Machine Learning Course Project - Weight Lifting Exercise Prediction

jhmedeiros

August 20, 2016

1 - Executive Summary

The main goal of this project is to create a machine learning algorith which predicts, within the given data, the manner that a set of weight lifting is being performed. The objective here is to explore the relationship between the measured data with accelerometers on the belt, forearm, arm and dumbell of six participants, in order to find out which data corresponds to which set of exercises. According to the obtained results, the random forest model was the one that provided the best prediction results, followed closely by the generalized boosted model.

2 - Load the Data

The first necessary step is to load the data from the internet, which is done by calling the download.file() function along with the specified arguments. The Sys.time() function is also called to avoid reproducibility issues, by printing a statement with the time and date regarding the download of the dataset. Note that the download will only be performed if the destination file doesn't already exists in the working directory. Then, both testing and training files are stored into the *data.testing* and *data.training* objects, respectively.

```
setwd("D:/PDFs escola/COURSERA/Data Science - JHU/8 - Practical Machine Learning/Course Project")

path <- getwd()

url.testing <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
url.training <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"

file.testing <- "pml-testing.csv"; file.training <- "pml-training.csv"

if (!file.exists(file.testing)) {
        download.file(url.testing, file.path(path, file.testing))
        Sys.time()
}

if (!file.exists(file.training)) {
        download.file(url.training, file.path(path, file.training))
        Sys.time()
}

rm(file.testing, file.training, url.testing, url.training, path)

data.testing <- read.csv(file = "pml-testing.csv")
data.training <- read.csv(file = "pml-training.csv")
```

Then, some basic packages are loaded to be used during the project.

```
library(caret)
library(rattle)
```

3 - Basic Exploratory Analyses and Treatment

In order to check the datasets' dimentions, the dim() function is used on *data.testing* and *data.traing*. Then, neat data frame is built and treated in order to compare both objects' dimensions.

```
test.dim <- dim(data.testing); training.dim <- dim(data.training)
data.dim <- rbind(test.dim, training.dim)
colnames(data.dim) <- c("Rows", "Columns")
rownames(data.dim) <- c("Testing  Dataset", "Training Dataset")
data.dim
```

```
##                   Rows Columns
## Testing  Dataset    20     160
## Training Dataset 19622     160
```

As stated in the output, the *data.testing* has 20 observations, while the *data.training* has 19.622 observations, which is a grotesque difference. According to the book Elements of Statistical Learning (HASTIE et al., 2nd edition, p. 222), a good split between train, validation and test groups would be to put 50% of the observations into the training set and the other observations equally split into validation and test groups. It is easy to note that testing and training sets being studied were not properly partitioned, hence, it is necessary to remanage *data.training* into more suitable proportions.

Before doing so, it is worth removing the NAs and zero variate variables. Afterall, they do not contribute to the prediction algarythm.

```
na.testing <- sapply(data.testing, function(y) sum(length(which(is.na(y)))))
na.training <- sapply(data.training, function(y) sum(length(which(is.na(y)))))

nzv.testing <- nearZeroVar(data.testing, saveMetrics = TRUE)
nzv.training <- nearZeroVar(data.training, saveMetrics = TRUE)

nas.df <- data.frame()
nas.df <- cbind(na.testing, nzv.testing$nzv, na.training, nzv.training$nzv)
colnames(nas.df) <- c("na.testing", "nzv.testing", "na.training", "nzv.training")
nas.df <- data.frame(nas.df)
nas.df[c(5,10,15,20,25,30,35,40,45,50,55,60),]
```

```
##                      na.testing nzv.testing na.training nzv.training
## cvtd_timestamp                0           0           0            0
## yaw_belt                      0           0           0            0
## skewness_roll_belt           20           1           0            1
## max_yaw_belt                 20           1           0            1
## amplitude_pitch_belt         20           1       19216            0
## var_roll_belt                20           1       19216            0
## stddev_yaw_belt              20           1       19216            0
## accel_belt_x                  0           0           0            0
## magnet_belt_z                 0           0           0            0
## var_accel_arm                20           1       19216            0
## stddev_pitch_arm             20           1       19216            1
## gyros_arm_x                   0           0           0            0
```

According to the randomly selected columns in both datasets, there are plenty of columns with missing values and near zero variance that need to be cleansed before the prediction begins. In order to do it, both *data.training* and *data.testing* are subsetted accordingly, and the resulting objects stored into *data2.training* and *data2.testing*. It is also worth mentioning that the first six columns of each dataset are labels, which can be safely removed.

```
data2.training <- data.training[ , nas.df$nzv.training == 0 &
                                   nas.df$na.training == 0]
data2.training <- data2.training[ ,-c(1:6)]

data2.testing <- data.testing[ , nas.df$nzv.training == 0 &
                                   nas.df$na.training == 0]
data2.testing <- data2.testing[ ,-c(1:6)]

test2.dim <- dim(data2.testing); training2.dim <- dim(data2.training)
data2.dim <- rbind(test2.dim, training2.dim)
colnames(data2.dim) <- c("Rows", "Columns")
rownames(data2.dim) <- c("Testing  Dataset", "Training Dataset")
data2.dim
```

```
##                    Rows Columns
## Testing  Dataset    20      53
## Training Dataset 19622      53
```

Note that the new *data2.testing* and *data2.training*, after being treated, have only 53 variables that will be used in the prediction. The, the *data2.training* is partitioned into a test set and a training set, in order to start the prediction, using the createDataPartition() function, passing the classe column as the argument. A seed is also specified to ensure reproducibility. Thus, the objects *data.subtraining* and *data.subtesting* are partitioned from the *data2.training* object.

```
set.seed(5)
subtraining <- createDataPartition(y = data2.training$classe, p = 0.75, list = FALSE)
data2.subtraining <- data2.training[subtraining, ]
data2.subtesting <- data2.training[-subtraining, ]

subtest.dim <- dim(data2.subtesting); subtraining.dim <- dim(data2.subtraining)
subdata2.dim <- rbind(subtest.dim, subtraining.dim)
colnames(subdata2.dim) <- c("Rows", "Columns")
rownames(subdata2.dim) <- c("subTesting  Dataset", "subTraining Dataset")
subdata2.dim
```

```
##                       Rows Columns
## subTesting  Dataset  4904      53
## subTraining Dataset 14718      53
```

Note that the *data2.subtraining* and *data2.subtesting* are partitions of the *data2.training* object by classe, the first containing 75% of the observations an the latter 25%.

4 - Fitting Prediction Models

The fitted models in this project are decision trees, random forest and generalized boosted regression models.

4.1 - Decision Tree Model

The goal here is to split the outcome into several different subgroups, evaluating the homogeneity within each subgroup, in order to build a decision tree. The split is done by finidng which variable best separates the outcomes into homogeneous subgroups, and then repeat it until all variables are used.

First, the prediction model is created using the function train() and passing the argument method = "rpart" over the *data2.subtraining object*. Note that a seed was set in order to avoid reproducibility issues.

```
set.seed(5)
model.dt <- train(classe ~ .,data = data2.subtraining, method = "rpart")
```

Then, the achieved model is used to predict, with the predict() function, what are the classes of the *data2.subtesting* object. The confusionMatrix() function is also used to check the accuracy of the predictions.

```
pred.model.dt <- predict(model.dt, data2.subtesting)
confusionMatrix(data2.subtesting$classe, pred.model.dt)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1250   30   94    0   21
##          B  415  323  211    0    0
##          C  406   33  416    0    0
##          D  362  143  299    0    0
##          E  120  122  238    0  421
##
## Overall Statistics
##
##                Accuracy : 0.4914
##                  95% CI : (0.4774, 0.5055)
##     No Information Rate : 0.5206
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3352
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4896  0.49616  0.33068       NA  0.95249
## Specificity            0.9383  0.85281  0.87959   0.8361  0.89242
## Pos Pred Value         0.8961  0.34036  0.48655       NA  0.46726
## Neg Pred Value         0.6287  0.91707  0.79205       NA  0.99475
## Prevalence             0.5206  0.13275  0.25653   0.0000  0.09013
## Detection Rate         0.2549  0.06586  0.08483   0.0000  0.08585
## Detection Prevalence   0.2845  0.19352  0.17435   0.1639  0.18373
## Balanced Accuracy      0.7140  0.67448  0.60514       NA  0.92246
```

According to the output, the results of the prediction based on a decision tree model were not amazing, specially regarding the prediction of the classe d activities, which were not captured at all by the model. Overall, it is possible to state that the model is not useful in the current project, as it presents an accuracy

of only 0.4914356, a rather low value. In this case, the obtained in-sample-error is 0.5085644, a very high number.

4.2 - Random Forest Model

The goal of a random forest model is to take repeated resamples of the training dataset in order to build classification trees on each of the bootsraped samples. Within each split, not only the data is split, but also the variables are bootstraped, so that only a subset of the variables is considered at eat potential split, building a whole forest of classification trees (each tree based on a bootstrap sample), while averaging for the best prediction.

First, the prediction model is created using the function train() and passing the argument method = "rf" over the *data2.subtraining* object. Note that a seed was set in order to avoid reproducibility issues.

```
set.seed(5)
model.rf <- train(classe ~ ., data = data2.subtraining, method = "rf")
```

Then, the achieved model is used to predict, with the predict() function, what are the classes of the *data2.subtesting* object. The confusionMatrix() function is also used to check the accuracy of the predictions.

```
pred.model.rf <- predict(model.rf, data2.subtesting)
confusionMatrix(data2.subtesting$classe, pred.model.rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    3  945    1    0    0
##          C    0    6  849    0    0
##          D    0    0   11  793    0
##          E    0    0    0    3  898
##
## Overall Statistics
##
##                Accuracy : 0.9951
##                  95% CI : (0.9927, 0.9969)
##     No Information Rate : 0.2851
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9938
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9979   0.9937   0.9861   0.9962   1.0000
## Specificity            1.0000   0.9990   0.9985   0.9973   0.9993
## Pos Pred Value         1.0000   0.9958   0.9930   0.9863   0.9967
## Neg Pred Value         0.9991   0.9985   0.9970   0.9993   1.0000
## Prevalence             0.2851   0.1939   0.1756   0.1623   0.1831
## Detection Rate         0.2845   0.1927   0.1731   0.1617   0.1831
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9989   0.9963   0.9923   0.9968   0.9996
```

According to the output, the results of the prediction based on a random forest model were very good. Overall, it presents an accuracy of 0.995106, a much higher value than the one achieved using the previous model. In this case, the obtained in-sample-error is 0.004894, a very low number, which indicates that the randon forest model was able to provide a reasonable prediction upon the *data2.subtesting* object.

4.3 - Generalized Boosted Regression Model

The basic idea of the boosting model is to take a large number of possibly weak predictors, and then combine and weight them into a stronger prediction.

First, the prediction model is created using the function train() and passing the argument method = "gbm" over the *data2.subtraining* object. Note that a seed was set in order to avoid reproducibility issues.

```
set.seed(5)
model.gb <- train(classe ~ ., data = data2.subtraining, method = "gbm", verbose = FALSE)
```

Then, the achieved model is used to predict, with the predict() function, what are the classes of the *data2.subtesting* object. The confusionMatrix() function is also used to check the accuracy of the predictions.

```
pred.model.gb <- predict(model.gb, data2.subtesting)
confusionMatrix(data2.subtesting$classe, pred.model.gb)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1370   20    4    1    0
##          B   25  891   32    1    0
##          C    0   32  812    9    2
##          D    0    0   29  773    2
##          E    1   14    7   15  864
##
## Overall Statistics
##
##                Accuracy : 0.9604
##                  95% CI : (0.9546, 0.9657)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.95
##  Mcnemar's Test P-Value : 2.362e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9814   0.9310   0.9186   0.9675   0.9954
## Specificity            0.9929   0.9853   0.9893   0.9924   0.9908
## Pos Pred Value         0.9821   0.9389   0.9497   0.9614   0.9589
## Neg Pred Value         0.9926   0.9833   0.9822   0.9937   0.9990
## Prevalence             0.2847   0.1951   0.1803   0.1629   0.1770
## Detection Rate         0.2794   0.1817   0.1656   0.1576   0.1762
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9871   0.9582   0.9539   0.9800   0.9931
```

According to the output, the results of the prediction based on a generalzied boosted regression model were good, even tough not better than the random forest model. Overall, it presents an accuracy of 0.9604405, a much higher value than the one achieved through the decision tree model, but lower than the one obtained through the random forest model. In this case, the obtained in-sample-error is 0.0395595, a low number, which indicates that the generalized boosted model was able to provide a reasonable prediction upon the *data2.subtesting* object.

4.4 - Overall Training Results

The obtained results can be simplified as follows:

```
df.results <- data.frame()

dtmodel <- c(unname(confusionMatrix(data2.subtesting$classe, pred.model.dt)$overall[1]),
             1 - unname(confusionMatrix(data2.subtesting$classe, pred.model.dt)$overall[1]))

rfmodel <- c(unname(confusionMatrix(data2.subtesting$classe, pred.model.rf)$overall[1]),
             1 - unname(confusionMatrix(data2.subtesting$classe, pred.model.rf)$overall[1]))

gbmodel <- c(unname(confusionMatrix(data2.subtesting$classe, pred.model.gb)$overall[1]),
             1 - unname(confusionMatrix(data2.subtesting$classe, pred.model.gb)$overall[1]))

df.results <- rbind(dtmodel, rfmodel, gbmodel)
colnames(df.results) <- c("Accuracy", "In-Sample-Error")
rownames(df.results) <- c("Decision Tree Model", "Random Forest Model", "Gene. Boosted Model")

df.results
```

```
##                       Accuracy In-Sample-Error
## Decision Tree Model 0.4914356     0.508564437
## Random Forest Model 0.9951060     0.004893964
## Gene. Boosted Model 0.9604405     0.039559543
```

As conveyed by the output, the most accurate model was the random forest, presenting an accuracy of 0.995106 and an in-sample-error of 0.004894, followed by the generalized boosted model, that presented an accuracy 0.9604405 and an in-sample-error of 0.0395595. The decision tree model, as noted before, severely under-performed, with an accuracy of 0.4914356 and an in-sample-error of 0.5085644.

5 - Prediction

The achieved model *model.rf*, the one that achieved the best results in the training process, is used to predict, with the predict() function, what are the classes of the *data2.testing* object. Also, the other models *model.dt* and *model.gb* are also used to predict, for comparison purposes.

```
pred.model.rf2 <- predict(model.rf, data2.testing)
pred.model.dt2 <- predict(model.dt, data2.testing)
pred.model.gb2 <- predict(model.gb, data2.testing)

df.final <- data.frame()
df.final <- rbind(pred.model.dt2, pred.model.rf2, pred.model.gb2)
rownames(df.final) <- c("Decision Tree Model", "Random Forest Model", "Gene. Boosted Model")

write.table(df.final, col.names = FALSE)
```

```
## "Decision Tree Model" 3 1 3 1 1 3 3 1 1 1 3 3 3 1 3 1 1 1 1 3
## "Random Forest Model" 2 1 2 1 1 5 4 2 1 1 2 3 2 1 5 5 1 2 2 2
## "Gene. Boosted Model" 2 1 2 1 1 5 4 2 1 1 2 3 2 1 5 5 1 2 2 2
```

Note that, even though the random forest model was slightly more accurate than the generalized boosted model, both were able to achieve the same practical results upon the *data2.testing*, predicting the same classes for all twenty samples. As expected, the decision tree model achieved different results, as a consequence of its excessively low accuracy.

Keeping in mind that the in-sample-error of the random forest model was measured as 0.004894, it is possible to state the the out-of-sample error of the same model would be a higher number, due to the fact that the achieved model problably overfitted the training the data. It means that it is capturing not only the signal, but also [a part of] the noise of the training data. Hence, when used to predict the testing data, it is inevitable to obtain a higher out-of-sample error, when compared to the obtained in-sample-error.