Machine Learning: Course Project Write-Up

Author: Jens Berkmann

Abstract

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. The goal of this project, is to predict the manner in which they did the exercise. By building a random forest predictor based on just 6 variables we could achieve an accuracy of 91% where training/testing were performed on 25%/75% of the training data set.

Preparation and Exploratory Data Analysis

The data for this project come from the source
http://groupware.les.inf.puc-rio.br/har which is highly appreciated. We load the test and training data sets first and perform an initial cleaning of data. As some data contained "#DIV/0!" we replaced it by "NA" while reading the data in.

```
library(knitr)
library(plyr)
library(stringr)
library(caret)
```

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

```
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(312342)
tr <- read.csv("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",na.strings=c("NA",
te <- read.csv("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv" ,na.strings=c("NA",
```

Inspecting the data revealed that one column was duplicated and just had a similar name. Moreover, keeping both standard deviation and variance of the same underlying variable in the prediction process I did not regard useful. I therefore decided to do some initial deletion of columns as follows:

```
badCols  <- grep("stddev|skewness_roll_belt.1", colnames(tr))
tr  <- tr[, -badCols]
te  <- te[, -badCols]
```

Much data in the data frame is classified as non-numerical despite their obvious numerical nature. I therefore converted the numerical data (columns 8 to 146) to numerical data. I also extracted the numerical variables from the data set for further analysis.

```
for(i in 8:146){
    tr[,i] <- as.numeric(as.character((tr[,i])))
    te[,i] <- as.numeric(as.character((te[,i])))
}
trNum      <- tr[,8:146]
teNum      <- te[,8:146]
```

Further inspection of data revealed that many columns contained lots of NAs. We decided to keep only variables having no NAs. We further tried a reduction of variables by making use of the function nearZeroVar and looked at the correlation of the variables. Only nearly uncorrelated variables were kept. For the parameter choice below I ended up with 6 variables only. We finally added the column containing the desired output *classe* to the data frame containing our numerical predicting variables. This strong reduction of variables was necessary due to the fact that I have a very old laptop with Windows Vista running on it.

```
trNumNASum  <- apply(trNum, 2, function(x) sum(is.na(x)))
trNum       <- trNum[,trNumNASum==0]
ColsNotOk   <- nearZeroVar(trNum, saveMetrics=T)
trNum       <- trNum[,ColsNotOk$nzv==FALSE]
C           <- abs(cor(trNum)); C[lower.tri(C)] <- 0; diag(C) <- 0
corr_thresh <- 0.25#1.0#0.75#0.5#0.33#0.25
ColsNotOk   <- apply(C,2,function(x) any(x > corr_thresh))
trNum       <- trNum[,!ColsNotOk] # columns > threshold deleted
finalTr     <- data.frame(classe=tr$classe, trNum)
```

Model Building

According to the lecture notes a model approach based on random forests usually produces very accurate predictors. Originally I planned to a do random forest approach with all numerical variables included and a PCA-preprocessing step followed by looking at the importance of the individual variables. Based on this I would have tried to prune the tree and come up with an optimized model. HOwever, due to the mentioned limitations of my hardware I was forced to perform this course project in a suboptimum fashion and limit the number of variables and samples a priori, e.g. reduction of feature variables above and the size of the training set to 25%.

We partition the training data set in a true training set and a set for cross validation and build the model based on the training set.

```
inTrain     <- createDataPartition(finalTr$classe, p = 0.25)[[1]]
training    <- finalTr[ inTrain,]
testing     <- finalTr[-inTrain,]
modFit      <- train(classe~.,data=training,method="rf")
```

I also played around initially with different settings of trainControl based on the "cv"-method, could however not significantly increase accuracy of the prediction. The improvement were in the order of 0.2% accuracy. I finally excluded trainControl from the model building step and relied on the default settings.

The in-sample and out-of-sample errors were computed by letting the predictor run on the training set and the testing set.

```
pred_in  <- predict(modFit,training)
cf_in    <- confusionMatrix(pred_in,training$classe)
pred_out <- predict(modFit,testing)
```

```r
cf_out   <- confusionMatrix(pred_out,testing$classe)
cf_in
cf_out
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  950    0    0    0
##          C    0    0  856    0    0
##          D    0    0    0  804    0
##          E    0    0    0    0  902
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.999, 1)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             1.000    1.000    1.000    1.000    1.000
## Specificity             1.000    1.000    1.000    1.000    1.000
## Pos Pred Value          1.000    1.000    1.000    1.000    1.000
## Neg Pred Value          1.000    1.000    1.000    1.000    1.000
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.284    0.194    0.174    0.164    0.184
## Detection Prevalence    0.284    0.194    0.174    0.164    0.184
## Balanced Accuracy       1.000    1.000    1.000    1.000    1.000
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3939   71   12   28   13
##          B  115 2513   99   62   69
##          C   82  191 2360  148   92
##          D   39   55   72 2145   37
##          E   10   17   23   29 2494
##
## Overall Statistics
##
##                Accuracy : 0.914
##                  95% CI : (0.909, 0.919)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
```

```
##                   Kappa : 0.891
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.941    0.883    0.920    0.889    0.922
## Specificity             0.988    0.971    0.958    0.983    0.993
## Pos Pred Value          0.969    0.879    0.821    0.914    0.969
## Neg Pred Value          0.977    0.972    0.983    0.978    0.983
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.268    0.171    0.160    0.146    0.169
## Detection Prevalence    0.276    0.194    0.195    0.160    0.175
## Balanced Accuracy       0.965    0.927    0.939    0.936    0.958
```

Conclusions

It is seen that within the training set the prediction works 100% perfectly while on the cross-validation test set an accuracy of 91% is achieved. We remark once more that due to hardware limitations we stop here and hope that the accuracy is sufficient to pass the testcases for the true testing set.

Preparation of testset for testcase submission

The (true) testset still has to undergo the very same data cleaning procedure as the training set. Parts of it has already been done above. what is left over is the final selection of the used feature columns followed by running the predictor model on the cleaned test set.

```
teNum       <- teNum[,names(teNum) %in% names(finalTr)]
finalTe     <- data.frame(classe=rep("NA", 20),teNum)
answers     <- predict(modFit, newdata=finalTe)
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(answers)
```