Machine Learning Project. Human activity recognition in R

Artem Reshetnikov

23 October 2018

Intriduction

Human activity recognition is one of the interesting spheres for machine learning. We can measure human activity by several methods and then try suggest right decisions about conditions of health of person. Smartphones help us to provide all necessary information for making such kind of models. This project is about using randomForests and SVA algorithms in making decision models and comparison of this algorithms.

Data set

As a main data set I decided to use data from UCI Machine Learning Repository .The experiments have been carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) wearing a smartphone (Samsung Galaxy S II) on the waist. Using its embedded accelerometer and gyroscope, we captured 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually. The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.

The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

The training sample contains 7352 cases, the test sample is 2947. Both samples are marked with tags corresponding to six activities: walking, walking_up, walking_down, sitting, standing and laying.

Algorithms

As a basic algorithm for experiments, I chose Random Forest. The choice was based on the fact that RF has a built-in mechanism for assessing the importance of variables. I also decided to try the support vector machine (SVM) to compare the quality of the models for both algorithms.

Packages

For realization of idea of project I used Caret and randomForest package

```
library(caret)
library(randomForest)
```

Loading data

The problem of dataset that all information divided in a lot of files and for loading whole data we need to load data and then combine it.

```
a_names <- read.table("./UCI HAR Dataset/activity_labels.txt", stringsAsFactors=F)
v_names <- read.table("./UCI HAR Dataset/features.txt",  stringsAsFactors=F)

## train data
tr_data <- read.table("./UCI HAR Dataset/train/X_train.txt")
tr_activity <- read.table("./UCI HAR Dataset/train/y_train.txt")

## test data
tt_data <- read.table("./UCI HAR Dataset/test/X_test.txt")
tt_activity <- read.table("./UCI HAR Dataset/test/y_test.txt")
```

It was necessary to prohibit the automatic conversion of rows into factor variables, as it turned out that there are duplicate variable names. In addition, the names contained incorrect characters. This problem was solved by the following construction with the sapply () function.

```
editN <- function(x) {
  y <- v_names[x,2]
  y <- sub("BodyBody", "Body", y) #subs duplicate names
  y <- gsub("-", "", y) # global subs for dash
  y <- gsub(",", "_", y) # global subs for comma
  y <- sub("\\()", "", y) # subs for ()
  y <- gsub("\\)", "", y) # global subs for
  y <- sub("\\(", "_", y) # subs for (
  y <- paste0("v",v_names[x,1], "_",y) #add number, prevent duplicat.
  return(y)
}
## edit names
new_names <- sapply(1:nrow(v_names), editN)

## work with training data
names(tr_data)<-new_names
tr_data <- cbind(tr_activity[,1], tr_data)
names(tr_data)[1]<-"Activity"

## work with test data
names(tt_data)<-new_names
tt_data <- cbind(tt_activity[,1], tt_data)
names(tt_data)[1]<-"Activity"

a_names[2,2] <- substr(a_names[2,2], 1, 10) #cut long names
a_names[3,2] <- substr(a_names[3,2], 1, 12)

tr_data <- transform(tr_data, Activity=factor(Activity))
tt_data <- transform(tt_data, Activity=factor(Activity))
levels(tr_data[,1])<-a_names[,2]
levels(tt_data[,1])<-a_names[,2]
```

Preparation of data

After I prepared a training and test sample. The question arose of the need for data preprocessing. Using the range () function in a loop, I calculated a range of values. It turned out that all signs are within [-1,1] and it means that neither normalization nor scaling is needed.

```
rang <- sapply(new_names, function(x){
    range(tr_data[,x])
})
min(rang)
```

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

```
max(rang)
```

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

Learning

randomeForest

It provides k-fold cross-validation with k = 5. By default, three models are trained with different mtry values (the number of signs that are randomly selected from the entire set and are considered as a candidate for each branching tree), and then the best one is selected by accuracy. The number of trees for all models is the same ntree = 100.

To determine the quality of the model on the test sample, I took the confusionMatrix (x, y) function from caret, where x is the vector of predicted values, and y is the vector of values from the test sample.

```
##RM
fitControl <- trainControl(method="cv", number=5)
set.seed(123)
tstart <- Sys.time()
forest_full <- train(Activity~., data=tr_data,
                        method="rf", do.trace=10, ntree=100,
                        trControl = fitControl)
```

```
## ntree      OOB      1      2      3      4      5      6
##    10:  19.95% 10.13% 13.03% 16.35% 29.70% 27.43% 20.14%
##    20:  12.70%  4.59%  8.86% 10.14% 22.64% 17.38% 10.84%
##    30:   9.78%  3.57%  6.18%  7.60% 20.12% 13.74%  6.13%
##    40:   7.26%  2.75%  4.55%  7.22% 14.29% 11.10%  3.11%
##    50:   6.60%  2.24%  3.73%  5.20% 14.29% 10.46%  2.75%
##    60:   6.12%  1.83%  3.26%  5.07% 13.31%  9.92%  2.49%
##    70:   6.19%  2.55%  3.03%  5.20% 13.80%  9.65%  2.13%
##    80:   5.32%  2.24%  2.56%  4.82% 12.24%  8.37%  1.15%
##    90:   5.19%  2.04%  2.68%  5.20% 11.56%  8.19%  1.07%
##   100:   4.85%  1.73%  2.33%  4.44% 11.66%  7.55%  0.89%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.77%  5.57%  5.52%  6.61%  8.06%  9.27%  0.00%
##    20:   3.52%  3.16%  2.45%  4.06%  4.86%  6.64%  0.00%
##    30:   2.75%  2.34%  1.40%  2.66%  4.76%  5.19%  0.00%
##    40:   2.58%  2.45%  1.28%  2.28%  4.47%  4.82%  0.00%
##    50:   2.45%  2.34%  1.17%  2.03%  4.28%  4.64%  0.00%
##    60:   2.31%  1.94%  1.05%  1.52%  4.28%  4.64%  0.09%
##    70:   2.28%  1.73%  0.93%  1.65%  4.57%  4.46%  0.00%
##    80:   2.18%  1.73%  0.70%  1.65%  4.37%  4.28%  0.00%
##    90:   2.11%  1.63%  0.70%  1.52%  4.28%  4.19%  0.00%
##   100:   2.18%  1.73%  0.82%  1.39%  4.76%  4.00%  0.00%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.36%  4.53%  6.60%  5.36%  8.01%  8.07%  0.09%
##    20:   4.03%  4.18%  4.31%  3.93%  5.93%  6.01%  0.09%
```

```
##    30:   3.50%  3.36%  3.85%  3.04%  5.54%  5.28%  0.09%
##    40:   3.52%  3.26%  3.38%  2.92%  5.64%  5.82%  0.09%
##    50:   3.21%  2.45%  3.50%  3.17%  4.96%  5.28%  0.09%
##    60:   3.09%  2.45%  3.26%  3.04%  4.96%  4.91%  0.09%
##    70:   3.08%  2.34%  3.15%  3.04%  5.15%  4.82%  0.09%
##    80:   3.03%  2.34%  3.26%  3.04%  5.05%  4.55%  0.09%
##    90:   3.13%  2.55%  3.38%  3.04%  5.34%  4.55%  0.09%
##   100:   3.08%  2.24%  3.26%  2.92%  5.34%  4.73%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:  18.65%  9.35% 14.57% 14.47% 28.28% 26.64% 16.26%
##    20:  11.04%  3.78%  8.39%  8.49% 21.21% 16.92%  6.13%
##    30:   9.18%  2.24%  6.18%  6.34% 17.80% 15.01%  5.95%
##    40:   7.26%  2.14%  4.08%  5.83% 15.66% 11.37%  3.46%
##    50:   6.55%  2.04%  3.38%  6.21% 14.49%  9.28%  3.20%
##    60:   5.75%  1.84%  3.15%  5.58% 12.65%  8.37%  2.40%
##    70:   5.29%  1.94%  2.80%  5.83% 12.26%  6.92%  1.78%
##    80:   5.05%  1.94%  2.91%  4.56% 11.96%  7.01%  1.51%
##    90:   4.91%  2.04%  3.26%  4.31% 11.28%  6.64%  1.60%
##   100:   4.59%  1.84%  2.80%  4.56% 10.70%  6.10%  1.33%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.29%  3.20%  5.64%  6.17%  7.62%  9.17%  0.27%
##    20:   2.84%  2.24%  2.21%  2.66%  4.77%  5.00%  0.09%
##    30:   2.53%  2.04%  1.17%  2.79%  4.28%  4.64%  0.18%
##    40:   2.28%  1.63%  0.93%  2.28%  4.18%  4.37%  0.09%
##    50:   2.11%  1.73%  1.40%  2.15%  3.79%  3.46%  0.09%
##    60:   2.18%  1.43%  1.40%  2.15%  4.09%  3.82%  0.09%
##    70:   2.18%  1.53%  1.52%  1.90%  4.09%  3.82%  0.09%
##    80:   2.01%  1.33%  1.28%  1.77%  4.09%  3.37%  0.09%
##    90:   2.06%  1.22%  1.40%  1.77%  4.09%  3.64%  0.09%
##   100:   2.06%  1.43%  1.40%  2.15%  3.79%  3.46%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.24%  4.54%  5.39%  6.01%  8.15%  7.72%  0.09%
##    20:   3.52%  2.96%  3.50%  2.79%  7.00%  4.82%  0.09%
##    30:   3.42%  2.45%  3.96%  2.92%  6.52%  4.73%  0.09%
##    40:   3.32%  2.45%  3.38%  3.04%  6.61%  4.46%  0.09%
##    50:   3.13%  2.24%  3.38%  2.66%  6.42%  4.09%  0.09%
##    60:   3.23%  2.45%  3.38%  2.92%  6.52%  4.19%  0.09%
##    70:   3.35%  2.35%  3.61%  2.79%  6.81%  4.55%  0.09%
##    80:   3.32%  2.35%  3.61%  2.79%  6.71%  4.46%  0.09%
##    90:   3.33%  2.35%  3.61%  2.79%  6.61%  4.64%  0.09%
##   100:   3.25%  2.24%  3.38%  2.66%  6.71%  4.46%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:  20.44%  7.37% 13.72% 16.05% 30.51% 30.73% 20.71%
##    20:  13.23%  4.18%  8.27%  9.38% 25.17% 19.38% 10.67%
##    30:  10.03%  3.57%  5.59%  6.97% 19.92% 15.47%  6.84%
##    40:   8.19%  3.47%  4.19%  5.96% 16.33% 13.74%  4.09%
##    50:   7.41%  2.75%  3.61%  5.58% 16.03% 11.83%  3.47%
##    60:   6.94%  2.34%  3.38%  4.69% 15.35% 11.46%  3.11%
##    70:   6.53%  2.04%  2.91%  4.82% 14.97% 10.28%  3.02%
##    80:   5.76%  2.04%  2.56%  3.68% 13.02%  9.83%  2.31%
##    90:   5.64%  2.14%  2.68%  3.68% 13.02%  9.10%  2.22%
##   100:   5.24%  2.04%  2.91%  3.30% 11.76%  9.01%  1.51%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.55%  4.00%  5.65%  7.42%  6.89%  9.34%  0.54%
```

```
##    20:   3.37%  2.75%  2.44%  3.42%  5.83%  5.46%  0.27%
##    30:   2.86%  2.04%  1.63%  2.79%  5.73%  4.55%  0.27%
##    40:   2.53%  1.83%  1.51%  2.53%  5.05%  4.09%  0.09%
##    50:   2.36%  2.04%  0.93%  1.77%  5.05%  3.91%  0.18%
##    60:   2.31%  1.94%  0.58%  2.03%  5.05%  4.00%  0.00%
##    70:   2.19%  2.04%  0.47%  2.03%  4.96%  3.46%  0.00%
##    80:   2.09%  1.63%  0.47%  1.65%  4.76%  3.64%  0.09%
##    90:   2.16%  1.83%  0.58%  2.15%  4.66%  3.46%  0.09%
##   100:   2.19%  1.73%  0.58%  2.03%  5.15%  3.37%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.57%  5.05%  6.60%  6.57%  8.46%  7.52%  0.00%
##    20:   3.94%  3.06%  3.61%  4.06%  7.09%  6.01%  0.00%
##    30:   3.76%  3.06%  3.61%  3.42%  7.00%  5.46%  0.09%
##    40:   3.66%  2.96%  3.26%  3.30%  6.90%  5.55%  0.00%
##    50:   3.59%  3.06%  3.26%  3.04%  6.51%  5.64%  0.00%
##    60:   3.54%  3.06%  3.14%  3.04%  6.71%  5.19%  0.09%
##    70:   3.43%  2.55%  3.03%  3.04%  6.61%  5.28%  0.09%
##    80:   3.21%  2.45%  3.03%  2.66%  6.61%  4.46%  0.09%
##    90:   3.26%  2.65%  2.91%  2.79%  6.12%  5.00%  0.09%
##   100:   3.20%  2.75%  2.68%  2.41%  6.32%  4.82%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:  18.23%  8.94% 12.08% 13.59% 30.92% 24.66% 16.38%
##    20:  11.63%  4.18%  7.34%  7.86% 24.30% 17.45%  6.76%
##    30:   8.94%  3.47%  5.36%  7.86% 17.98% 13.55%  4.44%
##    40:   7.28%  3.06%  3.73%  6.34% 16.23% 10.18%  3.29%
##    50:   6.63%  2.24%  3.61%  5.96% 14.38%  9.73%  3.11%
##    60:   5.92%  2.04%  3.26%  4.94% 13.61%  8.64%  2.31%
##    70:   5.46%  2.14%  2.91%  4.94% 12.34%  7.73%  2.13%
##    80:   5.29%  2.24%  3.15%  4.94% 11.95%  7.64%  1.42%
##    90:   5.07%  1.83%  3.03%  4.56% 11.66%  7.55%  1.33%
##   100:   5.07%  1.83%  3.03%  4.82% 11.86%  7.73%  0.80%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.61%  4.93%  7.20%  6.68%  7.09%  8.34%  0.18%
##    20:   3.45%  2.45%  3.85%  3.42%  5.34%  5.64%  0.18%
##    30:   2.77%  2.65%  2.45%  3.30%  3.79%  4.55%  0.09%
##    40:   2.62%  2.24%  1.98%  2.53%  4.86%  4.00%  0.09%
##    50:   2.64%  2.34%  1.86%  2.92%  4.76%  3.91%  0.09%
##    60:   2.35%  1.94%  1.63%  2.41%  4.76%  3.27%  0.09%
##    70:   2.28%  1.94%  1.17%  2.53%  4.66%  3.27%  0.09%
##    80:   2.19%  1.83%  1.28%  2.15%  4.47%  3.27%  0.09%
##    90:   2.13%  1.83%  1.05%  2.03%  4.66%  3.00%  0.09%
##   100:   2.18%  1.73%  1.17%  2.03%  4.86%  3.09%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.49%  3.60%  6.24%  7.58%  7.49%  8.69%  0.09%
##    20:   3.96%  3.06%  4.90%  4.56%  6.32%  5.36%  0.09%
##    30:   3.77%  2.85%  3.73%  4.69%  6.32%  5.36%  0.09%
##    40:   3.77%  2.75%  3.61%  5.20%  6.22%  5.27%  0.09%
##    50:   3.42%  2.75%  2.91%  4.44%  5.83%  4.82%  0.09%
##    60:   3.42%  2.75%  2.80%  4.44%  5.83%  4.91%  0.09%
##    70:   3.26%  2.75%  2.68%  4.31%  5.34%  4.73%  0.09%
##    80:   3.21%  2.75%  3.03%  3.93%  5.64%  4.18%  0.09%
##    90:   3.08%  2.65%  2.91%  3.30%  5.34%  4.36%  0.09%
##   100:   3.03%  2.65%  3.15%  3.17%  5.15%  4.18%  0.09%
## ntree      OOB      1      2      3      4      5      6
```

```
##    10:  18.54%  9.14% 12.63% 14.03% 27.49% 27.85% 17.09%
##    20:  12.04%  4.38%  7.10%  8.26% 21.48% 19.51%  9.24%
##    30:   9.04%  3.47%  4.54%  6.73% 18.95% 14.19%  4.88%
##    40:   7.74%  3.06%  3.38%  5.84% 16.03% 12.92%  3.82%
##    50:   6.82%  1.83%  3.26%  5.46% 16.03% 10.37%  2.93%
##    60:   6.10%  2.55%  2.44%  5.84% 14.58%  8.55%  2.04%
##    70:   6.05%  1.94%  2.79%  5.08% 14.09%  9.83%  1.78%
##    80:   5.42%  1.94%  2.44%  5.08% 12.73%  8.83%  0.98%
##    90:   4.98%  1.83%  2.21%  4.95% 11.47%  8.28%  0.71%
##   100:   4.66%  1.73%  2.10%  3.81% 11.37%  7.64%  0.71%
## ntree      OOB      1      2      3      4      5      6
##    10:   6.01%  4.63%  6.50%  6.95%  9.42%  8.93%  0.18%
##    20:   3.59%  2.45%  2.91%  3.81%  6.03%  6.37%  0.00%
##    30:   2.89%  1.94%  1.63%  2.66%  6.22%  4.73%  0.00%
##    40:   2.74%  2.14%  1.86%  2.16%  5.73%  4.28%  0.09%
##    50:   2.72%  2.04%  1.86%  2.66%  5.34%  4.37%  0.00%
##    60:   2.58%  1.94%  1.63%  2.66%  5.15%  4.09%  0.00%
##    70:   2.47%  1.94%  1.51%  2.28%  5.15%  3.82%  0.00%
##    80:   2.38%  1.83%  1.28%  2.16%  5.05%  3.82%  0.00%
##    90:   2.36%  1.73%  1.28%  2.41%  5.05%  3.64%  0.00%
##   100:   2.30%  1.63%  1.05%  2.54%  5.05%  3.46%  0.00%
## ntree      OOB      1      2      3      4      5      6
##    10:   4.85%  4.34%  5.90%  6.23%  7.26%  6.12%  0.09%
##    20:   4.18%  3.47%  4.89%  4.19%  7.68%  5.19%  0.09%
##    30:   3.74%  2.65%  4.07%  3.43%  7.39%  5.00%  0.09%
##    40:   3.67%  2.75%  3.96%  3.05%  7.48%  4.82%  0.09%
##    50:   3.55%  2.75%  3.49%  3.43%  7.19%  4.55%  0.09%
##    60:   3.35%  2.75%  3.26%  3.05%  7.00%  4.09%  0.09%
##    70:   3.25%  2.45%  3.61%  2.79%  6.61%  4.09%  0.09%
##    80:   3.15%  2.24%  3.49%  2.79%  6.61%  3.82%  0.09%
##    90:   3.09%  2.34%  3.14%  2.66%  6.61%  3.82%  0.09%
##   100:   3.20%  2.75%  3.49%  2.66%  6.71%  3.64%  0.09%
## ntree      OOB      1      2      3      4      5      6
##    10:   5.02%  3.30%  5.18%  6.66%  7.10%  7.79%  0.65%
##    20:   2.92%  2.12%  2.52%  4.16%  4.43%  4.29%  0.36%
##    30:   2.54%  2.04%  2.05%  2.74%  4.20%  4.00%  0.28%
##    40:   2.31%  2.12%  1.58%  2.64%  3.89%  3.64%  0.07%
##    50:   2.04%  1.79%  1.21%  2.23%  3.27%  3.64%  0.07%
##    60:   2.08%  1.88%  1.30%  2.33%  3.50%  3.49%  0.00%
##    70:   1.99%  2.12%  0.93%  1.83%  3.50%  3.42%  0.00%
##    80:   1.77%  1.79%  0.84%  1.42%  3.34%  3.06%  0.00%
##    90:   1.92%  1.96%  0.93%  1.42%  3.73%  3.28%  0.00%
##   100:   2.00%  2.12%  1.03%  1.52%  3.89%  3.28%  0.00%
```

```
tend <- Sys.time()
print(tend-tstart)
```

```
## Time difference of 17.57281 mins
```

```
## predict and control Accuracy

prediction <- predict(forest_full, newdata=tt_data)
cm <- confusionMatrix(prediction, tt_data$Activity)
print(cm)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      WALKING WALKING_UP WALKING_DOWN SITTING STANDING LAYING
##   WALKING           483         35           17       0        0      0
##   WALKING_UP          5        430           39       0        0      0
##   WALKING_DOWN        8          6          364       0        0      0
##   SITTING             0          0            0     428       45      0
##   STANDING            0          0            0      63      487      0
##   LAYING              0          0            0       0        0    537
##
## Overall Statistics
##
##                Accuracy : 0.926
##                  95% CI : (0.916, 0.9352)
##     No Information Rate : 0.1822
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9111
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: WALKING Class: WALKING_UP Class: WALKING_DOWN
## Sensitivity                  0.9738            0.9130              0.8667
## Specificity                  0.9788            0.9822              0.9945
## Pos Pred Value               0.9028            0.9072              0.9630
## Neg Pred Value               0.9946            0.9834              0.9782
## Prevalence                   0.1683            0.1598              0.1425
## Detection Rate               0.1639            0.1459              0.1235
## Detection Prevalence         0.1815            0.1608              0.1283
## Balanced Accuracy            0.9763            0.9476              0.9306
##                      Class: SITTING Class: STANDING Class: LAYING
## Sensitivity                  0.8717          0.9154        1.0000
## Specificity                  0.9817          0.9739        1.0000
## Pos Pred Value               0.9049          0.8855        1.0000
## Neg Pred Value               0.9745          0.9812        1.0000
## Prevalence                   0.1666          0.1805        0.1822
## Detection Rate               0.1452          0.1653        0.1822
## Detection Prevalence         0.1605          0.1866        0.1822
## Balanced Accuracy            0.9267          0.9447        1.0000
```

SVM

Package Caret supports several SVM implementations. I chose svmRadial (SVM with a core - a radial basis function), it is often used with caret, and is a general tool when there is no special information about the data. To train a model with SVM, it is enough to change the value of the method parameter in the train () function to svmRadial and remove the do.trace and ntree parameters. The algorithm showed the following results: Accuracy on the test sample - 0.952.

```
## SVM, full set
fitControl <- trainControl(method="cv", number=5)
tstart <- Sys.time()
svm_full <- train(Activity~., data=tr_data,
                  method="svmRadial",
```

```
                    trControl = fitControl)
tend <- Sys.time()
print(tend-tstart)
```

```
## Time difference of 7.954228 mins
```

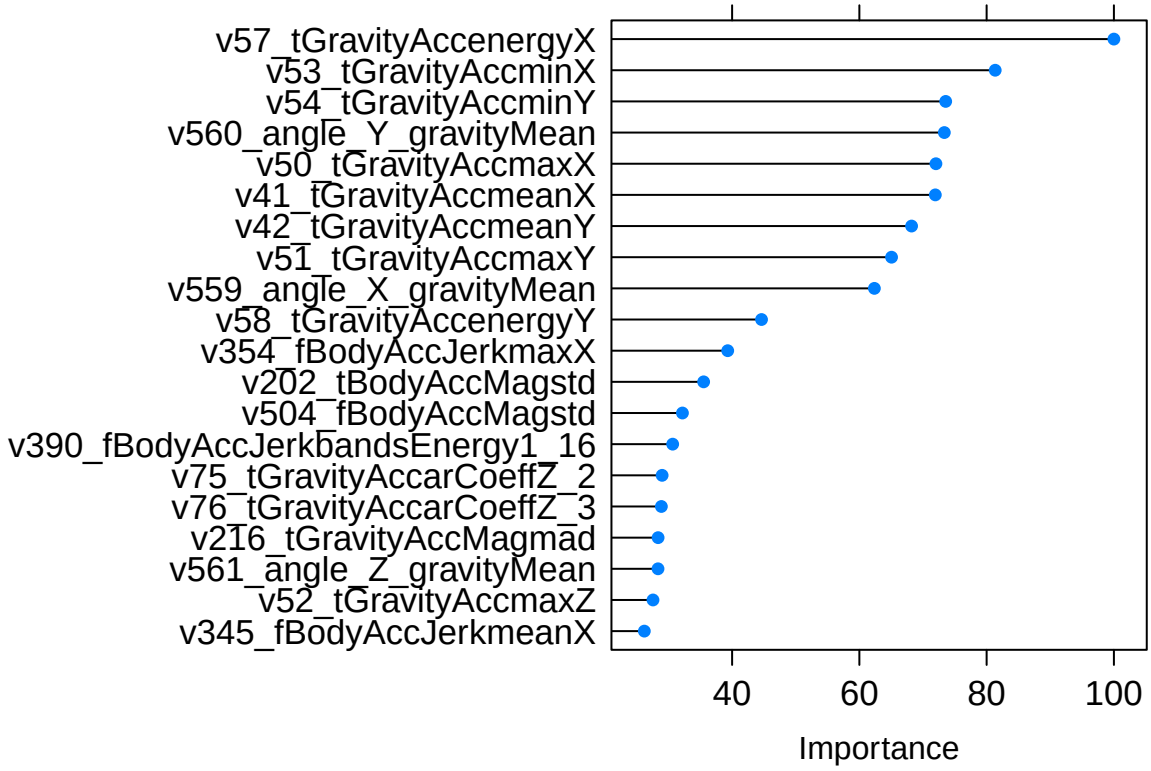
```
## predict and control Accuracy
prediction <- predict(svm_full, newdata=tt_data)
cm <- confusionMatrix(prediction, tt_data$Activity)
print(cm)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      WALKING WALKING_UP WALKING_DOWN SITTING STANDING LAYING
##   WALKING           482         13            7       0        0      0
##   WALKING_UP          5        457           32       1        0      0
##   WALKING_DOWN        9          1          381       0        0      0
##   SITTING             0          0            0     442       26      0
##   STANDING            0          0            0      46      506      0
##   LAYING              0          0            0       2        0    537
##
## Overall Statistics
##
##                Accuracy : 0.9518
##                  95% CI : (0.9435, 0.9593)
##     No Information Rate : 0.1822
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9421
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: WALKING Class: WALKING_UP Class: WALKING_DOWN
## Sensitivity                  0.9718            0.9703              0.9071
## Specificity                  0.9918            0.9847              0.9960
## Pos Pred Value               0.9602            0.9232              0.9744
## Neg Pred Value               0.9943            0.9943              0.9847
## Prevalence                   0.1683            0.1598              0.1425
## Detection Rate               0.1636            0.1551              0.1293
## Detection Prevalence         0.1703            0.1680              0.1327
## Balanced Accuracy            0.9818            0.9775              0.9516
##                      Class: SITTING Class: STANDING Class: LAYING
## Sensitivity                  0.9002          0.9511        1.0000
## Specificity                  0.9894          0.9810        0.9992
## Pos Pred Value               0.9444          0.9167        0.9963
## Neg Pred Value               0.9802          0.9891        1.0000
## Prevalence                   0.1666          0.1805        0.1822
## Detection Rate               0.1500          0.1717        0.1822
## Detection Prevalence         0.1588          0.1873        0.1829
## Balanced Accuracy            0.9448          0.9660        0.9996
```

Importance of variables

We can get the results of the built-in assessment of the importance of variables in RF using the varImp () function from the caret package. The construction of the plot view (varImp (model), 20) will reflect the relative importance of the first 20 signs:

```
plot(varImp(forest_full),20, scales=list(cex=1.1))
```



If we look closely at the plot, we can make sure that there are no data from the gyroscope among the most important variables.

Results

1. We received two good models with high level of accuracy. Human activity recognition is one of ways to make life of people more comfortable. With the help of such models we can significantly improve the performance of fitness applications by constantly tracking a person's physical activity without additional actions on his part. It means that's the user does not need to switch application modes when he goes on a run or cycling. However, this can be realized with the use of sensors integrated with the application.Such models can help people prone to migraine attacks, epilepsy or fainting. Applications of smartphone is used to monitor patients and prevent critical situations, monitors their neurobiological signals through the sensors of smartphones and processes information using a machine learning model. If there is a risk of an attack, the application "signals" this by sending a message to who may come to the rescue.
2. SVM much more effectively than Random Forest in that task, it works twice as fast and gives a better model. Of course it depend of dataset. randomForest is more effectively for datasets with the less number of variables.
3. It would be correct to understand the physical meaning of variables. According to results of measuring

of importance of variables we can see, that some parameters of measuring are not so important for experiment in general.