Machine Learning Project

Luis

March 25, 2016

Introduction

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).

Objective

The goal of the project is to predict the manner in which the users did the exercise. This is the "classe" variable in the training set. We will also use the prediction model to predict 20 different test cases.

Data

The testing and training datasets are available from here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data is loaded from these urls:

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
url1 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url2 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
training <- read.csv(url1)
ftesting <- read.csv(url2)
```

Data Preprocessing

Selection of variables

We will consider potential candidates to be predictor variables among the "belt","arm","dumbbell" and "forearm" variables without NA's or empty values in their records.

```
VarsWithNA <- sapply(training, function (x) any(is.na(x) | x == ""))
PotPred <- !VarsWithNA & grepl("belt|[^(fore)]arm|dumbbell|forearm", names(training))
PredNames <- names(training)[PotPred]
print(PredNames)
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"
```

Now we will create a new dataset as a subset of the training data including only the potential predictors plus the variable to predict. We will mirror all procedures in our testing dataset.

```
newtrain <- training[,c("classe",PredNames)]
fin_test <- ftesting[,c("problem_id",PredNames)]
```

And convert the variable "classe" to a factor:

```
newtrain$classe <- as.factor(newtrain$classe)
fin_test$problem_id <- as.factor(fin_test$problem_id)
dim(newtrain)
```

```
## [1] 19622    53
```

```
dim(fin_test)
```

```
## [1] 20 53
```

Since our train dataset is very large compared with our test dataset we can divide our training dataset in training and validation. We will set a seed and split 70-30.

```
set.seed(5678)
inTrain <- createDataPartition(newtrain$classe, p=0.7, list=FALSE)
train1 <- newtrain[inTrain,]
test1 <- newtrain[-inTrain,]
```

Data Standardarizing

Before starting with the model fitting, we standarize de datasets: centering and scaling of all datasets (training, testing, final test) using the values from the training dataset as reference.

```
CSobj <- preProcess(train1[,-1], method = c("center","scale"))
CStrain <- predict(CSobj,train1)
CStest  <- predict(CSobj,test1)
CSftest <- predict(CSobj,fin_test)
```

Model Fitting

The fact that we are working in an old laptop and the deadline is today impedes us regarding the use of "fancier" (and more adequate) algorithms like random forest ("rf") or boosting (i.e. "gbm"), we will have to content ourselves with a simple forest "rpart" model, pay no mind to cross-validation either. At least we can tune our model, let's start with that: we create a function based on the "trainControl" in order to tune the only parameter for the "rpart" algorithm: the "cp" complexity parameter.

```
folds=10
repeats=10
fitControl <- trainControl(method="repeatedcv",number=folds,repeats=repeats, classProbs=T,
allowParallel=T, summaryFunction=defaultSummary)

train.rpart <- train(classe ~ ., data=CStrain, method="rpart", tuneLength=10, trControl=fitControl)
```

```
## Loading required package: rpart
```

```
print(train.rpart)
```

```
## CART
##
## 13737 samples
##    52 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 12363, 12365, 12363, 12363, 12364, 12363, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.01139253  0.7068309  0.6280659  0.02103012   0.02628229
##   0.01342691  0.6728038  0.5865905  0.01705436   0.02093624
##   0.01922490  0.6434774  0.5490703  0.03055793   0.04330296
##   0.02014037  0.5997888  0.4869213  0.04583446   0.06770896
##   0.02034381  0.5881488  0.4698292  0.04434622   0.06584589
##   0.03193978  0.5322426  0.3908959  0.01727902   0.02661631
##   0.03234666  0.5212218  0.3760718  0.02044529   0.03131363
##   0.03956871  0.4949792  0.3404268  0.02917367   0.04556763
##   0.05009663  0.4262271  0.2290473  0.05647881   0.09753320
##   0.11168752  0.3196820  0.0538673  0.03780343   0.05760320
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.01139253.
```

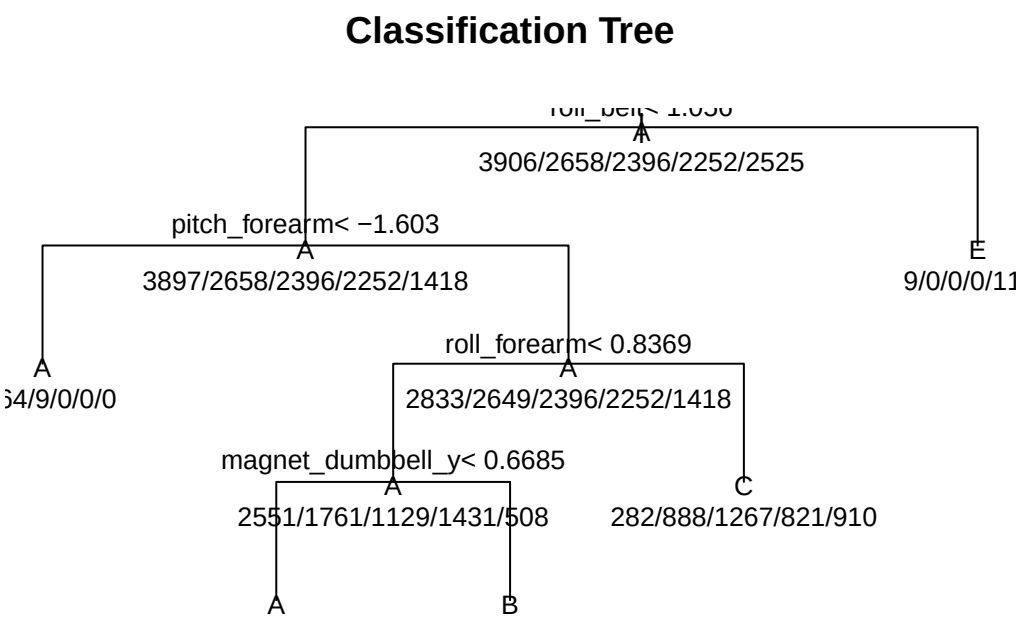
The best cp value is "cp = 0.01139253". We will continue by training the model with our training data:

```
modelFit <- train(classe ~ ., data = CStrain, method = "rpart", cp = 0.01139253)
```

Plotting the result tree

Here we can visualize the results of our model

```
plot(modelFit$finalModel, uniform = T, main = "Classification Tree")
text(modelFit$finalModel,use.n = T, all = T, cex = .8)
```



Evaluate in the training dataset

```
ResTrain <- predict(modelFit,CStrain)
confusionMatrix(ResTrain,CStrain[,"classe"])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3559 1106 1091 1035  361
##          B   56  664   38  396  147
##          C  282  888 1267  821  910
##          D    0    0    0    0    0
##          E    9    0    0    0 1107
##
## Overall Statistics
##
##                Accuracy : 0.4802
##                  95% CI : (0.4718, 0.4886)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.3213
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9112  0.24981  0.52880   0.0000  0.43842
## Specificity            0.6345  0.94250  0.74420   1.0000  0.99920
## Pos Pred Value         0.4976  0.51038  0.30398      NaN  0.99194
## Neg Pred Value         0.9473  0.83966  0.88201   0.8361  0.88765
## Prevalence             0.2843  0.19349  0.17442   0.1639  0.18381
## Detection Rate         0.2591  0.04834  0.09223   0.0000  0.08059
## Detection Prevalence   0.5206  0.09471  0.30341   0.0000  0.08124
## Balanced Accuracy      0.7728  0.59616  0.63650   0.5000  0.71881
```

As we can see in the "confusionMatrix" output the Accuracy is really poor, in fact we can expect our algorithm to wrongly classify roughly about half of the sample.

Evaluate in the testing dataset

```
ResTest <- predict(modelFit,CStest)
confusionMatrix(ResTest,CStest[,"classe"])
```

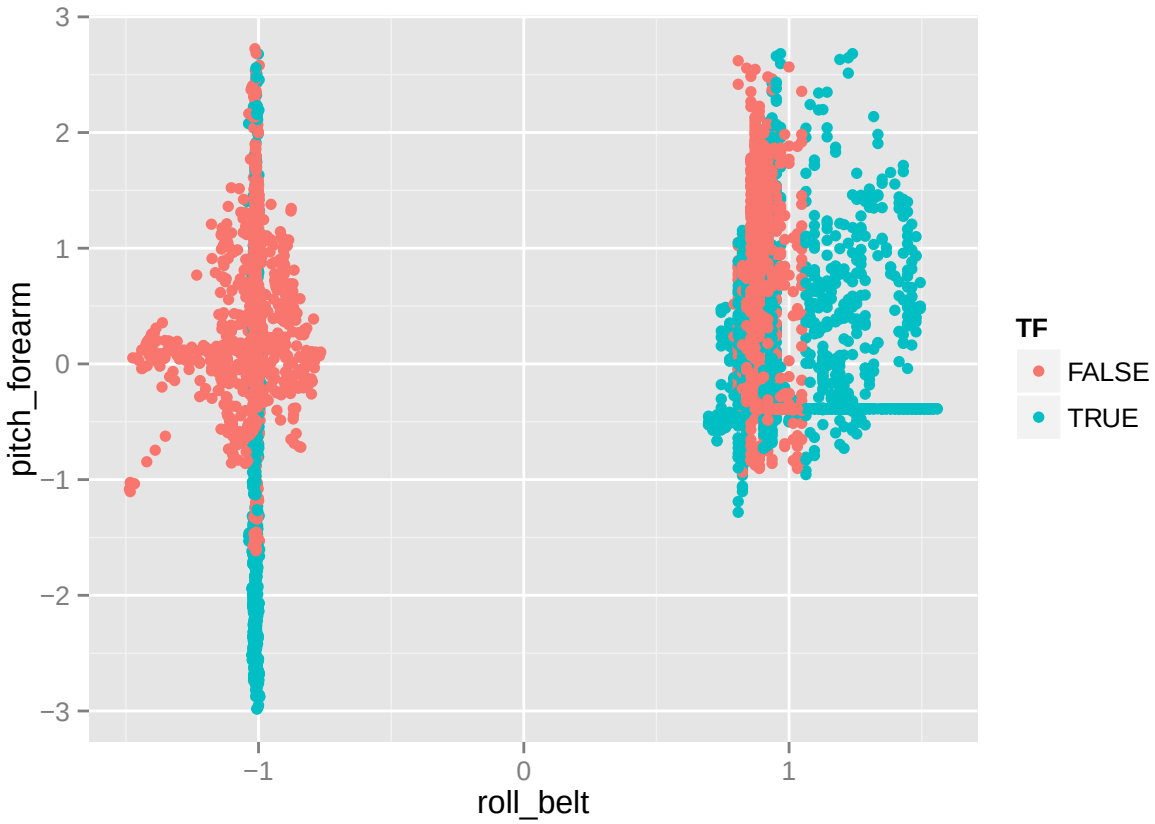
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1525  486  502  421  163
##          B   24  290   15  146   62
##          C  120  363  509  397  333
##          D    0    0    0    0    0
##          E    5    0    0    0  524
##
## Overall Statistics
##
##                Accuracy : 0.4839
##                  95% CI : (0.4711, 0.4968)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3255
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9110  0.25461  0.49610   0.0000  0.48429
## Specificity            0.6267  0.94796  0.75036   1.0000  0.99896
## Pos Pred Value         0.4924  0.54004  0.29559      NaN  0.99055
## Neg Pred Value         0.9466  0.84125  0.87581   0.8362  0.89582
## Prevalence             0.2845  0.19354  0.17434   0.1638  0.18386
```

```
## Detection Rate         0.2591  0.04928  0.08649   0.0000  0.08904
## Detection Prevalence   0.5263  0.09125  0.29261   0.0000  0.08989
## Balanced Accuracy      0.7688  0.60128  0.62323   0.5000  0.74162
```

No surprises here, at least the Accuracy doesn't get worse.

We can visualize the distribution of True and False prediction.

```
TF <- ResTest == CStest[,"classe"]
qplot(roll_belt,pitch_forearm,data = CStest,colour = TF)
```



There is a clear pattern that our model should be able to distinguish. Tough luck.

Final Model and Prediction

Crossed fingers:

```
modelFit$finalModel
```

```
## n= 13737
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 13737 9831 A (0.28 0.19 0.17 0.16 0.18)
##    2) roll_belt< 1.055594 12621 8724 A (0.31 0.21 0.19 0.18 0.11)
##      4) pitch_forearm< -1.60277 1073    9 A (0.99 0.0084 0 0 0) *
```

```
##      5) pitch_forearm>=-1.60277 11548 8715 A (0.25 0.23 0.21 0.2 0.12)
##       10) roll_forearm< 0.8369374 7380 4829 A (0.35 0.24 0.15 0.19 0.069)
##         20) magnet_dumbbell_y< 0.6685318 6079 3584 A (0.41 0.18 0.18 0.17 0.059) *
##         21) magnet_dumbbell_y>=0.6685318 1301  637 B (0.043 0.51 0.029 0.3 0.11) *
##       11) roll_forearm>=0.8369374 4168 2901 C (0.068 0.21 0.3 0.2 0.22) *
##    3) roll_belt>=1.055594 1116    9 E (0.0081 0 0 0 0.99) *
```

```
ResFinal <- predict(modelFit,CSftest)
ResFinal
```

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