Practical Machine Learning Assignment

Felix Dijkstal

2022-09-19

Load libraries

First we load the required libraries.

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Loading required package: lattice

## randomForest 4.7-1.1

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

Load data

We set the seed for reproducibility. While importing the provided training and test data we also specify the strings that should be read as missing values as the data is somewhat messy in this regard. To get a sense of the data we check the number of observations per group. It seems like the data is slightly skewed towards group A but not to an extent that would impact the model.

```r
set.seed(666)

training_data = read.csv('/Users/felixdijkstal/Downloads/pml-training.csv',
                         na.strings = c("NA","#DIV/0!",""))
test_data = read.csv('/Users/felixdijkstal/Downloads/pml-testing.csv',
                     na.strings = c("NA","#DIV/0!",""))

# Inspect data
training_data %>%
  group_by(classe) %>%
  summarise(n = n())
```

```
## # A tibble: 5 x 2
##   classe     n
##   <chr>  <int>
## 1 A       5580
## 2 B       3797
## 3 C       3422
## 4 D       3216
## 5 E       3607
```

Clean data

Prior to building the model it is necessary to select only the variables that are needed to predict our outcome variable 'classe'. First, we drop the variables that are obviously not predictors. Secondly, we find that several variables contain little to no data (near-zero), and these must be dropped as well. Finally, we transform 'classe' into a factor.

```r
# Drop unnecessary variables
training_data = training_data %>%
  select(-X, -user_name, -contains('timestamp'), -new_window, -num_window)
test_data = test_data %>%
  select(-X, -user_name, -contains('timestamp'), -new_window, -num_window, -problem_id)

# Drop near zero variables
training_data = training_data[, colSums(is.na(training_data)) == 0]
test_data = test_data[, colSums(is.na(test_data)) == 0]

# Make 'classe' factor
training_data = training_data %>%
  mutate(classe = as.factor(classe))
```

Cross validation

The model will be cross validated as follows. First we split the training data into 'training' and 'testing' sub-samples. First, the model will tested on the 'testing' sub-sample and secondly on the original testing data.

Build model

We split the training data into a 'training' and 'testing' sub-sample accounting for 70% and 30% of the training data respectively. We then fit a random forest model on the training sub-sample, using 'classe' as the outcome variable and all other variables as predictors. After fitting the model, we use the model to predict the 'testing' sub-sample and compare it to the actual values of 'classe'. The corresponding confusion matrix shows that the accuracy of the model is 0.99 meaning that we would expect the out of sample error to be very low (less than 0.01).

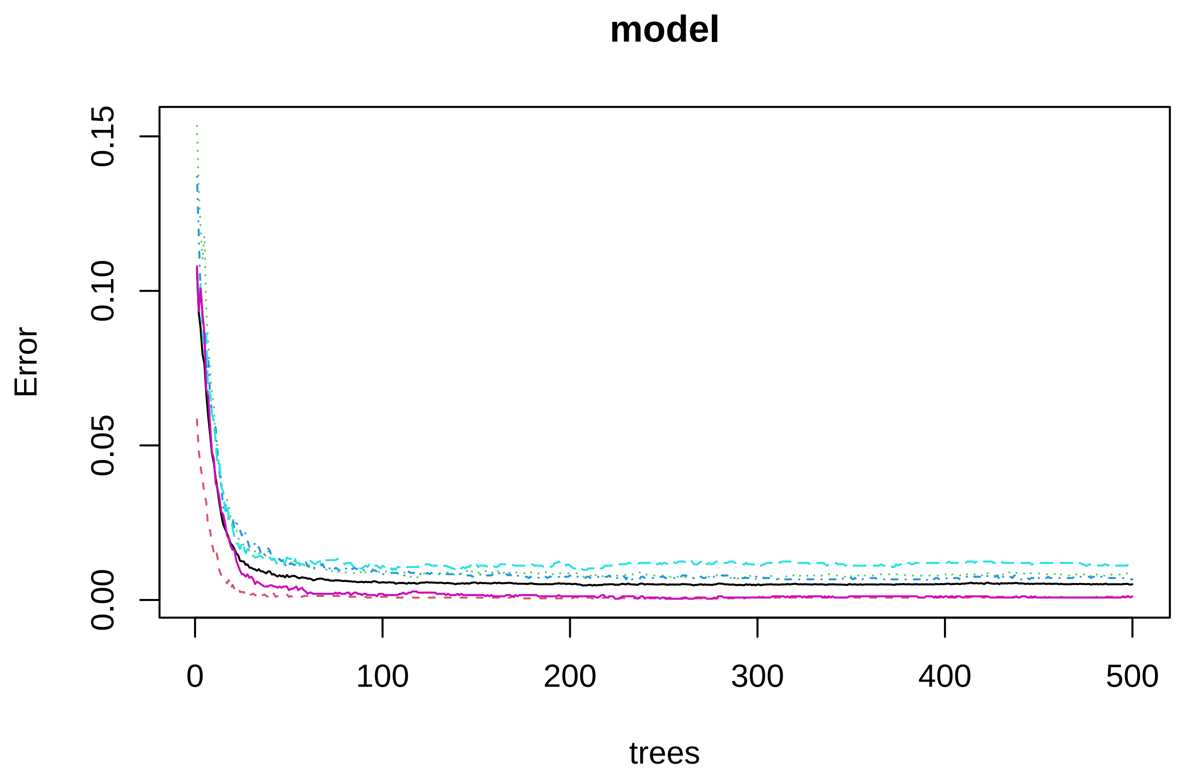
```
# Split data
inTrain = createDataPartition(training_data$classe, p = 0.7, list = F)
training = training_data[inTrain,]
testing = training_data[-inTrain,]

# Fit model
model = randomForest(classe ~ ., data = training, type = 'class')

# Predict the test sub-sample
predictions = predict(model, testing)
confusionMatrix(predictions, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    4    0    0    0
##          B    0 1134    3    0    0
##          C    0    1 1022   12    2
##          D    0    0    1  951    1
##          E    0    0    0    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9958
##                  95% CI : (0.9937, 0.9972)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9946
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9956   0.9961   0.9865   0.9972
## Specificity            0.9991   0.9994   0.9969   0.9996   0.9998
## Pos Pred Value         0.9976   0.9974   0.9855   0.9979   0.9991
## Neg Pred Value         1.0000   0.9989   0.9992   0.9974   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1927   0.1737   0.1616   0.1833
## Detection Prevalence   0.2851   0.1932   0.1762   0.1619   0.1835
## Balanced Accuracy      0.9995   0.9975   0.9965   0.9931   0.9985
```

```
plot(model)
```



Predict

Given the accuracy of our model, we can use it to predict the 20 test cases.

```
predictions2 = predict(model, test_data, type = 'class')
write.csv(predictions2, 'pml_assignment_predictions.csv')
```