Practical ML : Course Project

Marcos Gestal

Friday, May 19, 2015

Background

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. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

See: Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013. online

Goal

The goal of this project is to predict the manner in which users did the exercise. This is the `classe` variable in the training set. The other variables should predict it. The present report describes how the model was built, the cross validation validation, an explanation about the choices, and so on.

Finally, the prediction model is used to predict 20 different test cases

Data

The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises.

Load data and pre-process steps

```
trainData <- file.path(getwd(), "Data/pml-training.csv")
testData <- file.path(getwd(), "Data/pml-testing.csv")

originalTrain <- read.csv(trainData, na.strings=c("NA", "#DIV/0!"))
originalTest <- read.csv(testData, na.strings=c("NA", "#DIV/0!"))
```

The original train dataset has 19622 observations and the test dataset has 20 observations. Both datasets have 160 features each one.

The dataset is filtered to remove variables with all values missing and to discard the irrelevant variables (mainly the descriptive ones) from the train and test sets.

```r
# Pre-processing of datasets: clean features (columns) with all values of NAs in
# the train dataset
notNullFeatures <- colSums(is.na(originalTrain))==0

train <- originalTrain[, notNullFeatures]
test <- originalTest[, notNullFeatures]

irrelevantFeatures <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2",
                        "cvtd_timestamp", "new_window", "num_window")

train <- train[, !names(train) %in% irrelevantFeatures]
test <- test[, !names(test) %in% irrelevantFeatures]
```

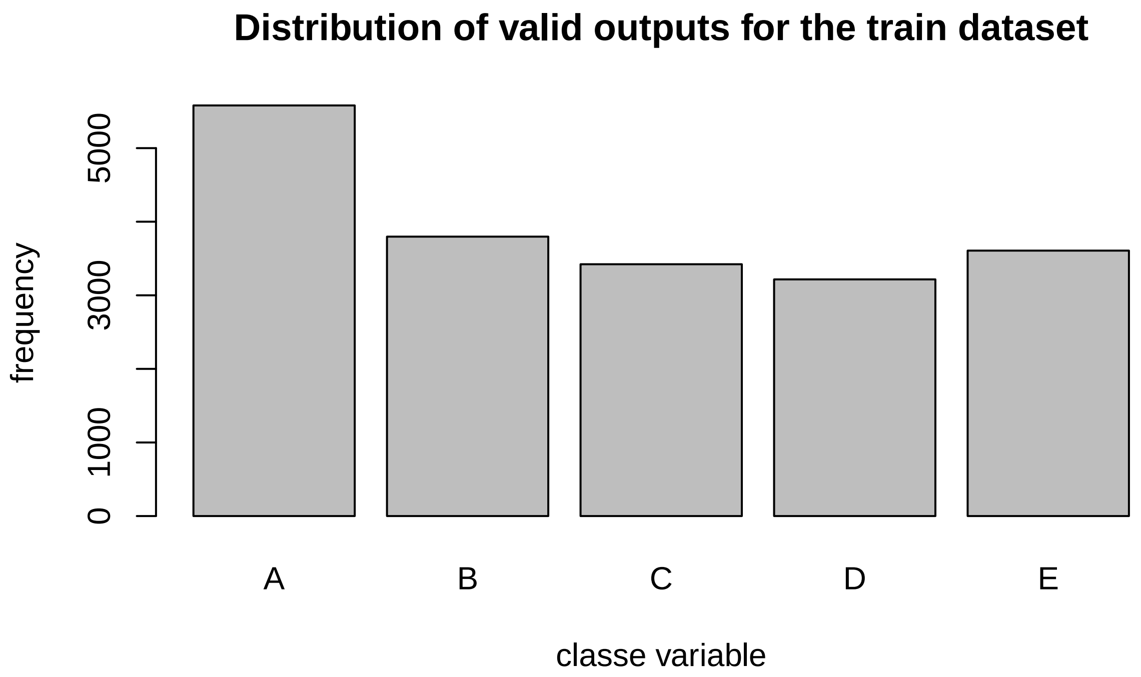
The final datasets used for the train and test phase have 53 features each one. Those features are the *same* in both datasets.

Data overview

Data contains info about how the users perform the exercises. The classe variable is a factor with 5 levels about the way one user perform one set of 10 repetitions of the unilateral dumbbell biceps curl. . Class A : according to the specification . Class B : throwing the elbows to the front . Class C : lifting the dumbbell only halfway . Class D : lowering the dumbbell only halfway . Class E : throwing the hips to the front

Figure 1 shows the distribution of the 5 different levels.

```r
plot(train$classe,
     main="Distribution of valid outputs for the train dataset",
     xlab="classe variable", ylab="frequency")
```



As the previous graphic shows, class A is a little more present in the dataset, but there is not any class either over- or under- represented. As we noted previously, Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

Partitioning the training set

In order to allow ***cross-validation***, We split the original data set, with the 60% of that samples used for train phase and the 40% for the validation (using a random subsampling without replacement approach).

The seed for the random number generation was set at 12345, so in order to reproduce the results below, the same value should be used in other studies.

```r
set.seed(12345)
inTrain <- createDataPartition(train$classe, p = 0.6, list = FALSE)

train <- train[inTrain, ]
validation <- train[-inTrain, ]
```

Predictions Models

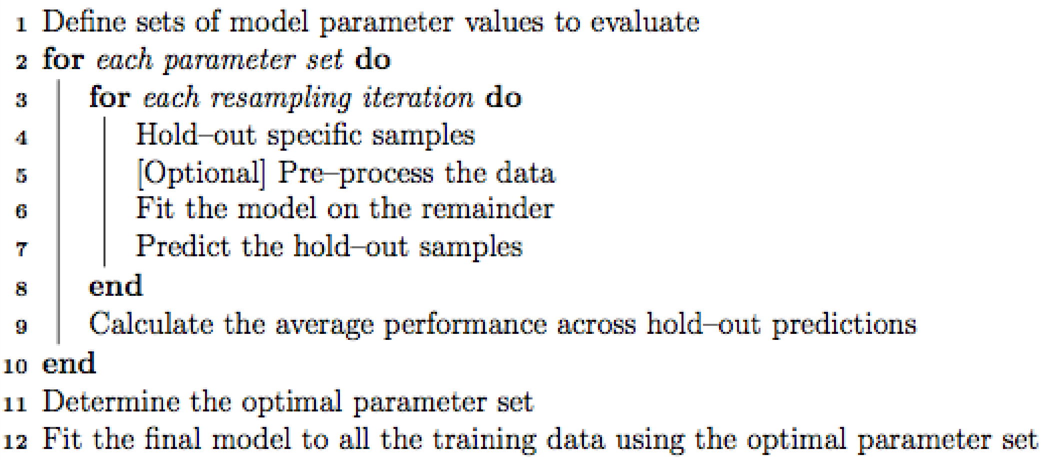
Several predictions models were tested: neural networks, regression trees and random forest. For each one, the confusion matrix shows the main statistic measures to check their performance.

The performance of the models is checked using the cross validation set to check problems like overfitting.

The confusionMatrix methods provides the Accuracy in the cross validation dataset, so we can calculate the ***expected out-of-sample error*** as 1-accuracy to check the percentage of missclassified observations. We will consider good models those that present an expected out-of-sample error below 1% (or 0.01)

Two approaches for each selection are used.

First, the function *train* from library caret is also used to compare the results. This function can be used to: . evaluate, using resampling, the effect of model tuning parameters on performance . choose the "optimal" model across these parameters . estimate model performance from a training set



```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

Once the model and tuning parameter values have been defined, the type of resampling should be also be specified (k-fold cross-validation (once or repeated), leave-one-out cross-validation and bootstrap). After resampling, the process produces a profile of performance measures is available to guide the user as to which tuning parameter values should be chosen. By default, the function automatically chooses the tuning parameters associated with the best value.

After that, a specific library implementation of the method is used and tested. This function receives a fixed set of parameters, so the performance related with computation time should be higher.

A 10-fold Cross Validation was used as trainControl parameter. Furthermore, libraries *parallel* and *doParallel* were used to improve the time responses.

```
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

trControl <- trainControl(method="cv", number=10,
                          allowParallel = TRUE)
```

Artificial Neural Networks

First, ANNs are used as model.

```
modelANN_caret <- train(classe ~ . , data=train, method="nnet",
                 trControl = trControl, verbose = FALSE, trace=FALSE)

## Check accuracy over the validation dataset
predictionsANN_caret <- predict(modelANN_caret, newdata = validation, type="raw")
confusionMatrix(predictionsANN_caret, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 908 137 130  89  41
##          B  25 144  56  50 126
##          C  81 213 196  49 102
##          D 193 115 110 390 146
##          E 118 314 334 199 464
##
## Overall Statistics
##
##                Accuracy : 0.4444
##                  95% CI : (0.4302, 0.4587)
##     No Information Rate : 0.2801
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2996
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.6853  0.15601  0.23729  0.50193   0.5279
## Specificity            0.8834  0.93249  0.88601  0.85732   0.7494
## Pos Pred Value         0.6958  0.35910  0.30577  0.40881   0.3247
## Neg Pred Value         0.8782  0.82005  0.84593  0.89751   0.8743
## Prevalence             0.2801  0.19514  0.17463  0.16427   0.1858
## Detection Rate         0.1920  0.03044  0.04144  0.08245   0.0981
## Detection Prevalence   0.2759  0.08478  0.13552  0.20169   0.3021
## Balanced Accuracy      0.7843  0.54425  0.56165  0.67963   0.6386
```

```
modelANN <- nnet(classe ~ . , data = train, size=17, maxit=2000,
                 abstol=1e-3, algorithm = "backprop",
                 preProc=c("center", "scale"), trace=FALSE, verbose = FALSE)

predictionsANN <- predict(modelANN, newdata = validation, type="class")
confusionMatrix(predictionsANN, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1140  104   90   82   69
##          B   32  484   50   26  180
##          C   58  125  537  199  209
##          D   90   79   93  391  105
##          E    5  131   56   79  316
##
## Overall Statistics
##
##                Accuracy : 0.6063
##                  95% CI : (0.5923, 0.6203)
##     No Information Rate : 0.2801
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5012
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8604   0.5244   0.6501  0.50322  0.35950
## Specificity            0.8987   0.9243   0.8486  0.90716  0.92963
## Pos Pred Value         0.7677   0.6269   0.4761  0.51583  0.53833
## Neg Pred Value         0.9430   0.8891   0.9198  0.90282  0.86411
## Prevalence             0.2801   0.1951   0.1746  0.16427  0.18584
## Detection Rate         0.2410   0.1023   0.1135  0.08266  0.06681
## Detection Prevalence   0.3140   0.1632   0.2385  0.16025  0.12410
## Balanced Accuracy      0.8795   0.7244   0.7494  0.70519  0.64456
```

ANN model does not provide good results in the classification or at least as not good as we expected. More tests should be done, but the neither nnet nor caret packages provides a good/easy tuning of the parameters like number of hidden layers, number el Process Elements per layer, training algorithms…

The required computation time is the greatest of the three methods probed.

Regression Trees

The second model test is based on Regression Trees (CART Algorithm)

```
modelTree_caret <- train(classe ~ . , data=train, method="rpart2",
                         preProc=c("center", "scale"),
                         trControl = trControl)
```

```r
## Check accuracy over the validation dataset
predictionsTree_caret <- predict(modelTree_caret, newdata = validation, type="raw")
confusionMatrix(predictionsTree_caret, validation$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   A   B   C   D   E
##          A 792  61   1  27   9
##          B 198 573 134  70 152
##          C 221 193 551 115 128
##          D 108  93 140 565  94
##          E   6   3   0   0 496
## 
## Overall Statistics
## 
##                Accuracy : 0.6294
##                  95% CI : (0.6154, 0.6432)
##     No Information Rate : 0.2801
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.5377
##  Mcnemar's Test P-Value : < 2.2e-16
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.5977   0.6208   0.6671   0.7272   0.5643
## Specificity            0.9712   0.8545   0.8317   0.8900   0.9977
## Pos Pred Value         0.8899   0.5084   0.4561   0.5650   0.9822
## Neg Pred Value         0.8612   0.9029   0.9219   0.9432   0.9093
## Prevalence             0.2801   0.1951   0.1746   0.1643   0.1858
## Detection Rate         0.1674   0.1211   0.1165   0.1195   0.1049
## Detection Prevalence   0.1882   0.2383   0.2554   0.2114   0.1068
## Balanced Accuracy      0.7845   0.7376   0.7494   0.8086   0.7810
```

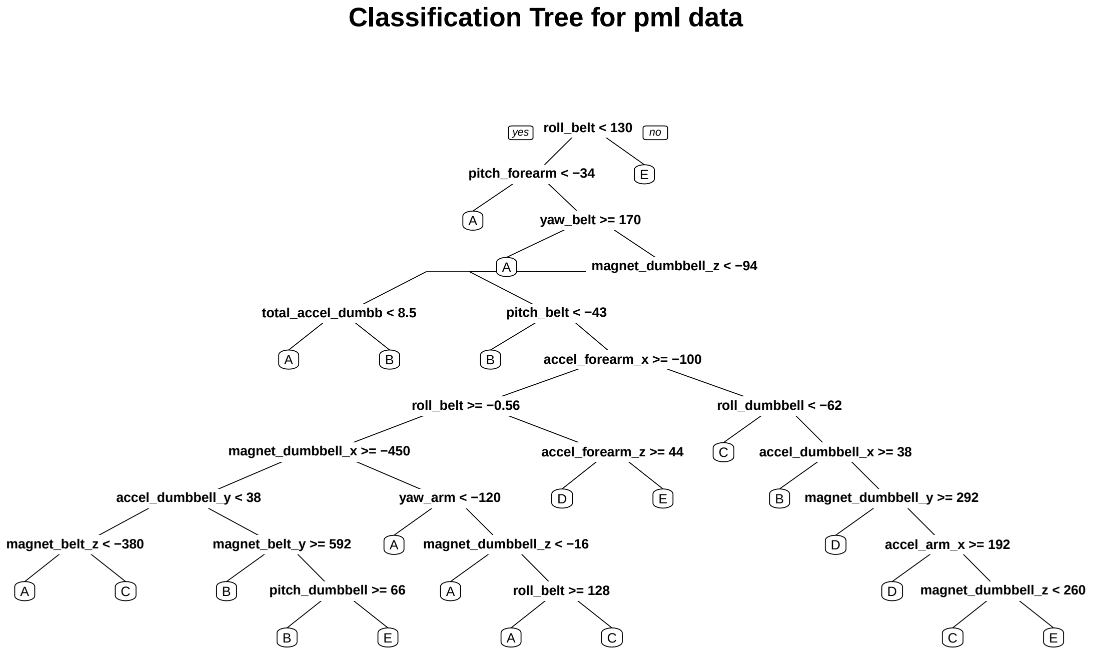
```r
system.time(modelTree <- rpart (classe ~ ., data=train, method="class"))
```

```
##    user  system elapsed
##    2.04    0.01    2.04
```

```r
predictionsTree <- predict(modelTree, newdata = validation, type="class")
confusionMatrix(predictionsTree, validation$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1108  163   18   40   37
##          B   24  491   62   22   28
##          C   67  177  739  190  138
##          D  103   82    7  514   38
##          E   23   10    0   11  638
##
## Overall Statistics
##
##                Accuracy : 0.7378
##                  95% CI : (0.7251, 0.7503)
##     No Information Rate : 0.2801
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6691
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8362   0.5320   0.8947   0.6615   0.7258
## Specificity            0.9242   0.9643   0.8535   0.9418   0.9886
## Pos Pred Value         0.8111   0.7831   0.5637   0.6909   0.9355
## Neg Pred Value         0.9355   0.8947   0.9746   0.9340   0.9405
## Prevalence             0.2801   0.1951   0.1746   0.1643   0.1858
## Detection Rate         0.2342   0.1038   0.1562   0.1087   0.1349
## Detection Prevalence   0.2888   0.1326   0.2772   0.1573   0.1442
## Balanced Accuracy      0.8802   0.7481   0.8741   0.8017   0.8572
```

```
rpart.plot(modelTree, main="Classification Tree for pml data")
```



In this case, the specific library provides better results than caret CART implementation.

Random Forest

Finally, we test a randomForest approach using a specific library (randomForest) and the generic caret library that performs a more exhaustive (and slow) search with k-fold cross validation.

```
modelRF_caret <- train(classe ~ . , data=train, method="rf",
                       preProcess = c("center", "scale"),
                       trControl = trainControl(method = "cv"))

## Check accuracy over the validation dataset
predictionsRF_caret <- predict(modelRF_caret, newdata = validation, type="raw")
confusionMatrix(predictionsRF_caret, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1325    0    0    0    0
##          B    0  923    0    0    0
##          C    0    0  826    0    0
##          D    0    0    0  777    0
##          E    0    0    0    0  879
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9992, 1)
##     No Information Rate : 0.2801
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  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   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2801   0.1951   0.1746   0.1643   0.1858
## Detection Rate         0.2801   0.1951   0.1746   0.1643   0.1858
## Detection Prevalence   0.2801   0.1951   0.1746   0.1643   0.1858
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

```
modelRF <- randomForest(classe ~ ., data=train, method="class")

predictionsRF <- predict(modelRF, newdata = validation, type="class")
confusionMatrix(predictionsRF, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1325    0    0    0    0
##          B    0  923    0    0    0
##          C    0    0  826    0    0
##          D    0    0    0  777    0
##          E    0    0    0    0  879
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9992, 1)
##     No Information Rate : 0.2801
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  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   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2801   0.1951   0.1746   0.1643   0.1858
## Detection Rate         0.2801   0.1951   0.1746   0.1643   0.1858
## Detection Prevalence   0.2801   0.1951   0.1746   0.1643   0.1858
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

In both cases, we obtain an accuracy of 100% (out-of-sample error = 0%)

Conclussions

Once the tests were performed several conclussions can be drawn

1. From the tested methods, randomForest achieves the best results (Expected out-of sample: 0%)
2. The caret package provides an easy an common interface to test different classification/regression methods instead of using specific libraries (although the last ones provides detailed graphs and functions but specific to a subset of its implemented methods)

Submission

The code to generate the files with the predictions is the next (Predictions are genereted from the Random Forest method):

```
predictionsFinal <- predict(modelRF, newdata = test, type="class")
predictionsFinal
```

```
##  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
```

```
# Create individual files for the submissions of the predictions

pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}

pml_write_files(predictionsFinal)
```