Machine learning project: Human activity recognition

Synopsis

This analysis shows that machine learning methods can be used to identify in which way a weightlifting exercise has been done. In the experiment different weight lifters were asked to do their exercises in five different ways; using random forests, it was possible to correctly identify the type in which the exercise was done with an accuracy of 98% and better.

Introduction

This report analyses the measurement data from an experiment in which different weight lifters were asked to do their exercises in five different ways: In the correct way, i.e. according to exercise specifications (method A), and in 4 different wrong ways (methods B-E). Throughout these exercises, movement data were collected from sensors at 4 different points, which were located at the forearm, arm, belt and dumbbell. The collected data are analysed in this report.

Loading and processing the data

Downloading and loading the original data for training and testing.

```r
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
              destfile = "training.csv", method="curl")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
              destfile = "testing.csv", method="curl")
training_orig <- read.csv("training.csv")
testing_orig <- read.csv("testing.csv")
dim(training_orig)
```

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

Many columns have 19216 NA entries, i.e. are mostly empty, which is reason enough to get rid of those columns. We do this for both training and test set, but since the last column in the test set ("problem_id") is named differently than in the training set ("classe"), we have to take care of that, too. As we are at it, we will also drop the first five columns, which include index, user name and time data and which should be of no importance for our model fit.

```r
drop_cols <- colnames(training_orig[colSums(is.na(training_orig)) > 1000])
drop_cols <- c(colnames(training_orig)[1:5],drop_cols)
training_small <- training_orig[, !(names(training_orig) %in% drop_cols)]
testing_small <- testing_orig[, !(names(testing_orig) %in% drop_cols)]
colnames(testing_small)[length(colnames(testing_small))] <- "classe"
```

We are now down to a training set with 88 columns, i.e. to 87 possible predictors. Time to split the training set into a smaller training set and a validation set.

```r
set.seed(1904)
inTrain <- createDataPartition(training_small$classe, p=0.75, list=FALSE)
train <-training_small[inTrain,]
valid <-training_small[-inTrain,]
```

When looking closely at our data, we see a number of factor columns for variables that should in fact be numeric, i.e. for the kurtosis and the skewness for many of the measurements. So we should convert these columns to numeric, but we have to do this stepwise.

```
bla <- train
ctn <- colnames(train)[2:(length(colnames(train))-1)]
bla[,ctn] <- lapply(bla[,ctn,drop=FALSE],as.character)
suppressWarnings(bla[,ctn] <- lapply(bla[,ctn,drop=FALSE],as.numeric))
```

We find that these converted columns now mostly consist of NA's, which means that they were practically empty to begin with. Time to drop them, and we will do so for all data frames of importance. We also drop the columns "new_window" and "num_window", since they don't seem to contain sensory data.

```
drop2 <- colnames(bla[colSums(is.na(bla)) > 1000])
drop2 <- c("new_window","num_window",drop2)
train <- train[, !(names(train) %in% drop2)]
valid <- valid[, !(names(valid) %in% drop2)]
training_small <- training_small[, !(names(training_small) %in% drop2)]
testing_small <- testing_small[, !(names(testing_small) %in% drop2)]
```

Now we are down to a training set with 53 columns, i.e. to 52 possible predictors. It's finally time for some fitting action.

Fitting of models

Since I'm doing this on my trusty laptop (which was build in 2006), I will start with some smaller models. Let's go and see how well a single decision tree can work by fitting a maximum tree:

```
suppressMessages(library(caret))
suppressMessages(library(rpart))
rpartfit<- rpart(classe~., data=train)
rpartpred <- predict(rpartfit, newdata=valid, type="class")
suppressMessages(rpartconf<- confusionMatrix(rpartpred,valid$classe))
rpartconf$overall[1]
```

```
## Accuracy
##   0.7282
```

Well, that's not too good, and the accuracy on the validation set is similar to the accuracy for the training set (0.7392). We should probably try another model... How about linear discriminant analysis?

```
suppressMessages(ldafit<- train(classe~., data=train, method="lda"))
ldapred <- predict(ldafit, newdata=valid)
ldaconf <- confusionMatrix(ldapred,valid$classe)
ldaconf$overall[1]
```

```
## Accuracy
##   0.7094
```

Even worse, and again the accuracy on the validation set is similar to the accuracy for the training set (0.7042). Seems like it's time get out the big guns - random forests. Since this takes a very long time on my laptop (I have done it in an interactive console), I'm not going to do this again in this write-up; instead, I will show how to do the calculations, but then go ahead and load the saved model.

```r
rffit<- train(classe~., data=train, method="rf")
save(rffit, file="RFfit.RData")
```

```r
suppressMessages(library(randomForest))
load("RFfit.RData")
rffit <- rffit
rffit$results
```

```
##   mtry Accuracy  Kappa AccuracySD  KappaSD
## 1    2   0.9893 0.9865   0.001975 0.002498
## 2   27   0.9891 0.9862   0.001677 0.002126
## 3   52   0.9800 0.9746   0.004438 0.005626
```

```r
rffit$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: 2
##
##         OOB estimate of  error rate: 0.66%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4182    3    0    0    0   0.0007168
## B   14 2828    6    0    0   0.0070225
## C    0   17 2544    6    0   0.0089599
## D    0    0   41 2369    2   0.0178275
## E    0    0    2    6 2698   0.0029564
```

The random forest model gives us an accuracy of 98% on the training set, and the confusion matrix is nearly diagonal. Let's see how the model does on the validation set.

```r
rfpredict <- predict(rffit,newdata=valid)
suppressMessages(rfconf <- confusionMatrix(rfpredict,valid$classe))
rfconf$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    3    0    0    0
##          B    1  944    7    0    0
##          C    0    2  847   12    0
##          D    0    0    1  792    1
##          E    0    0    0    0  900
```

```r
rfconf$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##         0.9945         0.9930         0.9920         0.9964         0.2845
## AccuracyPValue  McnemarPValue
##         0.0000            NaN
```

Now that's not a bad accuracy we get after applying our fitted model to the validation set! Recall that we did not use this validation set to fit any of the parameters, so this accuracy of 99% is easily good enough to call this model our final model. The only thing left to do is to apply the model to our test set, which we do twice: Once to see the probabilities with which a test case is assigned to each class, and the second time with the final classification.

```
predict(rffit,testing_small,type="prob")
```

```
##        A     B     C     D     E
## 1  0.120 0.698 0.124 0.036 0.022
## 2  0.822 0.060 0.098 0.008 0.012
## 3  0.118 0.646 0.156 0.026 0.054
## 4  0.880 0.018 0.060 0.034 0.008
## 5  0.952 0.010 0.030 0.000 0.008
## 6  0.018 0.126 0.088 0.034 0.734
## 7  0.036 0.018 0.114 0.792 0.040
## 8  0.122 0.622 0.102 0.104 0.050
## 9  0.994 0.002 0.002 0.002 0.000
## 10 0.978 0.008 0.008 0.006 0.000
## 11 0.120 0.616 0.138 0.078 0.048
## 12 0.058 0.060 0.798 0.024 0.060
## 13 0.016 0.950 0.002 0.012 0.020
## 14 1.000 0.000 0.000 0.000 0.000
## 15 0.024 0.044 0.024 0.026 0.882
## 16 0.028 0.056 0.010 0.016 0.890
## 17 0.960 0.004 0.002 0.000 0.034
## 18 0.048 0.860 0.008 0.050 0.034
## 19 0.086 0.828 0.040 0.034 0.012
## 20 0.004 0.992 0.000 0.000 0.004
```

```
predict(rffit,newdata = testing_small)
```

```
##  [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
```

And that's it - we now classified each of the test cases to a class. The expected error rate of our classifcation should be more or less the same as the error rate we saw for the validation set, since the model was fitted independently of both the validation and test set. Since we saw a classification accuracy of 99% in the validation set and a accuracy of 98% in the training set, we should expect a similar accuracy for the test set.