Machine learning and Human Activity Recognition

Practical Machine Learning course project assignment

StefMT2970

Overview

This project is a result of a project assignment for the JHU course on Machine Learning (part of the data science specialisation, offered via Coursera).

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, the 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).

Geting the data set

The data is provided by :

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.

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz3cSdc7Ewm

The raw data sets are loaded into two data frames:

```r
setwd("d:/dev/app/gitrepo/PML_HAR")
if(!file.exists("./data")){
  dir.create("./data")
}
setwd("./data")

## check if file exists first, then download and unzip
if(!file.exists("train.csv")){
  file_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
  download.file(url = file_url, destfile = "train.csv")
}
if(!file.exists("test.csv")){
  file_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
  download.file(url = file_url, destfile = "test.csv")
}
trainDS <- read.csv("train.csv", na.strings = c("NA", "#DIV/0!", ""))
testDS <- read.csv("test.csv", na.strings = c("NA", "#DIV/0!", ""))

table(trainDS$user_name, trainDS$classe)
```

```
##
##              A    B    C    D    E
##   adelmo   1165  776  750  515  686
##   carlitos  834  690  493  486  609
##   charles   899  745  539  642  711
##   eurico    865  592  489  582  542
##   jeremy   1177  489  652  522  562
##   pedro     640  505  499  469  497
```

The 2 datasets have the same structure except for the last variable which is *classe* in the training set (the outcome) and *problem_id* in the test set (to be submitted). There is a fair distribution between outcomes by test person. The *na.strings* variable in *read.csv* is important to capture all NA's.

Cleaning the data

Is.NA analysis

On first glance, a lot of the variables have NA's. The choice is to remove columns with more than 95% NA's. Pay attention to the na.strings variable in the initial read.csv instruction. (That was actually the first step in the cleaning.)

```
sum(colSums(is.na(trainDS)) > 0.95 * dim(trainDS)[1])
```

```
## [1] 100
```

```
removeColumns <- colSums(is.na(trainDS)) > 0.95 * dim(trainDS)[1]
trainNoNa <- trainDS[ , !removeColumns]
testNoNa <- testDS[ , !removeColumns]
```

Removing additional columns

The goal of this algorithm is to predict the manner in which the participants did the exercise, captured in the outcome variable *classe*. We will not complicate matters by considering time stamps. The variable *new_window* and *num_window* capture the time window and are removed. *user_name* has been removed to avoid the risk that 1 test person does a better job than another. We want to train from the sensor data.

```
trainNZV <- nearZeroVar(trainNoNa, saveMetrics=TRUE)
sum(trainNZV$nzv)
```

```
## [1] 1
```

```
trainNZV[trainNZV$nzv,]
```

```
##            freqRatio percentUnique zeroVar  nzv
## new_window  47.33005    0.01019264   FALSE TRUE
```

```
extraColumns <- names(trainNoNa) %in%
   c("X",
     "user_name",
     "raw_timestamp_part_1",
     "raw_timestamp_part_2",
     "cvtd_timestamp",
     "new_window",
     "num_window"
     )
trainClean <- trainNoNa[!extraColumns]
testClean <- testNoNa[!extraColumns]
dim(trainClean); dim(testClean)
```

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

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

Exploring the data set

With so many columns left, this is not easy to see if some variables are more important than others. The goal of this paper is to provide an algorithm to predict the outcome, inference questions are subordinate but nevertheless interesting. In the prediction chapter we will take some sidesteps and try to get information on importance of variables.

Predictive analysis

Data Splitting

The clean training data set is split in a training and testing set (50/50, to reduce excessive runtimes later)

```
set.seed(2015) # For reproducibile purpose
inTrain <- createDataPartition(trainClean$classe,p=0.5, list=F)
training <- trainClean[inTrain, ]
testing <- trainClean[-inTrain, ]
```

Training the individual algorithms

A couple of algorithms are trained (using caret package) and then compared. - RF random forests - SVM with radial basis - QDA quadratic discriminant analysis (LDA was performing very poor) - GBM boosting

Cross validation is used with a K-fold=5 parameter, or standard Caret bootstrap settings (QDA only).

```
fitControl <- trainControl(method = "cv",number = 5)

t.RF <- system.time(
   {fit.RF <- train(classe ~ ., data=training,
                method="rf",
                trControl=fitControl)}
)
```

```r
t.GBM <- system.time(
   {fit.GBM <- train(classe ~ ., data=training,
                     method="gbm",
                     trControl = fitControl, verbose=FALSE)}
)
```

```r
t.QDA <- system.time(
   {fit.QDA <- train(classe ~ ., data=training,
                     method="qda", verbose=FALSE)}
)
```

```r
t.SVM <- system.time(
   {fit.SVM <- train(classe ~., data = training,
                     method="svmRadial",
                     trControl = fitControl)}
)
```

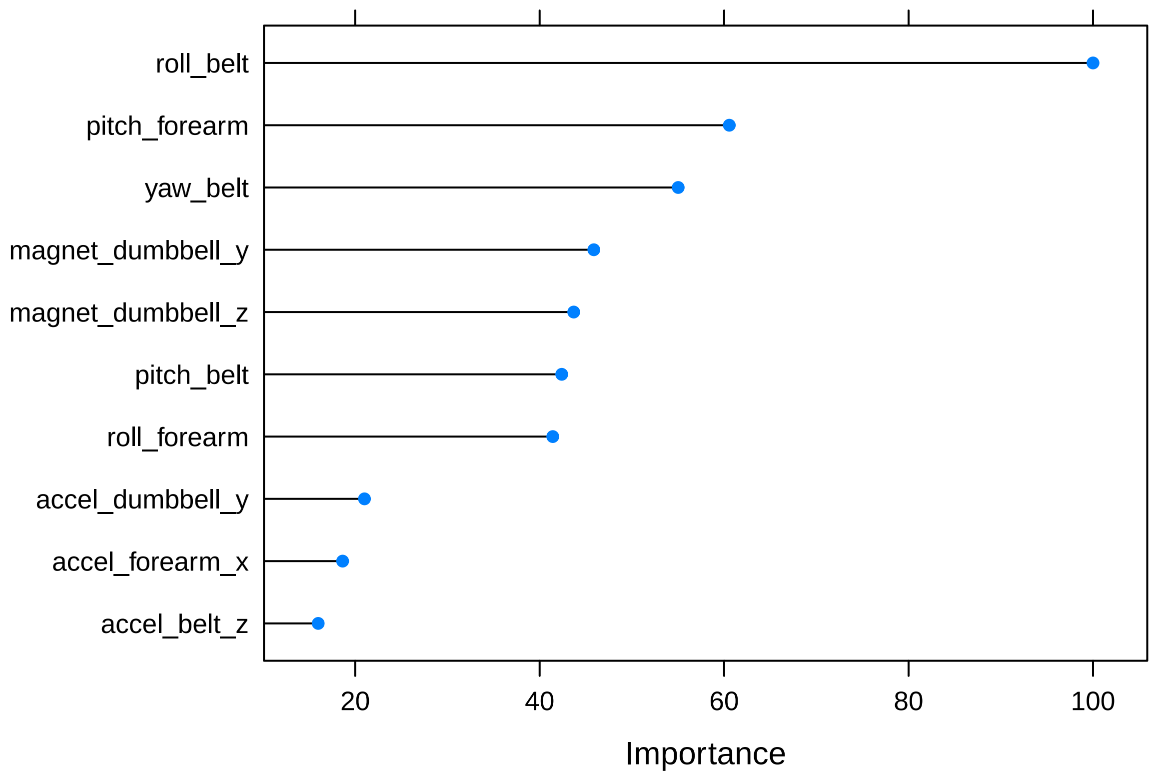
Testing the algorithms

Use the testing data set to predict for the 4 algorithms and then report the training runtime, test accuracy and test error.

```r
pred.RF <- predict(fit.RF, testing)
pred.GBM <- predict(fit.GBM, testing)
pred.QDA <- predict(fit.QDA, testing)
pred.SVM <- predict(fit.SVM, testing)

c.RF <- confusionMatrix(pred.RF, testing$classe)
c.GBM <- confusionMatrix(pred.GBM, testing$classe)
c.QDA <- confusionMatrix(pred.QDA, testing$classe)
c.SVM <- confusionMatrix(pred.SVM, testing$classe)

a.RF <- c.RF$overall[1]
a.GBM <- c.GBM$overall[1]
a.QDA <- c.QDA$overall[1]
a.SVM <- c.SVM$overall[1]
oos.RF <- 1 - a.RF
oos.GBM <- 1 - a.GBM
oos.QDA <- 1 - a.QDA
oos.SVM <- 1 - a.SVM

r1 <- c("RF",  round(t.RF[3],  0), round(a.RF, 3), round(oos.RF, 3))
r2 <- c("GBM", round(t.GBM[3], 0), round(a.GBM, 3), round(oos.GBM, 3))
r3 <- c("QDA", round(t.QDA[3], 0), round(a.QDA,3), round(oos.QDA, 3))
r4 <- c("SVM", round(t.SVM[3], 0), round(a.SVM, 3), round(oos.SVM, 3))
resultSummary <- as.data.frame(rbind(r1,r2,r3,r4))
names(resultSummary) <- c("Method",
                          "training runtime",
                          "Test accuracy",
                          "Test OOS error")
rownames(resultSummary) <- NULL
resultSummary
```

```
##   Method training runtime Test accuracy Test OOS error
## 1     RF              495         0.988          0.012
## 2    GBM              229         0.959          0.041
## 3    QDA                7         0.887          0.113
## 4    SVM              343         0.912          0.088
```

```
plot(varImp(fit.RF), top=10)
```



Conclusion

Although Random forests has the longest training runtime, it is selected because it has the lowest error on the test set (1.2%) and will be used to submit the results. The variance plot for RF sheds some light on which sensors are most important to predict. QDA gives the highest test error (but still reasonable) but is very fast to train compared to the other algorithms.

Appendix : submit section

```
prediction <- as.character(predict(fit.RF, testClean ))
# write prediction files
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(prediction)
```