Practical machine learning - final project

Tuomo Kässi

2023-04-23

Synopsis

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.

The purpose of this project is to predict the type of activity undertaken by people exercising. This is the `classe` variable in the data.

Loading data and required libraries

We first load the relevant data and libraries.

```
library(knitr)
library(caret)
```

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

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

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
dataurl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
data <- read.csv(url(dataurl))
```

Split into separate training and test sets.

```
inTrain  <- createDataPartition(data$classe, p=0.7, list=FALSE)
TrainSet <- data[inTrain, ]
TestSet  <- data[-inTrain, ]
```

Data preprocessing

Check training and test set dimensions.

```
dim(TrainSet)
```

```
## [1] 13737   160
```

```
dim(TestSet)
```

```
## [1] 5885  160
```

Then we remove variables with Nearly Zero Variance, since we expect them not to provide any predictive power.

```
NZV <- nearZeroVar(TrainSet)
TrainSet <- TrainSet[, -NZV]
TestSet  <- TestSet[, -NZV]
dim(TrainSet)
```

```
## [1] 13737   106
```

We also remove variables that are mostly NA

```
AllNA    <- sapply(TrainSet, function(x) mean(is.na(x))) > 0.95
TrainSet <- TrainSet[, AllNA==FALSE]
TestSet  <- TestSet[, AllNA==FALSE]
```

We also variables that are not used for prediction such as identification only variables (columns 1 to 5)

```
TrainSet <- TrainSet[, -(1:5)]
TestSet  <- TestSet[, -(1:5)]
```

We are left with the following data

```
dim(TrainSet)
```

```
## [1] 13737    54
```

```
dim(TestSet)
```

```
## [1] 5885   54
```

Modeling: Random forest with cross-validation

Cross-validation is a often used method of checking the variation of the statistical data. The data is divided to three or five (or n) folds. One of the folds is reserved for testing and the rest of the folds are used for training. When the procedure is repeated n times and the average is calculated we have an estimate for statistical performance.

```
set.seed(111)
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
modFitRandForest <- train(classe ~ ., data=TrainSet, method="rf",
                          trControl=controlRF)
modFitRandForest$finalModel
```

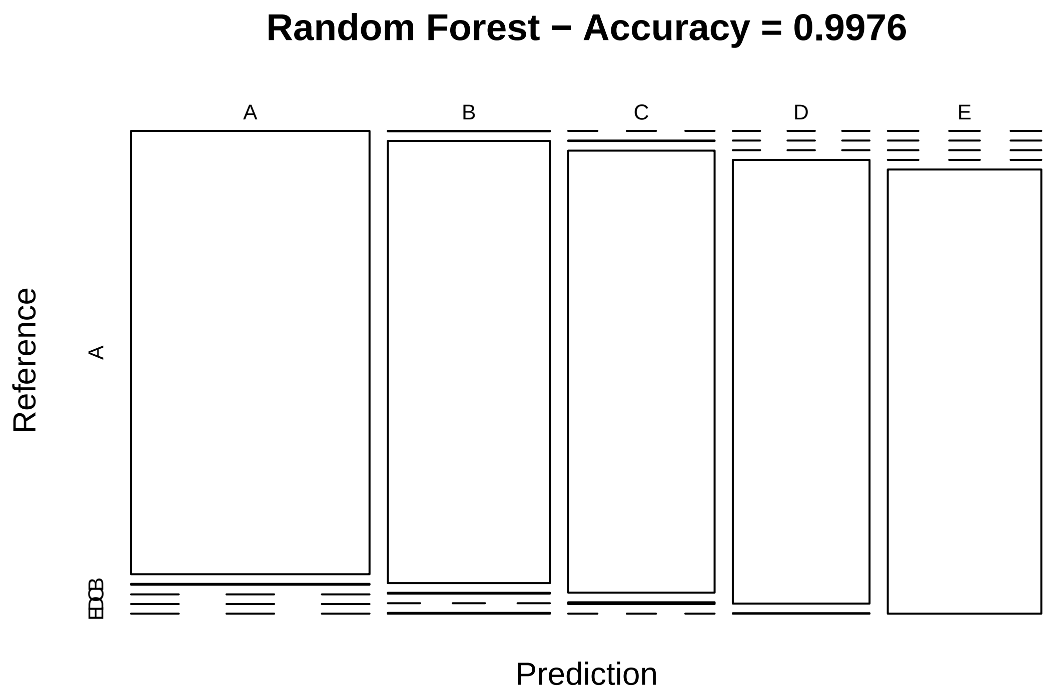
```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.25%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    0    0    0    1 0.0002560164
## B    7 2648    3    0    0 0.0037622272
## C    0    6 2390    0    0 0.0025041736
## D    0    0    8 2243    1 0.0039964476
## E    0    0    0    8 2517 0.0031683168
```

Prediction on Test dataset

```r
predictRandForest <- predict(modFitRandForest, newdata=TestSet)
confMatRandForest <- confusionMatrix(predictRandForest, as.factor(TestSet$classe))
confMatRandForest
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    3    0    0    0
##          B    1 1135    2    0    2
##          C    0    1 1024    4    0
##          D    0    0    0  960    1
##          E    0    0    0    0 1079
##
## Overall Statistics
##
##                Accuracy : 0.9976
##                  95% CI : (0.996, 0.9987)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.997
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9965   0.9981   0.9959   0.9972
## Specificity            0.9993   0.9989   0.9990   0.9998   1.0000
## Pos Pred Value         0.9982   0.9956   0.9951   0.9990   1.0000
## Neg Pred Value         0.9998   0.9992   0.9996   0.9992   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1929   0.1740   0.1631   0.1833
## Detection Prevalence   0.2848   0.1937   0.1749   0.1633   0.1833
## Balanced Accuracy      0.9993   0.9977   0.9985   0.9978   0.9986
```

Visualise prediction in test data set.



Modeling: Generalized boosted models with cross-validation

```
set.seed(111)
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
controlGBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
modFitGBM  <- train(classe ~ ., data=TrainSet, method = "gbm",
                    trControl = controlGBM, verbose = FALSE)
```

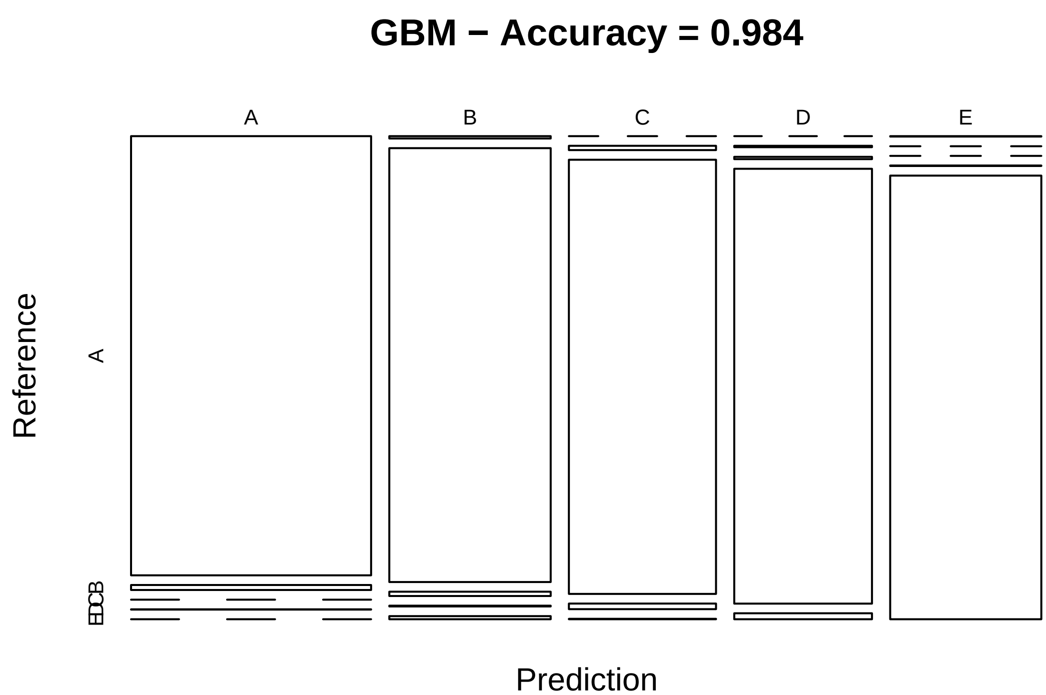
Prediction in Test dataset

```
predictGBM <- predict(modFitGBM, newdata=TestSet)
confMatGBM <- confusionMatrix(predictGBM, as.factor(TestSet$classe))
confMatGBM
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    A    B    C    D    E
##          A 1667   19    0    1    0
##          B    6 1107   11    2    8
##          C    0   10 1010   13    1
##          D    0    3    5  947   13
##          E    1    0    0    1 1060
##
```

```
## Overall Statistics
##
##                Accuracy : 0.984
##                  95% CI : (0.9805, 0.9871)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9798
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9958   0.9719   0.9844   0.9824   0.9797
## Specificity            0.9953   0.9943   0.9951   0.9957   0.9996
## Pos Pred Value         0.9881   0.9762   0.9768   0.9783   0.9981
## Neg Pred Value         0.9983   0.9933   0.9967   0.9965   0.9954
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2833   0.1881   0.1716   0.1609   0.1801
## Detection Prevalence   0.2867   0.1927   0.1757   0.1645   0.1805
## Balanced Accuracy      0.9955   0.9831   0.9897   0.9890   0.9896
```

Again we plot performance of the model.



We find that both models perform extremely well, but the random forest model has a slight edge over generalized boosted trees.

Based on the prediction error in test data set, we expect the random forest model to be able to predict 99.8% of the cases correctly.

Prediction in unseen test data

Finally, we put the preferred model to a true test by checking how it performs in a completely unseen data.

First we load the test data.

```
UrlTest  <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
newdata <- read.csv(url(UrlTest))
```

Then we clean up the test data

```
newdata <- newdata[,-NZV]
newdata <- newdata[,-AllNA==FALSE]
newdata <- newdata[, -(1:5)]

dim(newdata)
```

```
## [1] 20 54
```

Then we can apply the model to the new data. As we argue above, we expect that 99.8% of these classes are correct. :)

```
predictTEST <- predict(modFitRandForest, newdata=newdata)
predictTEST
```

```
##  [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
```