Practical Machine Learning

Jorge B

Saturday, March 21, 2015

In this project the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbellThe to predict the manner in which 6 participants did their exercise.

- The data for this project comes from this source: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)
- The "classe" variable in the training set is the manner in which they did their exercise

The output of this report should include:

- Description how you built your model
- How cross validation is used
- The expected out of sample error

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

Exploring and understanding the data:

```
dim(data)
```

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

```
str(data[,15:20])
```

```
## 'data.frame':    19622 obs. of  6 variables:
##  $ skewness_roll_belt  : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1: Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt        : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Cleaning the data

As we can see there are many NAs and "#DIV/0!" that will be irrelevant for our analysis. I'll now load the data setting all those to be NAs. Additionally the first 7 variables are not useful for our analysis as they are time related and non-movement variables. Finally I'll remove all columns where any elements are NAs. As we can see below all columns with NA values have over 19k NAs, which represents most of its rows, so we can safely delete them.

```
data <- read.csv("pml-training.csv", na.strings = c("NA","", "#DIV/0!"))
table(colSums(is.na(data)))
```

```
## 
##     0 19216 19217 19218 19220 19221 19225 19226 19227 19248 19293 19294 
##    60    67     1     1     1     4     1     4     2     2     1     1 
## 19296 19299 19300 19301 19622 
##     2     1     4     2     6
```

```
data <- data[,colSums(!is.na(data))==nrow(data)]
data <- data[,c(-1:-7)]
```

Cross validation

I'll divide the data into a training set and a test set. The split being 70/30. The idea behind is:

- Build the model in the training set
- Evaluate on the test
- Estimate the out of sample error

```
TrainData <- createDataPartition(y =data$classe , p=0.7, list = FALSE)
training <- data[TrainData,]
testing <- data[-TrainData,]
dim(training)
```

```
## [1] 13737    53
```

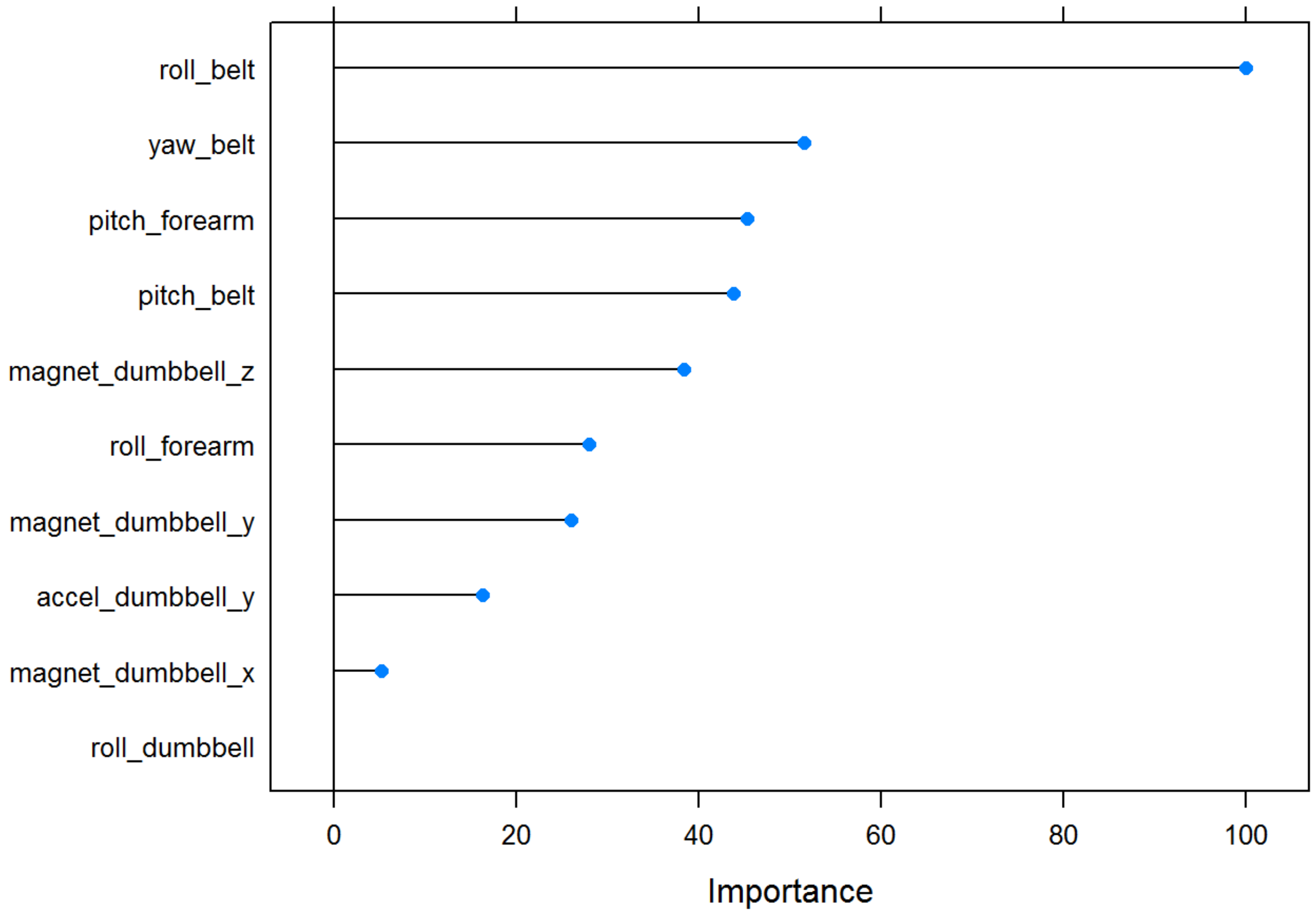
Now we have brought the varibles down to 52 (+1 that is the output) so probably any algorithm will be rather slow as we have arround 13k observations.

Buiding the model Machine Learning algorithm

In order to see if we can reduce them we can run a test (random forrest) with just a few iterations and rank the importance of the variables. This could provide a good overview of the dataset we have and if we can eliminate some variables:

```
modFit <- train(classe ~ ., data=training, methods='rf', number = 3, repeats = 3, ntree=5)
modfitImp <- varImp(modFit)
```

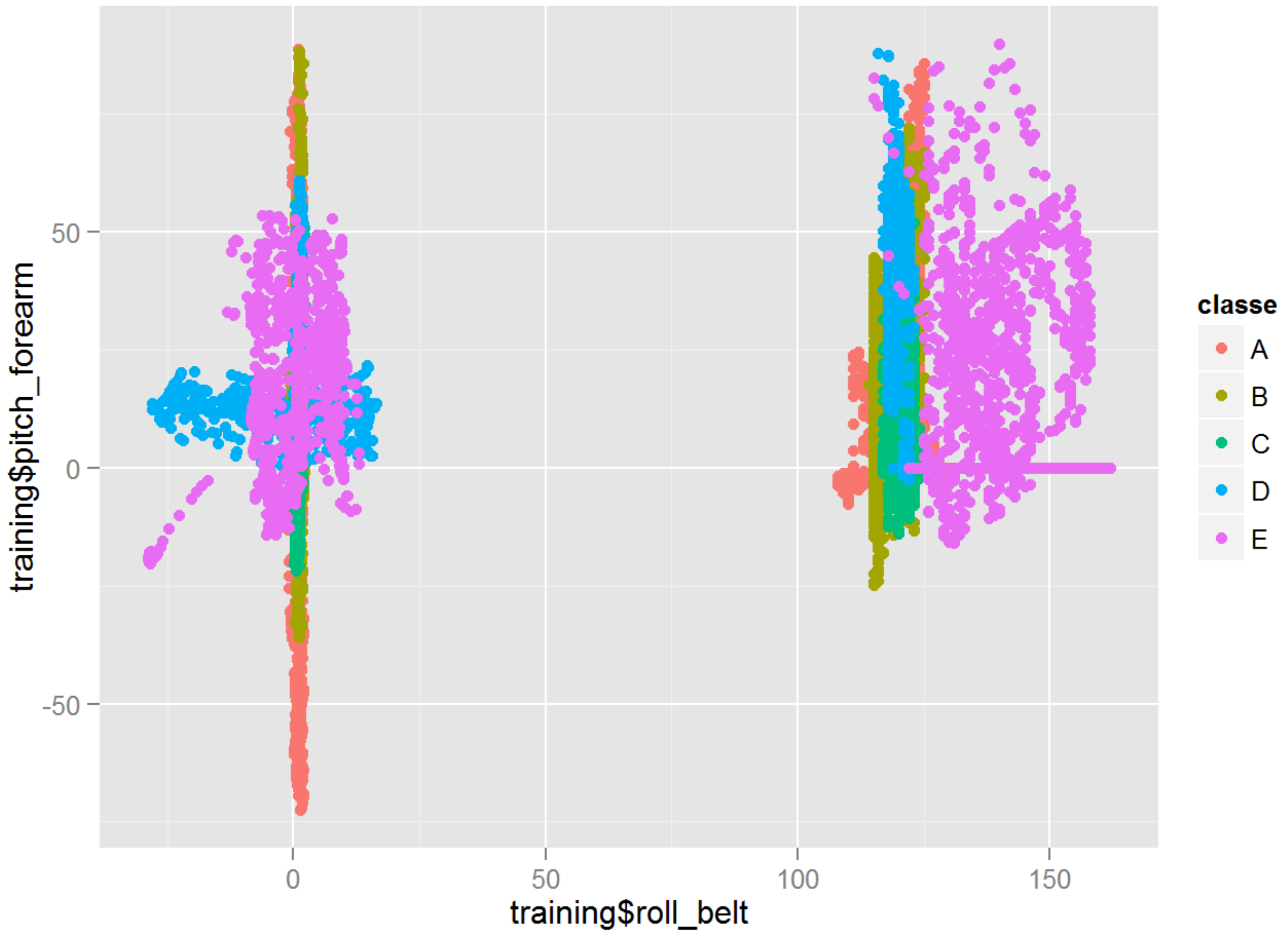
```
plot(modfitImp, top = 10)
```



As we can see from this plot some variables have relatively high importance when it comes to explaining the outcome.

Let’s explore the top two variables in terms of importance:

```
qplot(training$roll_belt, training$pitch_forearm, colour = classe, data = training)
```



There seems to be some paterns that explain the different exercises.

Building the final model

I'll include the first 10 predictors and see what the new model looks like. Now that the dimensions will be reduced we can increase the number of trees in the simulation to a more significant number (500).

```
variables <- c("roll_belt", "pitch_forearm", "yaw_belt", "pitch_belt", "magnet_dumbbell_z", "magne
t_dumbbell_y", "roll_forearm", "accel_dumbbell_y", "magnet_dumbbell_x", "roll_dumbbell", "classe")
```

```
training <- (training[, variables])
modFit2 <- train(classe ~ ., data=training, methods='rf', number = 3, repeats = 3, ntree=500)
```

Cross Validation

Let's test the model in our training dataset and see the results in a table:

```
testing <- (testing[, variables])
pred <- predict(modFit2, testing)
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.1.3
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
table(pred, testing$classe)
```

```
##
## pred    A    B    C    D    E
##    A 1661    3    0    0    0
##    B    7 1121    8    0    2
##    C    6   11 1016    1    0
##    D    0    4    2  962    0
##    E    0    0    0    1 1080
```

Out of sample error

From here we can calculate the out of sample error with the function confusionMatrix:

```
confusionMatrix(testing$classe,pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1661    7    6    0    0
##          B    3 1121   11    4    0
##          C    0    8 1016    2    0
##          D    0    0    1  962    1
##          E    0    2    0    0 1080
##
## Overall Statistics
##
##                Accuracy : 0.9924
##                  95% CI : (0.9898, 0.9944)
##     No Information Rate : 0.2828
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9903
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9851   0.9826   0.9938   0.9991
## Specificity            0.9969   0.9962   0.9979   0.9996   0.9996
## Pos Pred Value         0.9922   0.9842   0.9903   0.9979   0.9982
## Neg Pred Value         0.9993   0.9964   0.9963   0.9988   0.9998
## Prevalence             0.2828   0.1934   0.1757   0.1645   0.1837
## Detection Rate         0.2822   0.1905   0.1726   0.1635   0.1835
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9976   0.9906   0.9903   0.9967   0.9993
```

The model seems to be pretty accurate (0.9874)