Practical Machine Learning Project

Alastair Mak

22 April 2017

Introduction

Here we cover predicting efficiency and effectiveness of different exercises using wearable technology. We use a Random Forests model to predict which of five outcome classifications to assign to 20 observations in the `testing` dataset, with our predictions given at the end of the document.

Data and packages

First, let's download the data and load the packages we'll use later.

```
set.seed(76437)
library(caret)
library(randomForest)
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
                     na.strings=c(""," ","NA"))
testing  <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
                     na.strings=c(""," ","NA"))
dim(training)
```

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

```
dim(testing)
```

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

```
unique(training$classe)
```

```
## [1] A B C D E
## Levels: A B C D E
```

Looking at the outcome variable `classe` we see that it has five levels, with values from `A` to `E`.

We want to leave the `testing` set for the very end, once we have built and verified our model. Let's split the `training` data set into two, using one dataset to build the model, and the other to test it.

```
set.seed(76437)
inTrain <- createDataPartition(training$classe, p=0.7, list=FALSE)
train1 <- training[inTrain,]
pretest <- training[-inTrain,]
```

We'll use the `train1` dataset to build the model, the `pretest` data to test the model, leaving the `testing` dataset to the very end.

Data cleaning

Were we to perform `head(training)`, we would see that there are two columns, `X` and `user_name` that are identifying columns and that we don't want to use to build our model; let's exclude them from all three data sets. This is simple as they're the first two columns in all cases.

```
train1 <- train1[,-c(1,2)]
pretest <- pretest[,-c(1,2)]
testing <- testing[,-c(1,2)]
```

There is a lot of NA data in the `train1` dataset. Rather than imputing the NA values, let's remove any columns that contain at least one NA value.

```
colkeep <- colSums(is.na(train1)) == 0
train1 <- train1[colkeep == TRUE]
dim(train1)
```

```
## [1] 13737    58
```

We can see that the `train1` data now has 58 columns now, whereas before it had 160. Let's ensure that the `pretest` and `testing` data contain only these columns as well. We'll also remove the `problem_id` column from `testing`, as this column is not present in either `train1` or `pretest`.

```
pretest <- pretest[colkeep == TRUE]
testing <- testing[colkeep == TRUE]
testing <- testing[,-58] #remove the problem_id column in testing data
```

Building the model

Let's use the Random Forests model building methodology. While it is more computationally intensive and less interpretable than others such as Decision Trees, it typically gives more accurate predictions, which is our priority in this particular instance.

```
#Model build
set.seed(76437)
RF <- randomForest(classe ~ ., data=train1, ntree=500, importance=TRUE)

#Display model details
RF
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = train1, ntree = 500,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.12%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3906    0    0    0    0 0.0000000000
## B    2 2656    0    0    0 0.0007524454
## C    0    5 2388    3    0 0.0033388982
## D    0    0    6 2246    0 0.0026642984
## E    0    0    0    0 2525 0.0000000000
```

We see that the out-of-sample error is low, at 0.12%. We can now use the `RF` model to predict the outcomes on the `pretest` data, and measure the accuracy.

```
set.seed(76437)
#Predicting using this model on the pretest data
RFp <- predict(RF, newdata=pretest)
```

```r
#Confusion matrix
confusionMatrix(RFp, pretest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    0 1139    0    0    0
##          C    0    0 1026    1    0
##          D    0    0    0  960    0
##          E    0    0    0    3 1082
##
## Overall Statistics
##
##                Accuracy : 0.9993
##                  95% CI : (0.9983, 0.9998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9991
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   0.9959   1.0000
## Specificity            1.0000   1.0000   0.9998   1.0000   0.9994
## Pos Pred Value         1.0000   1.0000   0.9990   1.0000   0.9972
## Neg Pred Value         1.0000   1.0000   1.0000   0.9992   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1935   0.1743   0.1631   0.1839
## Detection Prevalence   0.2845   0.1935   0.1745   0.1631   0.1844
## Balanced Accuracy      1.0000   1.0000   0.9999   0.9979   0.9997
```

We see that the RF model does a very good job at predicting the outcomes on the `pretest` data, with accuracy of 0.999.

Predicting on the testing data

Before predicting on the testing data, we need to make sure the factor variables in the training and test sets have the same levels.

```r
#create single vector with common names (to be 100% sure)
intersection <- intersect(names(train1), names(testing))

#convert data types in testing data to match those in train1
  for (i in intersection)
    {
    if (class(train1[[i]]) == "factor") #if a factor variable,
      {
      #set the levels in testing to be equal to those in train1
      levels(testing[[i]]) <- levels(train1[[i]])
      }
    }
```

We can now predict as normal using the testing data.

```
set.seed(76437)
RFt <- predict(RF, newdata=testing)
RFt
```

```
##  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
```

And we can see that the values are each one of the five levels we saw earlier.