Cousera 06: Practical Machine Learning

DatSci007

3 Mai 2019

00_Information

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 â□□ 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 (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

01_Prep

load packages and customized functions if neccessary

02_Analysis

http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

This human activity recognition research has traditionally focused on discriminating between different activities, i.e. to predict “which” activity was performed at a specific point in time (like with the Daily Living Activities dataset above). The approach we propose for the Weight Lifting Exercises dataset is to investigate “how (well)” an activity was performed by the wearer. The “how (well)” investigation has only received little attention so far, even though it potentially provides useful information for a large variety of applications,such as sports training.

In this work (see the paper) we first define quality of execution and investigate three aspects that pertain to qualitative activity recognition: the problem of specifying correct execution, the automatic and robust detection of execution mistakes, and how to provide feedback on the quality of execution to the user. We tried out an on-body sensing approach (dataset here), but also an “ambient sensing approach” (by using Microsoft Kinect - dataset still unavailable)

Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience. We made sure that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz5mqflBUdB (http://groupware.les.inf.puc-rio.br/har#ixzz5mqflBUdB)

02_01_Cleaning

Load the data (test and train) Note: Y we want to predict using all other vars is “classe”. This variable is missing in dat_test_raw, because that’s the variable we want to predict afterwards.

First step is data prep. Chance the class of our DV into factor, so it’s easier to calculate the confusion matrix later on. Then we exclude all variables with a variance of nearly zero and all variables with more than 95% of missings.

After this cleaning step 55 variables including name of participant, num_window and classe.

```
dim(dat_train_raw)
```

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

```
dat_train_raw <- dat_train_raw %>%
  dplyr::select(-dplyr::one_of(List_ZeroVar, List_MostNA), # delete them with sum NA > 95% an
d them with nearly zero veraiance
                -c(V1, dplyr::contains("timestamp"))) # other vars without useful information
dim(dat_train_raw)
```

```
## [1] 19622    55
```

```
# now we have 55 cols to work with
```

02_02_Splitting

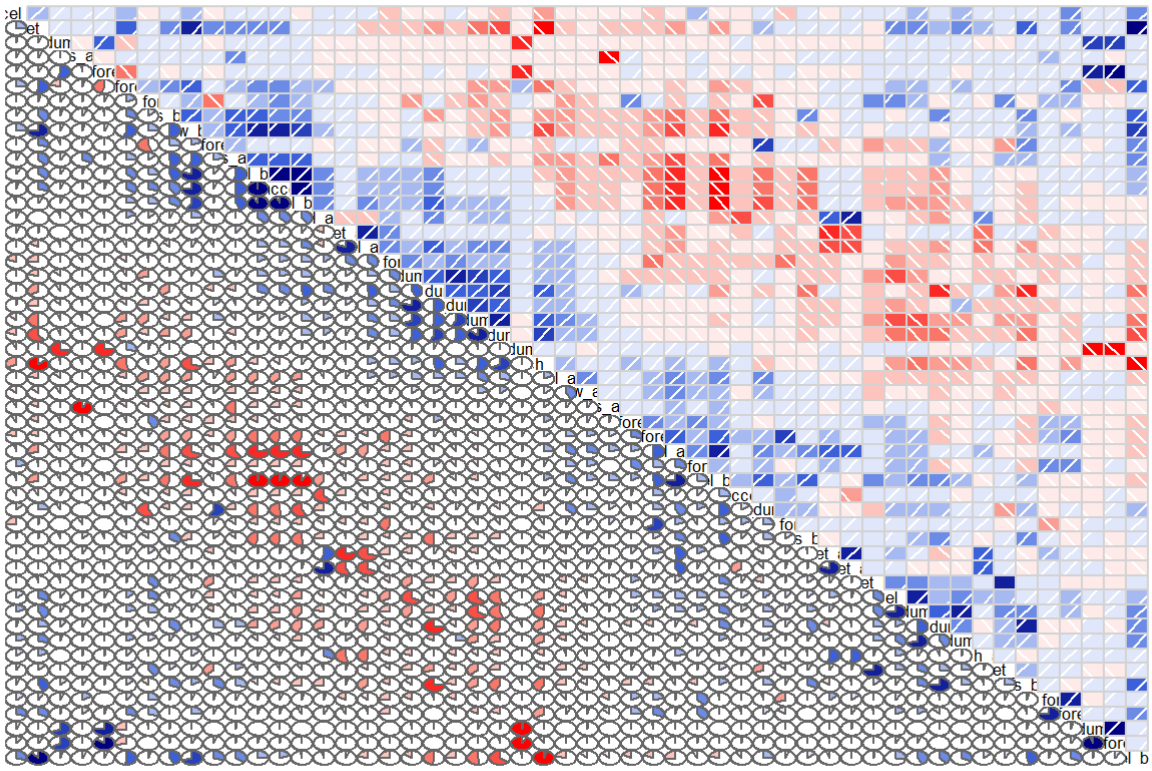
After the cleaning is done, divide the dat_train_raw into one test (don’t get confused with dat_test_raw) and one training sample. We use 75% for training and 25% for testing

```
## [1] 14718    55
```

```
## [1] 4904   55
```

02_03_Vizualisation

let’s have a closer look on the data. By calculating the correlation of all variables we notice, that there are some highlig correlated variables. Our plot proofs it.



On top of that, the number of variables might be too high, so let's use principal component analysis to reduce the number of variables. We need only 12 components to explain at leat 80% of the variance.

```
## Created from 14718 samples and 52 variables
##
## Pre-processing:
##   - centered (52)
##   - ignored (0)
##   - principal component signal extraction (52)
##   - scaled (52)
##
## PCA needed 12 components to capture 80 percent of the variance
```

02_04 Modeling

Finaly the fun part begins :) Methods we want to use: 1) rpart (Regression Tree) 2) naive_bayes (Naive Bayes) 3) xgbTree (XGBoost) 4) stack (Combination of three methods above)

For every method we stop the running time and calcuate the accuracy/error in order to compare them.
We use 5-k-crossvalidation in order to avoid over-fitting, but still expect the out-of-sample (test) error to be higher then the training error.

02_04_01 rpart

We try both approaches, one with pca and one without pca. But we notice, that the approach with pca (accuracy = 0.36) performs much worse than the approach without pca (accuracy = 0.56). so let's drop this approach for the other methods.

```
start_pca_rpart <- proc.time()
model_pca_rpart <- caret::train(dat_train[, -c(1, 2, 55)], dat_train$classe, method = "rpart"
,
                                preProcess = "pca",
                                trControl = trainControl(preProcOptions = list(thresh = 0.8
)))
end_pca_rpart <- proc.time()
pred_pca_rpart <- predict(model_pca_rpart, dat_test)
confusionMatrix(dat_test$classe, pred_pca_rpart)$overall['Accuracy']
```

```
##  Accuracy
## 0.3209625
```

```
### using pca result in bad prediction --> don't use it


### model we test
start_rpart <- proc.time()
model_rpart <- caret::train(dat_train[, -c(1, 2, 55)], dat_train$classe, method = "rpart",
                            trControl = trainControl(method = "cv", number = 5))
end_rpart <- proc.time()
pred_rpart <- predict(model_rpart, dat_test)
(conf_rpart <- confusionMatrix(dat_test$classe, pred_rpart)$overall['Accuracy'])
```

```
##  Accuracy
## 0.4971452
```

```
model_rpart_wo_cv <- caret::train(dat_train[, -c(1, 2, 55)], dat_train$classe, method = "rpar
t")
pred_rpart_wo_cv <- predict(model_rpart_wo_cv, dat_test)
(conf_rpart_wo_cv <- confusionMatrix(dat_test$classe, pred_rpart_wo_cv)$overall['Accuracy'])
```

```
##  Accuracy
## 0.4971452
```

02_04_02 naive_bayes

Naive Bayes performs much better than rpart with an accuracy of 0.74

```
start_naive_bayes <- proc.time()
model_naive_bayes <- caret::train(dat_train[, -c(1, 2, 55)], dat_train$classe, method = "naiv
e_bayes",
                                  trControl = trainControl(method = "cv", number = 5))
end_naive_bayes <- proc.time()

pred_naive_bayes <- predict(model_naive_bayes, dat_test)
(conf_naive_bayes <- confusionMatrix(dat_test$classe, pred_naive_bayes)$overall['Accuracy'])
```

```
##  Accuracy
## 0.7473491
```

02_04_03 xgbTree

xgbTree delivers nealy perfect prediction on our test sample with an accuracy of 0.996.

```
# setup
grid_default <- expand.grid(nrounds = 100,  max_depth = 6, eta = 0.2,   gamma = 0,   colsampl
e_bytree = 0.7,  min_child_weight = 1,  subsample = 0.7)
train_control <- caret::trainControl(method = "none", verboseIter = FALSE, allowParallel = TR
UE)

start_xgb <- proc.time()
model_xgb <- caret::train(x = dat_train[, -c(1, 2, 55)], y = dat_train$classe,
                          trControl = train_control, tuneGrid = grid_default,
                          method = "xgbTree", verbose = TRUE)
end_xgb <- proc.time()

pred_xgb <- predict(model_xgb, dat_test)
(conf_xgb <- confusionMatrix(dat_test$classe, pred_xgb)$overall['Accuracy'])
```

```
##  Accuracy
## 0.9965334
```

02_04_04 stack

xgBoost on its own performs outstanding. therefora combining all three methods won't result in better precidtion.

```
# stack
dat_stack <- data.frame(pred_rpart, pred_naive_bayes, pred_xgb , classe = dat_test$classe)
model_stack <- randomForest(classe ~., data = dat_stack)
pred_stack <- predict(model_stack, dat_test)

(conf_stack <- confusionMatrix(dat_test$classe, pred_stack)$overall['Accuracy'])
```

```
##  Accuracy
## 0.9942904
```

02_05_Conclusion

The runningtime of xgbTree with 0.185 minutes is higher than the running time of rpart (0.089) and naive_bayes (0.065). Nevertheless, xgbTree delivers nearly 100% accuracy and performs way better than rpart and naive bayes. We don't get any furhter improvements by stacking the three methods.

```
##                                method  time accuracy  error
## 1                               rpart 0.083     0.50 0.5029
## 2                         naive_bayes 0.063     0.75 0.2527
## 3                             xgbTree 0.167     1.00 0.0035
## 4 stack: rpart + naive_bayes + xgbTree    NA     0.99 0.0057
```

02_06_Prediction on Test data wihout class information

For the final prediction of our 20 new observation we use the model of xgbTree.

```
predict(model_xgb, dat_test_raw[, -c(1, 2, 55)])
```

```
##  [1] 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
```

END