Peer-graded Assignment

Brynjólfur Gauti Jónsson

21 September 2017

Contents

```
library(ggplot2); library(ggthemes); library(caret); library(parallel)
library(doParallel); library(randomForest)
```

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. The purpose of this analysis is to utilize data from accelerometers such as the above fastened on the belt, forearm, arm and dumbbell of participants to predict how they performed certain exercises. Six participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways. Those 5 ways will be the output classes of our model.

More information is available here

Getting and Cleaning the Data

Reading the data

```
set.seed(101)
training <- read.csv('training.csv')
testing <- read.csv('testing.csv')
dim(training)
```

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

In the data there are lots of NA's and lots of reduntant features. Next we will remove unneeded features and features with a high number of NA's.

Cleaning the Data

There are a lot more variables than we need. We will filter out all variables pertaining to means, standard deviations, kurtosis etc. since they are combinations of all other variables. To do this I have written a function called removeFeatures.

```r
removeFeatures <- function(xtrain) {
    xtrain <- xtrain[,-grep('[Kk]urtosis',names(xtrain))]
    xtrain <- xtrain[,-grep('[Ss]kewness',names(xtrain))]
    xtrain <- xtrain[,-grep('[Ss]tddev',names(xtrain))]
    xtrain <- xtrain[,-grep('[Aa]vg',names(xtrain))]
    xtrain <- xtrain[,-grep('[Vv]ar',names(xtrain))]
    xtrain <- xtrain[,-grep('[Mm]in',names(xtrain))]
    xtrain <- xtrain[,-grep('[Mm]ax',names(xtrain))]
    xtrain <- xtrain[,-c(1,2)] # Remove the name of participant and obs. number.
    xtrain <- xtrain[,-c(12, 43, 59)] # These variables were all na in the test set.
    xtrain <- xtrain[,colMeans(is.na(xtrain)) < 0.9]
    return(xtrain)
}
train <- removeFeatures(training)
xtrain <- train[,-ncol(train)]
ytrain <- train[,ncol(train)]
test <- testing[,names(testing) %in% names(train)]
```

Modelling

Model Settings and PreProcessing

Now we prepare the data for modelling. There is a lot of missing data so we will utilize knnImpute to make new data points where needed. Since there are so many variables and no need for interpretability of the model, we will center and scale the data, and reduce the dimensions of the variable space via Principal Components Analysis. This will decrease the amount of space needed to store the data and shorten training time.

```r
library(caret)
preObj <- preProcess(x = xtrain, method=(c('knnImpute','center','scale', 'pca')))
xtrainfit <- predict(object = preObj, xtrain)


# Settings for the model. We use 15-fold Cross-Validation
fitSettings <- trainControl(method = 'cv', number = 5,
                            allowParallel = TRUE)



preObj
```

```
## Created from 19622 samples and 57 variables
##
## Pre-processing:
##   - centered (55)
##   - ignored (2)
##   - 5 nearest neighbor imputation (55)
##   - principal component signal extraction (55)
##   - scaled (55)
##
## PCA needed 26 components to capture 95 percent of the variance
```

We are able to explain 95% of the variance in the variables using only 26 principal components. This will help our model, since random trees are susceptible to redundant features. Now we have centered, scaled and reduced data ready for modelling.

Creating a smaller model

Since we don't have access to the real values of the test set, we first train a model on a subset of our training set and rate it using a test set.

```r
inTrain <- createDataPartition(train[,ncol(train)], p = 4/5, list=FALSE)

# Training set
smalltrain <- train[inTrain,]
smalltrainx <- predict(preObj, smalltrain[,-ncol(smalltrain)])
smalltrainy <- smalltrain[,ncol(smalltrain)]
# Testing set
smalltest <- train[-inTrain,]
smalltestx <- predict(preObj,smalltest[,-ncol(smalltest)])
smalltesty <- smalltest[,ncol(smalltest)]
```

Training the smaller model

Now we are ready to train the smaller model.

```
#cluster <- makeCluster(detectCores() - 1)
#registerDoParallel(cluster)
#smallmodel <- train(x = smalltrainx[,-1], y = smalltrainy, method = 'rf',
#                     trControl = fitSettings)
#saveRDS(smallmodel, 'smallrfModel.rds')
#stopCluster(cluster)
#registerDoSEQ()
smallmodel <- readRDS('smallrfModel.rds')
```

Testing the smaller model

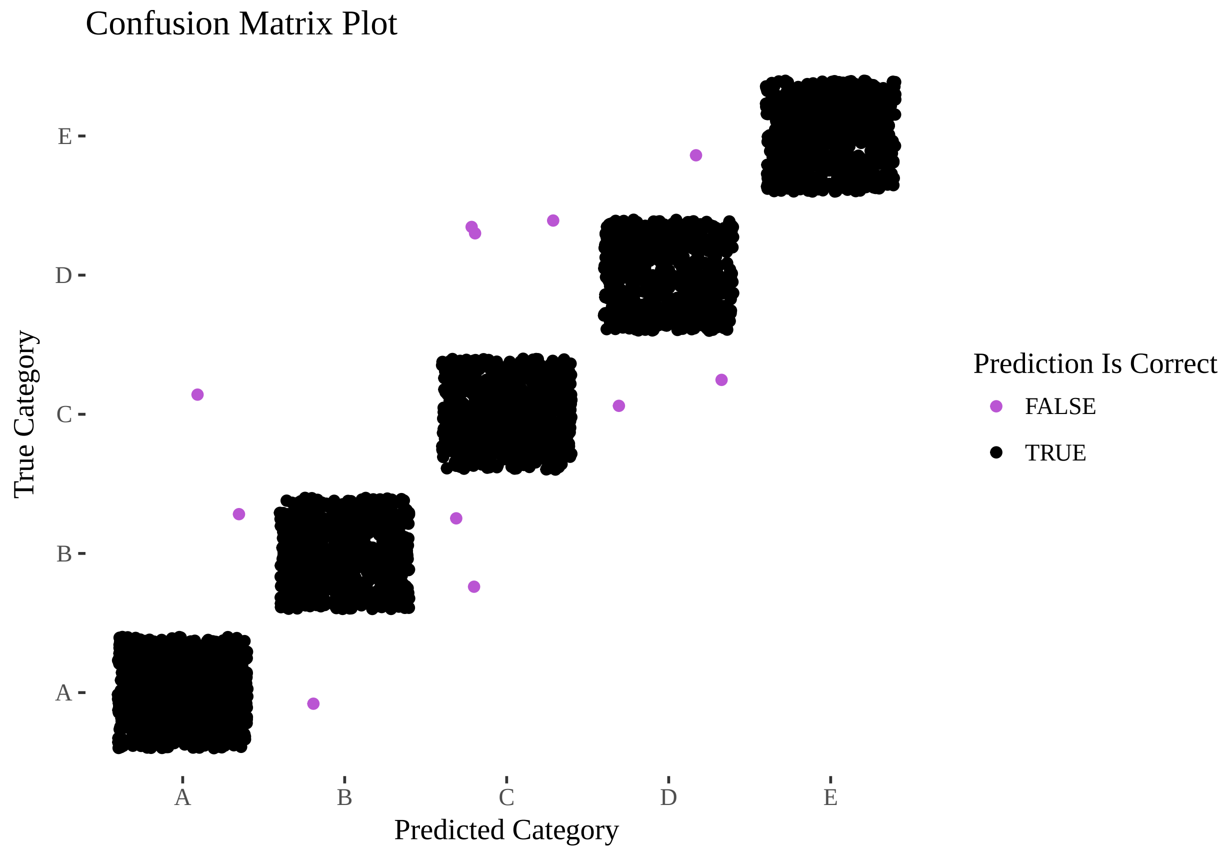
Now we see how well the smaller model works on the test holdout set.

```
smallpred <- predict(smallmodel, smalltestx)
confusionMatrix(smallpred, smalltesty)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1115    1    1    0    0
##          B    1  756    0    0    0
##          C    0    2  681    3    0
##          D    0    0    2  640    1
##          E    0    0    0    0  720
##
## Overall Statistics
##
##                Accuracy : 0.9972
##                  95% CI : (0.995, 0.9986)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9965
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9960   0.9956   0.9953   0.9986
## Specificity            0.9993   0.9997   0.9985   0.9991   1.0000
## Pos Pred Value         0.9982   0.9987   0.9927   0.9953   1.0000
## Neg Pred Value         0.9996   0.9991   0.9991   0.9991   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1927   0.1736   0.1631   0.1835
## Detection Prevalence   0.2847   0.1930   0.1749   0.1639   0.1835
## Balanced Accuracy      0.9992   0.9979   0.9970   0.9972   0.9993
```

```r
g <- ggplot(data = data.frame(pred = smallpred, True=smalltesty,
misclas = (smallpred==smalltesty)))

g + geom_jitter(aes(x = pred, y = True, col=misclas)) + theme_tufte() +
    xlab('Predicted Category') + ylab('True Category') +
ggtitle('Confusion Matrix Plot') + labs(color='Prediction Is Correct') +
    scale_colour_manual(values = c('mediumorchid', 'black'))
```



There seems to be no pattern in our prediction errors. We are now ready to train a full-size model and use it to predict on our original test set and answer all questions on the quiz.

Training the Full Model

We utilize the parallel and doParallel packages to quicken the training process. The model has previously been saved into the file ‘rfMOdel.rds’ and is loaded from there instead of training, but the code for the model is availiable below.

This model has previously been trained and saved into a file, but the code follows for reproducibility.

```
#library(parallel)
#library(doParallel)
#cluster <- makeCluster(detectCores() - 1)
#registerDoParallel(cluster)
#model <- train(x = xtrain, y = ytrain, method = 'rf',
#               trControl = fitSettings)
#saveRDS(model, 'rfModel.rds')
#stopCluster(cluster)
#registerDoSEQ()

model <- readRDS('rfModel.rds')
```

Testing the Full Model

Now we apply the same pre-processing to the testing data, as we did to the training data, after which we find our predictions.

```
model
```

```
## Random Forest
##
## 19622 samples
##    26 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (15 fold)
## Summary of sample sizes: 18314, 18314, 18313, 18314, 18314, 18314, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9859345  0.9822053
##   14    0.9823165  0.9776325
##   26    0.9774240  0.9714399
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
xtest <- predict(preObj, test)
pred <- predict(model, xtest)
pred
```

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

Summary

In this assignment I read and cleaned data meassured by motion sensors mounted on various parts of participants' bodies. This data was then used to predict the manner in which they did certain exercises. After cleaning the data and preprocessing it by centering, scaling, and reducing its dimensions, random forests was the preferred model. There was little need for interpretability of the model and no need for major scalability, and RF is an extremely accurate algorithm that sacrifices both of those perks.