Human Activity Recognition analysis

ptitmatheux

July 19, 2017

Summary

We train a random forest model on the dataset provided by http://groupware.les.inf.puc-rio.br/har in order to predict if a physical exercise (namely the Unilateral Dumbbell Biceps Curl) was performed correctly or not knowing the measurements provided by a panel of four devices weared by the person performing the exercise. We perform a Principal Component Analysis as feature extraction in the preprocessing step which reduces the number of predictors 18. The results obtained on a separate testing set exhibit an overall "out of sample" estimated accuracy of 97%.

Exploring and preparing the data

After loading the data, we remove columns containing essentially NA or empty values. These columns seem to correspond to agregated features; we shall disregard them in the present analysis and keep only the raw measurements. We can do this as follows:

```
modeling <- read.csv("data/pml-training.csv")
predicting <- read.csv("data/pml-testing.csv")

# checking for NA and empty values:
check.NA.model <- apply(modeling, MARGIN=2, FUN=function(COL) { sum(is.na(COL)) })
print(table(check.NA.model))
```

```
## check.NA.model
##     0 19216
##    93    67
```

```
check.empty.model <- apply(modeling, MARGIN=2, FUN=function(COL) { sum(COL == "") })
print(table(check.empty.model))
```

```
## check.empty.model
##     0 19216
##    60    33
```

```
# removing those columns from modeling set:
modeling <- subset(modeling, select=names(which(check.empty.model == 0))) # this removes also the NA va
```

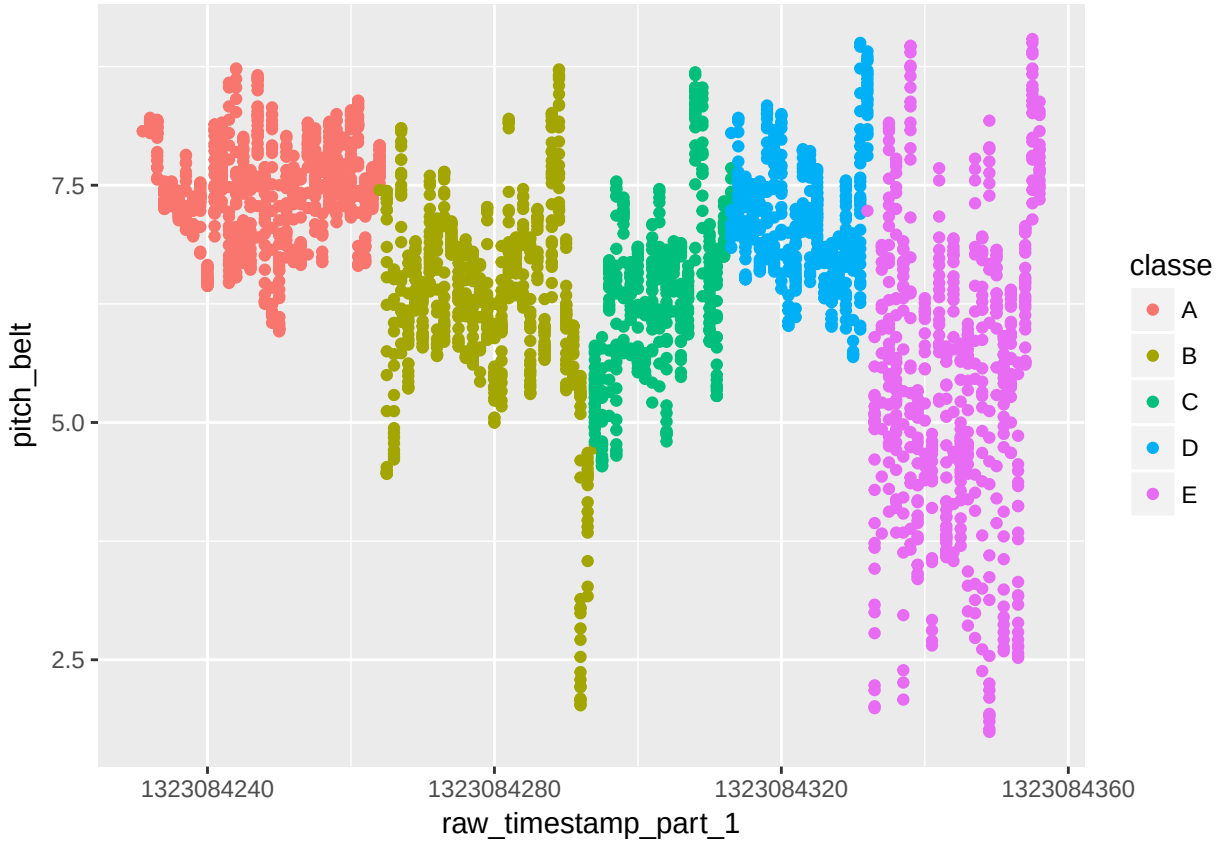
```
## check.NA.pred
##   0  20
##  60 100
```

```
## check.empty.pred
##  0
## 60
```

As we see, in the modeling data set, there are 67 columns which are essentially (97% of the rows) filled with 'NA', and 33 which are empty (empty string "" in the csv). We perform a similar treatment with the predicting dataset (code not shown).

Let us have a look at some data by considering raw measurements recorded on the user "Carlitos"; for example, the variable "pitch_belt":

```
library(ggplot2)
CHECK <- modeling[modeling$user_name == "carlitos",]
qq <- qplot(raw_timestamp_part_1, pitch_belt, colour=classe, data=CHECK)
qq
```



There are approximately 20 records by second. As we shall see, the dispersion of the records during a given 1-second-long window can be quite noisy. In a finer tuning of the model, we could perform a tailored summarizing feature extraction such as taking the mean, variance, range, etc. on sliding windows of fixed width in order to reduce the noise. However, as a first approach and for simplicity, we shall keep the raw measurements so far.

Before turning to the training of the model, we remove variables related with the user and the timestamps, as well as the very first indexing variable (which is very strongly but artificially correlated with the output variable). Note that by removing all references to time, we implicitly perform the assumption that the variables are not significantly autocorrelated in time. This is certainly not completely true, but hopefully this assumption won't have a too bad impact on our model performance.

```
submodeling <- subset(modeling, select=-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvt
subpredicting <- subset(predicting, select=-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2,
```

Splitting the data for train and testing

We now split the data from the first dataset (submodeling) into a training and a test set using a random sampling which equally dispatches the values of the output variable; we keep 40% of the data for testing while the remaining will be used for training:

```
library(caret)
```

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

```
set.seed(1234)
trainIndex = createDataPartition(submodeling$classe, p = 0.6)[[1]]
training = submodeling[trainIndex,]
testing = submodeling[-trainIndex,]
```

Preprocessing

As the number of raw input variables is quite large (52), we perform a feature selection using a Principal Component Analysis in which we set the threshold for explained variance set at 90%. This reduces the number of input variables to 18. (This step will be performed computationally in the next section.)

Model selection and training

In our case, we face with a classification problem in which the output variable can take 5 different levels. Due to the noisy nature of the data, we shall consider a random forest model with bootstrap resampling (we leave the default boostrap settings, i.e., 25 repetitions). In order to speed up the computation, we shall take advantage of the parallelization facilities provided by R:

```
library(doMC)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
registerDoMC(cores=4)

fit.rf <- train(classe ~ ., method="rf",
                preProcess="pca",
                trControl=trainControl(preProcOptions = list(thresh=0.9)),
                data=training)
```

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

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

N.B. The preprocessing step using PCA is performed in the previous command, just before training the random forest.

Model assessment

We now can test our model by performing a prediction on the testing set, then comaring with the true values:

```
test.rf <- predict(fit.rf, testing)
confMat <- confusionMatrix(test.rf, testing$classe)
print(confMat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2210   30    7    9    0
##          B    3 1458   26    0    7
##          C    9   26 1319   62   11
##          D   10    1   13 1213   12
##          E    0    3    3    2 1412
##
## Overall Statistics
##
##                Accuracy : 0.9702
##                  95% CI : (0.9662, 0.9738)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9623
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9901   0.9605   0.9642   0.9432   0.9792
## Specificity            0.9918   0.9943   0.9833   0.9945   0.9988
## Pos Pred Value         0.9796   0.9759   0.9243   0.9712   0.9944
## Neg Pred Value         0.9961   0.9906   0.9924   0.9889   0.9953
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2817   0.1858   0.1681   0.1546   0.1800
## Detection Prevalence   0.2875   0.1904   0.1819   0.1592   0.1810
## Balanced Accuracy      0.9910   0.9774   0.9738   0.9689   0.9890
```

As we can see, we obtain an overall "out of sample" accuracy of 97%, which seems a good result. The respective accuracies by output classes are 99% for class A, 96% for class B, 96% for class C, 94% for class D and 98% for class E.