Practical Machine Learning - Prediction Assignment

xbuRAw (GH)

December 27, 2019

0. Introduction

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, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants and try to predict the manner in which they did the exercise. This is the "classe" variable in the training set.

This write up will consist of three sections:

1. Data loading and cleaning
2. Model testing, building and evaluation
3. Conclusion: Model selection, and final prediction.

1. Loading data and data pre-processing

```r
library(caret)
library(rpart)
library(randomForest)
library(nnet)
set.seed(111)
```

```r
trainUrl <- url("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
testUrl <- url("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
rawTrainData <- read.csv(trainUrl, na.strings = c("NA","#DIV/0!",""))
rawTestData  <- read.csv(testUrl, na.strings = c("NA","#DIV/0!",""))
dim(rawTrainData)
```

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

The data set has 19622 observations of 160 variables. The first seven variables have no predictive value and can be discarded.

```r
# Excluding the first seven columns and all variables with NA entries
trainData <- rawTrainData[, -c(1:7)] # sample(c(1:19622), size = 100)
trainData <- trainData[,colSums(is.na(trainData)) == 0]
testData <- rawTestData[, -c(1:7)]
dim(trainData)
```

```
## [1] 19622    53
```

The tidied up data set has 53 remaining variables. We can now use the caret's `createDataPartition` to split the data in training and test sets.

```r
partition <- createDataPartition(trainData$classe, p = .8, list = FALSE)
trainDF <- trainData[partition, ]
testDF <- trainData[-partition, ]
```

2. Model testing

All the models will be evaluated using caret's `confusionMatrix` function. And make use of cross validation, via the `trainControl parameter method = repeatedcv`. We will only include the output for one model, otherwise the write up will be too long.

2.1 Model testing: Decision trees

```
system.time(
  modelDT <- rpart(classe ~ ., method = "class", data = trainDF)
  )
predictDT <- predict(modelDT, testDF, type = "class")
cM <- confusionMatrix(predictDT, testDF$classe)
cM
```

The accuracy of the decision tree is 76.22%.

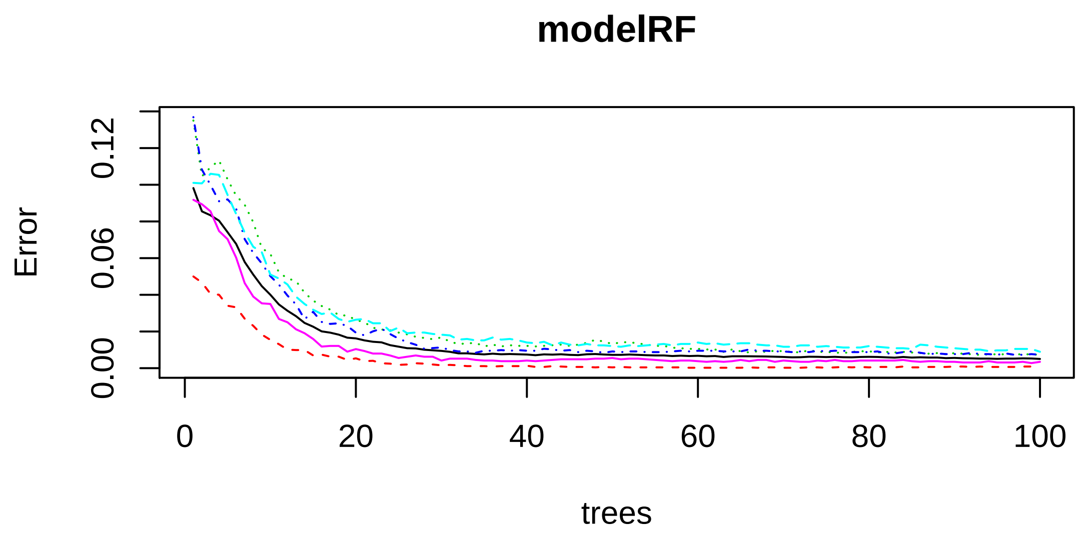
2.2 Model testing: Random Forest

The logical next step is to make the jump from a decision tree to a random forest model. We use the `randomForest` library instead of carets own implementation, because it performs better.

```
system.time(
  modelRF <- randomForest(classe ~ ., data = trainDF, ntree = 100)
  )
```

```
##    user  system elapsed
##   8.125   0.141   8.263
```

```
plot(modelRF)
```



```r
predictRF <- predict(modelRF, testDF, type = "class")
cM <- confusionMatrix(predictRF, testDF$classe)
cM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    5    0    0    0
##          B    0  754    5    0    0
##          C    0    0  678    5    0
##          D    0    0    1  638    3
##          E    0    0    0    0  718
##
## Overall Statistics
##
##                Accuracy : 0.9952
##                  95% CI : (0.9924, 0.9971)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9939
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9934   0.9912   0.9922   0.9958
## Specificity            0.9982   0.9984   0.9985   0.9988   1.0000
## Pos Pred Value         0.9955   0.9934   0.9927   0.9938   1.0000
## Neg Pred Value         1.0000   0.9984   0.9981   0.9985   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1922   0.1728   0.1626   0.1830
## Detection Prevalence   0.2858   0.1935   0.1741   0.1637   0.1830
## Balanced Accuracy      0.9991   0.9959   0.9948   0.9955   0.9979
```

The accuracy of this model is 99.52%.

2.3 Model testing: GBM

Another popular machine learning algorithm is the gradien boosting machine (GBM).

```r
system.time(
  modelGBM <- train(classe ~ ., method = "gbm", data = trainDF,
    trControl = trainControl(method = "repeatedcv", number = 5, repeats = 1))
  )

predictGBM <- predict(modelGBM, testDF)
cM <- confusionMatrix(predictGBM, testDF$classe)
cM
```

This model has an accuracy of 95.92%.

2.4 Model testing: LDA

An LDA (Linear Discriminant Analysis) model can be seen as an close relative of logistic regression and is often used for classification problems with more than two classes.

```r
system.time(
  modelLDA <- train(classe ~ ., method = "lda", data = trainDF,
    trControl = trainControl(method = "repeatedcv", number = 5, repeats = 1))
  )
predictLDA <- predict(modelLDA, testDF)
cM <- confusionMatrix(predictLDA, testDF$classe)
cM
```

The accuracy of the LDA model is 70.43%.

2.5 Model testing: Deep learning with `nnet`

At last we will try a deep learning model using the `nnet` package.

```r
system.time(
  modelDL <- nnet(classe ~ ., trainDF, size = 5, rang = .1, decay = 5e-4, maxit = 100,
    trControl = trainControl(method = "repeatedcv", number = 5, repeats = 1))
  )
predictDL <- predict(modelDL, testDF, type = "class")
cM <- confusionMatrix(as.factor(predictDL), testDF$classe)
cM
```

The estimated accuracy of the DL model is 40.79%. The reason of this low score could be the one hidden layer limitation of the `nnet` library.

3. Conclusion

We trained five models and random forests performed best (accuracy score of 99.52%).

3.1 Score overview

Decision trees: 76.22%, Random Forest: 99.52%, GBM: 95.92%, LDA: 70.43%, DNN: 40.79%

3.2 Time overview

The fastest algorithm was LDA with 1.75s system time. Decision trees: 1.875s, Random Forest: 7.500s, GBM: 3.75s, LDA: 1.75s, DNN: 8.9s

3.3 Final prediction

Although it was slower by the factor of two than the GBM model, the random forest had by far the best predictive strength. For this reason we will use the random forest model to make the final predictions on the `testData`.

```r
predict(modelRF, testData, type = "class")
```

```
##  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
```