Machine learning algorithm to predict activity quality from activity monitors

Asier

Friday, July 24, 2015

Preprocessing data

First, we begin by loading the libraries, including a performance enhancing library in order to reduce the computation time.

```
library(caret)
```

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

```
library(rpart)
#Working in two cores did not work for me
#library(doParallel)
#registerDoParallel(cores=2)
library(randomForest)
```

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

Next, we load the two sets of data assuming that is already downloaded. We also set a seed for reproducibility purposes.

```
testing = read.csv("pml-testing.csv")

training = read.csv("pml-training.csv")

set.seed(145)
```

The NA values play a mayor role in this calculation, an columns either have 0 NA values or almost all of them. This is the reason why we will erase the latter columns from our datasets. Note that the resulting amount of columns is different for each set.

```
table(colSums(is.na(training)))
```

```
##
##     0 19216
##    93    67
```

```
table(colSums(is.na(testing)))
```

```
##
##   0  20
##  60 100
```

```
training<-training[,colSums(is.na(training)) == 0]
testing<-testing[,colSums(is.na(testing)) == 0]
```

We also get rid of columns that do not contribute to accelerometer measurements, for the sake of performance. After this code the amount of columns in both datasets are equal.

```
classe <- training$classe
trainera <- grepl("^X|timestamp|window", names(training))
training <- training[, !trainera]
traincle <- training[, sapply(training, is.numeric)]
traincle$classe <- classe
testera <- grepl("^X|timestamp|window", names(testing))
testing <- testing[, !testera]
testcle <- testing[, sapply(testing, is.numeric)]
```

Now we have to slice the data into a training data set (70%) and a validation data set (30%). We will use the first to train our model and the second to check the validity of our results. In other words, we will perform a cross-validation.

```
inTrain = createDataPartition(traincle$classe, p=0.7, list=F)
train01 = traincle[ inTrain,]
test01 = traincle[-inTrain,]
```

Model creation

We will create our model using the random forests technique for improved accuracy but giving away speed and interpretability, risking overfitting.

```
modFit <- train(classe ~ ., data=train01, method="rf",prox=T )

saveRDS(modFit, file="modFit.rds")
```

Due to the long time it requires to train this set we have saved modFit in a rds file that we proceed now to load instead of training again.

```
modFit = readRDS("modFit.rds")

modFit
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9890506  0.9861396  0.001886162  0.002390656
##   27    0.9891405  0.9862550  0.001828086  0.002311992
##   52    0.9825421  0.9779030  0.004174685  0.005280988
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

In order to check our model we will cross-validate it using the "test" data. Below we show several tables and values indicating the amount of correct and incorrect guesses.

```
pretest<-predict(modFit, newdata = test01)
test01$predRight<-pretest==test01$classe
table(pretest,test01$classe)
```

```
##
## pretest    A    B    C    D    E
##       A 1672    7    0    0    0
##       B    2 1131    4    1    1
##       C    0    1 1020   11    1
##       D    0    0    2  952    1
##       E    0    0    0    0 1079
```

```
confusionMatrix(test01$classe, pretest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    2    0    0    0
##          B    7 1131    1    0    0
##          C    0    4 1020    2    0
##          D    0    1   11  952    0
##          E    0    1    1    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9947
##                  95% CI : (0.9925, 0.9964)
##     No Information Rate : 0.2853
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9933
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9958   0.9930   0.9874   0.9969   1.0000
## Specificity            0.9995   0.9983   0.9988   0.9976   0.9994
## Pos Pred Value         0.9988   0.9930   0.9942   0.9876   0.9972
## Neg Pred Value         0.9983   0.9983   0.9973   0.9994   1.0000
## Prevalence             0.2853   0.1935   0.1755   0.1623   0.1833
## Detection Rate         0.2841   0.1922   0.1733   0.1618   0.1833
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9977   0.9956   0.9931   0.9972   0.9997
```

We obtain a surprising accuracy of %99.47 with a very low p value and Kappa=0.9933. According to these values the random forest model is extremely good in prediction for this set. Note that as we are performing cross-validation overfitting should not be a concern, in principle.

Prediction for testing data set

Now we apply the tested model to the testing data that we downloaded in the first place. This gives us the result we were looking for.

```
results<-predict(modFit, newdata = testcle)
results
```

```
##  [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
```