Prediction Assignment Writeup

Davy Meesemaecker

22/1/2018

Executive summary

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

data processing and exploratory data analysis

Our very first step is to download the two datasets required to perform our analysis. First one is called pml with 160 columns and 19622 rows while second one is called validation with 160 variables and 20 rows.

```
url1 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url2 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(url = url1, "pml-training.csv", method = "auto")
download.file(url = url2, "pml-testing.csv", method = "auto")
pml <- read.csv("pml-training.csv", na.strings = c("", "NA"))
dim(pml)
```

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

```
validation <- read.csv("pml-testing.csv", na.strings = c("", "NA"))
dim(validation)
```

```
## [1]  20 160
```

Now it's time to split our pml dataset into two different paritions. Training, composed of 80% random rows from pml and testing with the remaining 20%

```
library(caret)
set.seed(1305)
partition <- createDataPartition(y = pml$classe, p = 0.8, list = FALSE)
training <- pml[partition, ]
(dim(training)[1]/dim(pml)[1])*100
```

```
## [1] 80.00713
```

```
testing <- pml[-partition, ]
dim(testing)[1]/dim(pml)[1]*100
```

```
## [1] 19.99287
```

We can start exploring our training data, let's check if we have NAs :

```
sum(is.na(training))/(15699*160)
```

```
## [1] 0.6118224
```

We have an important rate (0.61) of NAs in the training dataset, so we'll avoid using variables/columns with NAs on both training and testing.

```
NAlocation <- apply(training, 2, function(x) sum(is.na(x)))
training <- training[, which(NAlocation == 0)]
dim(training)[2]
```

```
## [1] 60
```

```
testing <- testing[, which(NAlocation == 0)]
dim(testing)[2]
```

```
## [1] 60
```

```
sum(is.na(training))
```

```
## [1] 0
```

We'll also remove columns with unique values or very few variability in the dataset.

```
zeroVar <- nearZeroVar(training)
training <- training[, -zeroVar]
dim(training)[2]
```

```
## [1] 59
```

```
testing <- testing[, -zeroVar]
dim(testing)[2]
```

```
## [1] 59
```

Finally we'll remove the X column(corresponding to the index), the user_name column and the timestamp columns (raw_timestamp_part1, raw_timestamp_part2 and cvtd_timestamp) as these columns shouldn't have any effect on the classe variable.

```
training <- training[, -c(1:5)]
dim(training)[2]
```

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

```
testing <- testing[, -c(1:5)]
dim(testing)[2]
```

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

The model

We've finished processing the data (54 columns remaining), it's now time to fit a model to predict the manner in which people people do the exercices. We'll use the classe column as the outcome and all other remaining variables as predictors. In my opinion, random forest seems to be the best model to use as we want to classify results in 5 different classes. We could use a simple decision tree but we'd like to use cross validation with enough trees (500) to avoid overfitting.

```
library(randomForest)
set.seed(2610)
mdlRF <- randomForest(classe~., data = training, ntree = 500)
mdlRF
```

```
##
```

```
## Call:
##  randomForest(formula = classe ~ ., data = training, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.27%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4463    1    0    0    0 0.0002240143
## B    4 3031    3    0    0 0.0023041475
## C    0   10 2727    1    0 0.0040175310
## D    0    0   19 2553    1 0.0077730276
## E    0    0    0    3 2883 0.0010395010
```

```
predRF <- predict(mdlRF, testing)
conf <- confusionMatrix(predRF, testing$classe)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    1    0    0    0
##          B    0  758    1    0    0
##          C    0    0  683    4    0
##          D    0    0    0  639    2
##          E    0    0    0    0  719
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.996, 0.9991)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9987   0.9985   0.9938   0.9972
## Specificity            0.9996   0.9997   0.9988   0.9994   1.0000
## Pos Pred Value         0.9991   0.9987   0.9942   0.9969   1.0000
## Neg Pred Value         1.0000   0.9997   0.9997   0.9988   0.9994
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1932   0.1741   0.1629   0.1833
## Detection Prevalence   0.2847   0.1935   0.1751   0.1634   0.1833
## Balanced Accuracy      0.9998   0.9992   0.9987   0.9966   0.9986
```

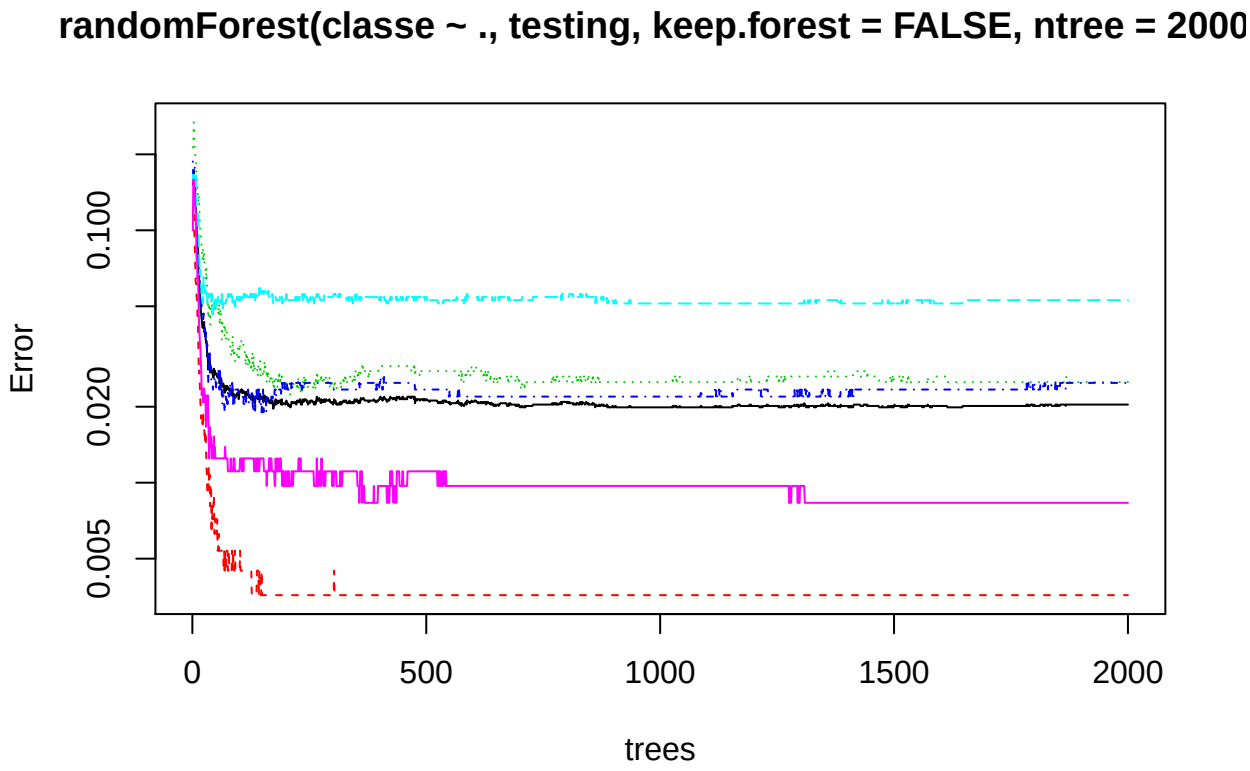
The model is very efficient, the accuracy is 99.847% on the test set and only misclassified 5/3923 rows. The expected out of sample error is 1 - accuracy for predictions made against the test set = 1 - 0.9985 = 0.0015 meaning there's only 0.15% chances we missclassify the way the exercice was performed. As our validation test has only 20 rows, we can expect our data to be correctly classified. Let's check out

```
predValidation <- predict(mdlRF, validation)
predValidation
```

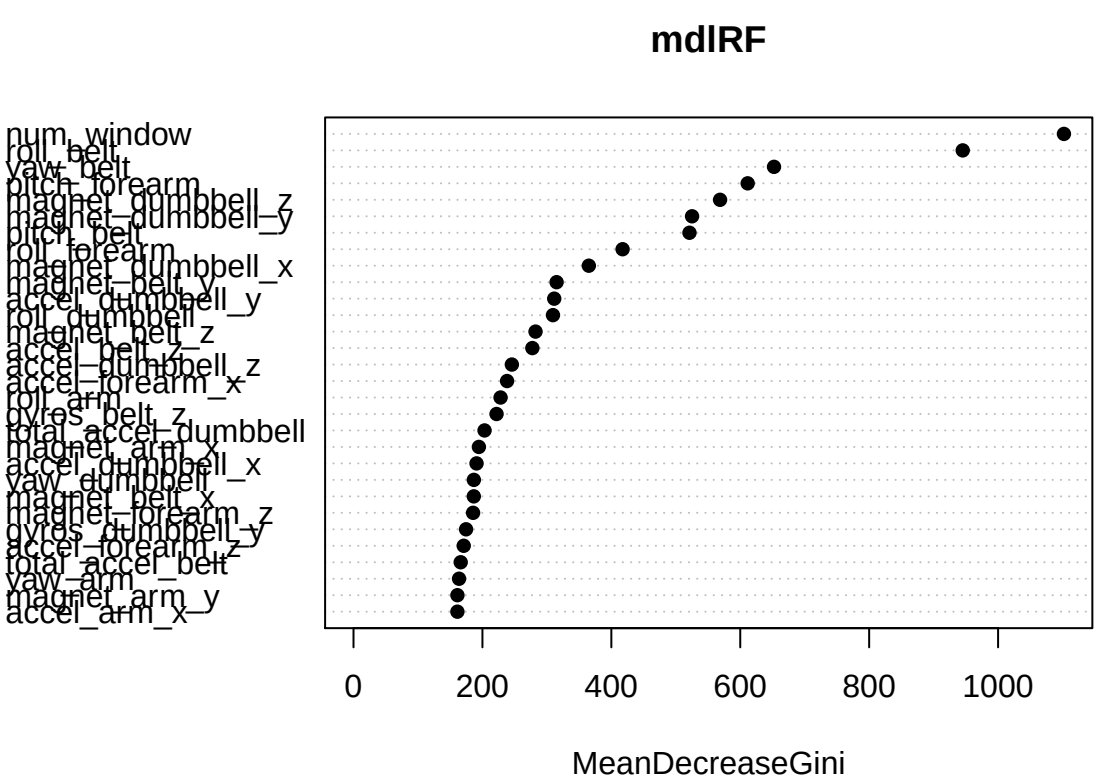
```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  B  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B
## Levels: A B C D E
```

Finally, what would happen if we use a bigger number of trees in our random forest ? As we had 53 predictors, which ones are the most influencial on the classe variable ?

```
plot(randomForest(classe ~ ., testing, keep.forest=FALSE, ntree=2000), log="y")
```




```
varImpPlot(mdlRF, pch = 16)
```



The error rate is stable when ntree gets bigger than 200, meaning we wouldn't improve much our model accuracy with a ntree equals to 2000. On the second plot, we can easily see the most influential variables are num_window and roll_belt.