.1450   0.1490   0.1368   0.1665
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9391   0.8557   0.8688   0.9100   0.9634
```